Systemresurspartitionering under Linux

Johan Metzner, Erik Persson

Examensarbete för 30 hp Institutionen för datavetenskap, Naturvetenskapliga fakulteten, Lunds universitet

Thesis for a diploma in computer science, 30 ECTS credits Department of Computer Science, Faculty of Science, Lund University

Sammanfattning

Vi presenterar i detta examensarbete verktyg för mätning av systemresursanvändning och verktyg för systemresursstyrning på inbäddade Linux-system i form av övervakningskameror från Axis Communications AB. Vi presenterar också en implementation av systemresursstyrning på målarkitekturen. De systemresurser som arbetet huvudsakligen inriktats mot är CPU-tid, internminne och nätverksbandbredd.

Vi fann att vanligen använda verktyg, som `top` och `iperf`, var adekvata för värdering av systemresursutnyttjande.

Gruppschemaläggning med Linuxkärnans cgrupper kan tillhandahålla såväl tillräcklig CPU-tid som låg variabiliet i periodiska signaler.
Variabiliteten i periodiska signaler var vid hård schemaläggning av CPU-tid med VServer högre än för cgrupper.
Lägst nivå av signalvariabilitet erhölls med realtidsschemaläggning.
Realtidsschemaläggning kan baseras på cgrupper. Såväl cgrupper som VServer medger minnesbegränsning för grupper av processer.

Att fullt implementera systemresursstyrning med cgrupper på målarkitekturen bedöms vara möjligt, men kräver förändringar av viss programvara på målarkitekturen. Alternativt kan i vissa fall annan programvara ersätta aktuell programvara.

Ett antal intressanta frågeställningar, som t.ex. en värdering av OpenVZ som resursstyrningsverktyg, kvarstår.

Abstract

In this thesis we present tools to meassure system resource usage and tools to partition system resources on embedded systems from Axis Communications AB. We also present an implementation of a resource partitioning system for the target architechture. The main focus has been on CPU load, RAM, and network bandwidth.

Common tools for system resource usage messurements were in most cases adequate.

Group scheduling using cgroups, which are part of the standard Linux kernel, could maintain adequate CPU load as well as low variability in periodic signals. The variability in periodic signals were higher with hard CPU scheduling using VServer compared to cgroups. The lowest levels of signal variability was achieved with real time scheduling of processes. Cgroups can be used as a basis for real time scheduling. Both cgroups and VServer can limit memory usage for groups of processes.

A full implementation of system resource partitioning with cgroups on the target architechture could be possible, but changes in the software on the target architechture may in some cases be necessary. As an alternative, cgroup aware software may in some cases replace current software.

A number of questions, such as the performance of system resource patitioning with OpenVZ, remain.

Acknowledgement

Vi tackar våra handledare Ferenc Belik och Martin Samuelsson för deras hjälp och stöd under arbetets gång. Vi uttrycker också vårt tack till Axis Communications AB och anställda på Axis Communications AB som hjälpt oss under arbetes gång.

Ordlista

fork

`fork` är ett systemanrop som skapar en, med vissa undantag, exakt kopia av den process som gör systemanropet. När en process anropar `fork` så skapas sålunda en ny process som är en kopia av den gamla.

sourca

I skalprogramsammanhang använder man ibland termen sourca (eng: source). Detta innebär att en skalkommandofil (eng: shell script file) inkluderar en annan fil med skalkommandon och exekverar kommandona i aktuellt skal.

pipe

En pipe är ett sätt att föra över data från en process till en annan, därav jämförelsen med ett rör (eng: pipe). Det fungerar i praktiken genom att utdata från en process hamnar i en anonym buffert som matas in som indata till nästa process. En pipe skapas genom systemanropet `pipe` vilket skapar två fildeskriptorer, varefter fildeskriptorerna för *stdin* och *stdout* dupliceras. En namngiven pipe (eng: named pipe, FIFO special file) fungerar enligt samma princip, fast i detta fall syns bufferten som en fil i filsystemet. Processer kan skriva till och läsa från denna fil precis som en vanlig fil.

daemon

En daemon är en bakgrundsprocess utan någon kontrollerande terminal, d.v.s. utanför direkt kontroll av användaren, och som i allmänhet är barn till `init`. Oftast ansvarar en daemon för funktionalitet som den i t.ex. en http-server eller en smtp-server. Daemonprocessen sköter sig sålunda själv; användaren av systemet ser inte processen verka direkt. I Linux skapas daemonprocesser traditionellt genom systemanropet `fork` varefter föräldraprocessen avslutas. Därigenom blir init-processen förälder till daemonen. Ofta så stängs eller omdirigeras även fildeskriptorerna för *stdin* och *stdout*. Systemanropet `daemon` gör, om det förekommer, allt detta. I Unix brukar daemonprocesser ha ett "d" i slutet av sitt namn, t.ex. så kan en ftp-daemon kallas ftpd.

nice

En process nice-värde (eng: niceness) bestämmer processens statiska prioritet vid normal schemaläggning. Den statiska prioriteten är en av oftast flera faktorer som bestämmer en process dynamiska prioritet, vilken i sin tur är den prioritet som bestämmer hur processen ska shemaläggas i förhållande till andra processer.

Innehåll

Figurer

1 Introduktion

En kritiskt funktionalitet i övervakningskameror är att utan varaktig störning kunna skicka bilder. Det är därför viktigt att annan funktionalitet på kameran inte påtagligt stör bildströmmen. Därmed är det av intresse att på något sätt avsätta delar av systemets resurser till prioriterade processer. [22]

Tredjepartsprogramvara blir alltmer förekommande på inbäddade system (eng: embedded devices) som övervakningskameror, inte minst då detta kan innebära marknadsmässiga fördelar för den aktuella produkten. Förekomsten av sådan programvara medför dock att behovet av att garantera systemresurser för viss funktionalitet ökar eftersom systemtillverkaren inte längre kan bibehålla samma kontroll över vilken programvara som används på systemet och eftersom antalet möjliga kombinationer av programvara på systemet ökar. [22]

På inbäddade system, d.v.s. datorsystem konstruerade för en eller ett fåtal dedikerade funktioner, ofta med begränsad CPU-kraft och begränsad mängd interminne, är det önskvärt om kritisk funktionalitet kan garanteras åtminstone i mjuk bemärkelse. En vanlig metod för att garantera svarstider är att använda någon form av realtidsfunktionalitet. Med realtid avses strikta prioriteter samt garanterade svarstider. [1, 2, 11, 13]

En garanti kan möjligen också åstadkommas genom att partitionera delar av systemresurserna så att mer kritiska processer garanterades en viss andel av dessa, men utan att svälta annan funktionalitet; alltså någon form av systemresurskvoter (eng: system resource quotas).

Tredjepartsprogramvara gör det sålunda svårare att garantera att viktiga processer tilldelas relevanta systemresurser. Standardmetoden för att garantera CPU-tid och begränsade latenstider för högprioriterade processer är att använda något realtidsoperativsystem. Finns inte några formella latenskrav föreligger dock inte nödvändigtvis något behov av att använda ett realtidsoperativsystem, och det kan möjligen vara adekvat att använda en partitionering av systemresurser för att på så sätt tilldela processer en genomsnittlig CPU-tidsandel. Även andra resurser än CPU-tid kan behöva liknande hantering, som t.ex. minne och nätverksbandbredd.

Linux-kärnan har stöd för s.k. task control groups[21] (cgrupper, cgroups), vilka är grupper av processer som kan tilldelas en andel av tillgänglig CPU-tid, tillgängligt internminne m.m. Linux-kärnan har även stöd för olika schemaläggningspolicyer (eng: scheduling policies), varibland två stycken realtidspolicyer, som tillhandahåller *mjuk realtid* (se 2.2.1), ingår. Vissa verktyg för virtualisering på operativsystemsnivå, som VServer,

tillhandahåller även systemresursstyrning utöver vad Linux-kärnan normalt medger.

1.1 Syfte och mål

Målet är att presentera verktyg och tekniker för att mäta och styra systemresursanvändningen, i form av minne, CPU-tid, intern bandbredd och nätverksbandbredd, per process eller per grupp av processer på inbäddade Linux-system huvudsakligen på den av Axis Communication AB:s (Axis) egenutvecklade ARTPEC-3-kretsen. ARTPEC-3 används i Axis P3301 övervakningskamera, på vilken vi ska undersöka möjligheterna att använda verktygen och teknikerna.

ARTPEC-3 är en implementation av Axis egenutvecklade CRISv32-arkitektur och har på en krets förutom CPU även integrerat ett flertal övriga funktioner.

Under aktuell arkitektur och systemprogramvara så anses att CPU-tid är den i praktiken mest begränsande resursen. Minnesanvändning kan i samband med tillåtande av 3:e-partsprogramvara också vara en teoretiskt begränsande faktor, men detta är i någon mån ett mindre problem då en förändring av systemet på detta område är mer tillgänglig. Trots detta är det av intresse att utvärdera metoder för styrning av minnesanvändning om så är möjligt. På samma sätt kan det vara av vikt att begränsa nätverksbandbredd samt i mån av möjlighet även intern bandbredd. Möjligheterna till resursstyrning av I/O mot sekundärminne kan även vara av översiktligt intresse. [22]

Lastverktyg för att belasta olika systemresurser måste utvärderas och portas till målarkitekturen och testfall som visar hur systemet beter sig under last måste utarbetas. Detta kräver en utvärdering av verktyg för att mäta olika systemresurser, samt en undersökning av huruvida dessa verktyg är möjliga att porta till aktuell målarkitektur. Axis har kamerasystem som använder sig av olika arkitekturer, och Linux-miljöerna i dessa kan variera med avseende på t.ex. vilka delar av Linux-kärnan som portats till systemen. Viss funktionalitet kan därmed tillhandahållas av en arkitektur men inte av en annan, samtidigt som i görligaste mån verktyg ska vara kompatibla med samtliga arkitekturer.

När de begränsade systemresurserna har hittats, är målet att tekniker ska utvecklas för att hantera dessa systemresurser så att kritiska processer kan garanteras tillräckliga systemresurser.

Målen för arbetet är sålunda att:

- utvärdera, anpassa och eventuellt utveckla verktyg för att mäta systemresursåtgång med inriktning mot Axis övervakningskameror,
- utveckla och utvärdera belastningsverktyg och testfall som lastar CPU, minne, nätverksbandbredd och intern bandbredd,
- utvärdera och anpassa systemresursstyrningsverktyg med inriktning mot Axis övervakningskameror, samt
- utveckla ett ramverk för systemresursstyrningsverktygens användning på Axis övervakningskameror.

Rapportens initiala del beskriver grundläggande relevanta aspekter av operativsystem i allmänhet och av inbäddade Linux-system i synnerhet (kapitel 2). Därefter kommer vi att beskriva de systemresursmätnings-, belastnings- och resursstyrningsverktyg som hittats (kapitel 3, 4, 5 och 6). Slutligen kommer vi att presentera resultatet från en kvantitativ utvärdering av systemresursstyrningsverktygen och det ramverk som utvecklats för systemresursstyrning (kapitel 7 och 8).

2 Bakgrund

Arbetet inriktar sig huvudsakligen mot inbäddade system i form av Axis övervakningskameror. Dessa kameror använder sig av ett Linux-baserat operativsystem, vilket är ett Unix-likt operativsystem. Då arbetet berör centrala funktioner i ett operativsystem, och då speciellt i Linux, beskrivs nedan sådana funktioner som är av betydelse för systemresursmätning och -styrning samt motiv till att använda Linux på inbäddade system. Även Unix historik med fokus på vad som har spekulerats vara av betydelse för åtminstone Linux tidiga framgång beskrivs, liksom orsaker till att använda Linux på inbäddade system. Vidare presenterar vi en beskrivning av de övervakningskameror som vi använt under arbetet, samt vissa aspekter på systemresursmätning, belastningsverktyg samt resursstyrningsverktyg.

Styrning av ett programs utnyttjande av CPU-tid och minne involverar de mest centrala delarna av ett operativsystems funktioner, som schemaläggning av när ett program får köras och av hur systemets internminne används. På motsvarande sätt är schemaläggning av I/O-operationer till sekundärt lagringsutrymme av vikt, liksom styrning av nätverksbandbredd.

2.1 Unix och Linux

Unix började utvecklas 1969 på AT&T Bell labs. Tidigt började University of California Berkley (UCB) att använda Unix, och UCB gjorde ett flertal förbättringar och utökningar av Unix, vilka kallades Berkley Software Distribution, BSD. Detta ledde till en splittring av Unix i två parallella huvudfåror; AT&T:s version samt UCB:s version.
För att kunna tillhandahålla BSD fritt så skrevs Unix och flera Unix-verktyg om enbart utifrån sina funktionsbeskrivningar. I samband med detta arbete skapades NetBSD[1], och något senare kom FreeBSD[2] som fokuserade på Intel-platformen, medan OpenBSD[3] fokuserade på säkerhet.
Unix System Laboratories (USL), ett av huvudsakligen AT&T ägt företag som övertagit Unix från AT&T, stämde dock under början av 1990-talet bl.a. UCB för att BSD distribuerades fritt. USL köptes upp av Novell och motsättningarna löstes utanför rättssystemet, med endast ett fåtal förändringar av såväl BSD som av den kommersiella AT&T-baserade Unix-versionen, som hade inkorporerat kod från BSD utan

[1] NetBSD:s webbplats: `http://www.netbsd.org/`
[2] FreeBSD:s webbplats: `http://www.freebsd.org/`
[3] OpenBSD:s webbplats: `http://www.openbsd.org/`

att nödvändig upphovsrättsinformation angavs, som enda resultat. Novell överförde sedermera varumärket Unix till The Open Group, som numera äger rätten att besluta vilka operativsystem som får kallas Unix. [3, 4]

Linux[4] är en operativsystemskärna och GNU/Linux är en grupp Unix-lika operativsystem som använder denna kärna. GNU/Linux är ett bland flera andra Unix-lika operativsystem, och är liksom t.ex. FreeBSD, OpenBSD, NetBSD och delvis Darwin[5] open source. Den första versionen av Linux släpptes under GNU GPL (GNU General Public Licence) 1991 av Linux Torvalds. GNU GPL, som är en open source-licens och idag finns i 3 versioner, medger att vem som helst kan få tillgång till och får ändra det licensierade materialets källkod, och det är därmed möjligt för alla att ändra i Linux funktionalitet. GPL kräver dock att deriverade arbeten släpps under samma licens.
Linux har sedan lanseringen kontinuerligt utvecklats till ett moget Unix-likt system, och idag används Linux på allt från små inbäddade system till massiva superdatorer, där majoriteten av de snabbaste superdatorerna i världen använder Linux-kärnan på något sätt[6]. Ett Linux-system innehåller dock mycket mer än enbart Linux-kärnan, och en stor del av de verktyg som används till och för att utveckla Linux har utvecklats av GNU (GNU's Not Unix) med stöd av FSF (Free Software Foundation). Vissa förespråkar därför användningen av termen GNU/Linux för sådana system. [5, 42, 20, 8, 7]

Det har spekulerats i att delar av Linux initiala framgång var en konsekvens av de tveksamheter som uppstod kring BSD under början av 1990-talet i samband med stämningen mot UCB[9].

2.2 Linux och inbäddade system

Då Linux-kärnan har varit fri så har det varit möjligt att anpassa den för speciella ändamål, bl.a. för användning på inbäddade system. Det finns inte någon entydig definition av "inbäddat system", men generellt kan sägas att ett inbäddat system är ett system som har designats för att utföra en eller ett fåtal funktioner, ofta med realtidskrav[10]. I allmänhet uppfattas inte inbäddade system som datorer av användaren, som t.ex. är fallet med moderna tvättmaskiner, och antalet inbäddade system är mycket stort. Enbart ca 2% av alla mikroprocessorer som säljs används i traditionella persondatorer.

[4]Kernel.org webbplats: `http://www.kernel.org/`

[5]Darwin är operativsystemet i Mac OS X. Webbplats: `http://developer.apple.com/opensource/`

Det inbäddade systemets fysiska storlek och funktionalitetsbehov begränsar och styr vilket operativsystem som är lämpligt att använda på systemet. Inbäddade system delas, bl.a. på basis av ovanstående faktorer, in i olika grupper.

[15, 20, 11, 12]

2.2.1 Indelningen av inbäddade system

Inbäddade system kan klassificeras utifrån ett antal faktorer.

Storlek

Den fysiska storleken på ett inbäddat system kan variera från väldigt små system, som armbandsur, till stora system. Systemets storlek styr naturligtvis vilken hårdvara som kan användas och vilken prestanda denna kan erbjuda. De minsta systemen har i allmänhet begränsade systemresurser vad avser CPU-kraft och minne, vilket kan göra det svårt att använda en komplex kärna som Linux på systemet. Något större system karakteriseras av en mindre begränsad mängd minne och något kraftfullare CPU:er, och systemen är ofta tillräckligt kraftfulla för att användas med Linux. I denna grupp finns olika PDA:er[6], MP3-spelare etc. Stora system, som t.ex. telefoneväxlar, karakteriseras av kraftfulla eller många CPU:er, stora mängder RAM och permanent sekundärt lagringsutrymme.

Tidskrav

Inbäddade system har grovt något av två tidskrav; stringenta eller milda.
Med ett stringent tidskrav avses att det finns ett krav på att systemet ska reagera inom en predefinierad tid. Förutom kravet på att beräkningar ska genomföras på ett korrekt sätt, så finns sålunda även krav på att beräkningarna ska levereras inom en given tidsram. Detta innebär sålunda realtidskrav.
Tidskraven kan vara absoluta, t.ex. för styrning av industrirobotar, där systemet måste garantera kraven annars inträffar något katastrofalt. Detta innebär *hård realtid*. I andra fall, t.ex. för MP3-spelare och övervakningskameror, räcker det med att systemet generellt uppfyller tidskraven och det är tolerabelt om kraven vid vissa tillfällen inte uppfylls. Dessa systemen tillhandahåller *mjuk realtid*.

[6]Personal digital assistant.

För övriga system finns enbart milda tidskrav där det inte är avgörande om systemet tillhandahåller en funktionalitet inom en stringent tidsram.

Nätverkskopplat system

Idag ställer kunderna allt större krav på att system ska vara nätverkskopplade, och t.ex. en PDA förväntas ha sådan funktionalitet. Däremot finns det få krav på att tillhandahålla nätverkskoppling för t.ex. kylskåp eller kaffekokare. Behov av nätverkskoppling leder, precis som övriga klassificerande faktorer, till krav på det operativsystem som används på systemet.

Användarinteraktion

Olika system behöver olika grad av användarinteraktion. Medan en mobiltelefon och en PDA förutsätter en hög grad av interaktion med användaren, så kan andra system enbart kräva minimal interaktion med användaren, som t.ex. motorstyrningssystemen i en bil eller styrsystemen i en tvättmaskin.

[15, 20, 11, 12]

2.2.2 Linux på inbäddade system

Ofta så används på inbäddade system någon form av operativsystem som är konstruerade för sådana system, som t.ex. eCos[7] och TinyOS[8]. Detta eftersom dessa operativsystem i allmänhet är små, kan tillhandahålla realtidsschemaläggning och begränsade exekveringstider för systemanrop, fokuserar på en process- och trådmodell med snabba kontext-byten (eng: context switches) m.m.

Det finns dock ett flertal motiv för att välja Linux som operativsystem på ett inbäddat system. Bland dessa kan noteras att Linux-koden har hög kvalitet. Koden är modulär, tydligt läsbar och rimligt enkelt utökbar med ny funktionalitet. Linux har ett brett hårdvarustöd, bra stöd för kommunikationsstandarder och följer i stort standarder vad gäller mjukvara. Det finns därtill ett stort utbud av verktyg. Vidare så tillhandahålls en känt gränssnitt mot systemet.

[7] eCos webbplats: `http://ecos.sourceware.org/`
[8] TinyOS webbplats: `http://www.tinyos.net/`

Förutom dessa tekniska aspekter så medför även licensen att användaren kan göra vad han eller hon själv önskar med Linux källkoden. Att använda Linux innebär också oberoende från en eventuell systemleverentör, och därtill är Linux inte bara fritt i betydelsen att källkoden är tillgänglig utan även fritt i betydelsen att Linux inte har någon licenskostnad.

[1, 11, 13, 14, 15, 20, 42, 12]

2.3 Testplattformar

Nedan beskriver vi de testplattformar som vi använt oss av i vår undersökning. P3301 och P1311 är releaseversioner av nätverksbaserade kameror, men vi har även använt en intern prototyp.

Realtidskraven på en övervakningskamera är låga; det gör inget om en enstaka bild kommer iväg försent. Det är dock viktigt att bildströmmen i sin helhet är någorlunda ostörd. Därför kan man klassa tidskraven för nedanstående produkter som mjuka. Samtliga produkter använder sig av TCP/IP för att kommunicera med omvärlden. De kan strömma bilder både i H.264- och Motion JPEG-formatet.

2.3.1 P3301

Detta är vår primära målplatform och den kamera vi huvudsakligen har undersökt mät-, belastnings- och resursstryrningsverktyg på. Kameran använder Axis egenutvecklade ARTPEC-3-krets, vilken baseras på Axis likaledes egenutvecklade CRISv32-arkitektur. ARTPEC-3-kretsen består av en 200 MHz 32-bit RISC ISA CPU, MMU, tre programmeringsbara I/O-processorer, och har stöd för hårdvaruaccellererad kryptografi och för gigabit ethernet. Kameran har 128 MB RAM och 128 MB flashminne. [150, 151]

2.3.2 P1311

Denna kamera använde vi framförallt för att undersöka funktionalitet som inte var implementerade på CRIS-arkitekturen. P1311 har en ARM[9]-baserad ARTPEC-B-krets. Denna består av en 225 MHz CPU, en bildsignalprocessor, och har stöd för hårdvarubaserad video-kodning, 100 Mbit/s ethernet m.m. Den har 64 MB RAM och 32 MB flashminne. [153, 154]

[9]ARM är en processorarkitektur. Mer information finns t.ex. här: `http://en.wikipedia.org/wiki/ARM_architecture`

Figur 1: Axis Communications AB:s kameramodell P3301

Figur 2: Axis Communications AB:s kameramodell P1311

2.3.3 Systemprototyp

Systemprototypen är en efterföljare till aktuell ARTPEC-3-krets. Vi använde prototypen för att testa viss funktionalitet som inte var möjlig att testa på kamerasystemen baserad på ARTPEC-B- eller ARTPEC-3-kretsarna. Närmare specifikation av denna systemprototyp kan inte göras då den är företagsintern. [155]

2.4 Operativsystem - kärnor

Program kan grovt indelas i systemprogram, d.v.s. sådana program som hanterar datorns arbete, och applikationsprogram, d.v.s. sådana program som gör det som användaren önskar. Ett operativsystem, som är det mest grundläggande systemprogrammet, har som funktion att bl.a. hantera och abstrahera datorns hårdvara och att hantera och fördela datorsystemets resurser. I sin enklaste form kan ett operativsystem utgöras av en mängd

mjukvarurutiner för att kommunicera med bakomliggande hårdvara. Det går dock inte att tydligt avgränsa vad som avses med ett operativsystem, även om kärnan (eng: kernel), d.v.s. den del av operativsystemet som i allmänhet tillhandahåller dess centrala funktioner, ibland avses med begreppet operativsystem. I system där flera program tillåts köra så att de upplevs vara samtidiga så måste det finnas funktionalitet för att schemalägga olika uppgifter i relation till varandra, för att byta mellan uppgifter, för att tillse att olika uppgifter inte samtidigt (d.v.s. så att de inte stör varandra) försöker använda samma hårdvara, samt för att tilldela systemresurser till uppgifter. Sådan schemaläggning sker av systemets schemaläggare (eng: scheduler). Operativsystemets processhantering ombesörjer delar av detta.

Viss hårdvara kan informera systemets processor eller systemets processorer om att någon extern händelse har inträffat. Detta görs genom s.k. hårdvaruavbrott (eng: interrupts), vilka till sin natur är asynkrona, d.v.s. sker oberoende av processorns aktuella aktivitet. När ett ej maskerat avbrott når processorn så avbryts processorns aktuella exekvering samtidigt som processorn försätts i ett nytt läge och exekveringssammanhanget lagras. Därefter kallas en förutbestämd rutin för att hantera hårdvaran som initierat avbrottet. Avbrottshantering är sålunda en del av abstraktionen mot hårdvaran och är därtill en fundamental del i ett operativsystem som tillhandahåller s.k. preemptivitet, d.v.s. där processer kan avbrytas av operativsystemet när ny schemaläggning av processer måste göras. Sådana preemptivitet möjliggörs av periodiska klockbaserade avbrott.

Moderna operativsystem använder sig av virtuellt minne och sidhantering (eng: paging). Hur dessa båda begrepp definieras varierar men generellt kan sägas att med virtuellt minne så åsyftas ofta att adresseringing av fysiskt minne inte sker direkt utan att adresseringen sker indirekt. Detta sker via ett sidhanteringssystem som översätter en lokal minnesadress till en fysisk adress. Detta system ansvarar också för att frigöra minne genom att lägga delar av detta på sekundärt lagringsutrymme (eng: backing storage i detta fall) och hämta sådant minne i vissa fall. Virtuellt minne medför att en process har en kontinuerligt adresserbar mängd minne, vilket i verkligheten dock kan vara segmenterad, och att en process minnesutrymme delvis kan vara placerad på sekundära lagringsutrymmen, snarare än i fysiskt RAM, utan att processen behöver vara medveten om att så är fallet. Operativsystemets minneshantering tillhandahåller också oftast minnesskydd, vilket innebär att en process inte kan komma åt det minne som någon annan process använder, om processerna inte uttryckligen delar delar av sitt minne. Virtuellt minne och sidhantering kräver hårdvarustöd i form av en MMU (memory management unit), även om det finns andra sätt att implementera till viss

del liknande funktionalitet. Processen kan därmed vara i stort agnostisk vad gäller den fysiska minneslayout som processens exekverbara kod och dess data har.

Ett operativsystem tillhandahåller även skydd för processer. Med skydd avses här att en process inte tillåts störa en annan process annat än genom den resurskonkurrens som uppstår mellan processerna. Som beskrivits ovan så innebär virtuellt minne att en process skyddas från andra processer, men även andra resurser kan skyddas. Abstraktionen av hårdvaran innebär också ett skydd, då resurser kan ägas av en viss process under den tid processen behöver resursen.

Annan funktionalitet som ett operativsystem tillhandahåller kan t.ex. vara virtuella filsystem, d.v.s. ett allmänt gränssnitt (eng: interface) som abstraherar det eller de underliggande filsystemens funktionalitet. Därmed behöver inte enskilda program innehålla programkod för att hantera enskilda filsystem utan operativsystemet presenterar ett enhetligt gränssnitt vilket programmet kan använda sig av oberoende av det underliggande filsystemets implementation.

Ovanstående funktioner implementeras oftast i en del av operativsystemet som kallas för operativsystemets kärna. För att på ett säkert sätt kunna utestänga processer från delar av ovanstående funktioner krävs att processorn inte tillåter alla processer att använda processorns samtliga funktioner. Detta sker genom att processorn tillhandahåller minst två olika exekveringstillstånd, vilka ofta kallas user mode respektive kernel mode, eller liknande. Normala användarprocesser körs i user mode och kan inte utföra vissa funktioner som t.ex. att hantera det fysiska minnet. Vissa anrop, s.k. systemanrop, liksom då ett avbrott mottas av kärnan, medför att processorn går över i kernel mode och att en förutbestämd rutin i kärnan kallas, avbrottshanteraren (eng: interrupt handler). Systemanrop kan implementeras som mjukvarugenererade avbrott, även kallade traps, varvid processen genererar en avbrottssignal till processorn. Vissa processorinstruktioner kräver att processorn kör i kernel mode och dessa instruktioner kan därmed enbart nås från användarprogram genom systemanrop. Därmed kan ett operativsystem och mer specifikt dess kärna tillhandahålla en tvingande abstraktion mot delar av hårdvaran.

