hyväksymispäivä arvosana

arvostelija

Reaaliaikakäyttöjärjestelmän tarjoama tuki

Sami Saarinen

Helsinki 19.4.2001

Multimedia ja tietoliikenne / Lea Kutvonen

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Reaaliaikakäyttöjärjestelmän tarjoama tuki

Sami Saarinen

Multimedia ja tietoliikenne / Lea Kutvonen

Tietojenkäsittelytieteen laitos

Helsingin yliopisto

19.4.2001, 17 sivua

Tässä tutkielmassa tarkastellaan erilaisten multimediasovellusten reaaliaikavaatimuksia käyttöjärjestelmältä. Näiden vaatimusten ratkaisemiseksi tarkastellaan järjestelmän suorittimen skedulointimenetelmiä, sekä levypyyntöjen skedulointimenetelmiä. Lisäksi tarkastellaan jonkin verran multimediajärjestelmän yleisiä vaatimuksia mm. resurssien hallinnan ja muistinkäsittelyn osalta.

Aiheluokat(Computing Reviews 1998): D.4.7, H.5.1

Avainsanat: Käyttöjärjestelmät, Multimedia, Reaaliaikaisuus

Sisältö

1 Johdanto **1**

2 Multimediasovellukset **2**

3 Multimediakäyttöjärjestelmän toteutus **4**

4 Suorittimen skedulointi **6**

4.1 Reaaliaikaskedulointi 8

4.2 Hierarkinen skedulointi 10

5 Levypyyntöjen skedulointi **13**

6 Yhteenveto **16**

Lähteet **16**

1 Johdanto

Jatkuvaa datavirtaa käsittelevät multimediasovellukset, kuten videokonferenssit, ja jopa tavallinen audion ja videon toistaminen, asettavat käyttöjärjestelmille vaatimuksia, jotka alkavat muistuttaa reaaliaikakäyttöjärjestelmien vaatimuksia. Datavirtaa on pystyttävä käsittelemään reaaliajassa ilman viiveitä. Ei riitä pelkästään se, että verkkoyhteydestä on pystytty varaamaan tarpeeksi leveä kaista datavirralle, jos joko lähettävä tai vastaanottava järjestelmä ei pysty käsittelemään datavirtaa tarpeeksi tehokkaasti.

Tärkeimmät vaatimukset koskevat resurssien varausta ja hallintaa, ja siinä erityisesti suorittimen varausta. Suoritin täytyy voida varata periodisesti, jotta esimerkiksi jokainen pakattu videon kuva pystytään purkamaan ja näyttämään ajallaan. Lisäksi audioraitakin on usein purettava erikseen ja vietävä äänikortille toistettavaksi. Jos data itsessään tulee vielä tietoliikenneyhteyksien kautta on samaan aikaan onnistuttava hoitamaan vielä protokollapinon mukaiset käsittelyt tulevalle datalle.

Vaikka toisissa sovelluksissa, kuten audion ja videon toistossa, voidaan puskuroimalla dataa helpottaa käyttöjärjestelmän reaaliaikavaatimuksia, on mm. videokonferensseissa pitkä puskurointi mahdotonta, sillä itse data tuotetaan myös reaaliaikaisesti. Multimediasovelluksia tukevien käyttöjärjestelmien onkin toteutettava reaaliaikajärjestelmien ominaisuuksia [BlS98]. Esimerkiksi reaaliaikakäyttäytyminen täytyy olla ennustettavissa ja toteutettu tehokkaasti. Myös erilaisiin palvenlun laatu —pyyntöihin täytyy varautua, ja palveluille on voitava antaa erilaisia takuita. Lisäksi vielä järjestelmän käyttäytyminen ylikuormitustilassa on oltava hallittua.

Tässä tutkielmassa keskitytään suorittimen ja levypyyntöjen skeduloimiseen, sekä multimedian vaatimuksiin käyttöjärjestelmän rakenteelle. Aluksi katsotaan kuitenkin erilaisia multimediasovelluksia ja niiden vaatimuksia käyttöjärjestelmältä.