[1, 13, 14, 15, 35, 36, 37, 38, 39, 40, 16]

2.5 Trådar, processer och linuxprocesser

En process kan definieras som ett eller flera program som exekveras sekventiellt i en viss minneskontext. Medan ett program är en passiv entitet, d.v.s. en samling instruktioner lagrade t.ex. på sekundärt lagringsutrymme, så är en process en aktiv entitet. Till processen hör dess tillstånd, kontext, där aktuell instruktion, processens stack, processorregister m.m. ingår. En process kan befinna sig i ett av ett flertal tillstånd, t.ex. running, runnable (ready), sleeping (blocked), och stopped (möjligen kan även tillståndet zombie räknas in). En process kontext beskrivs i ett process controll block (PCB), ibland kallat task control block, vilket sålunda innehåller information om processen som programräknaren m.m., och PCB används vid kontext-byten.

En tråd är en grundläggande exekveringsenhet och delar i allmänhet exekveringskod, data, och andra operativsystemsresurser med övriga trådar i samma process. I Linux ombesörjs varje användartråd av en kerneltråd. Jämfört att starta en ny process är trådskapande i allmänhet mindre resurskrävande. Linux särskiljer ur exekveringshänseende inte mellan processer och trådar utan benämner varje exekveringsenhet med task, och en process är de tasks som delar samma s.k. trådgrupp. När vi skriver process menar vi alltså egentligen denna grupp av trådar.

Linux-processer och -trådar är preemptbara, d.v.s. de kan under exekvering bli avbrutna för schemaläggning, varvid andra processer eller trådar med högre prioritet kan bli schemalagda. Vid ett kontextbyte (eng: context switch), d.v.s. då en process eller en tråd avbryts och en annan tillåts exekvera[41], sparas aktuell kontext, varefter motsvarande information hämtas för en annan process eller tråd och denna process eller tråd tillåts exekvera. Kontext-byten görs av kärnan och utförs i kernel mode. Schemaläggning av processer och trådar tillsammans med kontext-byten gör att systemets processer och trådar upplevs som samtidiga, trots att systemet möjligen enbart har en CPU och därmed enbart kan exekvera en tråd vid ett givet tillfälle.

Man kan grovt dela in processer i tre typer:

- **Interaktiva processer**
  Det är ofta så att dessa processer under långa perioder väntar på I/O från användaren eller hårdvaran, d.v.s. processen är ofta blockerad i väntan på I/O. Det är viktigt att dessa processer tillåts köra i samband med att de I/O-händelser de väntar på inträffar, eftersom användaren

annars kommer att uppfatta systemet som långsamt och otillgängligt. Interaktiva processer är i allmänhet inte särskilt resurskrävande.

- **Batchprocesser**
  En batchprocess är en typisk bakgrundsprocess, som inte behöver någon indata från användaren och idealt inte heller från I/O-enheter. Detta gör att processen inte behöver ha bra responstid, eftersom användaren inte märker om processen har lång svarstid.

- **Realtidsprocesser**
  En realtidsprocess är en process som uppfyller realtidskrav. Detta kräver ofta att specialla hänsyn tas av systemet och realtidssytem har ofta strikta och statiska prioritet för schemaläggning av processer och trådar. Detta ska kontrasteras mot *best effort-schemaläggning* där processer med lägre prioritet under vissa förutsättningar körs trots att processer med högre statisk prioritet önskar processortid, d.v.s. sådana system är rättvisa och strävar efter att processer inte ska tillåtas svälta.

Denna indelning är givetvis en förenkling, men den är användbar för att, om än grovt, karakterisera processer.

Realtidsprocesser, som uppfyller hård realtid, finns inte i Linux i aktuella kärnversioner, bl.a. för att minneshanteringen inte har någon uppåt säkert begränsad körtid. Realtidssystem har även andra mål vid schemaläggning än vanliga användarsystem, där rättvisa enligt ovan är en aspekt. Medan schemaläggningen i ett realtidsoperativsystem fokuserar på att garantera begränsade svarstider så är det oftare mer relevant för ett ordinärt operativsystem att försöka nå en balans mellan svarstider och prestanda. En process i Linux kan dock schemaläggas som en mjuk realtidsprocess. Realtidsprocesser i Linux tilldelas statiska prioritet vilka är högre än varje vanlig process prioritet, och schemaläggs i någon av de realtidspolicyer som Linux-kärnan tillhandahåller. Det finns som standard två sätt att schemalägga realtidsprocesser i Linux, Round Robin (RR) och FIFO. I RR schemaläggs alla realtidsprocesser som har samma prioritet så att alla får lika mycket processortid. Om de schemaläggs enligt FIFO-policyn så avbryts de däremot aldrig av processer med samma realtidsprioritet. Alla realtidsprocesser schemaläggs i strikt prioritetsordning av kärnan och avbryts inte av processer med lägre prioritet under den tidsandel som avsats för realtidsprocesser[10]. Alla delar av kärnan är dock inte preemptbara, vilket

[10] I Linux kan en andel av CPU-tiden avsättas för realtidsprocesser. Denna andel är ett tak och sätts i proc-filsystemet (`/proc/sys/kernel/sched_rt_period_us` samt `/proc/sys/kernel/sched_rt_runtime_us`) som det antal mikrosekunder kärnan kan

bidrar till att det inte finns någon garanti för att en realtidsprocess ska kunna tillhandahålla en beräkning inom en vissa tidsram. Linux-kärnan kan kompileras med stöd för olika grad av preemptivitet av kod som körs i kernel mode, från en kärna utan tvingande preemption via en kärna med vissa preemption-punkter till en kärna som bortsett viss kritisk kod är fullständigt preemptbar. Högre grad av preemption gör systemet långsammare men med kortare svarstider. Det finns även patchar som gör Linux-kärnan väsentligen helt preemtbar.

[11, 1, 13, 14, 15, 20, 42, 43]

2.6 Linux schemaläggare

Schemaläggaren är den del av kärnan som bestämmer vilken kerneltråd, eller process, som ska exekvera och genererar kontextbyten. Schemaläggaren aktiveras bl.a. av hårdvarubaserade timer interrupts (tidsbaserade avbrott), vilka har hög prioritet, men kan också aktiveras av systemanrop som t.ex. då en process blockeras i samband med I/O.

Ett flertal krav kan ställas på en schemaläggare, där hänsyn måste tas till faktorer som snabb schemaläggning och att förhindra att processer får svälta. För multiprocessorsystem måste schemaläggningen även se till att en process inte schemaläggs samtidigt på flera processorer och att en process har en rimligt hög affinitet till en av processorerna för att minimera antalet cache-missar.

Olika användningssituationer ställer olika krav på schemaläggningen, och ingen schemaläggningsprincip fungerar bra i alla situationer. Ett kontextbyte, och till viss del även schemaläggningen som sådan, är beräkningsmässigt kostsam, varför en avvägning mellan hur ofta sådana önskas kunna ske och systemets extrabelastning p.g.a. kontextbyten måste göras. Schemaläggare kan indelas efter ett flera kriterier, vilket dock ligger utanför vad som beskrivs i denna raport

Olika versioner av Linux har haft olika schemaläggare. Schemaläggaren i 2.4-kärnan var $O(N)$, där N är antalet processer (eller snarare tasks). I 2.6-kärnan infördes en $O(1)$-schemaläggare, vilken senare byttes mot en $O(logN)$ rättvis schemaläggare (complete fair scheduler; CFS).

CFS, som introducerades i version 2.6.23 av Linux-kärnan, strävar efter att

använda för realtidsprocesser samtidigt som hela realtidsperioden anges.

efterlikna en ideal rättvis multitaskande CPU, i vilken varje tråd[11] tilldelas tid motsvarande andelen trådar i systemet och där alla trådar kör samtidigt på CPU:n. Medan en tråd väntar så beräknar kärnan den tid som tråden skulle tilldelats om den kört på en ideal rättvis CPU. Denna körtid, eller snarare avsaknade körtid, beräknas genom en klocka som uppräknas med en andel $1/N$ av verklig tid, där N är antalet trådar som körs på processorn. Den avsaknade körtiden använder schemaläggaren vid val av tråd vid kontextbyte, där tråden med högst avsaknad körtid väljs. Då denna tråd körs minskar dess avsaknade körtid samtidigt som övriga trådars avsaknade körtid ökar, och då en ny tråd har högst avsaknad körtid kommer vid schemaläggning ett nytt kontextbyte att ske varvid den nya tråden får CPU-tid. Ett rödsvart träd[12], sorterad efter avsaknad av körtid, håller trådarna. Användandet av denna datastruktur bestämmer CFS tidskomplexitet. Det är således principiellt lätt att välja den tråd som ska schemaläggas, och en nyligen körd tråd läggs in i det rödsvarta trädet baserat på aktuell avsaknad körtid. Statiska prioriteter (nice-värden) implementeras genom att tiden räknas snabbare för en tråd med låg prioritet då denna körs, vilket sålunda i praktiken innebär att tråden får en lägre andel av CPU-tiden.

2.6.23-kärnan introducerade även modularitet i schemaläggaren. Denna innebär att varje schemaläggningspolicy (t.ex. policyn för realtidsprocesser och för andra processer) kan implementeras oberoende av den grundläggande schemaläggningskoden.
I Linux finns fem olika schemaläggningspolicyer: SCHED_OTHER, SCHED_RR, SCHED_FIFO, SCHED_BATCH och SCHED_IDLE.
SCHED_OTHER är den normala schemaläggaren och använder CFS, medan SCHED_RR och SCHED_FIFO innebär realtidsschemaläggning där skillnaden mellan SCHED_RR och SCHED_FIFO är att den förra schemalägger realtidstrådar med samma prioritet round robin, medan den senare låter en tråd med en given prioritet köra tills den inte längre önskar köra och först därefter schemaläggs andra trådare med samma realtidsprioritet. Realtidsprocesser schemaläggs alltid i prioritetsordning. Realtidstrådar har högre prioritet än andra trådar.

Vissa förändringar gjordes i 2.6.24-kärnan där klockan för beräkning av avsaknad körtid ersattes med en klocka per tråd, `vruntime`, som håller den virtuella körtiden för tråden. I 2.6.24 introducerades också gruppschemaläggning (eng: group scheduling), där schemaläggningsprincipen utökades så att den omfattar schemaläggningsentiteter.

[11] Linux schemalägger egentligen inte trådar utan tasks. En process utan några trådar utgör en task, medan en process som startar N trådar har $N + 1$ tasks.

[12] Se `http://en.wikipedia.org/wiki/Red_black_tree`

En schemaläggningsentitet kan bestå av trådar tillsammans med andra schemaläggningsentiteter. Detta medför att schemaläggning också kan appliceras över grupper av processer och trådar, snarare än enbart över enstaka trådar. Dessa grupper kan baseras på t.ex. användare eller på cgrupper.

Ett flertal schemaläggningsparametrar går att ställa runtime för kärnan, som t.ex. önskad preemption latency och minimal schemaläggningsgranularitet.

[14, 42, 44, 45, 46, 47, 49, 50, 48, 19, 17, 18]

2.7 Minneshantering

En process minne måste inte nödvändigvis ligga i RAM (interminne). Ofta rapporteras minne som resident set size (RSS) vilket är den del av processens minne som finns i RAM, samt i virtual set size (VSZ) som är processens hela minne. Kärnans sidhanteringssystem kan överföra minne mellan internminnet och sekundärt lagringsutrymme. Placeringen av minnessidor på sekundärt lagringsutrymme kan stängas av i Linux, men aktiviteten i systemet kan också styras. Noterbart i sammanhanget är att på många små och mellanstora inbäddade system så används inte sekundärt lagringsutrymme för att lagra delar av processers minne.

Den verkliga mängd minne som en process ensam använder är inte tydligt definierad. När en process skapas, t.ex. med systemanropet `fork`, vilket skapar en kopia av aktuell process[52], så ska dotterprocessen ärva en kopia av föräldraprocessens minne. Detta skulle dock i många fall medföra en onödig tids- och minnesåtgång då minnessidor som varken föräldraprocessen eller dotterprocessen kommer att ändra onödigt hade kopierats. I allmänhet markeras istället minnessidorna som "copy on write", d.v.s. en kopiering sker först när någon av processerna försöker ändra i minnet. Två processer kan också dela delar av koden, som t.ex. biblioteksfunktioner. Liknande gäller även för minnesallokering där optimistic memory allocation används, d.v.s. den reella minnesallokeringen sker inte förrän minnet verkligen behövs. En process som inte använder allokerat minne kan således allokera mer minne än en process som verkligen använder det av processen allokerade minnet. Givet en viss mängd tillgängligt minne så kan det därför finnas en rimlighet i att kärnan överallokerar minne, d.v.s. att den tilldelar mer minne än vad som faktiskt finns tillgängligt. I Linux kan överallokgering styras liksom hur stor överallokering som tillåts.

Ovan framgår att ett verkligt minnesöverutnyttjande inte nödvändigtvis uppstår i samband med det systemanrop som försöker allokera minne. Kärnan måste dock hantera en situation där minnet tar slut. Linux-kärnan använder för detta out of memory kill (oom kill) för att döda någon process. Hur detta sker är konfigurerbart. Som standard beräknas en poäng baserat på ett flertal data som t.ex. minnesanvändning och körtid. Med utgångspunkt från detta värde så dödas en process. Alternativa metoder för oom kill existerar, som t.ex. att döda den process som gör den allokering som medför överallokering. oom kill kan stängas av helt för en process, och det finns även möjlighet att justera oom kill per process. Det är även möjligt att konfigurera kärnan så att en kernel panic[13] fås vid oom.

Algoritmen för att välja process att döda har varit omdebatterad och det finns ett antal kärn-patchar som ändrar kärnans standardbeteende i detta avseende. Normalt meddelar kärnan vilken process som dödats via ett kernel log-meddelande.

[51, 1, 53, 54, 55, 56, 57]

2.8 System V init

`init` är den process som startas först under uppstarten av Unix. `init` ansvarar efter sin uppstart för att starta resten av systemet och är därmed förfader till alla andra processer som körs på ett Unix-system. Vad `init` gör varierar mellan olika Unix-dialekter och även mellan olika Linux-distributioner, och det finns flera olika sätt som systemet kan startas på. De Axis-produkter som förekommer i detta arbete använder SysV-init. I SysV-init finns olika runlevels, vilket motsvara olika konfigurationer av systemet. Vissa runlevels är dock reserverade för speciella ändamål, t.ex. är runlevel 6 reserverad för reboot. `init` läser tidigt vid uppstart normalt en fördefinerad fil (`/etc/inittab`) och får från denna bl.a. en standard-runlevel. I den fördefinierade filen anges även hur systemet ska startas beroende på runlevel. I allmänhet innebär detta att ett script (`/etc/init.d/rc`) körs med runlevelvärdet som argument. Detta script exekverar i sin tur de symboliska länkarna *KXXnamn*, där *XX* är ett tal oftast i intervallet 00 till 99, i katalogen `/etc/rc<runlevel>.d/`, i alfabetisk ordning, med argumentet "stop". Detta innebär sålunda att vissa tjänster stoppas. Detta sker dock inte nödvändigtvis vid uppstart, d.v.s. när systemet inte har någon tidigare

[13]Kernel panic betyder att kärnan får ett allvarligt internt fel, så att den måste stänga av sig för att hindra att data blir korrumperad eller skadad. Se t.ex. `http://en.wikipedia.org/wiki/Kernel_panic`

runlevel, utan när runlevel förändras under det att systemet körs. Efter detta exekverar scriptet de symboliska länkarna *SXXnamn* med argumentet "start". De symbolisk länkarna refererar till script som ligger i katalogen `/etc/init.d/`. Siffrorna i de symboliska länkarnas namn är valda så att tjänster som beror av andra tjänster inte fallerar, d.v.s. eftersom scripten exekveras i bokstavsordning så kommer symboliska länkar med lägre nummer att köras först. Fördelen med SysV-init är att det är flexibelt och enkelt att konfigurera. En nackdel är att processerna startas sekventiellt, vilket gör att eventuella parallelliseringsvinster uteblir. Därtill medför SysV-init i allmänhet att många processer, delvis sådana som enbart behövs för att starta upp andra processer, körs. Detta bidrar till att SysV-init kan medföra att uppstartstiden blir lång. Om t.ex. en process har en långsam startprocess, kanske beroende på att den väntar på I/O, kan hela uppstartsprocessen förlångsammas.

Mot bakgrund av detta problem och en önskan om att `init`-processen även ska övervaka och starta om processer som avslutats prematurt finns ett stort antal alternativ till SysV-init. Bland dessa finns t.ex. Ubuntus Upstart[14], Apples launchd[15], initng[16], Sun Solaris Service Management Facility[17], runit[18], och systemd[19]. Vissa av dessa är inriktade mot vissa system medan andra är generella.

[157, 158]

2.9 I/O-schemaläggning i Linux

I/O-schemaläggning, vilket huvudsakligen rör hårddiskar, syftar till de delvis motstridiga målen att minska tidsåtgången för läsning, prioritera vissa processers I/O-förfrågningar och förhindra svält m.m. Förflyttning av hårddiskens läshuvud innebär i sammanhanget en betydande tidsåtgång.

En schemaläggare som alltid väljer den närmaste platsen är optimal vad avser antal operationer per tidsenhet, men tillåter svält och är orättvis i

[14]Se Ubuntus webbplats `http://www.ubuntu.com/` och Upstarts webbplats `http://upstart.ubuntu.com/`

[15]Se Apples webbplats `http://www.apple.com/` och webbplatsen för launchd `http://developer.apple.com/macosx/launchd.html`

[16]Se webbplatsen för initng `http://initng.sourceforge.net/`

[17]Se Solaris webplats `http://www.oracle.com/us/products/servers-storage/solaris/` och Romack, Service Management Facility (SMF) in the Solaris 10 Operating System, Sun, 2006. `http://www.sun.com/blueprints/0206/819-5150.pdf`

[18]Se runit - a UNIX init scheme with service supervision. `http://smarden.org/runit/`

[19]Se Poettering, Rethinking PID 1. `http://0pointer.de/blog/projects/systemd.html`

förhållande till data som ligger nära diskens kanter. FIFO-köer är rättvisa och förhindrar svält men har relativt dålig prestanda. Elevator-schemaläggare, vilka upprepat flyttar läshuvudet fram och tillbaka över disken från centrum till kant och tillbaka, ger relativt bra prestanda men är inte rättvisa. En variant på denna är en cyklisk elevator-schemaläggare, som flyttar läshuvudet enbart åt ett håll vid läs- och skrivoperationer. Denna är rättvis, förhindrar svält, men har något sämre prestanda.

Linux har fyra olika schemaläggare som kan väljas per blockenhet runtime. Deadline-schemaläggaren är en variant på en cyklisk schemaläggare, men där vissa förfrågningar kan ha en deadline. Om en förfrågans deadline hotas så ombesörjs omedelbart ifrågavarande förfrågan, och detta kan betraktas som realtids-I/O. Även den anticipatoriska schemaläggaren är en variant på en cyklisk schemaläggare, men med en förändring. När en förfrågan ombesörjts så avvaktar schemaläggare en kort stund för att se om någon ny näraliggande förfrågan inkommer. Sker så, så ombesörjs denna, och om så ej sker så följer denna schemaläggare samma princip som en cyklisk-elevator-schemaläggare. Schemaläggaren saknar stöd för realtids-I/O, svält förekommer i princip inte, och schemaläggaren förutsätter till viss del hur processer beter sig i verkligheten (avvaktan). Complete fair queuing-schemaläggaren (CFQ) är en rättvis schemaläggare med stöd för realtids-I/O och prioriteter. Vidare finns en FIFO-schemaläggare ("noop"-schemaläggaren).

[25, 26, 27, 28, 29, 30, 31]

2.10 Traffic shaping

Traffic shaping innebär att man styr hur paket prioriteras från ett nätverksgränssnitt och hur paket inom ett datornätverk skickas, i syfte att optimera prestandan för vissa flöden. Flöde i detta sammanhang betyder en ström av paket från en källa till ett mål. Denna styrning sker principiellt genom att paket fördröjs. Ett annan sätt är att kasta paket, vilket kallas traffic policing. Med traffic shaping och traffic policing kan man skapa oerhört komplexa nätverksbeteenden. Man kan därigenom säkerställa att viktiga flöden prioriteras, varvid de viktiga processerna som använder flödena får de nätverksresurser de kräver. [133, 134, 135]

2.11 Jitter

För en process som periodiskt tillhandahåller någon form av utdata, t.ex. i form av bilder eller ljud, så finns risk för att periodtiden och

periodvariabiliteten påverkas om t.ex. systemets CPU-belastning ökar. Schemaläggningen kan bli sådan att processens utdata kommer tidsförskjuten i förhållande till när den förväntats, men även variabilitet i periodtiden kan uppkomma. Delar av detta kan tänkas uppkomma även om processen får en genomsnittligt tillräcklig CPU-tid.

En vanlig allmän betydelse av jitter är ett tidsfel i en (periodisk) signal, d.v.s. att signalen kommer på en tidpunkt som skiljer från den tidpunkt då den förväntas. För periodiska signaler med en viss frekvens så är jitter sålunda ett fel där signalen är tidsförskjuten i förhållande till sin förväntade periodtid. [23]

I Realtime Transport Protocol, vilket används av målarkitekturen för att överföra bilder till klienten, så definieras, enligt RFC 3550, en form av jitter som kallas interarrival jitter. Definitionen inkluderar såväl samplingstiden som ankomsttiden till klienten. [81]

Ett alternativt sätt att kvantifiera jitter för periodiska processer är, i vissa fall, att bestämma kvadratroten av medelvärdet av periodernas kvadratiska avvikelse från den genomsnittliga periodtiden, d.v.s. ungefär standardavvikelsen av cykeltiden[23, 24].

2.12 Systemresursmätning

Ett system har dålig prestanda om någon resurs blir begränsande och processer tvingas vänta på resursen. Att hitta dessa begränsande resurserna är viktigt för att optimera systemets resursanvändning. När den begränsande resursen hittats krävs för att en analys ska vara meningsfull att orsaken till att resursen överanvänds härleds. Vissa grundläggande teoretiska aspekter finns för systemresursutnyttjande. [33]

Principiellt finns tre, eller fyra, olika sätt för prestandamätning; mjukvarubaserade metoder, hårdvarubaserade metoder, hybrider mellan dessa samt on-chip performance counters.
Mjukvarubaserade metoder lider av en del brister, t.ex. dålig upplösning på klockor för att mäta tidsåtgång, och risk att störa systemet. Det senare kan i vissa fall vara tolerabelt, medan störningen i andra fall kan påverka systemet signifikant. Vidare finns begränsningar i vad som kan mätas med mjukvarubaserade mätmetoder, där faktorer som cache-träffar och avbrott inte fullt kan mätas.
Hårdvarubaserade metoder kan komma runt vissa av begränsningarna som finns för mjukvarubaserade metoder genom att direkt läsa bustrafik. Störningen på systemet som mäts kan ofta hållas låg och upplösningen i

sådana system kan genom val av hårdvara anpassas till mätningens syfte. Hårdvarubaserade metoder kan dock vara problematiska då t.ex. enbart fysiska minnesadresser kan fås och att tillgång till relevant hårdvara krävs.
I hybridsystem kombineras styrkorna i mjukvaru- och hårdvarubaserade mätmetoder. Externa händelser loggas av hårdvarusystemet medan mjukvaruprober loggar händelser orsakade av programmet som analyseras.
Många moderna processorer har prestanda-räknare (eng: on-chip performance counters). Dessa kan tillhandahålla information som antal klockcykler, antal cache-missar och antal avbrott etc.
Frånsett helt hårdvarubaserade metoder, ger av övriga metoders användning av prestanda-räknare i allmänhet minst påverkan på det mätta systemet. [34]

Mätning på systemet riskerar att störa detsamma. På något sätt så måste mätverktyget erhålla mätdata från systemet, och ett system för att insamla denna data måste finnas. Mätsystemet måste på något sätt kopplas till det mätta systemet och därvid störs det senare. Körs mätsystemet på det system som ska mätas så kommer mätsystemet att använda såväl CPU-tid som minne och möjligen därtill sannolikt även systemresurser som sekundärt lagringsutrymme eller medföra störningar i form av systemanrop. Mätverktyget konkurrerar sålunda med det som det avser att mäta vilket kan påverka mätningen. [32, 33]

På motsvarande sätt kan antas att mätverktygets mätresultat kan störas av övrig last på det mätta systemet. Detta problem kan möjligen reduceras av att tilldela mätverktyget en tillräcklig prioritet och tillräckliga resurser så att verktyget kan köra väsentligen störningsfritt. Detta i sin tur medför dock en större påverkan på det mätta systemet.

En annan viktig fråga är vad mätverktyget leverar för data. De data verktyget tillhandahåller kan representera en eller en samling punktskattningar av den mätta resursen, eller medelvärdet över en tidsperiod. I det senare fallet erhålles utjämnade data, medan det i det förra fallet kan vara så att verktyget missar relevanta mätvärden. Medelvärden medför att resursutnyttjandegraden över en period kan erhållas och det är därmed lättare att se vilken kapacitet som resursen kan tillhandahålla över perioden. [33]

För vissa systemresurser är inte en punktskattning relevant, utan dess värde måste anges som ett medelvärde över tid. Exempel är CPU-andel i uniprocessorsystem. Det blir alltså inte särskilt meningsfullt att mäta dessa systemresurser i punkter, utan de måste mätas under någon tidsperiod. Minne är ett exempel på en resurs som kan mätas i punkter; vid varje given tidpunkt kan man se hur mycket minne en process använder.

2.13 Belastningsverktyg

Då ett system inte är under en sådan belastning att en konkurrenssituation uppstår om någon resurs, d.v.s. om varje process på systemet tilldelas de resurser den vid varje tillfälle behöver, eller om en sådan konkurrenssituation saknar relevans för den faktiska funktionen av systemet, så saknas motiv för att införa ett resursstyrningssystem. Resursstyrningssystem, såsom avses i detta examensarbete, syftar att styra en eller flera systemresurser på ett sådant sätt att vissa processer eller grupper av processer kan garanteras en viss resursnivå när konkurrens om resurserna uppstår. För att möjliggöra en utvärdering om en möjlig resursstyrningsmetod ger ett önskvärt resultat krävs sålunda att systemet kan belastas vad gäller CPU-användning, minnesanvändning respektive nätverksbandbredd. Samtidigt krävs en mätning av processens eller gruppen av processers resursanvändning för att värdera om styrningen på ett adekvat sätt tillhandahåller den resurs som avses.

2.14 Resursstyrningsverktyg

Med resursstyrning avses i detta examensarbete en styrning av resurser till en eller flera processer på ett sådant sätt att processen eller processerna under aktuella förhållanden kan garanteras, åtminstone i mjuk bemärkelse, en viss andel av ifrågavarande resurs, eller att resursen begränsas för processen eller gruppen av processer. Systemresurser som i detta sammanhang kan vara intressanta att styra innefattar CPU-tid, internminne samt i viss mån nätverksbandbredd och möjligen övrig I/O.

3 Verktyg för mätning av systemresursutnyttjande

De systemresurser vi vill mäta är framförallt CPU-tidsandel, minnesutnyttjande och nätverksutnyttjande. Principiellt skulle det även vara intressant att mäta intern bandbredd. Detta är dock inte möjligt på målarkitekturen[22]. Översiktligt har vi även eftersökt verktyg för att mäta I/O mot sekundärt lagringsutrymme.

3.1 Undersökning

För att mäta systemresursutnyttjande undersökte vi för ett flertal tillgängliga verktyg om det var möjligt att använda och porta dessa till Linux på CRIS-arkitekturen. Vi använde en kvalitativ evalueringsmetod. I den kvalitativa utvärderingsmetoden ingick huruvida det med rimlig enkelhet var möjligt att porta verktyget till målplattformen. Skulle en sådan portning kräva förändringar av Linux-kärnan eller om det krävde portning av i övrigt svårportade verktyg, så ansågs det inte vara möjligt att porta verktyget till målarkitekturen med en rimlig tidsåtgång. I övrigt studerade vi den information som programmet kunde leverera, programmets systemresursanvändning och dess stabilitet.

3.1.1 Portning

Det var inte möjligt att under den tid som fanns tillgänglig porta vissa av verktygen till den huvudarkitektur som arbetet inriktade sig mot, även om vissa verktyg portades till ARM-arkitekturen ARTPEC-B, exemplifierat med Axis P1311, för att testas där. I många fall förhindrades portning av verktyg av att Linux på målarkitekturen inte hade implementerat stöd för viss funktionalitet. Sålunda fanns i vissa fall arkitekturberoende begränsningar för vilka verktyg som var möjliga att använda. Dessa begränsningar kan i vissa fall vara överkomliga genom att porta funktionalitet i Linux-kärnan till den aktuella arkitekturen, men detta låg utanför arbetets omfattning. I andra fall förhindrades portning av vertyg av att vertyget använde större programpaket vilka bedömdes vara alltför arbetskrävande att porta. Sådana programpaket var t.ex. Python[20] och Perl[21]. I många fall krävdes att generella bibliotek, som t.ex. ncurses[22] och pcap[23], portades till målarkitekturen.

[20]Python webbplats: `http://www.python.org/`
[21]Pers webbplats: `http://www.perl.org/`
[22]GNU ncurses webbplats: `http://www.gnu.org/software/ncurses/`
[23]libpcap webbplats: `http://www.tcpdump.org/`

3.1.2 Stabilitet

Ett annat viktigt kriterium att beakta är verktygets stabilitet och hur stabil verktygets användning av systemresurser är. Om verktygets resursanvändning fluktuerar kraftigt kan detta göra mätningarna på kameran osäkra. Därför kan det vara fördelaktigt om verktygets resursanvändning är relativt stabil så att inte mätresultatet påverkas periodiskt. Huruvida detta i praktiken är betydelsefullt måste värderas från fall till fall. Det intressanta är huruvida verktygets påverkan är tillräckligt konstant i förhållande till hur övriga parametrar i testet påverkar testresultatet.

Det är också fördelaktigt om verktygets resursanvändning är låg. Eftersom det i de flesta fall enbart är den relativa skillnaden i testresultat som är intressant, så är detta sannolikt inte av större vikt så länge resursanvändningen håller sig på en rimlig nivå.

Ett teoretiskt problem i sammanhanget är att mätning av verktygets systemresursutnyttjande förutsätter att man redan har ett verktyg som kan mäta systemresursutnyttjande per process. Detta går dock inte att med för arbetet tillgängliga resurser att komma runt, eftersom vi inte har något annat sätt att mäta processers systemresursutnyttjande.

3.1.3 Typer av verktyg

Vi har inriktat oss på två huvudtyper av verktyg:

- sådana för att mäta systemresurstilldelning per process, samt
- mer generella verktyg som rapporterar statistik över systemets belastning

Vissa av de verktyg vi har undersökt kan mäta systemresurser såväl per process som för hela systemet. Generell information om belastningen av systemen är förvisso intressant, men är inte tillräcklig då per process-information krävs för att värdera hur olika processer i systemet påverkas av olika typ av belastning av systemet. Den första typen av verktyg krävs sålunda för att värdera om kritiska processer ges en tillräcklig resurstilldelning. Det kan dock även vara relevant att mäta den generella belastningen på systemet, inte minst för att sätta resurstilldelning per process i relation till systemets totala belastning. Många av de verktyg vi undersökt hämtar information från proc-filsystemet[24].

[24]För information om proc-filsystemet se: `http://en.wikipedia.org/wiki/Procfs`

I nedanstående så omnämns det Linux-system som finns på en viss arkitektur som t.ex. CRIS-arkitekturen, varmed underförstås aktuell Linux-version på den aktuella processorn.

3.2 Undersökta verktyg

top

Det finns flera olika varianter av top, som `top`, `atop`, `htop` m.fl, med varierande funktionalitet. `top` är ett verktyg där resursanvändingen för varje process i systemet presenteras dynamiskt och periodiskt med i allmänhet ett styrbart intervall. `top` är framförallt interaktivt verktyg som huvudsakligen ger information om CPU-användning och minnesanvändning för processer och trådar. Programmet kan presentera data om CPU-tidsanvänding sedan föregående presentation av datapresentation eller i absoluta värden. `top` kan också köras i batchmode. [58, 59, 60, 61]
`top`s resursutnyttjande är periodiskt varierande mot bakgrund av att information läses från framför allt proc-filsystemet periodiskt.
Busybox är en samling nedbantade standardvertyg för Unix och är avsett att användas på system med begränsade systemresurser, som inbäddade system. Busybox topversion saknar flera konfigurationsmöjligheter som finns i andra versioner av top. Antalet visade fält är t.ex. förutbestämt, och programmet kan inte att köras i batch mode. [62]
Målarkitekturens standardsystemprogramvara använder en version av top från Busybox. Av denna anledning portades en mer komplett version av top till målarkitekturen (eg. top från Debian paketversion procps 1:3.2.7-11). Denna version av top har bl.a. stöd för batch mode och kan följa en eller flera enstaka processer.

Beroende på uppdateringsintervall och antal övervakade processer varierar top:s resursutnyttjande vad avser CPU-tid. På målarkitekturen med övervakning av samtliga processer varierar tops grad av CPU-utnyttjande under uppdateringsintervallet från ca 2% vid ett uppdateringsintervall av 3 s till knappt 10% vid ett uppdateringsintervall av 0,5 s. tops CPU-uttnyttjande är dock tydligt periodiskt med tydliga toppar vid informationsinsamlingsögonblicket. Minnesutnyttjandet är mer konstant och också mer begränsat i förhållande till den minnesmängd som finns på målarkitekturen. Busybox version av top har en större grad av CPU-utnyttjande än den till målarkitekturen portade versionen av top.

vmstat

`vmstat` är ett verktyg för att undersöka huvudsakligen parametrar relaterade till användning av virtuellt minne. Då målarkitekturen inte använder virtuellt minne är detta verktyg av mindre betydelse. Verktyget kan även ge information om diskanvändning. [63]

free

Detta verktyg ger huvudsakligen information om systemets totala minnesanvändning, inklusive information om sidhantering. Programmet kan användas för att upprepat få sådan information, och resursutnyttjandet blir då periodiskt varierande. [64]
CPU-resursutnyttjandet är dock väldigt lågt liksom minnesutnyttjandet.

sysstat-verktygen

sysstat är en samling vertyg som mäter systemresursanvändning på olika sätt.

`sar` är ett verktyg som kan ge många olika sorters statistik över systemresursanvändningen. Verktyget kan ge statistik om t.ex. I/O- och CPU-resursanvändning, samt om sidhantering. `iostat` ger information om I/O-resursanvändning, medan `pidstat` kan ge information om CPU- och I/O-resursutnyttjande m.m. för varje process. En intressant detalj med `pidstat` är att verktyget presenterar den tid processer har exekverat i kernel- respektive user space. `pidstat` kan köras så att information erhålles periodiskt. [65]
Resursutnyttjandegraden vad avser CPU-tid är då dock påtagligt varierande och är också relativt hög, och högre än för `top`, medan minnesåtgången, liksom för `top`, är begränsad.

ps

`ps` är ett verktyg som listar systemets samtliga processer. Informationen som presenteras inkluderar för varje process bl.a. processens UID (d.v.s. ägaren till processen), PID, VSZ, RSS och CPU-tidsandel. Vilken information som presenteras är konfigurerbart. [66]
Liksom vad gäller top så har målarkitekturen i standardfirmware-utförande en nedbantad version av ps från Busybox. Denna har enbart begränsade möjligheter att styra de data som visas (PID, användare, status och minnesanvändning). En mer fullständig version där prioriteter, RSS, VSZ, CPU-användning och minnesanvändning m.m. framgår, portades därför till kameran.

procinfo

`procinfo` ger information om systemets status. Som namnet antyder samlar programmet in information från proc-filsystemet. Information som kan erhållas inkluderar data om minnesanvänding, tid som processorerna använt för att köra i user mode och kernel mode, antal kontextbyten, information om sidhanteringssystemet m.m. `procinfo` kan köras så att denna information kontinuerligt tillhandahålls, vilket ger ett periodiskt varierande resursutnyttjande. Såväl graden av CPU-resursutnyttjande som minnesutnyttjande är dock mycket lågt för procinfo på målarkitekturen. `procinfo` portades till målarkitekturen. [67]

dstat

`dstat` är ett statistikverktyg som kan ge allmän information om systemets resurser. Verktyget ger information om minne, sekundärt lagringsutrymme, CPU-utnyttjande, sidhantering m.m. `dstat` kan samla in data under valfri tidsperiod och sedan presentera insamlad data dynamiskt. `dstat` är skrivet i Python och ansågs därmed vara för tidsödande att porta till CRIS-arkitekturen eftersom det skulle kräva att även Python portades till målarkitekturen. [68]

psinfo

`psinfo` presenterar information om enskilda processer. Programmet kan visa en stor mängd information som CPU-användning, information om signaler, schemaläggningsinformation, sidfel, minnesanvändning fördelat mellan typer av minne som t.ex. delat minne, miljövariabler, aktuellt processtillstånd, processgrupp, sessionsid, nice-värde, realtidsprioritet, schemaläggningspolicy, antal trådar, oom kill-poäng m.m. Verktyget har inte stöd för att köras kontinuerligt. Resursutnyttjandegraden förefaller låg, men är svårvärderad. `psinfo` portades till målarkitekturen. [69]

pmap

`pmap` ger information om enskilda processers minnesåtkomst. Programmet, som portades till målarkitekturen, ger information om använda bibliotek, andel delat minne och annan minnesrelaterad information för processer. [70]

iftop

Detta verktyg ger information om nätverkstrafiken vad gäller bandbredd m.m. Informationen ges ej per process utan per interface, men det finns viss möjligt att filtrera erhållen information. `iftop` har dock en låg resursutnyttjandegrad vad avser CPU-tid och minne på målarkitekturen. [71].

iptraf

`iptraf` är ett verktyg som likt `iftop` visar information om nätverksanslutningar inkluderande bandbredd i vissa fall. Liksom vad gäller `iftop` finns viss möjlighet att filtrera trafiken. [72]
Liksom `iftop` behöver inte `iftraf` köras på den testade datorn, dock har `iptraf` ett lågt CPU- och minnesutnyttjande på målarkitekturen.

nethogs

`nethogs` mäter nätverkstrafiken per process och kan även visa den bandbredd en given process använder. `nethogs` körs kontinuerligt och uppdaterar data över trafiken vid specificerbara intervall. Detta gör att resursutnyttjandet blir cykliskt på målarkitekturen. CPU-tidsandelen varierar sålunda för `nethogs`, men håller sig relativt begränsad. Minnesutnyttjandet är lågt på målarkitekturen. [73]

iperf

`iperf` är ett verktyg för att mäta nätverksbandbredden. Data skickas mellan en server och en klient, och bandbredden beräknas utifrån detta. `iptraf` kan testa och rapportera bandbredden för såväl UDP- som TCP-trafik. Verktyget kan köras i batchmode. `iperf` portades till systemprototypen. [74]

ipbench

`ipbench` är ett distribuerat system för mätning av olika parametrar associerade till nätverks-prestanda. Programmet består av en del, som körs på en kontrolldator, som styr ett antal testdatorer. Dessa testdatorer kommunicerar i sin tur med måldatorn, vilket är den dator som testas. Programet är skrivet i Python. Python är inte portat till någon av de inbäddade arkitekturer som är aktuella och en portning av Python bedömdes som alltför tidskrävande. [78]

iotop

`iotop` är ett verktyg som visar I/O-statistik, d.v.s. läsning till och från disk, per process. Programmet kräver en version senare än 2.6.20 av Linux-kärnan och kräver att kärnan är kompilerad med stöd för I/O-accounting. [77]
`iotop` är skrivet i Python vilket bedömdes som alltför tidskrävande att porta till CRIS-arkitekturen. Vidare har CRIS-arkitekturen inte stöd för I/O-accounting.

Alternativa verktyg för undersökning av nätverkstrafik

Det finns många olika verktyg för att undersöka och filtrera nätverksdata, som t.ex. `Wireshark` och `tcpdump`. Dessa verktyg är dock huvudsakligen inriktade mot att filterera och undersöka nätverkstrafik, med ett stort antal funktioner för att värdera detta, och är mindre lämpade för att mäta nätverksbandbredd. Vi hade enbart begränsade behov av att filtrera och närmare undersöka nätverkstrafik. [75, 76]

Profileringsverktyg

Profileringsverktyg mäter dynamiskt, d.v.s. medan ett program körs, ett program beteende. Profileringsverktyget använder sedan denna information för att analysera t.ex. programmets prestanda. Dessa verktyg används oftast för att analysera vilka delar av programmet som kan behöva optimeras, t.ex. hur ofta olika fuktionsanrop körs och hur lång tid varje sådant anrop tar.

För Linux-kärnan existerar ett stort antal profilerings-verktyg. Dessa verktyg ger dock mest information om vilka funktioner i ett visst program som tar upp CPU-tid, vilket inte är intressant för oss. Därför har vi inte fokuserat på att undersöka dessa verktyg.

[156]

Verktyg för att mäta latens

`latencytop` är ett verktyg som anger den latens som uppstår då en process blockeras av kärnan då en resurs för tillfället ej är tillgänglig. `latencytop` kan inte mäta interrupt-latenser eller mer generellt schemaläggningslatensen. `latencytop` kräver att Linux-kärnan kompileras med stöd för detta, och därtill att den aktuella arkitekturen har stöd för stacktraces. Linux under CRIS-arkitekturen har för närvarande inte stöd för detta, medan ARM-arkitekturen har sådant stöd. [79, 80]

Linux-kärnan kan komplieras med stöd för ftrace, vilket är ett system för att från kärnan få trace-information. Systemet kan användas för att få viss latens-information. Tyvärr har inte Linux på CRIS-arkitekturen stöd för ftrace. [82, 83, 85, 84]

schedtop

Då Linux kompileras med schedstat-stöd så kan en mängd schemaläggnings-information erhållas från kärnan, dels generell sådan, dels per process. Vilken information som erhålls beror delvis av vilken version av schedstat som används. `schedtop` presenterar ifrågavarande information från kärnan. Då `schedtop` kräver Boost-biblioteket[25] och detta bedömdes vara för tidsödande att porta, så portades inte schedtop till målarktikturen. [86, 87, 88]

systemtap

Systemtap är ett utökbart program för att erhålla inforamtion från Linux-kärnan rörande processer som för tillfället exekverar i kernel mode. Systemet kan koppla script till händelser i kärnan, och exekverar dessa script då händelsen inträffar. Systemtap kräver att aktuell arkitektur har stöd för kprobes, vilket saknas för CRIS-arkitekturen. [89, 90, 91, 92, 93]

3.3 Diskussion

`top` är ett användbart verktyg för att övervaka CPU-resursutnyttjande. Problematiskt är såväl en relativt hög grad av CPU-utnyttjande, dock betydligt lägre än för Busybox-versionen, som en tydligt periodisk variation av denna. Verktygets minnessignatur är låg. `top` är alltså ett lämpligt verktyg för att övervaka processers minnesutnyttjande. `free` kan användas för att erhålla övergripande information om minnesutnyttjandet, och belastar varken CPU eller minne i någon större omfattning. Båda dessa verktyg, och då f.f.a. `top`, har använts vid senare utvärderingar.

`pidstat` ger information utöver vad som erhålles från `top`, men tillför i sammanhanget inte mycket information av värde. Resursutnyttjandegraden för `pidstat` är dessutom ofördelaktig. Vi använde inte `pidstat` i någon större utsträckning.

`ps` kan inte utan att ps-processer startas upprepat användas för att kontinuerligt erhålla information från det testade systemet. `ps` ger dock

[25] boost c++-bibliotek, webbplats: `http://www.boost.org/`

värdefull information, men kan alltså inte användas utan att det uppstår ett extrautnyttjande av resurser i samband med de upprepade processtarter det skulle innebära att använda verktyget. Vi har använt `ps`.

`procinfo` tillhandahåller information som i vissa situationer var användbar, och programmet har en låg resursutnyttjandegrad.

På samma sätt tillhandahöll `psinfo` i vissa fall intressant information, men det hade varit fördelaktigt om verktyget kunnat köras kontinuerligt.

`iperf` liksom `iptraf` och `nethogs` användes för att testa nätverksutnuttjandet. För våra syften passade `iperf` bättre för att testa nätverksutnyttjandet än de verktyg som enbart mätte detta. Orsaken till detta var att mätverktygen inte gick att köra i batchmode eller för att utdatan från dessa var mer svårparsad.

Verktyg för att mäta nätverkstrafiken behöver, om det nätverk där det testade systemet finns är isolerat, inte nödvändigtvis köras på det testade systemet, vilket medför att systemresursåtgången för verktyget inte lastar det testade systemet. Verktygen för att mäta nätverksresursutnyttjande hade därtill generellt en låg grad av CPU- och minnesutnyttjande på målarkitekturen.

Det är problematiskt att mäta systemresursutnyttjandet för de verktyg som inte körs kontinuerligt. Det går inte att garantera att verktyg som mäter systemresursutnyttjandet vid mätögonblicket lyckas fånga ett program vars exekveringstid är i sammanhanget är kort. Detta kan delvis överkommas genom att köra programmet upprepat t.ex. från ett shellscript, men detta garanterar ändå inte att mätverktyget hinner mäta systemresursutnyttjandet under den tid verktyget exekverar. Att använda ett program på detta sätt, vilket ofta varit önskvärt då kontinuerlig information eftersökts, skulle vidare medföra ett extra resursutnyttjande då processen startades på nytt upprepat. Det har därför varit önskvärt att i så stor utsträckning som möjligt använda mätverkyg som körs kontinuerligt.

Vi har inte haft tillgång till hårdvarubaserade verktyg för att få systeminformation, och vi har inte heller utvärderat verktyg för att läsa eventuella prestandaräknare eftersom de flesta sådana verktyg som vi hittat riktar sig mot Intels IA-32-arkitektur. Vår bedöming är dock att de verktyg som vi använt inte har störts det mätta systemet på ett sådant sätt att erhållen data är otillförlitlig.

4 Belastningsverktyg

Vårt behov av belastningsverktyg härör från ett behov av att undersöka hur systemet beter sig under last och hur resursstyrningsverktygen fungerar vid belastning. Dessa verktyg kan vara rätt triviala, då det i allmänhet inte behövs mycket kod för att processorn eller någon annan systemresurs ska belastas. Därför har vi inte behövt lägga mycket tid på att leta belastningsverktyg, eftersom det enda kravet är att det ska vara smidigt att använda. Förutom nedan beskrivna verktyg så har vi även utvecklat egna verktyg för att belasta vissa systemresurser.

stress

`stress` är ett verktyg som kan belasta många olika systemresurser som CPU, minne, sekundärt lagringsutrymme m.m. Programmet belastar systemresursen genom att starta nya processer som belastar resursen så specifikt som möjligt. Man kan ställa in hur många av dessa belastande processer som ska startas och därmed skräddarsy belastningen till önskad nivå. [118]

4.1 Diskussion

Det kan i vissa fall vara svårt att specifikt belast enbart en systemresurs. För att belasta nätverket från den testade enheten krävs t.ex. att nätverkspaket genereras på den testade enheten. Detta involverar systemets CPU. På inbäddade system, vars CPU-kraft kan vara begränsad, så kan detta medföra att CPU:n belastas signifikant. På aktuell målarkitektur var CPU-resurserna och tillgänglig nätverksbandbredd balanserade, vilket innebär att vid fullt utnyttjande av bandbredden så användes väsentligen all CPU-tid för att generera nätverkstrafik. På liknande sätt kan minnet vara begränsat vilket kan medföra att storleken av buffertar måste hållas låga, vilket i sin tur kan medföra en ökad belastning på systemets minne då önskan är att belasta t.ex. nätverket. Principiellt kan vid belastning av systemets CPU fördelningen mellan systemanrop och övriga anrop vara av betydelse, speciellt vad avser schemaläggning och huruvida processen är preempt-bar då linuxkärnan kan kompileras för olika grad av preemptbarhet.

5 Resursstyrningsverktyg

Resursstyrningsverktyg avser att begränsa, garantera eller styra tillgången till en resurs. De resurser som är av störst betydelse i detta arbete är CPU-tid, internminne samt nätverksbandbredd. I viss mån kan det även vara av intresse att översiktligt undersöka verktyg för att styra resurstilldelning vad avser prioritering till och tillgängligt sekundärminne.

5.1 Minnes- och processortidsstyrning

Klassiska prioriteter (nice) kan naturligtvis användas för att styra processers relativa prioritet, och därigenom ge olika processer mer eller mindre processortid. Nackdelen med detta är dock att det går att belasta systemet med ett stort antal processer vilket leder till att även processer med hög prioritet får begränsad CPU-tid.

Linux kan generellt binda processer till en viss virtuell CPU i system med flera kärnor eller med multithreading[26], vilket innebär en partitionering, om än grov, av tillgänglig CPU-tid.

5.1.1 cpulimit

`cpulimit` är ett userspace-verktyg som baserat på en process CPU-utnyttjande sänder SIGSTOP (stoppsignal) respektive SIGCONT (fortsättningssignal) till processen. Därmed blir begränsningen oberoende av den underliggande schemaläggaren i så mån att begränsningsfunktionen ligger utanför kärnans schemaläggare. Å andra sidan är `cpulimit` beroende av kärnans schemaläggning. `cpulimit` måste köras som root om en godtycklig process ska kontrolleras eller annars som samma användare som processen som ska kontrolleras. Verktyget justerar sin egen prioritet till nice-värdet -20 (högsta statiska prioritet för best effort-schemaläggning), och har en granularitet på ca 0,1 sekunder. `cpulimit` begränsar processer uppåt, vilket kan vara mindre intressant då i allmänhet en minsta garanterad CPU-användning är önskvärd snarare en övre begränsning. [119]

[26] Se `http://en.wikipedia.org/wiki/Simultaneous_multithreading`, `http://en.wikipedia.org/wiki/Multithreading_(computer_hardware)` eller `http://en.wikipedia.org/wiki/Hyperthreading`

5.1.2 Auto nice daemon

Auto nice daemon (AND) är ett program som justerar nice-värdet för processer eller skickar signaler till processer baserat på deras absoluta CPU-utnyttjande (total CPU-tid). Programmet körs i bakgrunden och justerar inte prioriteter för processer som ägs av root. Verktyget sänker enbart prioriteter. AND kan justera prioriteter baserat på processens ägare och grupptillhörighet, men även baserat på processens namn och förälder. Tre olika CPU-tidsgränser sätts generellt i verktyget, även om det finns vissa möjligheter att anpassa dessa så att de inte gäller alla processer som kontrolleras av verktyget. När CPU-tiden för en process överskrider någon av de tre CPU-tidsgränser som är möjliga att sätta, så justerar verktyget processens nice-värde, eller skickar en signal till processen. Vilket nytt nice-värde processen ska få eller vilken signal som ska skickas till processen är konfigurerbart. Verktygets funktionalitet begränsas signifikant av att absolut CPU-tid används för begränsningen, vilket gör det svårt att använda programmet för att begränsa processer som körs kontinuerligt. Vidare är CPU-tidsbegränsningarna väsentligen generella för hela systemet och den granularitet med vilken AND körs är relativt grov med ett standardvärde av en minut. [120]

5.1.3 Resource limits

Resource limits är ett system för att begränsa ett antal resurser för en process. Begränsningar, som införs genom enkla systemanrop för aktuell process, är antingen hårda eller mjuka. Kärnan tillser att resursen inte överskrider den mjuka begränsningen, där det maximala värde processen kan sätta motsvaras av den hårda begränsningen. Det är sålunda i allmänhet inte möjligt för en process att öka den hårda begränsning medan den mjuka begränsningen kan sättas upp till värdet av den hårda begränsningen. Processer med egenskapen CAP_SYS_RESOURCE kan dock förändra båda värdena godtyckligt[152]. Begränsningen ärvs av barn till processen, men den totala begränsningen för barn och förälder tillsammans ökar i och med detta då flertalet av begränsningarne gäller per process.

Resource limits kan begränsa ett antal olika resurser. Några i sammanhanget intressant är total minnesanvändning, total CPU-användning i sekunder, processens datasegments storlek, maximal filstorlek som processen kan skapa, antalet minnessidor som kan låsas till RAM, maximalt tak för nice (statisk prioritet för processer i schemaläggningspolicyn SCHED_OTHER), maximalt antal öppna filer, maximalt antal processer (inklusive trådar) som

kan skapas av användaren som äger processen, maximal realtidsprioritet för processen, maximal CPU-tid för realtidsprocesser och maximal stackstorlek för processen.

Pluggable authentication modules (PAM)[27] är ett modulärt biblioteksbaserat system för autensifiering av användare, och tillhandahåller ett system för att styra limits på användarnivå. PAM, genom pam_limits, tillhandahåller i stort sett samma begränsningar som limits-systemet, med några få tillägg som begränsning av antal möjliga samtidiga inloggningar. PAM tillser att limits införs genom att för program som har stöd för PAM införa begränsningarna om pam_limits konfigurerats för programmet. Eftersom PAM har utvecklats för att tillhandahålla autensifiering främst av användare, så finns PAM-stöd huvudsakligen för program som involverar initiering av användarsessioner, som t.ex. ssh och login. Detta är naturligtvis en fördel ur vissa aspekter, men gör också att systemet inte är generellt tillämpligt för att styra resursanvändningen i situationer när användarsessioner inte är involverade.

[121, 122, 123, 124]

5.1.4 Control task groups

Control task groups (cgrupper) är ett sätt att partitionera processer till grupper av sådana, vilka sedan kan tjäna som utgångspukt för tilldelning av systemresurser. Dessa grupper bildar hierarkier, vilka det kan finnas fler än en av. Varje enskild hierarki delar in systemets processer så att varje process är i exakt en cgrupp.

Ett (cgrupps-)subsystem är en modul som på något sätt använder cgruppens indelning av processer i sin funktion. Subsystemen tillhandahåller för det mesta någon form av resurskontroll, t.ex. genom att tilldela de olika grupperna av processer olika andel av en resurs. Till varje hierarki hör en grupp av subsystem som agerar på alla cgrupper i hierarkin.

Olika subsystem kan utnyttja samma hierarki eller använda olika hierarkier.

I sig gör alltså control taks groups inte något, utan det krävs att något subsystem kopplas till dessa. Det sker en ständig utveckling av Linux-kärnan och nya användningsområden av cgrupper tillkommer mer eller mindre kontinuerligt. Ett flertal olika subsystem existerar, varav några beskrivs kort nedan.

[27] Förutom referenser se `http://en.wikipedia.org/wiki/Pluggable_Authentication_Modules` och delvis `http://trac.des.no/openpam`

- cpuset är ett subsystem som är tänkt att användas för system med flera (virtuella) processorer och minnesnoder (non uniform memory access, NUMA) för att binda grupper av processer till vissa CPU:er eller minnesnoder. Av denna anledning är cpuset-subsystemet inte intressant att använda på målarkitekturen.

- Devices-subsystemet kan på bas av en process cgrupp-tillhörighet begränsa tillgången till olika enheter (eng: devices).

- Freezer-subsystemet möjliggör en gruppering av processer så att dessa kan stoppas och startas gemensamt, vilket kan vara av värde för att styra en grupp batch-arbeten.

- Memory-subsystemet medger en begränsning av mängden minne per cgrupp. Hanteringen av virtuellt minne är komplex och hur minne redovisas per cgrupp styrs delvis av hur Linux-kärnan kompilerats. Delat minne redovisas t.ex. under den cgrupp som först rör minnet, och när en process flyttas från en cgrupp till en annan cgrupp så flyttas inte redovisningen av allokerat minne med vid flytten. Memory-subsystemet medger också inspektion av mängden minne som allokerats och vilket typ av minne detta avser, som t.ex. cache-minne, RSS etc. Om minnet begränsas med detta system och redovisningen av minne övergår denna begränsning så initieras en oom kill av någon process under ifrågavarande cgrupp. Detta är dock i viss mån beroende av hur systemet är konfigurerat vad avser t.ex. överutnyttjande av minne. Minnesallokering av en tråd redovisas under den PID som tråden tillhör.