2 Multimediasovellukset

Multimediakäyttöjärjestelmää käytetään suorittamaan useita eri tyyppisiä sovelluksia. Nämä sovellustyypit vaativat suorittimen skeduloinnilta eri asioita. Seuraavassa listataan näiden sovellusten perustyypit ja selvitetään niiden vaatimusten eroja [GGV96]:

Kovat reaaliaikasovellukset: Nämä varsinaiset reaaliaikasovellukset vaativat tarkat suorittimen varaukset ja takuut siitä, että varaukset pidetään. Tähän luokkaan kuuluu sovellukset, joiden aikavaatimusten rikkominen johtaa jonkinlaiseen katastrofiin. Tällaisia sovelluksia ovat mm. CD-levyn polttoprosessi, jonka aikavaatimusten rikkominen johtaa levyn pilaantumiseen, sekä potilaan leikkaaminen telelääketieteen avulla, jossa datan käsittelyn tai siirron aikavaatimusten rikkominen voi pahimmillaan johtaa potilaan kuolemaan.

Pehmeät reaaliaikasovellukset: Tähän luokkaan kuuluvat sovellukset, joilla on reaaliaikavaatimuksia, mutta joissa vaatimusten rikkominen ei johda katastrofiin. Tähän luokkaan kuuluvat esimerkiksi multimediatiedoston katsominen ja etäkonferenssi. Aikavaatimusten rikkominen voi johtaa esimerkiksi videokuvan ja äänen pätkimiseen, jotka tietysti saattavat häiritä käyttäjää. Järjestelmä antaa tällaisille sovelluksille takuita esimerkiksi tietyn perustason kuvan ja äänen laadulle saavuttamiselle, mutta ei takaa, että paras mahdollinen laatu saavutetaan aina. Suorittimen käyttöasteen parantaminen on hyvä syy näiden heikompien reaaliaikavaatimusten käyttöön sovelluksissa. Esimerkiksi liikkuvan kuvan purkamisen vaatima prosessointiaika suorittimella vaihtelee yksittäisestä kuvasta toiseen ja olisi järjestelmän resurssien tuhlaamista varata kiinteä suoritinaika pahimman tapauksen mukaan.

Paras yritys -sovellukset: Tähän luokkaan kuuluvat sovellukset, joilla ei ole minkäänlaisia reaaliaikavaatimuksia. Esimerkkinä voisi olla multimediatiedoston muuntaminen toiseen formaattiin. Tiettyjen vaatimusten toteuttaminen on kuitenkin tarpeen myös näille sovelluksille, etteivät varsinaiset reaaliaikasovellukset vie kaikkea järjestelmän suoritinaikaa.

Multimediajärjestelmän on siis pystyttävä tarjoamaan reilua suorittimen skedulointia varsin erityyppisille sovelluksille. Järjestelmän antamat takuut on pidettävä, vaikka joku sovelluksista toimisikin hieman virheellisesti yrittäen esimerkiksi käyttää suoritinta enemmän kuin varaus antaa myöten. Järjestelmän on myös sopeuduttava uusiin tilanteisiin, siten että esimerkiksi uusien sovellusten käynnistäminen ei voi johtaa muiden sovellusten takuiden rikkomiseen. Järjestelmät käyttävät seuraavia keinoja pitääkseen annetut takuut [RJS00]:

Pääsyn valvonta: Pääsyn valvonnan avulla voidaan huolehtia siitä, ettei järjestelmä anna sellaisia takuita sovelluksille, joita se ei pysty pitämään. Jos järjestelmä on liian kuormitettu, kun uusia varauksia yritetään tehdä, niin järjestelmä voi estää sovelluksen käynnistymisen. Vaikka uusin sovellus ei todennäköisesti olekaan se, josta käyttäjä luopuu, ei järjestelmä voi suinpäin mennä sulkemaan muitakaan sovelluksia. Käyttäjä voi itse sulkea haluamiaan sovelluksia saadakseen haluamansa uuden sovelluksen käyntiin.