- Det finns även ett subsystem för att hantera I/O-schemaläggning. CFQ-I/O-schemaläggaren kan kompileras med stöd för grupp-I/O-schemaläggning, vilket medger stöd för prioritering av I/O på bas av cgrupp. Ifråga om detta kan även olika nivåer av isolering ställas mellan cgrupper och proportionerlig del av I/O kan sättas för olika cgrupper. Delar av detta har dock introducerats först i senare Linux-versioner (2.6.33).

- CPU-accouting-systemet gör det möjligt att erhålla information om CPU-användning per cgrupp och hur denna fördelas mellan user mode och kernel mode.

Gruppschemaläggning (eng: group scheduling) kan använda cgrupper som bas för grupp-schemaläggning. När gruppschemaläggning används så fås en

schemaläggning som är rättvis i så mån att grupperna tilldelas CPU-tid utifrån deras relativa CPU-andelar enligt specifikation i cgrupperna. Den relativa tilldelningen är en garanti snarare än en begränsning.

Vilka systemresurser som går att styra är således i hög grad beroende av vilka subsystem som implementerats.

För att använda cgrupper så monteras ett virtuellt filsystem. Initialt finns alla systemets processer i en rot-cgrupp. För att skapa en ny cgrupp så skapas en ny katalog inuti det virtuella filsystemet. I detta filsystem finns virtuella filer som man kan läsa och i vissa fall skriva konfigurationsdata till, som t.ex. vilka processer (e.g. i filen `tasks`) som ingår i ifrågavarande cgrupp. Den relativa CPU-tidsandelen sätts i filen `cpu.shares` medan minnebegränsningar kan sättas i `memory.limit_in_bytes` om relevanta subsystem är monterade. Dessa filer styr hur kärnan tilldelar systemresurser till de processer som finns i cgrupperna.

[21, 125, 126, 127, 128, 129, 130, 131]

Det är även möjligt att klassificera nätverkstrafik baserat på cgruppstillhörighet.

5.1.5 Virtualiseringverktyg

Det finns ett flertal olika virtualiseringsverktyg till Linux varav många tillhandahåller resursstyrning utöver vad som anges ovan. Virtualiseringsverktyg medger att ett system kan partitioneras så att flera operativsystem kan köras på systemet eller så att systemet upplevs köra flera operativsystem samtidigt. Virtualisering kan ske på flera olika nivåer. Virtualisering på operativsystemsnivå innebär att operativsystemets kärna tillhandahåller flera isolerade user space-instanser istället för enbart en. Ett operativsystem körs, men tillhandahåller en isolering mellan olika grupper av processer så att dessa upplevs vara helt skiljda. [100, 101, 95, 102, 103, 105, 106, 107, 104]

Generellt kan sägas att virtualisering på operativsystemsnivå, t.ex. med OpenVZ, överlag ger ett lägre extrautnyttjande av resurser än hypervisorbaserade virtualiseringstekniker som paravirtualisering, framför allt vad avser CPU-tid. Detta exemplifieras vid en jämförelse mellan Xen och OpenVZ där extrautnyttjandet av resurser jämfört en standardkärna är marginellt för OpenVZ men betydande för Xen. Skillnaden beror sannolikt på ett ökat antal cache-missar (L2-cache) i Xen jmf. OpenVZ. [94, 99, 108]

Xen finns bl.a. för modernare processorer i Intel[28] ia-32- och ia-64-arkitekturerna, men ej för målarkitekturen. Linux tillhandahåller som standard KVM (Kernel based Virtual Machine). KVM medger full virtualisering för arkitekturer med hårdvarustöd för detta, vilket för närvarande är processorer med Intels och AMD:s VT- respektive AMD-V-teknologi. KVM finns ej för målarkitekturen. I KVM är varje virtualiserat subsystem en egen process som styrs av Linux normala schemaläggare. Detta skiljer från Xen som tillhandahåller flera schemaläggare med olika egenskaper för de virtualiserade subsystemen. [109, 110, 19]

Linux tillhandahåller även User Mode Linux (UML), där en Linux-kärna kompileras för att köras helt i user mode. Linux-värden kan därvid sägas utgöra den maskin på vilken en Linux-kärna, gästen, kompilerad för denna maskin körs. UML möjliggör att en sådan Linux-gäst tilldelas en viss mängd minne. UML finns inte för målarkitekturen. [111, 112]

VServer, FreeVPS, Linux Containers och OpenVZ är, liksom Solaris Zones för Solaris och FreeBSD Jail för FreeBSD, system för virtualisering på operativsystemsnivå. VServer, FreeVPS och OpenVZ kräver att Linux-kärnan patchas och dessa tillhandahåller via dessa patchar nya möjligheter för resursstyrning utöver vad som normalt finns i Linux, bl.a. genom att införa nya schemaläggare. FreeVPS bygger ursprungligen på VServer, och tycks enligt sin hemsida inte uppdaterats sedan ca 2007. Linux Containers bygger enbart på användandet av Linuxkärnans stöd för namnutrymmen (eng: namespaces), d.v.s. möjlighet att gruppera processer i namnutrymmen. Linux har stöd för sådana namnutrymmen vad avser processid, datornamn, användare, nätverk, filsystem, IPC m.m. Linux Containers tillhandahåller inte resursstyrning utöver vad Linuxkärnan normalt tillhandahåller. VServer använder begreppet kontext (eng: context) för den grupp av processer som schemaläggs och isoleras tillsammans, och inför även en kontextidentifierare för dessa (eng: context id). VServer utökar resource limits så att begränsningarna enligt ovan kan sättas per virtualiserad kontext, och inte enbart per process, och introducerar även ett antal nya resurser som kan begränsas. En begränsning av en kontexts RSS ger oom kill vid överskridande av den övre gränsen vid normal minnesallokering, medan ett överskridande av en contexts hårda begränsning av AS (maximalt adressutrymme; eng: maximum address space) medför att minnestilldelningen inte tillåts. [95, 97, 98, 99, 96]

VServer introducerar även en token bucket-schemaläggare för virtualiserade kontexter. En token bucket är konceptuellt en struktur som kontinuerligt

[28] `http://www.intel.com/`

fylls med tokens tills den är full. När resursen som styrs av en token bucket används så plockas tokens ut ur strukturen motsvarande resursens användning. När det inte längre finns några tokens kvar så är resursen inte längre tillgänglig för den som utnyttjar resursen som är kopplad till aktuell tocken bucket, och resursen blir därmed tillgänglig för andra. VServer utökar dock detta något så att CPU-användning kan tillåtas rättvist när processorn annars inte skulle belastats. VServer medger också att via cgrupps-systemet sätta hårda gränser för CPU-utnyttjande, där således en övre gräns för CPU-utnyttjande kan sättas per virtualiserad kontext. Detta är av betydelse i virtualiseringssammanhang där det kan vara intressant att för t.ex. olika kunder som köper en virtualiserad miljö begränsa tillgänglig CPU-andel, men funktionaliteten torde i övrigt inte vara av samma signifikans då det är mer intressant att garantera en minsta CPU-tid snarare än att sätta en övre CPU-tidsbegränsning. I viss mån kan dock en hård övre begränsning medföra att resurser tillgängliggörs för andra processer. VServer saknar stöd för hantering av block-I/O utöver vad Linux-kärnan tillhandahåller. [113, 114, 115]

OpenVZ tillhandahåller i viss mån liknande funktionalitet med t.ex. isolering av nätverk mellan containers, quotas dels för varje container dels inom varje container, proportionerliga CPU-andelar för varje container eller övre begränsning av CPU-andelen för en container, och block-I/O-begränsning dels per container dels för processer inom en container. [116, 117].

5.1.6 Realtidsprocesser

Det problem som beskrivs ovan, d.v.s. att belasta systemet med ett flertal processer och därmed begränsa processortiden för en viktig process, kan övervinnas genom att använda realtidsprocesser. Realtidsprocesser har alltid högre prioritet än ordinära processer och de är strikt bundna av den realtidsprioritet som processen har. Genom att använda realtidsschemaläggning är det således möjligt att garantera vissa processer tillräcklig CPU-tid. Problemet härvidlag är att en realtidsprocess helt kan lägga beslag på all CPU-tid. Detta problem kan i viss mån reduceras då det är möjligt att ge kärnan parametrar som anger hur stor andel av CPU-tiden som maximalt tillåts för realtidsprocesser. I sammanhanget bör också påpekas att det är möjligt att använda cgrupper även för realtidsprocesser, där det går att begränsa tillgänglig tid för realtidsprocesser per cgrupp. Det är sålunda möjligt att begränsa CPU-tiden för realtidsprocesser baserat på vilken cgrupp processen ingår i. Detta fungerar så att för varje cgrupp så anges den andel av CPU:ns realtidsandel realtidsprocesser i cgruppen ska tilldelas. Rot-cgruppen tidsandel för realtidsprocesser måste vara mindre än

eller lika stor som systemets maxgräns för realtidsprocesser. Globalt för alla rot-cgruppens barn måste gälla att $\sum_{i=1}^{n} C_i \leq M$, där n är antalet cgrupper, C_i är CPU-andelen för realtidsprocesser i varje cgrupp och M är tiden för realtidsprocesser i rot-cgruppen. De CPU-andelar som tilldelas barn till rot-cgruppen kan dock få konkurrens från realtidsprocesser som ligger i själva rot-cgruppen. Realtidsprocesser i cgrupper hanteras dock för närvarande inte hierarkiskt.

5.2 Nätverksstyrningsverktyg

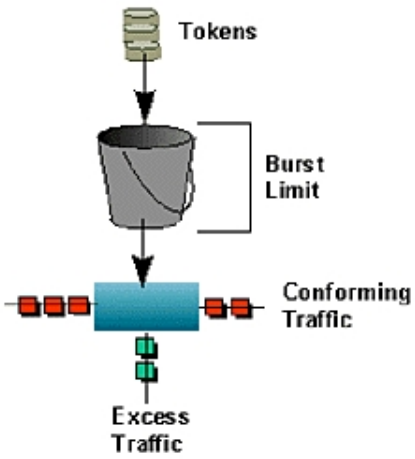
tc/tcng

tc är ett traffic shaping-verktyg till Linux. Såväl inkommande och utgående trafik kan styras, och komplexa trädstrukturer av olika s.k. qdiscs, queuing disciplines, kan användas för att erhålla komplex trafik-styrning. qdiscs styr nätverkstrafiken på olika sätt. Det finns enkla FIFO-qdiscs, som skickar inkomna paket i FIFO-ordning så snabbt som är möjligt. Det finns också mer komplicerade qdiscs, som htb (hierarchical token bucket). Denna qdisc fungerar som en vanlig token bucket (se avsnitt 5.1.5 och figure 3), fast det finns även en annan gräns utöver den vanliga, taket. Om qdiscen inte har några tokens, så försöker den låna tokens av föräldern upp till taket. Med flera lager av dessa htb:s går det att skapa komplexa beteenden. Till dessa qdiscs går det att koppla filter. Dessa kontrollerar alla paket som kommer och skickar vidare dem till rätt qdisc. Det finns även speciella filter, s.k. poliser, som kan kasta paket om flödet överstiger en viss gräns. `tc` interagerar även med netfilter[132]. Netfilter kan markera paket, bl.a. baserat på PID, UID, GID, och sessionsid, som sedan kan behandlas av `tc` baserat på markering. Det är även möjligt att klassificera paket baserat på genererande process cgruppstillhörighet, vilket dock inte undersökts närmare. Vidare finns ett antal kärnpatchar för att tillhandahålla ytterligare möjligheter till traffic shaping.

Det största problemet med `tc` är att syntaxen är arkaisk och otymplig. Exempelvis lägger följande tc-kommando till ett filter till en qdisc:

```
tc filter add dev eth0 parent 2:0 protocol all prio 1 \
handle 2:0:1 u32 ht 2:0:0 match u16 0x50 0xffff at 0 classid 2:1
```

`tcng` är ett verktyg som försöker åtgärda detta. Detta verktyg är skapat för att tolka ett ganska simpelt, Algol-influerat språk och sedan omtolka detta till `tc`-kommandon. Detta gör det enklare att hantera `tc`:s komplexa syntax.



Figur 3: Token bucket

Ett exempel på ett tcng-script är följande, som kastar alla utgående paket från http-porten:

```
//använd interface eth0
dev "eth0"{
        //egress styr utgående paket
        egress {
                /* kastar paket om källporten på tcp-headern
                är lika med http-porten  */
                drop if tcp_sport == PORT_HTTP;
        }
}
```

`tcng` slutade dock att utvecklas 2004, så all funktionalitet som tillhandahålls av `tc` är inte implementerad, och de tc-kommandon `tcng` genererar är inte alltid kompatibla med nyare versioner av `tc`. Det fungerar dock utmärkt att använda `tcng` för att generera ett script som sedan kan tjäna som utgångspunkt när det slutliga tc-scriptet skapas.

[135, 136, 137, 138, 139]

5.3 Prioritering av sekundärminne

Det har funnits flera möjligheter för bandbreddsbegränsning till sekundärt lagringsutrymme i form av kärnpatchar till Linux. Bland dessa kan nämnas

io-throttle controller[141, 142, 143, 144] som även haft stöd för cgrupper.

ionice

Verktyget `ionice` kan användas för att sätta I/O-schemaläggningsklass (idle, best effort eller real time) samt prioriteter för processer när de arbetar mot blockenheter som har stöd för respektive funktion, t.ex. när CFQ-schemaläggaren används. Då målarkitekturen inte använder hårddiskar utan flash-minne som sekundärt lagringsutrymme så är betydelsen av I/O-schemaläggare mindre, och en noop-schemaläggare är tillräcklig. På aktuella system med flash-minne finns inte heller stöd för annan I/O-schemaläggare än noop-schemaläggaren. [140]

ioband

Ioband och dm-ioband är ett system som bygger på device mapper och medger bandbreddsbegränsning mot block-enheter oberoende av typ och oberoende av schemaläggare. Systemet medger bandbreddskontroll baserat på process, cgrupp, användare eller grupptillhörighet, per filsystem m.m, och tillhandahåller flera begränsningsmöjligheter. Bandbredden kan tilldelas proportionellt mot en given vikt, som garanterade bandbredder eller som begränsande sådana, och det är även möjligt att skapa hierarkier för bandbreddsbegränsning. Systemet använder en token bucket-algoritm för att begränsa bandbredden. Med detta system är det möjligt att begränsa bandbredden även mot block-enheter som enbart har noop-schemaläggaren. [145, 146, 147, 148]

quota

Diskquotas är ett system för att begränsa diskanvändning för en användare eller en grupp. Begränsningar kan sättas för antal tillgängliga block och inoder[29] på en enhet, och det finns såväl en mjuk som en hård begränsning. Den mjuka begränsning kan överskridas under en viss tid, medan den hårda begränsning inte går att överskrida. [149]

5.4 Diskussion

För cpulimit krävs en cpulimit-process per begränsad process och det finns inte någon möjlighet att begränsa grupper av processer tillsammans. Detta

[29]Se `http://en.wikipedia.org/wiki/Inode`

är en betydande begränsning av verktyget liksom att cpulimit i sig, även om det körs högprioriterat, principiellt kan begränsas av andra processer på systemet, som t.ex. av realtidsprocesser. Vidare medför cpulimit ett tak för CPU-tiden för den övervakade processen. Detta medför att verktyget inte är tillgängligt för generell användning för att partitionera CPU-tid annat än i undantagsfall.

Vad gäller AND kan till del samma invändningar anföras. AND kan som egen process svälta. AND begränsar vidare processer baserat på total CPU-tid vilket gör att processer vilka ska köras kontinuerligt förr eller senare alltid kommer att nå begränsningarna. Vidare är granulariteten med vilka CPU-tiderna kan ställas in för grova och kan inte ställas för varje process eller för grupper av processer, och granulariteten för övervakningen är inte heller adekvat i sammanhanget.

Resource limits ger möjlighet att begränsa flera resurser för processer och begränsningar kan till viss del införas på UID-nivå. CPU-tidbegränsningen är av samma anledning som vad gäller AND inte lämplig då den avser total CPU-tid, och begränsningen är dessutom ett tak. Minnesbegränsningen är dock intressant och kan vara av värde i vissa sammanhang, och då speciellt om en process inte skapar nya processer. Det är dock problematiskt att vid `fork` så ärver den nyskapade processen föräldraprocessens begränsningar och begränsningarna ställs sålunda, med vissa undantag, inte för den grupp av processer som skapas vid upprepat användande av fork utan för varje process. Detta kan dock i viss mån motverkas genom att ange ett maximalt tak för antalet processer som får köras som en användare, men i det fallet blir begränsningarna för stränga eftersom den totala begränsing består av ett antal fragment av hårda lägre begränsningar. Flertrådning är dock inte ett problem i detta sammanhang utan begränsningarna gäller per PID. Notabelt i sammanhanget är att resource limits (se nedan) inte medför att oom kill startas utan att minnesallokering fallerar. Linux-versioner som skiljer från 2.4.x, med $x < 30$, kan dock inte med limits begränsa enbart RSS.

Cgrupper är ett behändigt verktyg för att styra resurser för grupper av processer. En fördel med cgrupper är att barnprocesser och trådar automatiskt hamnar i samma cgrupp som sin förälder och att resursstyrningen gäller för en hel cgrupp. Med existerande subsystemen och kopplingen till bl.a. gruppschemaläggaren är det därmed möjligt att på ett behändigt sätt styra CPU-tidsandelar för grupper av processer genom att ge dessa andelar av CPU-tiden. Denna partitionering innebär dock inte någon hård övre begränsning för processer som inte är satta som realtidsprocesser. Det går däremot att sätta en hård begränsning på realtidsprocesser. Det är möjligt att införa tak för minnesanvändningen per

cgrupp och detta system kan också användas hierarkiskt. Detta medger t.ex. att en lägre begränsning än summan av barnens begränsningar kan sättas för föräldern. I sammanhanget är det komplicerande att ett överskridande av minnesbegränsningen korrigeras med oom kill. En process kommer därvid att dödas, men detta kan leda till att systemet slutar fungera på avsett sätt. Vilken process som ska dödas kan dock delvis styras genom flaggor i proc-filsystemet. Någon varning inför en förestående oom kill fås dock ej, även om det finns kernel-patchar för att införa sådan funktionalitet. Det är inte heller säkert att den process som dödas är den process som utifrån förväntad funktionalitet överutnyttjar minnet. För system där en oom kill medför en omstart av systemet åstadkommer därför denna begränsning en ökad risk för att systemet startar om, med risk för att detta sker frekvent om någon process inte motsvarar förväntade kvalitetskrav. Detta kan medföra problem i relation till 3:e-partsprogramvara på inbäddade system då ett dysfunktionellt program inte kan tillåtas leda till upprepade frekvanta omstarter av systemet. Å andra sidan är det inte heller rimligt att ett inbäddat system som används t.ex. för övervakningsändamål tillåts vara i ett icke funktionellt tillstånd efter att någon nödvändig process har dödats. Det sätt som minnesredovisningen sker för trådar kan också vara problematiskt. För multiprocessorsystem eller system med hyperthreading kan även cpuset-subsystemet vara intressant. I appendix A och B redovisas vissa intressanta filer i proc-filsystemet och kompileringsflaggor för Linux som är relevanta i sammanhanget.

Virtualiseringsverktyg på operativsystemsnivå, som VServer och OpenVZ, medför ett visst begränsat extraarbete jämfört ett ej virtualiserat system. Linux-kärnan måste i vissa fall patchas, och verktyg för att hantera kontexter måste portas till målarkitekturen. Att använda en patchad kärna medför också risk för att systemet blir mindre stabilt. För samtliga virtualiseringsverktyg som utökar Linux-kärnans möjligheter till resursstyrning så krävs att kärnan patchas, och av de verktyg vi undersökt så är det endast KVM, UML och Linux Containers som använder en opatchad kärna. KVM medför dock sannolikt ett större extra-utnyttjande av systemresurser jämfört virtualisering på operativsystemsnivå och finns inte för målarkitekturen, och inget av dessa tre verktyg tillhandahåller resursstyrning utöver vad Linux-kärnan normalt erbjuder. Vissa verktyg, som VServer och OpenVZ, tillhandahåller dock möjligheter för resursstyrning utöver vad Linux-kärnan normalt tillhandahåller. VServer ger t.ex. möjlighet till minnesbegränsning utan att initiera oom kill i situationer där cgrupper skulle initierat oom kill, samt hård CPU-tidsbegränsning för kontexter. OpenVZ tillhandahåller liknande funktionalitet. Det kan därför i vissa fall vara intressant att använda virtualiseringstekniker för att begränsa

resurstillgången för grupper av processer. Virtualiseringstekniker som Xen medför däremot ett betydligt extrautnyttjande av resurser och borde därigenom inte vara lika intressanta att använda på inbäddade system. Den huvudsakliga extra funktionalitet dessa i grunden tillhandahåller är möjlighet att köra flera olika operativsystem; ett behov som i allmänhet inte föreligger på inbäddade system av aktuell storlek.

Av ovanstående virtualiseringsverktyg så är enbart OpenVZ, VServer, FreeVPS och Linux Containers så plattformsoberoende att de med en inte alltför stor tidsåtgång skulle kunna portas till målarkitekturen. Då inte alla verktygen kunde undersökas valdes VServer bl.a. p.g.a. stöd för hård CPU-begränsning och att ett tidigt försök att porta OpenVZ och VServer gav för handen att en portning av VServer skulle vara mer lättillgänglig. Övriga verktyg saknade resursbegränsningsmöjligheter utöver vad Linux-kärnan tillhandahåller, föreföll inte vara under aktiv utveckling eller bedömdes medföra ett onödigt stort extra resursutnyttjande. Det var dock inte möjligt att fullständigt porta VServer med rimlig tidsåtgång till varken ARTPEC-B eller ARTPEC-3. VServer portades dock till systemprototypen. För att kunna använda VServer krävs normalt ett antal verktyg, vilka i sin tur kräver ett stort antal verktyg med funktionalitet utöver vad som finns i Busybox-versionerna av motsvarande program (bash, grep, egrep, fgrep, mount, umount, pgrep m.fl.) VServer-verktygen samt de program dessa var beroende av portades till systemprototypen.

Användning av realtidsprocesser medför en strängare CPU-tidsfördelning än nice-prioriteter och sannolikt mindre jitter, men också risk för svält för lågt prioriterade processer. Det senare problemet kan överkommas genom att enbart tilldela realtidsprocesser en andel av tillgänglig CPU-tid, men en realtidsprocess kan fortfarande svälta en lägre prioriterad realtidsprocess. Användandet av realtidsprocesser medför inte heller någon partitionering av systemresurser för grupper av processer, men kan användas tillsammans med cgrupper varvid denna begränsning undanröjs eftersom varje cgrupp kan tilldelas en viss del av systemets realtidsandel. Den tid som systemet avsätter för realtidsprocesser är vidare uppåt begränsad, men med en som standard låg granularitet. Då realtidsprocesser har strikta prioriteter medför, speciellt om inte cgrupper används, ett överutnyttjande av dessa risk för svält för andra processer och då även andra realtidsprocesser.

Möjligheten att kombinera cgrupper och realtidsprocesser är intressant, eftersom detta kan kombinera styrkorna i båda metoderna. Realtidsprocesser kan användas i situationer där en process måste garanteras tillräcklig exekveringstid och låga nivåer av jitter, och speciellt om processen inte kräver så mycket systemresurser. Detta minskar risken för att andra

processer svälter. För att få ett någorlunda förutsägbart beteende hos realtidsprocesserna bör dock undvikas att dessa läggs i rot-cgruppen. Om processer läggs där, är det svårt att avgöra hur mycket CPU-tid de skulle få i konkurrans med andra realtidsprocesser.

Sammantaget medför detta att i en utvecklingssituation och -miljö där inte speciella hänsyn tagits till realtidsaspekter så bör realtidsprocesser användas enbart för särskilt utvalda processer.

Prioritering av nätverkstrafik är ett komplext område som involverar inte bara sändaren och mottagaren utan även eventuella mellanliggande system. IP-paketen tillhandahåller speciell header-information som delvis utgör bas för en sådan styrning. För sändaren kan sådan information tillsammans med även annan förbindelse-relaterad IP-header-information, information om sändaren som PID och UID m.m. samt processens cgrupps-tillhörighet utgöra grund för att prioritera nätverkstrafiken. Linux har stöd för avancerad och komplex styrning av utgående nätverkstrafik, men i sammanhanget kan det vara värdefullt att påpeka att de processer som genererar nätverkstrafik också lastar CPU och till viss del minne. Om en sådan process begränsas vad avser CPU, eller om systemet lastas så att en sådan process får mindre CPU-tid, kan detta innebära att processens nätverkstrafik samtidigt begränsas.

Quotas är ett klassiskt sätt för att begränsa användningen av sekundärt lagringsutrymme för användare och grupper av sådana, och kan vara av värde att använda även på inbäddade system. Vissa av virtualiseringsverktygen som OpenVZ medger också möjlighet att begränsa tillgången till sekundärt lagringsutrymme per container. För aktuellt målsystem är däremot prioritering av I/O till sekundärt lagringsutrymme inte en signifikant fråga av anledningar som beskrivits tidigare. Målarkitekturen använder sig, likt många andra små och medelstora inbäddade system, inte av hårddiskar, och de algoritmer som finns i Linux för prioterar I/O på bas av och optimerar I/O genom att att minska läs- och skrivhuvudenas rörelse över hårddisken. Möjligen hade ioband kunnat vara intressant att utvärdera.

Det är naturligtvis möjligt att kombinera olika verktyg och det som i så fall skulle vara mest intressant är att kombinera cgrupper och VServers möjlighet till hård begränsning av CPU-tiden och minnesbegränsning utan oom kill. Detta finns dock i viss mån redan i VServer, men är inte lika tillgängligt som en direkt användning av cgrupper.

6 Egenskapta verktyg

Vi har skapat flera olika verktyg för att hjälpa oss att undersöka resursanvändningen på kameran. Vissa verktyg skapade vi för att vi saknade viss funktionalitet som vi använde ofta. Andra skapade vi för att testa en viss aspekt av prestandan på kameran. Nedan beskrivs en delmängd av de verktyg vi har skapat.

Lastsimulerings- och signaleringsverktyg

För att få en uppfattning om variabilitet i svarstider under belastning skapade vi ett verktyg som skickar en tidsstämpel periodiskt till en klient. Inget av de verktyg vi lyckades porta till målarkitekturen kan ge en uppfattning om svarstider, och mätning av svarstiden i mjukvara är givetvis problematiskt. Programmet kan ge uppskattning av variabiliteten i svarstider under olika grader av belastning. Intervallet är konfigurerbart. Under detta intervall kan serverdelen av programmet sättas att utföra beräkningar för att därmed simulera viss last. Dessa beräkningar sker enbart i user mode. Sålunda sänder programmet periodiskt med en konfigurerbar sovtid en tidsstämpel, tagen på systemet där serverdelen av programmet körs, till klienten. Nästkommande signal är beroende av den föregående signalen, och bestäms inte då programmet startas. Alternativt så skulle signalernas önskade tidpunkt kunna bestämmas redan vid starten av programmet så att en signal alltid förväntades $t_n = t_0 + n * c$ där n är ett positivt heltal och c är cykeltiden, och om programmet fördröjdes så inträffade inte sovtiden förrän programmet kommit ifatt sin cykeltid. Detta skulle dock medföra olägenheten att programmet ändrade beteende så att sovtiden uteblev om cykeltiden överskreds, t.ex. om schemaläggningen påverkats, vilket i sin tur negativt påverkat en uppskattning av variabiliteten i signalen. Vid en belastning av processorn kan således något av följande inträffa för processen: cykeltiden förlängs, cykeltiden börjar variera, eller så medför belastningen inte någon signifikant påverkan på processen. Klienten tar enbart emot tidsstämplarna och sparar dessa till en fil.

Vi har även utvecklat verktyg för att belasta nätverket, såväl vad avser TCP- som UDP-trafik[30].

Axis har därtill ett färdigutvecklat system för att mäta jitter i den bildström som levereras från kameran.

[30]Se t.ex. The TCP/IP Guide: `http://www.tcpipguide.com/`

Realtidsprioritetsverktyg

Vi har utvecklat en variant av `renice`, som även kan användas för att förändra schemaläggningspolicy, till såväl någon av realtidspolycierna som till standardpolicyn, och därtill sätta realtidsprioritet för processer. Detta verktyg överensstämmer relativt väl med standardverktyget `chrt` men skiljer något i funktionalitet.

Processinformationsverkyg

Vi har även utvecklat ett verktyg som via nätverket skickar viss information från proc-filsystemet för alla processer till en klient. Denna information innehåller en stor mängd data, såsom PID, kommando, processtillstånd, föräldra-PID, processgruppid, sessionsid, sidfel för processen, tid som processen kört i user mode respektive i kernel mode m.m. Informationen skickas repetitivt med ett konfigurerbart intervall.

Detta verktyg skapade vi för att få ut rå information om processers resursutnyttjande, som vi sedan kunde analysera manuellt.

Verktyg för hantering av processer och cgrupper

Vi skapade ett antal verktyg för att enklare hantera processer och cgrupper. Dessa underlätade betydligt hanteringen och testandet av denna resursstyrningsmekanism.

`cgmv` är ett verktyg som flyttar alla processer med ett visst namn, eller med en viss PID, till en angiven cgrupp. Man kan även specificera att man vill ha med alla trådar som hör till en process.

`cgstart` startar en process i en angiven cgrupp.

7 Utvärdering av systemresursstyrningsverktyg på inbäddade system

För att undersöka dels CPU-tidsandel, dels försöka värdera jitter med olika resursstyrningsverktyg så utfördes ett antal tester.

7.1 Påverkan på bildströmningsprogramvaran av ökande CPU-belastning

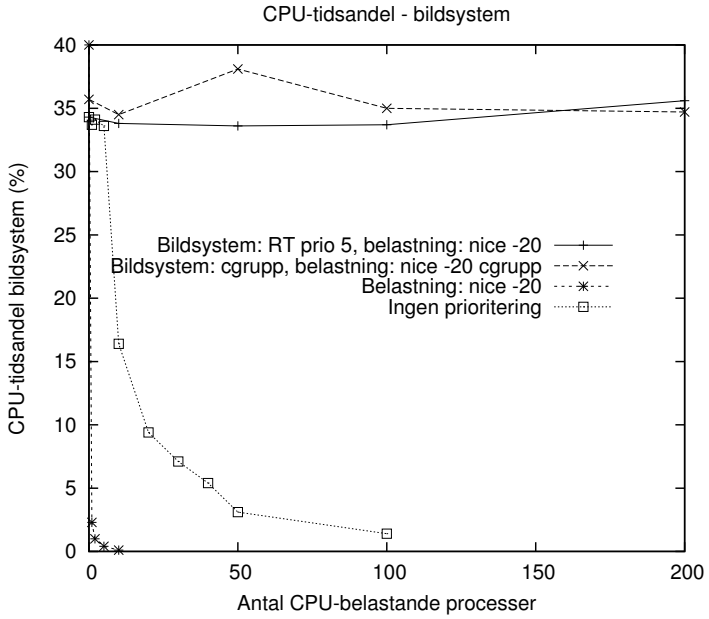
Först så undersöktes hur bildströmningsprogramvarna (hädanefter "bildsystemet") på den ARTPEC-3-baserade P3301 hanterade en situation där ett antal CPU-belastande processer startades samtidigt på enheten. Nedanstående beskriver vilka tester som gjordes.

- Bildsystemet och de CPU-belastande processerna med ordinär statisk prioritet (nice-värde 0) och utan något resursstyrningsverkyg,
- de CPU-belastande processerna med en hög statisk prioritet (motsvarande ett lågt nice-värde) och bildsystemet med ordinär statisk prioritet och utan något resursstyrningsverktyg,
- Bildsystemet med ordinär statisk prioritet i bas-cgruppen (vilken alltid har cpu.shares 1024) och de CPU-belastande processerna, med ordinär statisk prioritet, i en cgrupp med cpu.shares 256.
- Bildsystemet med ordinär statisk prioritet i en cgrupp med cpu.shares 2048, och de CPU-belastande processerna med hög statisk prioritet i en cgrupp med cpu.shares 256,
- Bildsystemet som en realtidsprocess enligt SCHED_RR, medan de CPU-belastande processerna kördes med hög statisk prioritet.

Antalet CPU-belastande processer (`stress`) var i varje test mellan 0 och 200, där de flesta tester kördes med 1, 10, 50 och 100 `stress`-processer och alla tester kördes med 0 `stress`-processer (se t.ex. figur 4). CPU-tidsandelen beräknades som ett medelvärde över erhållna värden från `top` för kortare konsekutiva tidsperioder, där datainsamlingstidens total längd var minst 1 minut och ofta mer än två minuter. Datainsamlingshastigheten samt antalet bilder per sekund togs som typvärden från erhållna bilder (denna data kan fås överlagrad de erhållna bilderna) efter det att bildströmmen

stabiliserats. Speciellt den angivna överföringshastigheten varierade under datainsamlingen. Noterbart är att om det finns en betydande variabilitet i de kortare tidsperioder mellan vilka `top` hämtade data så påverkas medelvärdet så att detta blir felaktigt. Genom att köra `top` som en realtidsprocess så försökte vi minimera en sådan variabilitet.

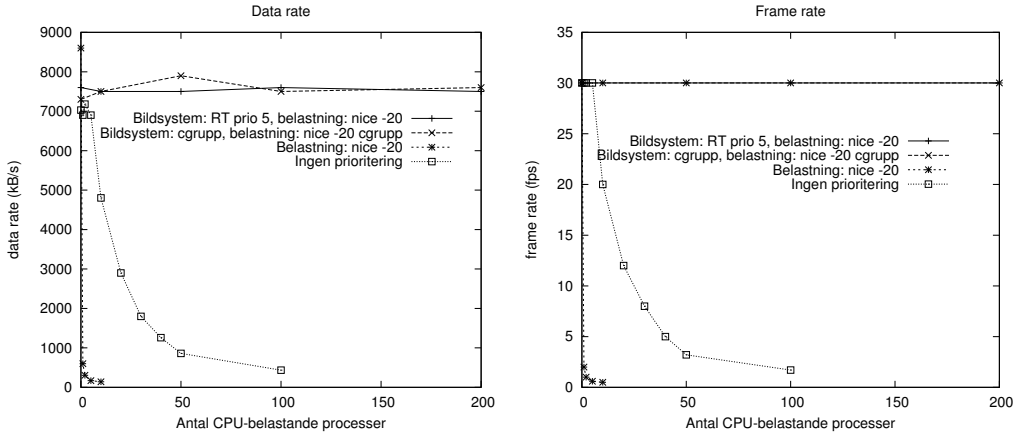
Hur mätningarna utfördes beskrivs i detalj i Appendix C.



Figur 4: Bildsystemets CPU-tidsandel vid olika belastning och med olika resursstyrningsverktyg.

Ur figur 4 ses att då ingen resursstyrning används så faller CPU-tidsandelen för bildsystemet drastiskt mellan 5 och 10 CPU-belastande processer respektive redan för 1 CPU-belastande process när denna har en hög statisk prioritet. Detta avspeglas också i sjunkande datainsamlingshastighet samt sjunkande antal bilder per sekund (se figur 5). När bildsystemet körs som en realtidsprocess samt då de CPU-belastande processerna körs i en cgrupp med låg cpu.shares så bibehålls dock en hög CPU-tidsandel för bildsystemet, liksom en hög datainsamlingshastighet och ett högt antal bilder per sekund.

Det är således klart att användade av cgrupper liksom användande av realtidsprocesser kan bibehålla en tillräcklig CPU-tidsandel för bildsystemet. Detta visar dock inte huruvida bildsystemet också kan tillhandahålla bilder med ett låg nivå jitter.



(a) Typvärden för bildsystemets datainsamlingshastighet.

(b) Typvärden för bildsystemets frame rate.

Figur 5: Typvärden för bildsystemets datainsamlingshastighet och frame rate.

7.2 Periodvariabilitet vid användning av cgrupper och realtidsprocesser

För att utreda huruvida jitternivåerna kunde hållas låga planerade vi initialt att använda jitter-mätningar enligt RFC 3550. Det visade sig dock inte vara lika enkelt som vi hoppats på, och vi använde därför ett enklare system med ett egentuvecklat enkelt signaleringsverktyg, vilket beskrivits tidigare.

Följande tester gjordes på den ARTPEC-3-baserade P3301:

- Signaleringsverktyget utan några arbetsiterationer, med ordinär statisk prioritet och utan någon resursstyrning. För att belasta systemet kördes samtidigt 0, 1, 2, 3, 4, 5, 10, 50 och 100 CPU-belastande processer med ordinär prioritet. Signaleringsverktygethade en egen CPU-belastning på ca 0%.
- Som ovanstående, fast med 200 arbetsiterationer, vilket gav en CPU-belastning för signaleringsverktyget på ca 20-30% då systemet i övrigt ej belastades.
- Som ovanstående, fast med 500 arbetsiterationer, vilket gav en CPU-belastning för signaleringsverktyget på ca 100% då systemet i övrigt ej belastades.

- Signaleringsverktyget med 200 arbetsiterationer, med ordinär statisk prioritet, men placerad i en cgrupp med cpu.shares 2048. Rot-cgruppen hade cpu.shares 1048, d.v.s. standardvärdet som inte går att förändra, och de processer som fanns i denna var genomgående i sovande tillstånd. För att belasta systemet kördes 0, 1, 2, 3, 4, 5, 19, 50 eller 100 CPU-belastande processer med ordinär statisk prioritet, men placerade i en cgrupp med cpu.shares 256.
- Som ovanstående, fast med 300 arbetsiterationer, vilket gav en CPU-belastning för programmet på ca 45% då systemet i övrigt inte belastades.
- Som ovanstående, fast med 400 arbetsiterationer, vilket gav en CPU-belastning för programmet på ca 100% då systemet i övrigt inte belastades. Data från detta redovisas inte i någon figur.
- Som ovanstående, fast med 500 arbetsiterationer, vilket gav en CPU-belastning för programmet på ca 100% då systemet i övrigt inte belastades.
- Signaleringsverktyget med 300 arbetsiterationer, men där programmet schemalades enligt SCHED_RR. CPU-belastande processer enligt tidigare med ordinär prioritet och utan användande av cgrupper.

Utöver detta så gjordes mätningar för att värdera och jämföra hur stor CPU-belastningen var för signaleringsverktyget i ovanstående situationer. Därvid uteslöts mätningar där signaleringsverktyget kördes utan några arbetsiterationer eftersom verktyget då inte mätbart belastade processorn och därmed någon reduktion i graden av CPU-belastning inte förväntades i denna situation.

Hur mätningarna utfördes beskrivs i detalj i Appendix C.

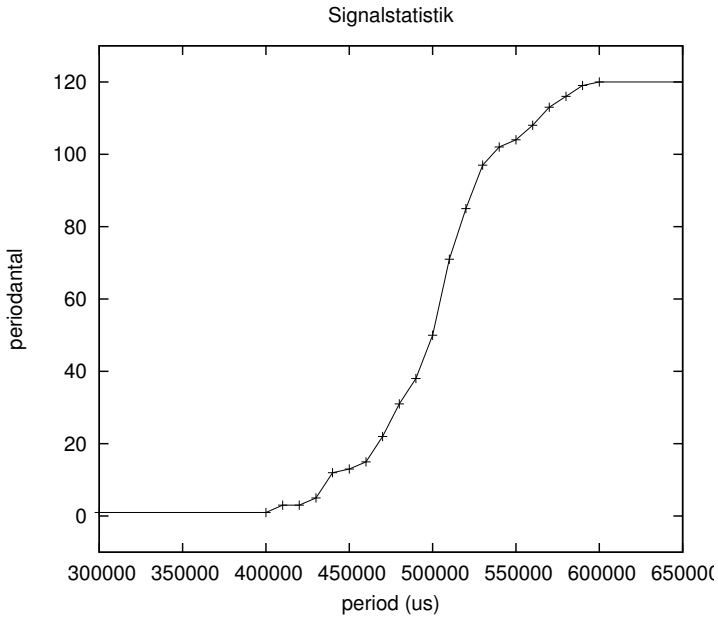
I efterhand kan konstateras att undersökningarna med 100% CPU-belastning var tveksamma. Det mest problematiska med dessa är att de representerar en situation där processen aldrig sov då sovtiden redan överskridits, ty hade det funnits någon kvarvarande tid till sovtidens utgång så skulle rimligen CPU-belastningen varit lägre än 100%. Det ses också i data från systemprototypen där 700 iterationer gav en CPU-belastning av 70-80% medan 710 iterationer gav en CPU-belastning av 100%.

Den genomsnittliga periodtiden tolkas delvis som ett uttryck för den CPU-tid programmet tilldelats under körperioden. Minskar den genomsnittliga CPU-belastningen för verktyget jämfört med när detta körs utan några CPU-belastande processer så medföra detta att den genomsnittliga periodtiden

förlängs. Variabiliteten i periodtiden, jämfört med när processen körs utan några CPU-belastande processer, är ett tecken på jitter. Denna variabilitet kvantifieras som periodtidens standardavvikelse.

Den reella periodtiden är oftast längre än den formella periodtiden, d.v.s. den önskade periodtiden såsom den anges till programmet, även när systemet i övrigt är obelastat. Detta beror sannolikt på ett antal faktorer. Om ett avbrott inträffar i vissa delar av programmet kan sovtiden vara oförändrad trots att den borde förkortats. Dessutom påverkas tiden mellan signaler av t.ex. schemaläggninslatensen, eftersom när den sovande processen väcks så måste den schemaläggas för att köras. Noterbart är att vid vissa iterationsantal så fås en kortare periodtid i tidssignalen än då signaleringsverktyget körs utan några arbetsiterationer alls (visas ej i någon figur). Orsaken till detta är sannolikt att sovtiden då precis överskridits, varvid processen aldrig sover och sålunda kärnresponstiden bortfaller.

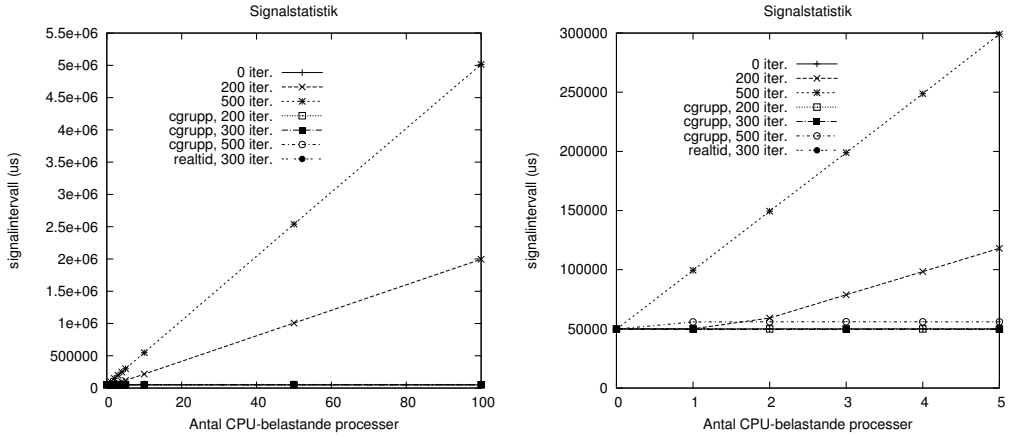
Det bör också poängteras att det på systemet fanns ett antal huvudsakligen sovande processer som kan tänkas vakna under körningen av programmet och som därmed kan påverka huvudsakligen variabiliteten i tidssignalerna.



Figur 6: Periodtidens fördelning för 500 arbetsiterationer och 100 CPU-belastande processer på P3301.

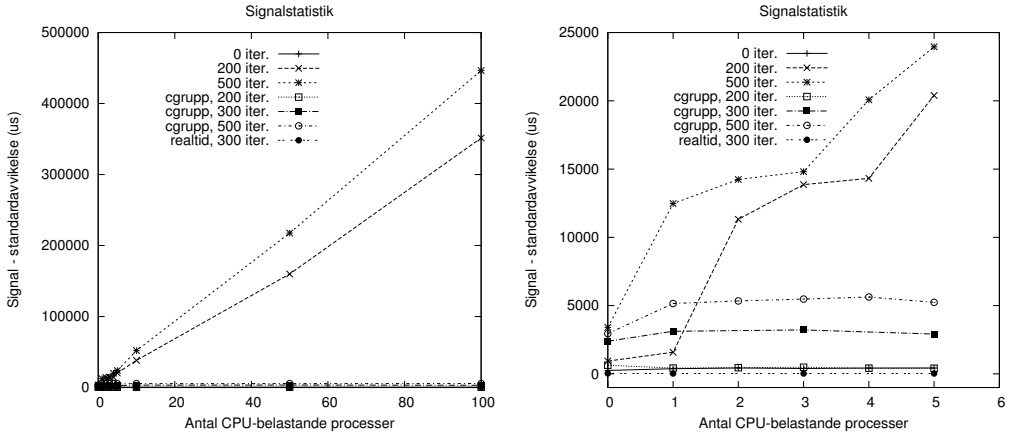
Figur 6 visar hur periodtiderna fördelas vid 500 arbetsiterationer och 100 CPU-belastande processer. Figuren visar väsentligen en fördelningsfunktion,

där alltså ordinatan anger antalet periodtider som är kortare eller lika långa som värdet angivet på abskissan. Denna har en viss likhet med normalfördelningens fördelningsfunktionen, och periodtiderna torde därmed i denna situation vara nära normalfördelade.



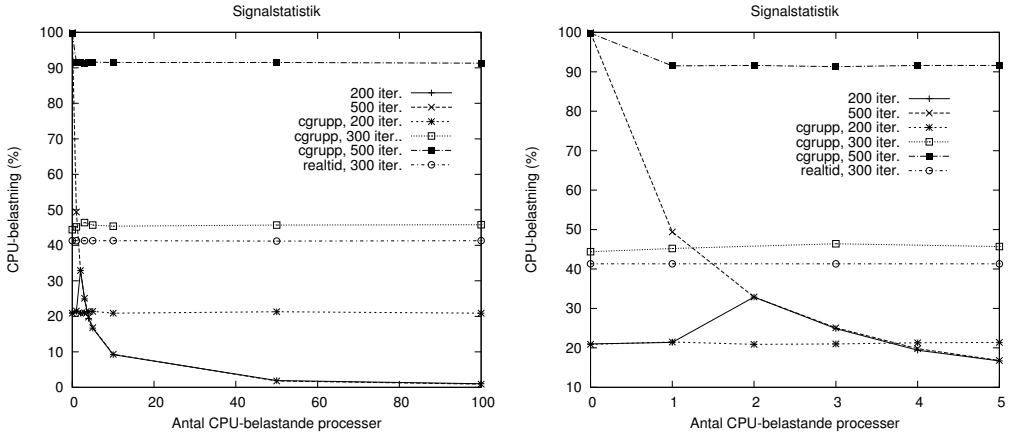
(a) 0-100 CPU-bel. processer (b) 0-5 CPU-bel. processer (delförstoring)

Figur 7: Genomsnittlig periodlängd vid användande av cgrupper, realtidsschemaläggning och utan någon av dessa.



(a) 0-100 CPU-bel. processer (b) 0-5 CPU-bel. processer (delförstoring)

Figur 8: Periodvariabilitet vid användande av cgrupper, realtidsschemaläggning och utan någon av dessa.



(a) 0-100 CPU-bel. processer (b) 0-5 CPU-bel. processer (delförstoring)

Figur 9: CPU-belastning vid användande av cgrupper, realtidsschemaläggning och utan någon av dessa.

Som ses i figurerna 7, 8 och 9 så får signaleringsverktyget då det körs utan någon signifikant egen CPU-belastning tillräcklig CPU-tid och låg nivå av jitter erhålls oberoende av antal CPU-belastande processer. Detta beror på att signaleringsverktyget huvudsakligen sover, och därmed i samband med schemaläggning alltid kommer att ha en lägre vruntime än någon av de CPU-belastande processerna. Således kommer signaleringsverktyget alltid att schemaläggas då det vaknar. När signaleringsverktyget körs så att CPU-belastningen obelastat för vertyget är 100% så kommer däremot varje ny CPU-belastande process, om inte någon resursstyrning el. dyl. används, att medföra att periodtiden förlängs motsvarande den obelastade periodtiden. Detta beror av att varje sådan CPU-belastande process genomsnittligen kommer att schemaläggas en gång per cykel i signaleringsverktyget, d.v.s. att CPU-tiden fördelas lika mellan de olika processerna i enlighet med CFS:s funktion. Standardavvikelsen ökar också betydligt i detta fall. När signaleringsverktyget har en obelastad egen CPU-belastning som är mellan dessa ytterligheter så fås också ett resultat som ligger mellan dessa, med en periodtid som ökar i något lägre grad och med en standardavvikelse som ökar nästan i samma omfattning som för det fullt belastande signaleringsverktyget.

Då cgrupper används så ses att signalintervallet och CPU-tidsandelen kan hållas konstanta jämfört det obelastade systemet utom i fallet där CPU-belastningen är 100%. Det senare är uppenbart eftersom alla processer

kommer att få köra även då cgrupper används, så om signalareren belastar systemet med 100% så kommer en introduktion av nya processer att medföra att signaleringsverktyget får en lägre CPU-tidsandel. Teoretiskt borde i detta fall, med CPU-andelarna (cpu.shares) 2048 och 256 för de båda cgrupperna, periodtiden bli $50 * (2048 + 256)/2048ms = 56ms$ där $50ms$ är den obelastade periodtiden. Detta stämmer bra med erhållen data. Sålunda medför användandet av cgrupper att såväl den genomsnittliga periodtiden som periodvariabiliteten hålls konstant. För den genomsnittliga periodtiden hålls nivån på samma nivå som när signaleringsverktyget körs utan några arbetsiterationer, utom i fallet där signaleringsverktyget självt belastar systemet 100% och periodtiden förlängs för den första belastande processen men därefter hålls konstant. Signalvariabiliteten ökar i vissa fall något med cgrupper jämfört med när signaleringsverktyget körs utan några arbetsiterationer, men variabiliteten hålls långt under nivåerna som ses då cgrupper inte används. Denna bild bekräftas också då CPU-belastningen för signaleringsverktyget studeras. När cgrupper används så hålls CPU-belastningen kostant (utom i fallet 100% vilket diskuterats ovan). När cgrupper inte används så faller CPU-belastningen med antalet CPU-belastande processer. Här ses dock i ett fall en paradoxal ökning av CPU-belastning samtidigt som periodvariabiliteten ökar kraftigt och periodtiden ökar lätt. Detta är svårt att förklara.

Det är i viss mån oklart varför en högre nivå av variabilitet ses utan CPU-belastande processer, såväl med som utan cgrupper, när den signaleringsverktyget belastar processorn i viss mån. Det kan tänkas bero av att andra processer på systemet, vilka huvudsakligen är sovande, behöver CPU-tid i vissa situationer under försöken.

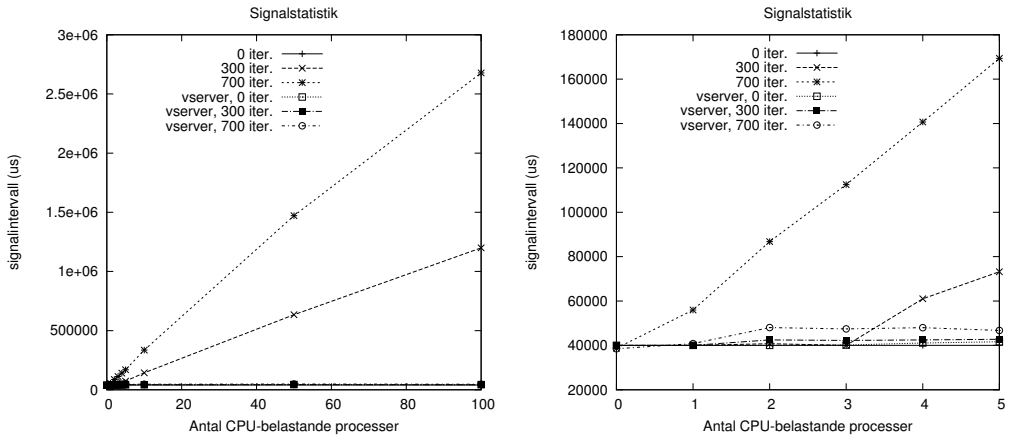
Om signaleringsverktyget körs som en realtidsprocess fås en signalvariabilitet som är mycket låg och närmast obefintlig. CPU-belastningen är då också närmast konstant. Detta torde vara ett uttryck för att verktyget får köra före alla andra på systemet förekommande processer, inkluderat sådana som normalt är sovande.

Ur figurerna torde det vara uppenbart att cgrupper kan användas för att partitionera total CPU-tid utom i de fall där en mycket låg nivå av signalvariabilitet eftersträvas. För ett system utan höga krav på låg nivå av jitter torde alltså cgrupper fungera adekvat, även om jitternivåerna är högre än i den obelastade situationen. Krävs låga nivåer av signalvariabilitet kan det vara lämpligare att köra processer som realtidsprocesser, vilket visas av den föga förvånande låga signalvariabilieteten när signaleringsverktyget körs som en realtidsprocess.

7.3 Periodvariabilitet vid användande av VServer

För att undersöka hur användning av VServer påverkade periodtidens längd, periodvariabiliteten och CPU-utnyttjandegraden gjorde vi undersökningar liknade dem i sektion 7.2. Dessa undersökningar gjordes på systemprototypen med 0, 300 eller 700 arbetsiterationer och där de CPU-belastande processerna kördes antingen i en hårt begränsad VServer-kontext eller utanför sådan. I det första fallet på en Linux-kärna som patchats med VServer, och i det senare fallet i såväl en ej patchad som en patchad Linux-kärna. För att jämföra med cgrupper gjordes även motsvarande undersökning där cgrupper användes. I detta fall exekverades såväl signaleringsverktyget, med 300 eller 700 arbetsiterationer, som de CPU-belastande processerna i cgrupper, där de förra fick cpu.shares 2048, medan de senare exekverades såväl med cpu.shares 256 som med cpu.shares 512. Som jämförelse gjordes även motsvarande undersökning utan att cgrupper användes. Dessa senare undersökningar kördes på en kärna där cgrupper aktiverats, men där Linux-kärnan inte patchats med VServer.

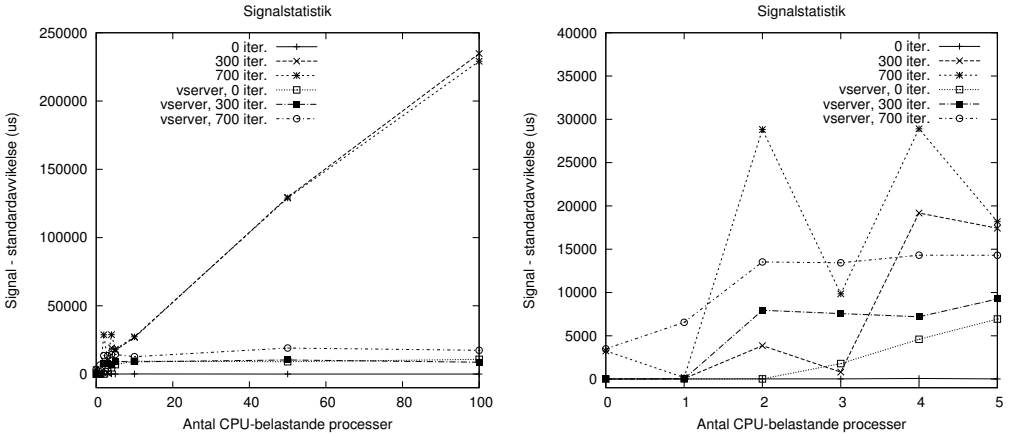
Hur mätningarna utfördes beskrivs i detalj i Appendix C.