Resurssien hallinta: Järjestelmä voi yrittää neuvotella käynnissä olevien sovellusten kanssa resurssien luovuttamista uudelle sovellukselle. Tällöin joidenkin sovellusten palvelun laatua voitaisiin heikentää, jotta uusi prosessi saataisiin käyntiin.

Oppivat sovellukset: Sovellukset voivat myös itse tarkkailla varsinaista suoritinaikaa, jonka ne ovat saaneet. Tällöin esimerkiksi videon toistosovellus voisi itse siirtyä käyttämään huonompaa resoluutiota, jos järjestelmä on niin ylikuor-

Kuva 1: Multimediasovellusten luokittelu [RJS00].

type	examples	period	amount	degrades gracefully?	latency sensitivity
stored audio	MP3, AAC	around 100ms	1%–10%	no	low
stored video	MPEG-2, AVI	33ms	large	yes	low
distributed audio	Internet telephone	bursty	1%–10%	no	high
distributed video	video conferencing	bursty	large	yes	high
real-time audio	software synthesizer	1–20ms	varies	no	very high
RT simulation	virtual reality, Quake	up to refresh period	usually 100%	yes	high
RT hardware	soft modem, USB speakers	3–20ms	up to 50%	no	very high

mittunut, ettei sovellus koskaan saa enempää suoritinaikaa, kuin järjestelmän lupaama minimi.

Multimediasovellusten reaaliaikavaatimukset voidaan jakaa useaan luokkaan [RJS00]. Sovelluksen vaatima *periodin pituus* kertoo kuinka tiheään sovellus vaatii suoritinaikaa. *Suoritinajan määrä* kertoo sovelluksen vaatiman suoritinajan osuuden koko järjestelmän käytettävissä olevasta resurssien määrästä. *Toipuminen (graceful degradation)* kertoo sen, että kestääkö sovellus reaaliaikavaatimusten rikkoontumisen. *Viiveherkkyys (latency sensitivity)* kertoo sen, kuinka herkkä sovellus on järjestelmän päästä-päähän viipeelle. Kuvassa 1 nähdään eri multimediasovellusten vaatimukset näiden luokkien mukaan.

3 Multimediakäyttöjärjestelmän toteutus

Multimediakäyttöjärjestelmän toteutuksessa tärkeintä on resurssien varaaminen ja sovelluksille annettujen takuiden pitäminen. Järjestelmän omien toimintojen kuluttama suoritusaika on tiedettävä ennakkoon ja ne on oltava toteutettu tehokkaasti, jotta pahimpiinkin vaatimuksiin järjestelmältä voidaan vastata. Erityisesti on kiinnitettävä huomio muistin käsittelyn ja datan siirron tehokkuuteen [StN95].

Useat multimediasovellukset siirtävät suuria määriä dataa ympäri järjestelmää ja joissain tapauksissa datan käyminen varsinaisella keskusmuistilla voidaan kiertää resurssien säästämiseksi. Tästä on esimerkkinä suora piuha cd-rom asemasta äänikortille musiikin kuunteluun. Tämä säästää varsinaiselta järjestelmän omalta väylältä paljon datan siirtämistä. CD:n soitto-ohjelman tarvitsee saada CD-asemalta ainoastaan tietoa siitä, missä kohtaa kappaleessa mennään. Multimediajärjestelmän ja sen laitteiston on tuettava tämän kaltaisia "ylimääräisiä" väyliä multimedialaitteiden välillä.

Muistin käsittelyssä on tärkeää, että voidaan välttää turhia aikaa vieviä kopiointeja. Vaikka multimediasovellus usein tarvitseekin suuria määriä dataa, on kuitenkin sen käsittelyssä estettävä virtuaalimuistijärjestelmää siirtämästä sitä levylle odottamaan vuoroaan. Kun sovellus sitten tarvitsee sitä dataa, on sen nouto levyltä liian hidasta. Multimediasovellusten on siis voitava varata multimediadatalle puskurimuistia, jota ei voida poistaa keskusmuistista levylle.