(a) 0-100 CPU-bel. processer (b) 0-5 CPU-bel. processer (delförstoring)

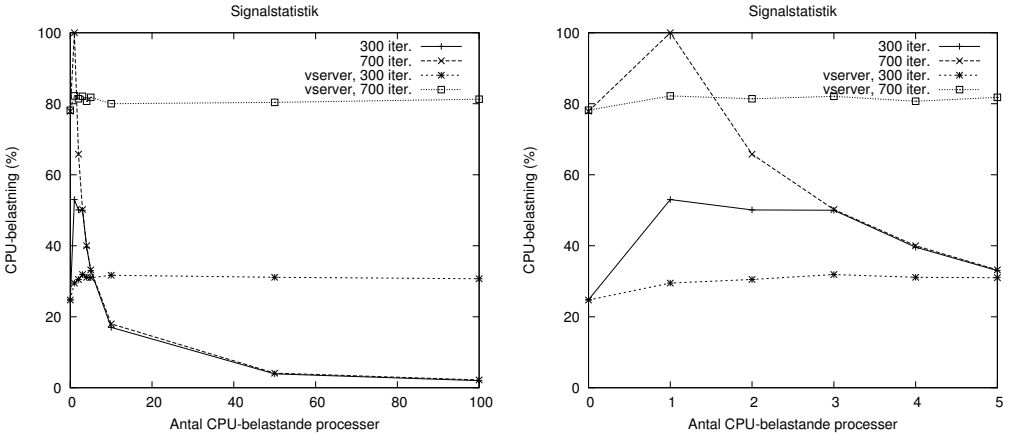
Figur 10: Genomsnittlig periodlängd med och utan VServer.

I figurerna 10, 11 och 12 ses att då de CPU-belastande processerna placeras i en hårt CPU-begränsad kontext så kan periodtiden behållas konstant under det att CPU-belastningen ökas, vilket inte är fallet då VServer inte används. Dock ökas periodtiden något initialt jämfört med situationen då vertyget körs utan några arbetsiterationer, för att sedan hållas konstant.



(a) 0-100 CPU-bel. processer (b) 0-5 CPU-bel. processer (delförstoring)

Figur 11: Periodvariabilitet med och utan VServer.

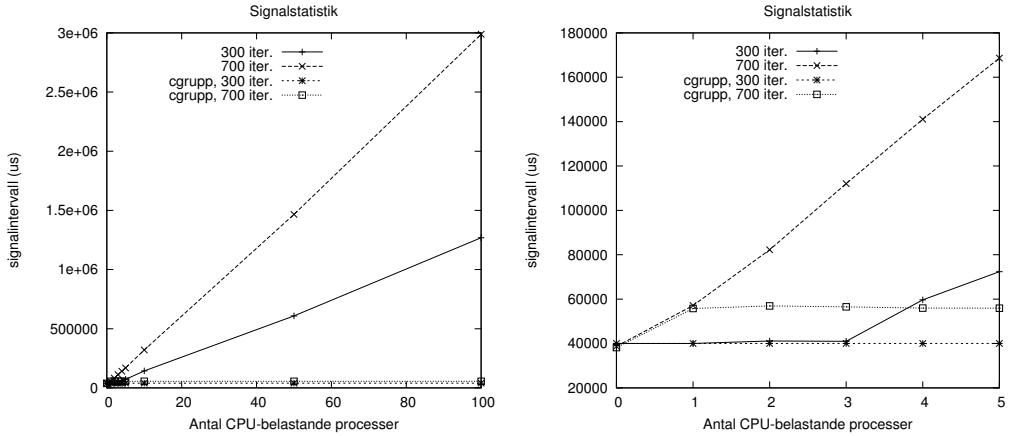


(a) 0-100 CPU-bel. processer (b) 0-5 CPU-bel. processer (delförstoring)

Figur 12: CPU-belastning med och utan VServer.

På samma sätt kan ses att periodvariabiliteten kan hållas konstant vid användande av VServer, även om variabiliteten ökar betydligt jämfört med när verktyget körs utan några arbetsiterationer. Detta kan också ses på att signaleringsverktygets CPU-belastningen kan behållas konstant under ökande antal CPU-belastande processer vid användande av VServer, medan använd CPU-andel sjunker ju fler CPU-belastande processer som körs om inte någon resursstyrning används för de CPU-belastande processerna. Notabelt är den med antalet CPU-belastande processer fluktuerande periodvariabiliteten som ses för vissa tester i figur 11. Detta tolkas som arkitekturberoende. Även den CPU-tidsandelsökning som ses i figur 12 när VServer inte används liksom för cgrupper senare (figur 15) tolkas som arkitekturberoende.

7.4 Jämförelse av signalstatistik mellan VServer och cgrupper

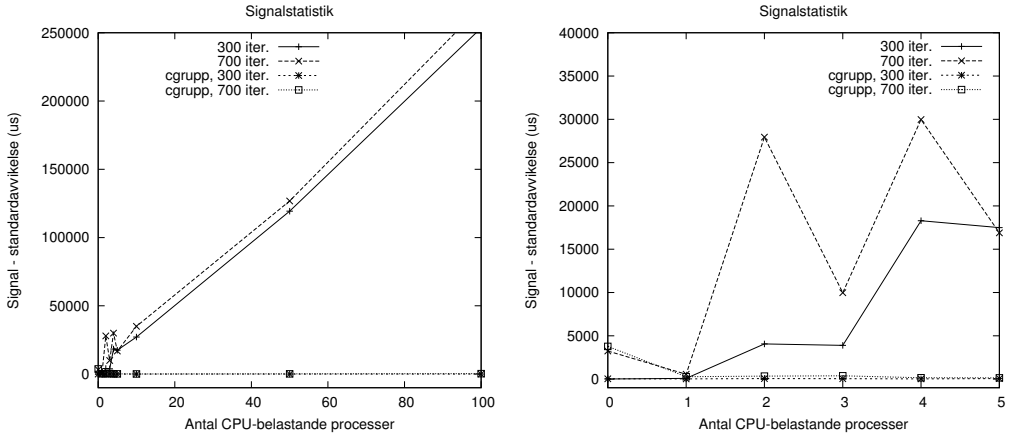


(a) 0-100 CPU-bel. processer (b) 0-5 CPU-bel. processer (delförstoring)

Figur 13: Genomsnittlig periodlängd på systemprototypen med och utan cgrupper.

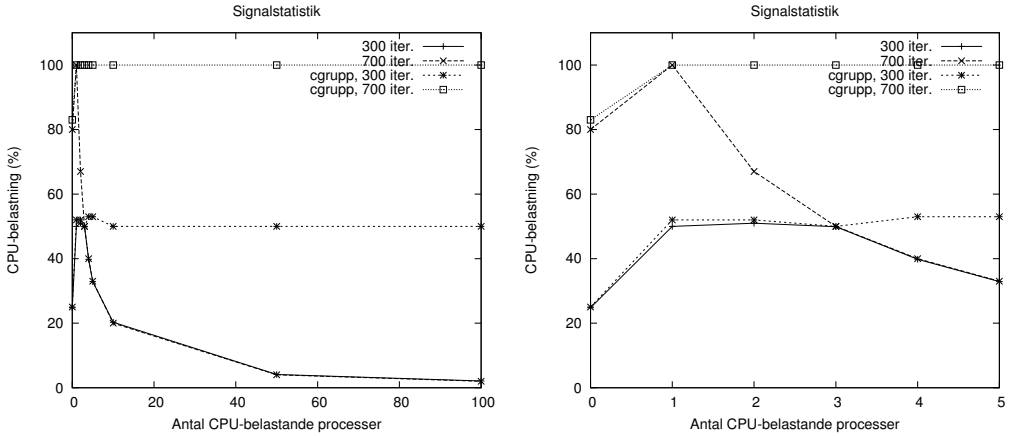
En direkt jämförelse mellan de två olika hårdvaruplatformarna är inte möjligt. Av denna anledning så genomfördes, enligt ovan, tester med cgrupper även på systemprototypen. Resultatet av dessa tester ses i figurerna 13, 14 och 15, och något nytt jämfört tidigare tester ses ej.

Vid en jämförelse mellan VServer och cgrupper, vilket visas i figurerna 16 och 17, noteras att redan vid ett relativt låg antal arbetsiterationer så fås en ökning av perioden vid användning av VServer. Samma ökning



(a) 0-100 CPU-bel. processer (b) 0-5 CPU-bel. processer (delförstoring)

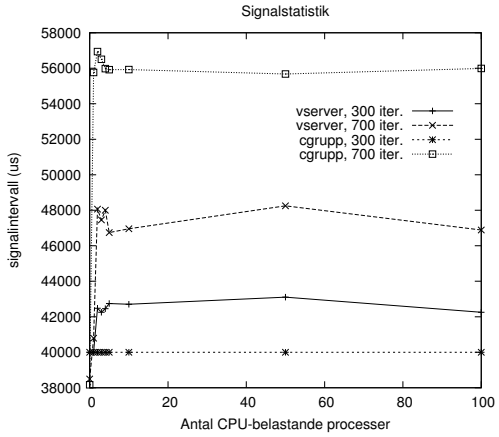
Figur 14: Periodvariabilitet på systemprototypen med och utan cgrupper.



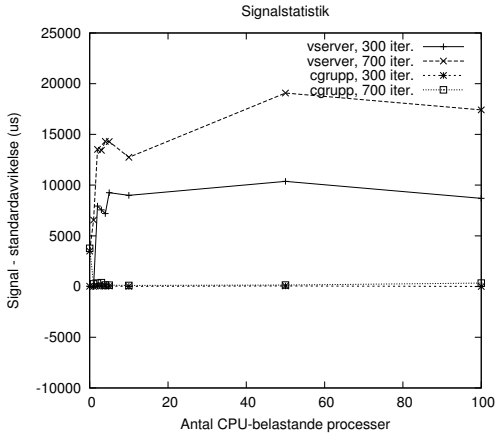
(a) 0-100 CPU-bel. processer (b) 0-5 CPU-bel. processer (delförstoring)

Figur 15: CPU-belastning på systemprototypen med och utan cgrupper.

av perioden ses inte för cgrupper utom i det fall där signaleringsverktyget CPU-tid är otillräcklig. I det fallet fås å andra sidan en tämligen krafigt ökad periodtid vid användning av cgrupper. I just denna situation är sålunda en användning av VServer att föredra framför en användning av cgrupper om målet är att upprätthålla en lägre periodtid. Om å andra sidan jämförelsen istället avser periodvariabiliteten så är denna i båda fallen betydligt mycket lägre vid använding av cgrupper än då VServer används. I en situation där CPU-tiden inte överskrider en övre gräns är sålunda cgrupper att föredra då dessa kan bibehålla såväl en låg periodvariabilitet som den önskade periodtiden. Naturligtvis är detta i viss mån beroende av hur respektive resursstyrningsverktyg konfigurerats. Allmänt kan sägas att cgrupper förefaller vara bättre på att upprätthålla såväl periodtid som en låg periodvariabilitet.



Figur 16: Jämförelse av periodtid vid användning av VServer och cgrupper.



Figur 17: Jämförelse av periodvariabilitet vid användning av VServer och cgrupper.

8 Ramverk för systemresursstyrningsverktyg på inbäddade system

Mot bakgrund av vad som angivits tidigare så valde vi att försöka implementera ett resursstyrningssystem baserat på cgrupper. För ett sådant system uppkommer i princip två problem. Dels så måste cgrupper konfigureras och processer placeras i cgrupper, dels så måste det för memory-subsystemet finnas ett system för att hantera en situation där oom kill initierats i en cgrupp. Dessa två problem har vi försökt lösa någorlunda separat för att få systemet så modulärt som möjligt, även om problemen delvis är relaterade.

8.1 Uppstart av processer i cgrupper

Antingen kan en process läggas i en cgrupp i samband med processens uppstart eller flyttas till relevant cgrupp efter det att processen redan startats. Det senare är omständligare som en generell metod och lider dessutom av problemet att en process minnesanvändningen inte följer med vid en flytt till en ny cgrupp. Vi har därför valt att försöka placera en process i den avsedda cgruppen i samband med att processen startas. Ett möjligt sätt att hantera uppstart är, med sysV-init-script, att varje sådant script som startar en process som ska ligga i en cgrupp skiljd från root-cgruppen definierar en variabel som sedan på något sätt medför att processen placeras i rätt cgrupp. Detta kan t.ex. göras genom att variabeln definierar en fil med data för cgruppen, och att denna fil sedan sourcas av SysV-init-scriptet. Ett alternativ är att ha en gemensam fil med konfigurerationsdata för samtliga cgrupper vilket ur användarperspektiv ger en enklare konfiguration, men detta blir något mer komplext att konstruera. Genom att placera alla konfigurationsscript i en gemensam katalog uppnås dock väsentligen samma överskådlighet. Den fil som sourcas kan definiera variabler som cgruppens namn, CPU-tidsandel för cgruppen, minnesbegränsning för cgruppen etc, och konfigurera cgruppen (d.v.s. skapa den och tilldela cgruppen dess CPU-tidsandel o.s.v). För en seriell uppstartsprocedur som SysV-init riskerar dock det senare att förlångsamma uppstarten eftersom ett init-script kan få för låg CPU-tidsandel om och efter att det placerats i en cgrupp med låg CPU-andel. Därmed förlängs init-scriptets exekveringstid, vilket är olyckligt när uppstartsprocessen är seriell. Det är därför önskvärt att tilldela cgrupperna begränsningarna först sent i uppstartsprocessen, och efter att processerna placerats i avsedda cgrupper. En invändning mot detta är att processer i

cgruppen redan kan överskrida begränsningarna när dessa införs.

Ytterligare ett problem uppstår eftersom det kan vara önskvärt att skapa hierarkier av cgrupper, t.ex. för memory-subsystemet. Om en cgrupp konfigureras i samband med att det init-script som behöver cgruppen körs finns risk att cgrupper närmare roten i hierarkin aldrig konfigureras eftersom dessa inte nödvändigtvis måste innehålla någon process som startas via sysV-init. Om en sådan cgrupp å andra sidan konfigureras av ett SysV-initscript för en process som ligger i en barn-cgrupp så uppstår en risk att föräldracgruppen konfigureras på olika sätt av olika SysV-initscript f.f.a. om dessa ska placeras i olika barn-cgrupper. Problemet är inte oöverstigligt, men det är önskvärt att varje cgrupp konfigureras en gång av en process skild från själva init-scriptet för den eller de processer som ska placeras i cgruppen, och att detta sker efter att init-scriptet körts. För att öka flexibiliteten och medge att tomma löv-cgrupper i cgrupps-trädstrukturen tillåts så kan även skapandet av cgrupper ske fritt från att de populeras av processer. Ovanstående medför alltså att det skapas ett sysV-init-script för att skapa samtliga cgrupper. Detta script bör köras tidigt i startproceduren. Varje sysV-init-script vars processer ska vara med i en cgrupp definierar därefter en variabel som anger vilket script som innehåller information för den cgrupp som processen ska ligga i. När sysV-init-scriptet körs med start-parametern sourcas ett script som medför att processen läggs i korrekt cgrupp. Sent i uppstartsprocessen körs ett sysV-init-script som konfigurerar alla cgrupper. Såväl det tidiga som det sena sysV-init-scriptet bör, för att undvika problem med underhåll av många filer, använda samma filer som de sysV-init-script som startar processer som ska ligga i cgrupper.

I princip kan ovanstående användas även för att starta processer med andra uppstartssystem än sysV-init. Dock kräver ovanstående lösning att cgrupper konfigureras m.h.a. shellscript, och för att minska belastningen på systemet kan det därför vara meningsfullt att konstruera denna uppstartsprocess som ett enhetligt program.

Ytterligare ett problem finns dock. Kamerorna använder en processövervakare, `respawnd`, för att tillse att vissa processer återstartas om de avslutats prematurt. Processövervakaren startas från processernas sysV-init-script och kontrollerar sedan regelbundet att processernas sökvägar finns i relevanta filer i proc-filsystemet. Om en process sökväg inte finns så startar processövervakaren processen utifrån den övervakade sökvägen. Processövervakaren tar i nuvarande utformning inte hänsyn till cgrupper, och om en övervakad process avslutats så kommer den sålunda återstartas i root-cgruppen. Ett enkelt sätt att korrigera detta är att till respawnd ange hur en avslutad process ska återstartas, d.v.s. via sysV-init-scriptet vilket

ombesörjer att processen placeras i korrekt cgrupp.

Ytterligare komplicerande faktorer är att tredjepartprogramvara kan nå kamerans bildhanteringssystemet på ett av tre olika sätt. Tredjepartsprogramvara kan nå bildsystemet direkt via de nativa gränssnitt som finns, varvid inte någon komplicerande faktor i relation till cgrupper finns. Däremot kan sådan programvaran även kontaka bildsystemet på kameran, vilket kan medföra att en ny tråd skapas av bildsystemet. Denna tråd kommer sålunda att belasta bildsystemets cgrupp. För att komma runt detta problem så måste kommunikationen mellan dessa processer förändras så att en sådan uppstartad tråd placeras i korrekt cgrupp. I sammanhanget bör påpekas att minnesredovisningen sker under den cgrupp som trådens PID ligger i, vilket inte nödvändigtvis är densamma som den cgrupp tråden ligger i. Ytterligare en möjlighet är att tredjepartsprogramvaran körs i form av lua-script[31] av bildsystemet. Här har vi inte hittar någon lösning. Lösning till ovanstående två problem har inte implementerats.

Ett alternativt system där sysV-init-scripten är helt agnostiska rörande cgrupp-tillhörighet kan tänkas. En möjlighet är då att låta rc-scriptet starta ett uppstartsscript för cgrupper som i sin tur startar sysV-init-scriptet. I princip kan det utformas på samma sätt som ovan, fast där uppstartsscriptets indata för att avgöra cgrupps-tillhörighet utgörs av vilket sysV-init-script som kallas. Detta skulle centralisera hanteringen av cgrupps-tillhörighet, och samtidigt knyta uppstartsprocessen hårdare till sysV-init-processen.

Ytterligare en möjlighet är att daemon-processer inte startas direkt av SysV-init-scripten, utan att varje daemonprocess startas av en process som i sin tur startar t.ex. en tråd som lägger processen i rätt cgrupp samtidigt som huvudtråden joinar denna tråd, varefter den kör något exec-systemanrop för att starta daemon-processen. Detta program för att starta daemoner i cgrupper hade kunnat läsa en konfigurationsfil vilken specificerar hur daemon-processerna ska placeras i cgrupper, och kallas från SysV-init-scripten om SysV-init används. På detta sätt hade uppstarten i cgrupper inte alls varit knuten till något speciellt system för system-initialisering. Möjligen hade detta dock i vissa fall kunnat interferera med något av de alternativa system för system-initialisering som kan övervaka processer och återstarta prematurt avslutade processer. Det hade sannolikt även blivit något mer komplext att avsluta och återstarta samtliga processer i en cgrupp.

Det mest behändiga vore om init-processen hade vetskap om förekomsten av cgrupper och hade möjlighet att starta om processer som avslutats och då placera dem i rätt cgrupp. Systemd, ett alternativ till sysV-init, verkar

[31] Se the programming language lua, webbplats: `http://www.lua.org/`

tillhandahålla möjligheter för detta och samtidigt möjliggöra övervakning av uppstartade processer. Vi har dock inte hunnit undersöka systemd. Problemet med att avsluta och återstarta samtliga processer i en cgrupp i samband med oom kill hade dock kvarstått.

Av ovanstående lösningar är systemd mest tilltalande, och därefter det system där daemon-processer inte startas direkt av SysV-init-scripten. Båda systemen riskerar dock att bli komplexa om de ska samexistera med ett system för att hantera oom kill. Vi valde dock, då det var enklast att implementera, den först beskrivna lösningen. Vårt system för att placera processer i cgrupper vid uppstart bygger sålunda på att en variabel sätts i SysV-init-scriptet. Därefter sourcas en fil då SysV-init-scriptet kallas med start-argumentet. Denna sourcade fil sourcar, baserat på variabeln, en fil med konfigurationsdata och lägger in processen i rätt cgrupp. Tidigt i sysV-init-processen har ett init-script körts som baserat på samtliga filer med konfigurationsdata skapat alla cgrupper och monterat relevanta subsystem, och i slutet körs ett script som baserat på samtlig konfigurationsdata konfigurerar cgrupperna. Därtill finns en mindre gemensam fil som håller gemensam data. Då systemet är enkelt och inte omfattar många rader kod finns det bifogat i appendix D

8.2 Daemon för övervakning av oom kill

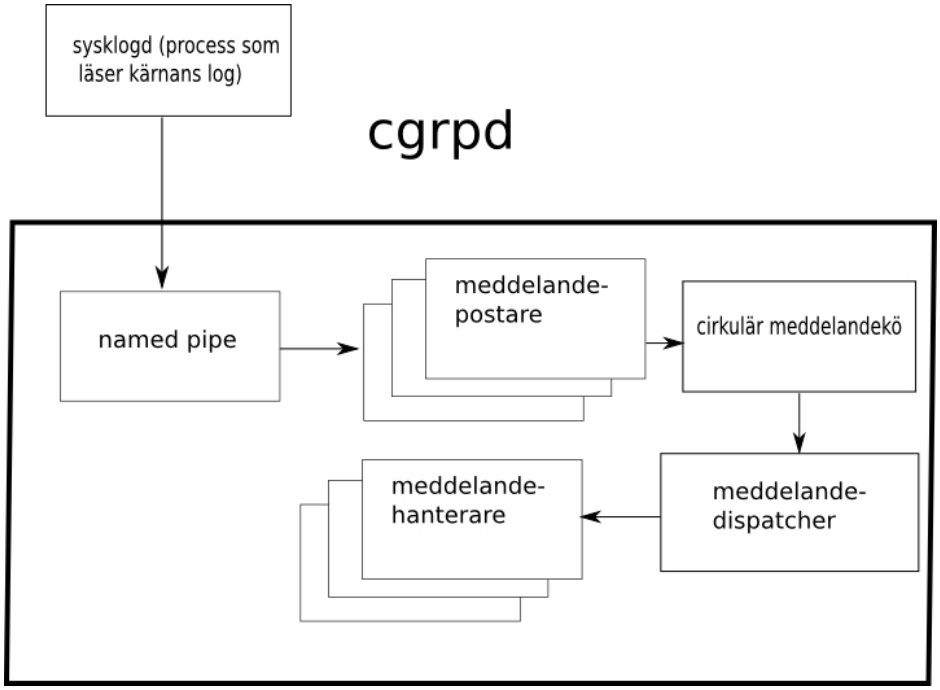
Mer problematiskt är hur hanteringen av oom kill ska ske. I nuläget så medför en oom kill, för att systemets integritet skall behållas, att systemet startas om. Detta är inte en adekvat hantering om cgrupper med minnesbegränsningar används eftersom ett sådant beteende enbart skulle resultera i att systemets stabilitet minskade utan att tillföra något. Felaktig programvara, t.ex. tredjepartsprogramvara, i en cgrupp kan inte heller tillåtas starta om systemet upprepat. Å andra sidan kan det i vissa situationer vara rimligt att systemet startas om vid oom kill, om t.ex. denna sker i vissa fundamentala processer. Därför utvecklade vi en daemon som kan hantera vad som ska ske vid oom kill beroende på i vilken cgrupp oom kill skett. Vilken process som dödas vid oom kill beror av konfigurationsparametrar i proc-filsystemet. Antingen väljs den process vilken initierat överanvändningen av minne eller så väljs, som tidigare beskrivits, en process baserat på ett flertal data rörande processerna. Inte i något av fallen är det säkert att det är den process som verkligen överanvänder minne som utsätts för oom kill. Detta medför, under antagande att processerna i en cgrupp konceptuellt hör samman, att vid en oom kill i en cgrupp så är det rimligt att starta om alla processer i cgruppen.

Deamonen består av flera delar (se figur 18). En del är meddelandepostarna. Varje meddelandepostare körs som en separat tråd och postar meddelanden, vilka är enkla strängar, till en cirkulär buffert. Vi har enbart implementerat en meddelandepostare, och denna har som uppgift att läsa kärnans loggmeddelanden via en namngiven pipe, leta fram oom kills och i vilken cgrupp en sådan skett, och sedan posta ett meddelande bestående av namnet på den cgrupp i vilken oom kill skett. Den cirkulära bufferten är implementerad som en monitor. Bufferten läses av en meddelandedispatcher, som då ett meddelande ankommit startar en tråd för att hantera meddelandet. Denna tråd startar ett godtyckligt antal meddelandehanterare i en given prioritetsordning. Vilka meddelandehanterare som startas för en given meddelandesträng styrs av en konfigurationsfil i vilken meddelandesträng (vilket alltså i detta fall motsvarar namnet på en cgrupp), prioritet och namnet på en meddelandehanterare anges. För att kunna skapa en ännu mer dynamisk meddelandehantering blir utdatan från föregående meddelandehanterare indata till nästa hanterare. På detta sätt möjliggörs att en hanterare skickar ett meddelande till den nästkommande hanteraren. Vi har skapat ett antal meddelandehanterare, som en sleep-handler vilken enbart medför att tråden sover, en reboot-handler som startar om datorn, en kill-handler som först ett antal gånger skickar SIGTERM till alla processer i en cgrupp för att sedan skicka SIGKILL till dem, och två metahierarki-hanterare som beskrivs mer senare.

Flödet av ett meddelande är alltså sådant att detta fås från kärnan via en meddelandepostare, där den implementerade postar namnet på en cgrupp i vilken det skett en oom kill. Dispatchern läser meddelandet och startar en tråd för att hantera meddelandet. Beroende på meddelande, d.v.s. cgrupp där oom kill körts, så exekveras ett godtyckligt och konfigurerbart antal meddelandehanterare i prioritetsordning.

Daemonen måste själv skyddas från oom kill och måste därtill köras med tillräckligt hög prioritet för att kunna utföra sitt arbete rimligt ostört.

Vi har även skapat ett system för att generera en meta-hierarki som representerar de processer som finns i cgrupper. Hierarkin har som syfte att underlätta omstart av en hel cgrupp om en process i den skulle dö p.g.a. oom kill. Meta-hierarkin består av en katalogstruktur som är identisk med cgruppernas katalogstruktur och hierarkin skapas av SysV-initscripten i samband med start-proceduren. De processer som ska finnas i cgruppen representeras av symboliska länkar till motsvarande SysV-initscript. Denna hierarki skapas alltså under uppstarten av processer, och de symboliska länkarna i meta-hierarkin läggs till och tas bort då sysV-init-scripten körs för att starta respektive avsluta processer. Vi har även skrivit två



Figur 18: cgrpd:s struktur

meddelandehanterare till cgrpd som är kopplade till denna metahierarki. Dessa metahierarki-hanterare har till uppgift att starta respektive stoppa alla processer som finns i en viss cgrupp baserat på informationen i metahierarkin. För att starta om en cgrupp efter att en oom kill har skett i cgruppen så stoppas först processerna i cgruppen med hjälp av metahierarkin och en dessa båda metahierarki-hanterare, därefter så kallas killhandlern för att säkert rensa ut alla processer ur cgruppen, och slutligen kallas metahierarki-hanteraren för att starta alla processer i cgruppen igen. Problemet med denna hierarki är att den inte löser problemet som uppstår när en process dör av andra orsaker än oom kill, t.ex. vid segmenteringsfel. Detta skulle i så fall kräva någon sorts integrering med respawnd. Vidare så kommer den process en tråd tillhör att dödas om SIGKILL skickas till tråden, vilket innebär att cgrupper andra än den i vilken oom kill skett kan förlora processer.

9 Diskussion

9.1 Verktyg för systemresursmätning, belastningsverktyg och egenskapta verktyg

Av de verktyg för systemresursmätning vi undersökt så använde vi huvudsakligen `top` och `iperf`. `top` är ett verkyg som ger en mängd information om de processer som körs på systemet, och presenterar bl.a. processernas CPU- och minnesanvändning. `top` är vältestat har funnits länge för Unix.

Många verktyg för att mäta CPU-användning och minnesutnyttjande använder sig av proc-filsystemet. `top` tillhandahåller dock bara en delmängd av den information som kan fås från proc-filsystemet, och det finns därför utrymme även för även andra verktyg. Som exempel kan nämnas `pidstat`, som t.ex. presenterar data om hur lång tid processer har exekverat i user- respektive kernel-mode.

`iperf` har varit användbart för att rapportera nätverksbandbredden under testning av framförallt `tc`. Det är ett smidigt och komplett verktyg som kan stresstesta nätverksbandbredden på flera olika sätt. Problemet är som nämnts tidigare att nätverksbandbredden på målarkitekturen huvudsakligen beror på hur mycket CPU-tid processen får. Detta problem löste vi delvis genom att använda den snabbare systemprototypen. Denna svaghet i mätningarna gick dock ej att korrigera helt.

I viss mån har vi även haft användning av `iptraf` för att värdera netverksbandbredden.

I det praktiska arbetet har vi haft begränsad nytta av de verktyg som presenterar mer generell statitisk över hur systemresurserna använts under längre tid. Detta beroende på att våra undersökningar fokuserades mot att mäta systemresursutnyttjande per process. Vi har inte heller haft större användning av verktyg som på olika sätt ger information om I/O mot hårdvara (förutom nätverksbandbredd) och då f.f.a. I/O vad avser sekundärt lagringsutrymme. Detta huvudsakligen p.g.a. att de inbyggda systemen vi arbetat med använder flash-minne.

Att hitta belastningsverktyg har som nämnts tidigare inte varit något problem. `stress` är ett dynamiskt verktyg som kan belasta de flesta systemresurser, och verktyget är enkelt att använda. Därför har vi använt huvudsakligen detta verktyg i den praktiska undersökningen. En svaghet med verktyget är dock att det för CPU-belastning enbart belastar systemet i

user mode. För att belasta minnesanvändningen har vi dock också använt egenskrivna program eftersom dessa kunnat skräddarsys för att få en långsammare ökning av minnesbelasatningen och det därmed blivit enklare att studera händelseförlopp. Värt att notera är att `iperf` i princip kan klassificeras som belastningsverktyg, även om vi inte använt det som sådant.

Av de verktyg vi skapat själva har vi huvudsakligen använt signalerings- och lastsimuleringsverktyget samt realtidsprioriteringsverktyget. Signaleringsoch lastsimuleringsverktygets lastning av processorn avviker sannolikt i viss mån från den last som det avseer att simulera då det exekverara nästan uteslutande i user mode. En fördel med detta är dock att verktyget blir mindre känsligt för hur Linux-kärnan kompilerats avseende preemptbarhet. Vid undersökningen av cgrupper har verktygen för att starta processer i cgrupper kommit till användning. Processinformationsverktyget användes initialt för att undersöka vissa aspekter av processers beteende.

I viss mån problematiskt ur kvantifieringshänseende var att mätsystemen delvis använde sig av det system som det avsåg att mäta och att de i vissa fall även belastade det uppmätta systemet cykliskt. För att minska denna påverkan kördes t.ex. `top` med PID-bevakning, vilket signifikant reducerade verktygets CPU-belastning. I flera fall är de relativa resultaten mer intressanta än absolutvärdena av mätningarna, vilket då störningen från mätningarna kan förväntas varit likartad under de olika testerna, gör att ovanstående invändning till viss del bortfaller. Resultaten är vidare tydliga och överensstämmer med vad som förväntats, vilket antyder att mätsystemets störning inte påverkat mätningarna på ett sådant sätt att resultaten bör förkastas eftersom det är osannolikt att resultatet skulle kunnat genereras av mätningsartefakter. Möjligen skulle t.ex. kompletterande mätningar med ett större interval för `top` kunnat användas, och om resultet då varit likartat, med en lägre nivå av störning, så hade sannolikheten för att resultatet berott på artefakter ytterligare minskat.

Ur ett teoretiskt hänseende är det naturligtvis besvärande att vi var tvugna att använda mätsystem som kördes som periodiska realtidsprocesser på målarkitekturen för att få information om bl.a. beteendet hos periodiska realtidsprocesser. Principiellt innebär detta att vi använt det system vars kvalitet vi avser att mäta för att få data för att bedöma samma systems kvalitet. Denna invändning försvagas dock i viss mån av att signaleringsverktygets periodvariabilitet och periodtid var konstant då verktyget såväl som mätsystemet kördes som realtidsprocesser.

För mätdata som representerar medelvärdet över en tidsperiod kan, om för långa tidsperioder används, data vara ointressant då en resurs under

delar av tidsperioden kan ha varit begränsande utan att detta avspeglas i medelvärdet. Principiellt gäller detta t.ex. för CPU-belastningen där en tillräcklig genomsnittlig CPU-tid kan ha erhållits över tid trots att CPU-tids-tilldelningen varit otillräcklig under delar av denna tid. Vad gäller mätning med `top` är detta dock inte intressant då verktyget inte har en tillräcklig granularitet i förhållande till våra mätningar. Vidare skulle det riskera att kräva CPU-tid långt utöver vad som är tillgängligt för att kunna köra med så hög granularitet.

Faktorer som variabiliteten av det tidskvanta under vilken en resurs mäts har även betydelse för en eventuell medelvärdesberäkning i efterhand. En stor variabilitet i den faktiska periodtiden för `top` skulle kunna påverka medelvärdesberäkningen av CPU-belastningen så att denna blev felaktig. Här antyder data att någon sådan betydande variabilitet inte förelegat.

En ytterligare invändning är att systemet i övrigt kan ha stört mätningen. Denna invändning reduceras dock av att då signaleringsverktyget kördes som en realtidsprocess så erhölls genomsnittligt konstanta periodtider oberoende av belastning på systemet och vidare en periodtidsvariabilitet som var mycket låg, och därmed kan samma beteenden rimligen förväntas för mätsystemet eftersom det körts som en realtidsprocess på samma sätt. För verktyg som inte kördes periodiskt var det svårt att få information om minnes- och CPU-belastning då dessa verktyg sällan kan fångas med `top`.

9.2 Verktyg och metoder för resursstyrning

Att garantera, i mjuk bemärkelse, CPU-tid till vissa processer kan grovt sägas ge upphov till två problem relativt en obelastad situation. Härvid avses dels huruvida processerna får tillräcklig CPU-tidsandel, dels om processerna får CPU-tid när de är i behov av denna. I princip kan detta ses som granulariteten för CPU-tids-tilldelningen till processen. Med en låg granularitet så kan processen förvisso få tillräcklig CPU-tid över en längre tidsperiod, men behöver inte nödvändigtvis få tillräcklig CPU-tid under mindre tidskvanta. Om det senare behovet fallerar så kan alltså problem som jitter i t.ex. bild- eller ljudströmmar uppstå. För I/O-bundna eller delvis I/O-bundna processer kan således en tillräcklig CPU-tidandel uppnås utan att det senare behovet tillfredställs. Värdering av jitter är dock inte trivialt. Vår intention var att kvantifiera jitter på målarkitekturen som RTP interarrival jitter enligt RFC 3550. Tyvärr var detta svårare än vad vi hoppats och vi övergick därför till att kvantifiera jitter som variabiliteten, i vårt fall standardavvikelsen, i en periodisk signal från målarkitekturen. Det hade

dock varit intressant att även undersöka RTP interarrival jitter. En tänkbar felkälla i vår värdering är vidare att den CPU-belastning som genererades av signaleringsverktyget var helt i user mode, och att därför verktyget skiljer sig från de processer som kan förväntas köra på målarkitekturen. Det innebär att signaleringsverktyget kan ha varit preempt-bar i större utsträckning än om större del belastningen utförts i kernel mode genom att systemanrop använts. Likaså kördes de CPU-belastande processerna huvudsakligen i user-mode, vilket introducerar motsvarande felkällor vad avser dessa. Ur värderingshänseende kan dock en fördel med detta sägas vara att preempt-barheten inte var lika beroende av hur Linux-kärnan kompilerats i detta avseende.

En arkitektur som inte uppfyller realtidskrav kan inte garantera schemaläggning av en process vid en given tidpunkt, och det går sålunda inte under Linux att garantera en låg nivå av jitter för periodiska processer. Bäst resultat i detta avseende erhölls dock för processer som körde som realtidsprocesser, men även cgrupper reducerade tydligt variabiliteten i den periodiska signalen från signaleringsverktyget. VServer tillhandahöll i sammanhanget en hård CPU-tidsbegränsning men med en högre nivå av variabilitet i den periodiska signalen. Å andra sidan föreföll VServer i vissa fall kunna garantera en lägre nivå av genomsnittlig periodtidsförlängning, men detta torde till del vara en konfigurationsfråga. Vi hade inte möjlighet att undersöka VServer mer utförligt t.ex. vad gäller användande av cgrupper för hård CPU-tidbegränsning, vilket skulle varit intressant. Likaså gjorde vi inte annat än en informell undersökning av hur parametrarna för den hårda CPU-tidsbegränsningen påverkade signalkarakteristiken. Vi undersökte inte heller VServers mjuka CPU-tidsbegränsning. Även denna ska dock använda token bucket-schemaläggaren, så i viss mån är en sådan undersökning av mindre värde. I detta sammanhang bör påpekas att det skulle varit intressant att undersöka även andra virtualiseringsverktyg som OpenVZ. Även en mer detaljerad undersökning av hur systemets realtidsandel kan fördelas mellan olika cgrupper och en undersökning av vilka möjligheter en sådan hantering kan tillhandahålla vore intressant. Ytterligare obesvarade frågor, dock sannolikt av mer perifer betydelse, är beteendet av processer i cgrupper och realtidsprocesser i VServer-kontexter. Eftersom allt fler CPU:er är flerkärniga eller har multithreading så skulle det också vara intressant att undersöka beteendet vid låsning av processer till en viss CPU, vilket kan göras även med cgrupper.

Vad gäller minnes-begränsning så var delvis VServer enklare att hantera än cgrupper, men att använda två system, ett för CPU-tidsbegränsning och ett annat för minnes-begränsning, skulle sannolikt införa en för stor

komplexitet. Här bör också beaktas att om en process havererat genom att använda för stora mängder minne så är det rimligt att processen dödas så att allokerat minne avallokeras och kan användas på nytt. Det kan därför vara fördelaktigt att hitta en oom kill och starta om de processer som kan ligga till grund för överanvändningen av minne, snarare än att förhindra förnyad minnesallokering för processerna, även om det senare är enklare att hantera.

I sammanhanget kan det också vara värt att påpeka att för inbäddade system där det finns krav på att systemet hela tiden har ett givet beteende, där avsaknad av en viss process kan medföra att systemet inte fungerar på ett adekvat sätt och där en omstart av systemet kan ske snabbt, så kan det vara lämpligt att, istället för att försöka korrigera ett uppkommet fel, starta om systemet. En nackdel med detta i relation till 3:e-partsprogramvara är att ett sådant förhållningssätt kan medföra att systemets tillgänglighet allvarligt kan påverkas.

Vid undersökning av nätverksbandbredd noterade vi att för trafik som genererades direkt från den ARTPEC-3-baserade P3301 så var CPU-tid och nätverksbandbredd balanserade, d.v.s. för att fylla nätverksbandbredden så krävdes 100% CPU-tid. Det är därför svårt att motivera en begränsning av nätverkstrafiken på ifrågavarande enhet från enbart enhetens perspektiv. Det förefaller helt enkelt inte meningsfullt att införa CPU-tidskrävande begränsningar av nätverkstrafiken för olika processer då en sådan begränsning redan existerar i och med att CPU-tid och nätverksbandbredd är balanserade. Det skulle möjligen kunna motiveras av att begränsa nätverkets hela bandbredd, men inte direkt från ett kameraperspektiv. Vi har inte undersökt närmare hur cgrupper kan användas för att styra nätverkstrafiken, men detta skulle kunna vara intressant.

Begränsning av sekundärt lagringsutrymme har inte undersökts närmare, men här kan ordinär quota förmodligen användas till del. Problemet är att dessa baseras på användare och grupper snarare än på processer. OpenVZ har dock system för att hantera quotas på container-nivå.

Vad gäller I/O-schemaläggning till sekundärt lagringsutrymme så syftar existerande algoritmer huvudsakligen på att minimera läs- och skrivhuvudets rörelser över en hårddisken. Då aktuella inbäddade system inte använder hårddiskar är en sådan schemaläggning inte aktuell, och det finns inte heller möjlighet att använda något annat än den grundläggande FIFO-schemaläggaren på de aktuella systemen.

Det är från utvärderingssektionen uppenbart att utan någon resursstyrning så kan en process som önskar all processortid, signifikant och självklart påverka tillgänglig processortid för andra processer. Det är också uppenbart

att detta påverkar systemets nivå av jitter. Som tidigare redovisats så kan användandet av cgrupper tydligt partitionera CPU-tid och även internminne så att systemet påverkas mindre av processer som beter sig illa. Om standard-Linux-kärnans cgrupper används kan tillräcklig CPU-tid upprätthållas för prioriterade processer. Inte nog med det, utan för periodiska processer så upprätthålls även en låg periodvariabilitet jämfört ett obelastat system. Detta samtidigt som systemets totala CPU-tid kan utnyttjas till fullo. En hård begränsning av CPU-tid med VServer ger liknande resultat. Dock upprätthåller cgrupper bättre en konstant periodtid och en låg variabilitet i periodtiden. Här ska dock påpekas att periodvariabiliteten mätts som standardavvikelsen av periodtiden, och i vissa situationer kan det vara mer adekvat att kvantifiera periodvariabiliteten som t.ex. den största avvikelsen från den genomsnittliga periodtiden. Vi har inte alls värderat detta.

9.3 Implementation av resursstyrning

Ett problem som inte tagits hänsyn till i cgrpd för att hantera oom kill, är om en oom kill sker p.g.a. ett överskridande av en begränsning i en intern nod. I ett sådant fall borde rimligen denna nod samt alla barn till denna startas om. Någon sådan hanterare har dock inte implementerats. Å andra sidan torde ett scenario med så komplexa cgrupps-hierarkier vara sällsynt, vilket gör problemet mindre relevant.

Ett problem med att implementera hanteraren av oom kill:s på det sätt som gjorts är att det kräver stabilitet över kärnversioner i de meddelande kärnan genererar vid oom kill. Ett tilltalande alternativ hade varit att kärnan själv vid oom hade kunnat döda en hel cgrupp. En intressant möjlighet skulle då vara att det i cgruppen definierades ett program att starta i den händelse en oom inträffade i cgruppen. Detta skulle möjligen kompliceras av oom i en föräldracgrupp p.g.a. ett överskridande genererat i ett barn. Detta problem finns dock även i aktuell implementation. En sådan implementation hade dock också varit känslig för förändringar i kärnan, och förmodligen mer underhållskrävande.

Att använda cgrupper för tredjepartsprogramvara skulle sålunda i princip vara möjligt. Komplicerande kan vara sådan programvaras interaktion med systemet i övrigt. Om tredjepartsprogramvara kommunicerar med övrig programvara så att de senare startar upp processer eller trådar så måste övrig programvara placera dessa i relevanta cgrupper. Skickas SIGKILL, t.ex. från cgrpd, till en tråd så dödas dock den process som tråden tillhör. Detta kräver vidare utredning, liksom huruvida det är

möjligt att placera tredjepartsprogramvara, i form av Lua-program som körs av systemets ordinära programvara, i cgrupper. Vid minnesbegränsning med cgrupper uppstår ytterligare en komplicerande faktor. För flertrådade program så räknas all minnesanvändning i den cgrupp som håller tasken med programmets PID.

Det bör också påpekas att det sannolikt varit lämpligare att implementera uppstarten av deamonprocesser antingen genom en alternativ init-process som kunnat handha detta, eller genom att använda ett eget program för detta, såsom beskrivits tidigare, för att på så sätt göra systemet mindre beroende av val av init-program.

9.4 Slutsats

Sammantaget så har vi utvärderat och utvecklat vertyg för systemresursmätning, där vi funnit att vanligt förekommande vertyg ofta varit tillräckliga. Vi har utvärderat och utvecklar belastningsverktyg för CPU-tid, minnesanvändning, och nätverksbandbredd. Intern bandbredd har inte kunnat utvärderas då målarkitekturen saknat stöd för att mäta detta med mjukvara. Vi har utvärderat systemresursstyrningsverktyg och utvecklat ett ramverk för deras användning på målarkitekturen.

Det finns många kvarvarande frågor, men sammantaget anser vi att det är möjligt att använda systemresursstyrning med cgrupper på målarkitekturen, men det kan kräva vissa arkitekturella förändringar.

A Relevanta virtuella filer i procfilsystemet

I denna sektion presenteras de virtuella filer i /proc- och /sys-filsystemen som har varit relevanta för vårt arbete. Vissa av flaggorna kan även kontrolleras med verktyget `sysctl`.

/proc/sys/vm/overcommit_memory och /proc/sys/vm/overcommit_ratio

overcommit_memory-flaggan styr hur kärnan hanterar överallokering av minne. Den kan vara i en av 3 lägen: kontrollera ej om överallokering sker, kontrollera med en heuristik eller kontrollera att det aldrig blir överallokering. overcommit_ratio styr hur beräkningen sker när man använder det sistnämnda läget.

/proc/sys/vm/oom_kill_allocating_task

Denna flagga styr vad som händer när en oom-situation uppstår. Om den är satt till 0 genomsöks hela listan av existerande tasks och en task med ofördelaktig oom-poäng dödas vid en oom-situation. Annars dödas den task som orsakade oom-situationen.

/proc/sys/vm/panic_on_oom

Om denna flagga är satt till 1 så aktiveras oom-killern när det uppstår en oom-situation. Om den är 0 blir det kernel panic istället.

/proc/<PID>/oom_adj och /proc/<PID>/task/<TID>/oom_adj

/proc/<PID>/oom_adj justerar oom-poäng för en task. Den har ett tal mellan -16 och 15, förutom specialvärdet -17 som utesluter denna task från att komma ifråga vid oom. Om värdet är positivt ökar sannolikheten för oom-kill i en oom-situation. Värdet är vad vi kan se samma för alla tasks i samma process, så värdet i /proc/<PID>/task/<TID>/oom_adj har egentligen ingen betydelse.

/proc/<PID>/stat och
/proc/<PID>/task/<TID>/stat

I denna virtuella fil, som finns för varje task och process, står information som PID, PPID, kommandorad, schemaläggningsprioritet,schemaläggningstyp, exekveringstid i user och kernel-mode, m.m. Den används av många av de verktyg för resursmätning som vi använt.

/proc/<PID>/status och
/proc/<PID>/limit/task/<TID>/status

Information om minnesanvändning, signalhantering m.m.

/proc/sys/kernel/sched_rt_period_us och
/proc/sys/kernel/sched_rt_runtime_us

Dessa båda flaggor reglerar hur länge realtidsprocesser får köra som längst. /proc/sys/kernel/sched_rt_period_us bestämmer under vilken tidsperiod (i mikrosekunder) schemaläggaren ska kontrollera exekveringstiden av realtidsprocesser. /proc/sys/kernel/sched_rt_runtime_us bestämmer hur lång tid (också i mikrosekunder) realtidsprocesser maximalt får exekvera under denna tidsperiod. Man får altså den maximala exekveringstiden (i procent) om man dividerar runtime med period.

/proc/<PID>/limit och
/proc/<PID>/limit/task/<TID>/limit

Resursbegränsningar (eng: resource limits) anges för aktuell process eller task.

/proc/<PID>/sched och
/proc/<PID>/limit/task/<TID>/sched

Data om schemaläggning, som antal kontext-byten, prioritet m.m.

/sys/block/<DEVICE>/queue/scheduler

Anger och konfigurerar vilken I/O-schemaläggare som kan används för aktuell enhet.

B Relevanta kompileringsflaggor för Linuxkärnan

Flaggor relevanta för cgrupper

CONFIG_GROUP_SCHED
CONFIG_FAIR_GROUP_SCHED
CONFIG_RT_GROUP_SCHED
CONFIG_CGROUP_SCHED
CONFIG_CGROUPS
CONFIG_CGROUP_DEBUG
CONFIG_CGROUP_NS
CONFIG_CGROUP_FREEZER
CONFIG_CGROUP_DEVICE
CONFIG_CPUSETS
CONFIG_PROC_PID_CPUSET
CONFIG_CGROUP_CPUACCT
CONFIG_RESOURCE_COUNTERS
CONFIG_CGROUP_MEM_RES_CTLR

Flaggor relevanta för tc

CONFIG_NET_SCH_CBQ
CONFIG_NET_SCH_HTB
CONFIG_NET_SCH_HFSC
CONFIG_NET_SCH_PRIO
CONFIG_NET_SCH_MULTIQ
CONFIG_NET_SCH_RED
CONFIG_NET_SCH_SFQ
CONFIG_NET_SCH_TEQL
CONFIG_NET_SCH_GRED
CONFIG_NET_SCH_DSMARK

CONFIG_NET_EMATCH
CONFIG_NET_EMATCH_STACK
CONFIG_NET_EMATCH_CMP
CONFIG_NET_EMATCH_NBYTE
CONFIG_NET_EMATCH_U32
CONFIG_NET_EMATCH_META
CONFIG_NET_EMATCH_TEXT

CONFIG_NET_CLS
CONFIG_NET_CLS_BASIC
CONFIG_NET_CLS_TCINDEX
CONFIG_NET_CLS_ROUTE4
CONFIG_NET_CLS_ROUTE
CONFIG_NET_CLS_FW
CONFIG_NET_CLS_U32
CONFIG_NET_CLS_ACT
CONFIG_NET_ACT_POLICE
CONFIG_NET_ACT_GACT
CONFIG_NET_CLS_CGROUP

CONFIG_NET_ACT_MIRRED
CONFIG_NET_ACT_IPT
CONFIG_NET_ACT_NAT
CONFIG_NET_ACT_PEDIT
CONFIG_NET_ACT_SIMP
CONFIG_NET_ACT_SKBEDIT
CONFIG_NET_CLS_IND

Flaggor relevanta för VServer

CONFIG_VSERVER
CONFIG_VSERVER_AUTO_LBACK
CONFIG_VSERVER_AUTO_SINGLE
CONFIG_VSERVER_COWBL
CONFIG_VSERVER_VTIME
CONFIG_VSERVER_DEVICE
CONFIG_VSERVER_PROC_SECURE
CONFIG_VSERVER_HARDCPU
CONFIG_VSERVER_IDLETIME
CONFIG_VSERVER_IDLELIMIT
CONFIG_VSERVER_PRIVACY
CONFIG_VSERVER_CONTEXTS
CONFIG_VSERVER_WARN
CONFIG_VSERVER_DEBUG

C Parametrar för och beskrivning av datainsamling

C.1 Mätning av bildsystemet

- På klientdatorn startades två förfrågningar efter bildströmmar i motion JPEG från kamerasystemet (P3301), med upplösning 640x480, och en kompressionsgrad av 30 (enligt kamerans interface för att ställa detta). Bildströmmarna var inställda så att det fanns en överlagring av dels datainsamlingshastigheten, dels antalet genererade bilder per sekund. Enbart från en av dessa två bildströmmar togs data vad gäller datainsamlingshastighet och antal biler per sekund.
- Ett antal CPU-belastande processer startades på kameran varefter systemet fick stabilisera sig under ca 5 minuter.
- netcat startas på klienten för att lyssna på inkommande trafik.
- `top` startades på kameran i batchmode och med ett uppdateringsintervall av 0,5 s samt med PID-övervakning av bildsystemet, d.v.s. enbart data från detta samlades in vilket minskade belastning på kameran från `top`. `top` kördes som en realtidsprocess enligt SCHED_OTHER och med samma prioritet som bildsystemet i förekommande fall.
- Data från `top` pipe:ades till netcat som kördes som realtidsprocess enligt SCHED_OTHER och med samma prioritet som `top`.
- Data insamlas under minst 1 minut, oftast mer än 2 minuter.

Relevant data sorterade därefter ut från erhållen data. Data från de första 5 s användes ej vid beräkningar av medelvärden för CPU-tid.

Exampel på kommando som kördes på kameran för att starta datainsamling[32]:

```
./chrt -r 5 /usr/bin/top-org  -p 608 -d 0.5 -b |
./chrt -r 5 nc 192.168.0.1 8080
```

[32]chrt är ett verktyg för att manipulera realtidsattribut för en process, och chrt -r 5 startar angiven process som en SCHED_RR-process med realtidsprioritet 5. Detta verktyg liksom vårt egenutvecklade verktyg användes för att starta realtidsprocesser.

C.2 Mätning av signalvariabilitet

Signalvariabilitet för signaleringsverktyget

På kameran hade initialt merparten av de processer som ansågs kunna störa datainsamlingen stängts ned. Bland dessa fanns t.ex. processer tillhörande bildsystemet, webserver. Vissa processer kunde dock inte stängas ned.

- Ett antal, 0, 1, 2, 3, 4, 5, 10, 50 eller 100, CPU-belastande processer startades på kameran, varefter systemet fick stabilisera sig beroende på antal uppstartade processer. För 100 processer fick systemet stabilisera sig under ca 5 minuter. Hur processerna skulle startas varierade beroende på om de belastande processer skulle köras i en cgrupp, i en VServer-kontext eller som vanliga processer.
- Signaleringsverktyget startades på kameran. Hur verktyget startades varierade beroende på om det skulle köras i en cgrupp, som en realtidsprocess eller utanför sådan kontroll. Antalet arbetsiterationer varierades och varje antal arbetsiterationer kördes med 0, 1, 2, 3, 4, 5, 10, 50 och 100 CPU-belastande processer.
- På den datainsamlande datorn startades omedelbart efter ovanstående det program som insamlade data.
- Datainsamlingen pågick mellan 2 och drygt 30 minuter beroende på antal erhållna tidssignaler. Målet var 2000 till 3000 datapunkter.

Från erhållen data beräknades skillnaden mellan varje par av konsekutiva tidssginaler. För medelvärdeberäkning och beräkning av standardavvikelse bortsågs från de första 10 datapunkterna.

Målvärdet för cykeltiden sattes till 30 ms. Antalet arbetsiterationer för signaleringsverktyget varierades.

Exempel på uppstart av signaleringsverktyget som en process med ordinär statisk prioritet och som lyssnar på en klient på tcp/8081. Ingen resurssstyrning. 30 ms målcykeltid och 200 arbetsiterationer[33]:

```
/tmp/c_server -s 30 -i 200 -p 8081
```

Som ovan fast med placering av processen i en cgrupp (/tmp/cg/high)[34]:

[33] c_server är signaleringsverktyget där parametern -s anger cykeltid i ms, -i anger antal arbetsiterationer och -p anger den port programmet ska lyssna på klienter på.

[34] cgstart startar angivet kommando i den av första parameter angivna cgruppen.

```
/tmp/cgstart /tmp/cg/high /tmp/c_server -s 30 -i 200 -p 8081
```

Som ovan fast med 300 arbetsiterationer och som realtidsprocess, med prioritet 5, i schemaläggningspolicyn SCHED_RR:

```
/tmp/chrt -r 5  /tmp/c_server -s 30 -i 300 -p 8081
```

Uppstart av CPU-belastande processer, här exemplifierat med 10 sådana med ordinär statisk prioritet och utan resursstyrning[35]:

```
/tmp/stress -c 10
```

Som ovan fast med de CPU-belastande processerna i en cgrupp:

```
/tmp/cgstart /tmp/cg/low /tmp/stress -c 10
```

Som ovan fast med de CPU-belastande proceserna i en VServer-kontext. Först starades skalprogrammet bash, som portats till målarkitekturen, i en ny kontext. Därefter sattes utanför denna kontext schemaläggningsprincipen till hård för den nyskapande kontexten, och CPU-andelen sattes för den nyskapade kontexten, dvs[36]:

```
vcontext --create --xid 101 bash
vattribute --xid 101 --flag sched_hard
vsched --xid 101 --fill-rate 1 --interval 5 --tokens 10
       --tokens-max 10 --tokens-min 0
```

De CPU-belastande processer startades sedan, som ovan, i den nyskapade kontexten, medan signaleringsverktyget startades utanför denna kontext.

Ovanstående kördes på den ARTPEC-3-baserade P3301 med version 2.6.31 av Linuxkärnan där cgrupper aktiverats men där kärnan inte var preemptible. Programvarans version var i övrigt LinuxFirmwarePlatform-2_1 beta8. Testen för VServer kördes dock på systemprototypen med version 2.6.31 av Linux-kärnan, som patchats för VServer version vs2.3.0.36.14-pre8, som inte var preemptible. På systemprototypen kördes också tester av cgrupper på en ej VServer-patchad Linux-kärna.

[35] stress är ett belastningsverktyg som med parametern -c anger det antal CPU-belastande processer som skall startas.

[36] vcontext är ett verktyg för att hantera VServer-kontexter och däribland skapa sådana. vattribute är ett verktyg för att ställa olika kontext-attribut, t.ex. som här om schemaläggningen av kontexter ska vara hård.
vsched är ett verktyg för att ställa schemaläggningsparametrar för kontexter.

Signaleringsverktygets CPU-belastning

- Signaleringsverktyget startades. Hur detta gjordes varierade beroende på om verktyget skulle köras i en cgrupp, som en realtidsprocess eller som en vanlig process (se föregående stycker). Antalet arbetsiterationer varierades så att varje serie av CPU-belastningar kördes för varje antal arbetsiterationer. Målcykeltiden sattes till 30 ms.

- Klienten för insamling av tidssignaler startades på klientdatorn. Mottagen data kastades.

- Ovanstående båda processer kördes oavbrutet under det att varje serie av CPU-belstningar kördes.

- Ett antal CPU-belastande processer, 0, 1, 2, 3, 4, 5, 10, 50 eller 100, staratdes därefter. Dessa startades beroende på test som vanliga processer, i en cgrupp, eller i en VServer-kontext. De startades som beskrivits tidigare.

- Systemet fick stabiliseras under ett ca 1 minut, dock beroende på hur många CPU-belastande processer som startats.

- netcat startades på klientdatorn för att samla in data.

- `top`, i batchmode, med PID-bevakning av signaleraren och med 0,5 s cykeltid startades med schemaläggningspolicy SCHED_RR med prioritet 5 och pipeade data till netcat, också med schemaläggningspolicy SCHED_RR och prioritet 5. Den senare överförde data till klienten.

Konsekutiva datapunkter användes för beräkning av periodtid, där data från de första 5 s kastades.

D System för uppstart av processer i cgrupper

Exempel på gemensam konfigurationsfil (`/etc/cgroup_base.conf`) för skapande och konfiguration av cgrupper, samt placering av daemon-processer i relevanta cgrupper. Enbart den första delen av filen ("configuration for the cgroup [...]") är av betydelse i detta sammanhang.

```
# Configuration for the cgroup startup system
#  ========================================
# Root dir for cgroups
CGROUP_ROOT_DIR=/tmp/cg
# subsystems to load
CGROUP_SUBSYS=cpu,memory
# Dir with config files per cgroup
CGROUP_CONFIG_DIR=/etc/cg_conf


# Configuration for the cgrpd oom_kill monitor daemon
# ================================================
# log file to monitor
export CGRPD_LOG_FILE=/var/log/cgrpd.log
# file to define cgroup, priority and handler triples
CGRPD_CONF_FILE=/etc/cgrp.conf


# Configuration for the exec_meta_handler
# =======================================
# Where to put the meta file structure for startup.
# cgroup metadir MUST be in tmp
CGROUP_META_DIR=/tmp/cgroup_meta/
# where to find the script to start links in the meta file structure
export CGROUP_SCRIPT_DIR=/etc/cgroup-scripts/
```

Exempel på konfigurationsfil för en cgrupp. Denna fil ska vara placerade i katalogen CGROUP_CONFIG_DIR och ha suffixet ".sh". Denna fil kan här antas nedan ha namnet `testcgrupp.sh`.

```
CGROUP_PATH=/subgroup
CPU_SHARES=2048
LIMIT_IN_BYTES=10M
```

Shell-script, vilket körs som ett SysV-init-script tidigt i uppstartsprocessen, för att skapa relevanta cgrupper.

```
#!/bin/sh

[ ! -e /etc/cgroup_base.conf ] && exit

source /etc/cgroup_base.conf

case $1 in
    start)
        echo " * Creating cgroups..."
        mkdir -p /$CGROUP_ROOT_DIR
        mount -t cgroup none /$CGROUP_ROOT_DIR -o $CGROUP_SUBSYS || exit

        for a in /$CGROUP_CONFIG_DIR/*.sh ; do
            source $a
            mkdir -p /${CGROUP_ROOT_DIR}/${CGROUP_PATH}
            unset CGROUP_PATH
        done
        echo `` * Done creating cgroups''
        ;;
    stop)
        # nothing to do
        ;;
    restart)
        # nothing to do
        ;;
esac
```

Script-fil, `/etc/init.d/add_to_cgroup.sh`, som sourcas för att placera en daemon i en cgrupp.

```
if [ ! -z $CGROUP_NAME ] && [ -e /etc/cgroup_base.conf ]; then
    echo " * Putting $0 in cgroup...''
    source /etc/cgroup_base.conf

    CONFIG_FILE=''$CGROUP_CONFIG_DIR/$CGROUP_NAME.sh''

    if [ -e  $CONFIG_FILE ]; then
        source $CONFIG_FILE

        TASKS_FILE=''$CGROUP_ROOT_DIR/$CGROUP_PATH/tasks''
        [ -e $TASKS_FILE ] && echo $$ >> $TASKS_FILE
    fi
    echo `` * $0 put in cgroup"
fi
```

Exempel på användning i ett SysV-init-script.

```
#!/bin/sh

CGROUP_NAME=testcgrupp

# övrig initiering

case $1 in
     start)
          . /etc/init.d/add_to_cgroup.sh
          # här sker på vanligt sätt start av daemon-processen
          ;;
    # övriga delar av case
esac
```

Shell-script för att konfigurera cgrupper. Körs som SysV-init-script sent i uppstartsprocessen, d.v.s. efter att alla SysV-initscript innehållande daemoner som ska köras i cgrupper har körts.

```
#!/bin/sh

[ ! -e /etc/cgroup_base.conf ] && exit

source /etc/cgroup_base.conf

case $1 in
    start)
        echo " * Setting cgroup attributes..."

        for a in /$CGROUP_CONFIG_DIR/*.sh ; do
            source $a

            CGROUP_DIR=/${CGROUP_ROOT_DIR}/${CGROUP_PATH}/
            [ -e ${CGROUP_DIR}/cpu.shares ] &&  \
                [ ! -z $CPU_SHARES ] && \
                echo $CPU_SHARES >> /${CGROUP_DIR}/cpu.shares
            unset CPU_SHARES

            [ -e ${CGROUP_DIR}/memory.limit_in_bytes ] && \
                [ ! -z $LIMIT_IN_BYTES ] && \
                echo $LIMIT_IN_BYTES >> \
                     ${CGROUP_DIR}/memory.limit_in_bytes
```

```
            unset LIMIT_IN_BYTES

            unset CGROUP_PATH
        done

        echo ‘‘ * Done setting cgroup attributes’’
        ;;

    stop)
        # noting to do
        ;;
esac
```

Referenser

[1] Galvin et al, Operating System Concepts, 7:e utg, Wiley (2005)

[2] Real-time computing, wikipedia.
`http://en.wikipedia.org/wiki/Real-time_computing` (Åtkommen 2010-04-21)

[3] McKusick, Twenty Years of Berkley Unix From AT&T owned to Freely Redistributable, Open Sources: Voices from the Open Source Revolution, 1:a utg, O'Reilly (1999).
`http://oreilly.com/catalog/opensources/book/kirkmck.html`
(Åtkommen 2010-05-16)

[4] The Open Group, webbplats.
`http://www.opengroup.org/` `http://www.unix.org/`

[5] The Open Source Initiative, webbplats.
`http://opensource.org/` (Åtkommen 2010-05-16)

[6] Top500, webbplats. `http://www.top500.org` (Åtkommen 2010-05-02)

[7] GNU Operating System, webbplats.
`http://www.gnu.org/` (Åtkommen 2010-05-16)

[8] Free Software Foundation, webbplats.
`http://www.fsf.org/` (Åtkommen 2010-05-16)

[9] The FreeBSD Diary, webbplats.
`http://www.freebsddiary.org/linux.php` (Åtkommen 2010-07-20)

[10] Heiser, The Role of Virtualization in Embedded Systems, 1st Workshop on Isolation and Integration in Embedded Systems, 11-16 (2008)

[11] Abbot, Linux for Embedded and Realtime Applications, Newnes (2003)

[12] Embedded system, Wikipedia.
`http://en.wikipedia.org/wiki/Embedded_system` (Åtkommen 2010-05-03)

[13] Bacon et al, Operating Systems Concurrent and Distributed Software Design, Addison-Wesley (2003).

[14] Tanenbaum, Operating Systems Design and Implementation, 3:e utg, Prentice Hall (2006).

[15] Stallings, Operating Systems Internals and Design Principles, 6:e utg, Prentice Hall (2009).

[16] LinuxMM, webbplats. `http://linux-mm.org/` (Åtkommen 2010-05-16)

[17] Scheduling, Wikipedia.
http://en.wikipedia.org/wiki/Scheduling_(computing) (Åtkommen 2010-05-16)

[18] Scheduling algorithm, wikipedia.
http://en.wikipedia.org/wiki/Scheduling_algorithm (Åtkommen 2010-05-16)

[19] Cherkasova, Comparison of Three CPU Schedulers in Xen, ACM SIGMETRICS Performance Evaluation Review, Vol. 35, No. 2, 42-51 (2007).
http://www.hpl.hp.com/personal/Lucy_Cherkasova/papers/per-3sched-xen.pdf

[20] Yaghmour et al, Building Embedded Linux Systems, 2:a utg, O'Reilly (2008)

[21] Linux kernel källkod, Documentation/cgroups/cgroups.txt (Linux kernel 2.6.33). http://git.kernel.org (Åtkommen 2010-03-29)

[22] Axis Communications AB, intern kommunikation.

[23] Jitter, wikipedia.
http://en.wikipedia.org/wiki/Jitter (Åtkommen 2010-04-16)

[24] Standard deviation, Wikipedia.
http://en.wikipedia.org/wiki/Standard_deviation (Åtkommen 2010-07-10)

[25] Disk Scheduling in Linux.
http://www.docstoc.com/docs/23937336/Disk-Scheduling-In-Linux (Åtkommen 2010-05-10)

[26] CFQ, Wikipedia.
http://en.wikipedia.org/wiki/CFQ (Åtkommen 2010-05-10)

[27] Noop scheduler, Wikipedia.
http://en.wikipedia.org/wiki/Noop_scheduler (Åtkommen 2010-05-10)

[28] Anticipatory scheduling, Wikipedia.
http://en.wikipedia.org/wiki/Anticipatory_scheduling (Åtkommen 2010-05-10)

[29] Deadline scheduler, Wikipedia.
http://en.wikipedia.org/wiki/Deadline_scheduler (Åtkommen 2010-05-10)

[30] Corbet, Disk I/O priorities, lwn.net (2003).
http://lwn.net/Articles/57732/ (Åtkommen 2010-05-10)

[31] Skeppstedt, Föreläsningsmaterial EDA050, LTH (2010).
http://www.cs.lth.se/EDA050/ (Åtkommen 2010-05-10)

[32] Observer effect, wikipedia.
http://en.wikipedia.org/wiki/
Observer_effect_(information_technology) (Åtkommen 2010-14-21)

[33] Majidimehr, Optimizing UNIX for Performance, Prentice Hall (1996).

[34] Ojha, Techniques in Least-Intrusive Computer System Performance Monitoring, SoutheastCon 2001. Proceedings. IEEE, 150-154 (2001)

[35] Operating System, Wikipedia.
http://en.wikipedia.org/wiki/Operating_system (Åtkommen 2010-05-02)

[36] Process management, computing, Wikipedia.
http://en.wikipedia.org/wiki/Process_management_%28computing%29
(Åtkommen 2010-05-02)

[37] Interrupt, Wikipedia.
http://en.wikipedia.org/wiki/Interrupt (Åtkommen 2010-05-02)

[38] Memory management, Wikipedia.
http://en.wikipedia.org/wiki/Memory_management (Åtkommen 2010-05-02)

[39] MMU, Wikipedia.
http://en.wikipedia.org/wiki/Memory_management_unit (Åtkommen 2010-05-02)

[40] Virtual Memory, Wikipedia.
http://en.wikipedia.org/wiki/Virtual_memory (Åtkommen 2010-05-02)

[41] The Linux information project.
http://www.linfo.org/context_switch.html (Åtkommen 2010-04-19)

[42] Bovet et al, Understanding the Linux Kernel, 3:e utg, O'Reilly (2005).

[43] preempt rt patch, webbplats. https://rt.wiki.kernel.org/ (Åtkommen 2010-03-29)

[44] Aas, Understanding the Linux 2.6.8.1 CPU Scheduler, Silicon Graphics inc. (2005). http://joshaas.net/linux/linux_cpu_scheduler.pdf

[45] Jones, Inside the linux scheduler, IBM Developerworks (2006).
http://www.ibm.com/developerworks/linux/library/l-scheduler/
(Åtkommen 2010-04-26)

[46] Jeremy, Linux: The complete fair scheduler, Kernel Trap (2007).
http://kerneltrap.org/node/8059 (Åtkommen 2010-04-26)

[47] Pabla, Completely Fair Scheduler, Linux Journal 184, 68-72 (2009).

[48] Corbet, CFS group scheduling, lwn.net (2007).
http://lwn.net/Articles/240474/ (Åtkommen 2010-04-26)

[49] Jones, Inside the Linux 2.6 Complete Fair Scheduler, IBM Developerworks (2009).
http://www.ibm.com/developerworks/linux/library/l-completely-fair-scheduler/ (Åtkommen 2010-05-10)

[50] Kumar, Multiprocessing with the Completely Fair Scheduler, Introducing the CFS for Linux, IBM Developerworks (2008).
http://www.ibm.com/developerworks/linux/library/l-cfs/ (Åtkommen 2010-05-10)

[51] Gorman, Understanding the Linux Virtual Memory Manager, Prentice Hall (2004).

[52] fork, manualsidor, Debian paketversion 3.05-1 av manpages-dev.

[53] Mulyadi Santosa, When Linux Runs out of Memory, linuxdevcenter.com (2006).
http://linuxdevcenter.com/pub/a/linux/2006/11/30/linux-out-of-memory.html (Åtkommen 2010-04-26)

[54] proc-filsystemet, manualsidor, Debian paketversion 3.05-1 för manpages.

[55] Corbet, Toward a smarter OOM killer, lwn.net (2009).
http://lwn.net/Articles/359998/ (Åtkommen 2010-04-26)

[56] Edge, Avoiding the OOM killer with mem_notify, lwn.net (2008).
http://lwn.net/Articles/267013/ (Åtkommen 2010-04-26)

[57] Corbet, Respite from the OOM killer, lwn.net (2004).
http://lwn.net/Articles/104179/ (Åtkommen 2010-04-26)

[58] top, wikipedia.
http://en.wikipedia.org/wiki/Top_(Unix) (Åtkommen 2010-03-29)

[59] top, manualsidor, Debian paketversion 1:3.2.7-11 av procps.

[60] atop, webbplats.
http://www.atoptool.nl/ (Åtkommen 2010-03-29)

[61] htop, webbplats.
http://htop.sourceforge.net/ (Åtkommen 2010-03-29)

[62] Busybox, webbplats.
http://www.busybox.net/ (Åtkommen 2010-04-08)

[63] vmstat, wikipedia.
http://en.wikipedia.org/wiki/Vmstat (Åtkommen 2010-03-29)

[64] free, manualsidor, Debian paketversion 1:3.2.7-11 av procps.

[65] sysstat, webbplats.
http://pagesperso-orange.fr/sebastien.godard/ (Åtkommen 2010-03-29)

[66] ps, manualsidor, Debian paketversion 1:3.2.7-11 av procps.

[67] procinfo, manualsidor, Debian paketversion 18-2+lenny1.

[68] dstat, webbplats.
http://dag.wieers.com/home-made/dstat/ (Åtkommen 2010-03-30)

[69] psinfo, webbplats.
http://www.ward.nu/computer/psinfo/ (Åtkommen 2010-03-30)

[70] pmap, manualsidor, Debian paketversion 1:3.2.7-11 av procps.

[71] iftop, webbplats.
http://www.ex-parrot.com/pdw/iftop/ (Åtkommen 2010-04-16)

[72] iptraf, webbplats.
http://iptraf.seul.org/ (Åtkommer 2010-05-01)

[73] nethogs, webbplats.
http://nethogs.sourceforge.net/ (Åtkommen 2010-04-16)

[74] iperf, webbplats.
http://iperf.sourceforge.net/ (Åtkommen 2010-04-16)

[75] Wireshark, webbplats.
http://www.wireshark.org/ (Åtkommen 2010-05-01)

[76] tcpdump, webbplats.
http://www.tcpdump.org/ (Åtkommen 2010-05-01)

[77] iotop, webbplats.
http://guichaz.free.fr/iotop/ (Åtkommen 2010-04-09)

[78] ipbench, webbplats.
http://ipbench.sourceforge.net/ (Åtkommen 2010-04-09)

[79] Latencytop, webbplats.
http://www.latencytop.org/ (Åtkommen 2010-04-08)

[80] Edge, Finding system latency with LatencyTOP, lwn.net (2008).
http://lwn.net/Articles/266153/ (Åtkommen 2010-04-08)

[81] Schulzrinne et al, RTP: A Transport Protocol for Real-Time Applications, RFC 2550 (2003).
http://www.ietf.org/rfc/rfc3550.txt

[82] Linux kernel källkod, Documentation/trace/ftrace.txt (kernel version 2.6.32)

[83] Rostedt, Debugging the Kernel using Ftrace, part 1, lwn.net (2009). http://lwn.net/Articles/365835/ (Åtkommen 2010-04-08)

[84] Rostedt, Debugging the Kernel using Ftrace, part 2, LWN.net (2009). LWN.net, http://lwn.net/Articles/366796/ (Åtkommen 2010-04-08)

[85] Brindley, RedHat Enterprise MRG 1.2, Realtime Tuning Guide, Section tion 3.7, Using the ftrace Utility for Tracing Latencies, RedHat (2009). http://www.redhat.com/docs/en-US/Red_Hat_Enterprise_MRG/1.2/html/Realtime_Tuning_Guide/sect-Realtime_Tuning_Guide-Realtime_Specific_Tuning-Using_the_ftrace_Utility_for_Tracing_Latencies.html (Åtkommen 2010-04-07)

[86] Linux kernel källkod, Documentation/scheduler/sched-stat.txt (kernel version 2.6.32)

[87] Linux Scheduler Statistics. http://eaglet.rain.com/rick/linux/schedstat/ (Åtkommen 2010-04-08)

[88] schedtop från preempt_rt_patch:s webbplats. https://rt.wiki.kernel.org/index.php/Schedtop_utility (Åtkommen 2010-04-08)

[89] Systemtap, webbplats. http://sourceware.org/systemtap/ (Åtkommen 2010-04-12)

[90] Linux kernel källkod, Documentation/kprobes.txt (kernel version 2.6.32)

[91] Cohen, Getting insight into the Linux kernel with Kprobes, RedHat (2005). http://www.redhat.com/magazine/005mar05/features/kprobes/ (Åtkommen 2010-04-12)

[92] Panchamukhi, Kernel debugging with Kprobes, IBM DeveloperWorks (2004). http://www.ibm.com/developerworks/library/l-kprobes.html (Åtkommen 2010-04-12)

[93] Goswami, An introduction to KProbes, LWN.net (2005). http://lwn.net/Articles/132196/ (Åtkommen 2010-04-12)

[94] Xen, webbplats. http://www.xen.org/ (Åtkommen 2010-05-16)

[95] vserver, webbplats. http://linux-vserver.org (Åtkommen 2010-05-10)

[96] System Administration Guide: Solaris Containers-Resource Management and Solaris Zones, Sun Microsystems (2009). http://docs.sun.com/app/docs/doc/817-1592 (Åtkommen 2010-05-16)

[97] FreeVPS, webbplats.
http://www.freevps.com (Åtkommen 2010-05-20)

[98] lxc Linux Containers, webbplats.
http://lxc.sourceforge.net/ (Åtkommen 2010-05-20)

[99] OpenVZ, webbplats.
http://wiki.openvz.org/ (Åtkommen 2010-05-01)

[100] Understanding Full Virtualization, Paravirtualization, and Hardware Assist, VMWare (2007).
http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf
(Åtkommen 2010-05-16)

[101] Jones, Virtual Linux, An overview of virtualization methods, architectures, and implementations, IBM Developerworks (2006).
http://www.ibm.com/developerworks/library/l-linuxvirt/
(Åtkommen 2010-05-20)

[102] Hardware virtualization, Wikipedia.
http://en.wikipedia.org/wiki/Hardware_virtualization (Åtkommen 2010-05-16)

[103] Hardware-assisted virtualization, Wikipedia.
http://en.wikipedia.org/wiki/Hardware-assisted_virtualization
(Åtkommen 2010-05-16)

[104] Hypervisor, wikipedia.
http://en.wikipedia.org/wiki/Hypervisor (Åtkommen 2010-05-16)

[105] Paravirtualization, Wikipedia.
http://en.wikipedia.org/wiki/Paravirtualization (Åtkommen 2010-05-16)

[106] Full virtualization, Wikipedia.
http://en.wikipedia.org/wiki/Full_virtualization (Åtkommen 2010-05-10)

[107] Operating system-level virtualization, Wikipedia.
http://en.wikipedia.org/wiki/Operating_system-level_virtualization
(Åtkommen 2010-05-16)

[108] Padala et al, Performace Evaluation of Virtualization Technologies for Server Consolidation, HP Technical Reports (2007).
http://www.hpl.hp.com/techreports/2007/HPL-2007-59R1.html

[109] KVM, Kernel Based Virtual Machine, webbplats.
http://www.linux-kvm.org (Åtkommen 2010-05-16)

[110] KVM - Kernel Based virtual Machine, Red Hat (2009).
http://www.redhat.com/f/pdf/rhev/DOC-KVM.pdf (Åtkommen 2010-05-16)

[111] User Mode Linux, webbplats.
http://user-mode-linux.sourceforge.net/ (Åtkommen 2010-05-16)

[112] Dike, User-Mode Linux.
www.kernel.org/doc/ols/2001/uml.pdf (Åtkommen 2010-05-16)

[113] vserver resource limits, webbplats.
http://linux-vserver.org/Resource_Limits (Åtkommen 2010-05-10)

[114] vserver CPU limits, webbplats.
http://linux-vserver.org/Applying_CPU_Limits (Åtkommen 2010-05-10)

[115] vserver cgroups, webbplats.
http://linux-vserver.org/util-vserver:Cgroups (Åtkommen 2010-05-10)

[116] OpenVZ resource management, webbplats.
http://wiki.openvz.org/Resource_management (Åtkommen 2010-05-10)

[117] OpenVZ Features, webbplats.
http://wiki.openvz.org/Features (Åtkommen 2010-05-10)

[118] stress, webbplats. http://weather.ou.edu/ apw/projects/stress/ (Åtkommen 2010-04-12)

[119] cpulimit, webbplats. http://cpulimit.sourceforge.net/ (Åtkommen 2010-05-05)

[120] Auto nice daemon, webbplats. http://and.sourceforge.net/ (Åtkommen 2010-05-05)

[121] setrlimit, manualsidor, Ubuntu paketversion 3.23-1 för manpages-dev.

[122] setrlimit, manualsidor, Ubuntu paketversion 2.16-1 för manpages-posix-dev.

[123] Linux PAM, webbplats.
http://www.kernel.org/pub/linux/libs/pam/ (Åtkommen 2010-05-05)

[124] pam_limits, manualsidor, Ubuntu paketversion 1.1.1-2ubuntu5 för libpam-modules.

[125] Linux kernel källkod, Documentation/scheduler/sched-design-CFS.txt (Linux kernel 2.6.35.2). http://git.kernel.org (Åtkommen 2010-07-29)

[126] Linux kernel källkod, Documentation/cgroups/memory.txt (Linux kernel 2.6.35.2). http://git.kernel.org (Åtkommen 2010-07-29)

[127] Linux kernel källkod, Documentation/cgroups/freezer-subsystem.txt (Linux kernel 2.6.35.2). http://git.kernel.org (Åtkommen 2010-07-29)

[128] Linux kernel källkod, Documentation/cgroups/devices.txt (Linux kernel 2.6.35.2). http://git.kernel.org (Åtkommen 2010-07-29)

[129] Linux kernel källkod, Documentation/cgroups/blkio-controller.txt (Linux kernel 2.6.35.2). http://git.kernel.org (Åtkommen 2010-07-29)

[130] Linux kernel källkod, Documentation/cgroups/cpusets.txt (Linux kernel 2.6.35.2). http://git.kernel.org (Åtkommen 2010-07-29)

[131] Linux kernel källkod, Documentation/cgroups/cpuacct.txt (Linux kernel 2.6.35.2). http://git.kernel.org (Åtkommen 2010-07-29)

[132] Netfilter, webbplats.
http://www.netfilter.org/ (Åtkommen 2010-05-16)

[133] Traffic shaping, Wikipedia.
http://en.wikipedia.org/wiki/Traffic_shaping (Åtkommen 2010-05-17)

[134] Traffic policing, Wikipedia.
http://en.wikipedia.org/wiki/Traffic_policing (Åtkommen 2010-05-17)

[135] Linux TC How-To, webbplats.
http://linux-ip.net/articles/Traffic-Control-HOWTO/ (Åtkommen 2010-05-17)

[136] tcng, webbplats. http://tcng.sourceforge.net/ (Åtkommen 2010-05-17)

[137] tcng referens, webbplats. http://linux-ip.net/gl/tcng/ (Åtkommen 2010-05-17)

[138] ALGOL, Wikipedia.
http://en.wikipedia.org/wiki/ALGOL (Åtkommen 2010-05-17)

[139] tc, manualsidor, Debian paketversion 20080725-2 för iproute.

[140] ionice, manualsidor, Ubuntu paketversion 2.17.2-0ubuntu1 av util-linux.

[141] Righi, Per-task I/O throttling, Linux kernel Mailing List 2008-01-10.
http://lwn.net/Articles/264770/

[142] Tsuruta, bio-cgroup: Introduction, Linux Kernel Mailing List 2008-09-19.
http://lwn.net/Articles/299731/

[143] Jianfeng, introduce bio-cgroup into io-throttle, Linux Kernel Mailing List, 2008-11-10.
http://lwn.net/Articles/308108/

[144] Righi, cgroup: io-throttle controller, Linux Kernel Mailing List, 2009-04-28.
http://lwn.net/Articles/330531/

[145] Ioband, webbplats. http://sourceforge.net/apps/trac/ioband/ (Åtkommen 2010-05-20)

[146] Takahashi et al, dm-ioband A disk IO bandwidth controller Implemented as a Device-mapper Module.
http://www.valinux.co.jp/documents/tech/presentlib/2009/jls/ioband-jls2009.pdf

[147] Device-mapper Resource Page, webbplats.
http://sources.redhat.com/dm/

[148] Device mapper, Wikipedia.
http://en.wikipedia.org/wiki/Device_mapper

[149] van Dooren, Quota mini-Howto, v0.5, 2003-08-09.
http://tldp.org/HOWTO/Quota.html

[150] Axis P3301, produktinformation.
http://www.axis.com/products/cam_p3301/ (Åtkommen 2010-05-27)

[151] Axis Communications, ARTPEC-3 Designer's reference, 2007.

[152] Capabilities, kernel FAQ.
http://ftp.kernel.org/pub/linux/libs/security/linux-privs/kernel-2.4/capfaq-0.2.txt (Åtkommen 2010-05-28)

[153] Axis P1311, produktinformation.
http://www.axis.com/products/cam_p1311/ (Åtkommen 2010-06-01)

[154] Företagsintern AXIS-dokumentation.

[155] Axis Communications, ARTPEC-4 Designers Reference, 2010.

[156] Profiling, Wikipedia.
http://en.wikipedia.org/wiki/Profiling_(computer_programming) (Åtkommen 2010-06-02)

[157] init, Wikipedia.
http://en.wikipedia.org/wiki/Init (Åtkommen 2010-07-13)

[158] init, manualsidor, Debian paketversion 2.86.ds1-61 sysvinit.