Myös turhaa keskusmuistin sisäistä kopiointia on vältettävä. Esimerkiksi tietoliikenneprotokollapinon käsittelyssä on pinon eri kerrosten voitava käyttää yhteistä muistia, ettei paketteja tarvitsisi fyysisesti kopioida paikasta toiseen matkalla verkkokortilta sovellukselle useammin kuin on tarpeen. Minimissään data kopioidaan verkkokortilta keskusmuistiin paikkaan, josta sitä sitten itse sovelluskin käyttää. Tämä kuitenkin vaatii, ettei yksikään dataa käyttävä prosessi halua muuttaa sitä, vaan ne ainoastaan lukevat sitä ja välittävät siitä halutun osan ylemmälle tasolle (esimerkiksi IP prosessi lähettää TCP prosessille pointterin IP paketin sisällä olevaan TCP pakettiin).

Lisäksi eri varatut resurssit on priorisoitava siten, ettei alemman prioriteetin varaus estä korkeamman prioriteetin prosessin käyttämästä jotain tiettyä resurssia. Esimerkiksi tietoliikennepaketeissa olevat prioriteetit on käytettävä hyödyksi siten, että korkeamman prioriteetin paketit ohittavat muut paketit, jotka jonottavat sa-

Kuva 2: Reaaliaikaskeduloijien jaottelu [RJS00].

programming model	examples	load isolation	prior knowledge	support for varying latency requirements?
rate-monotonic and other static priority	Linux, RTLinux, Solaris, Windows 2000	isolated from lower priority	priority	yes
CPU reservations	Nemesis, Rialto, Spring	strong	period, amount	yes
proportional share	BVT, EEVDF, SMART	strong	share (, latency)	varies
earliest deadline first	Rialto, SMART	strong / weak	amount, deadline	yes
feedback control	FC-EDF, SWiFT	varies	metric, set point	varies
hierarchical scheduling	CPU Inheritance, SFQ	varies	varies	varies

malla protokollapinon tasolla.

4 Suorittimen skedulointi

Kuten kuvasta 2 huomataan, on multimediasovelluksille olemassa monia skeduloijia, joiden ennakkoon vaatimat tiedot prosesseista, ja joiden antamat takuut vaihtelevat runsaasti. Seuraavassa listassa tarkastellaan näiden eri tyyppisten skeduloijien ominaisuuksia [RJS00]:

Prioriteettipohjaiset skeduloijat: Nämä skeduloijat toimivat suhteellisen hyvin, jos ajettavat sovellukset ja niiden vaatimukset tiedetään etukäteen. Ylikuormitus tilanteessa nämä eivät kuitenkaan enää toimi, sillä silloin alemman prioriteetin sovellukset eivät enää saa ollenkaan suoritinaikaa. Lisäksi jos ohjelmat saavat itse valita prioriteettinsä, niin ne valitsevat yleensä hieman korkeamman prioriteetin kuin niille kuuluisi, koska silloin ohjelmat tuntuvat toimivan nopeammin verrattuna muihin vastaaviin ohjelmiin. Nämä skeduloijat eivät juuri pysty antamaan takuita, jos ajettavat sovellukset eivät ole ennalta tiedossa.

Suorittimen varaus: Näillä skeduloijilla voidaan varata tarvittava määrä suoritinaikaa. Jos sovelluksen tarvitsema suoritinaika vaihtelee paljon, niin joko suorittimen käyttöaste laskee, tai skeduloijan antamat takuut heikkenevät. Joskus sovelluksen tarvitsema suoritinaika on vaikea arvioida etukäteen, sillä siihen vaikuttaa mm. suorittimen teho. Varattu suoritinaika on taattu useimmissa tämän tyypin skeduloijissa.

Suhteellinen osuus: Näiden skeduloijien käyttö vaatii vähemmän tietoa kuin suorittimen varaus —skeduloijien käyttö. Nämä eivät kuitenkaan voi taata kovien reaaliaikasovellusten tarvitsemia vaatimuksia. Nämä sopivat vain sovelluksille, jotka toipuvat hyvin vähempään suoritinaikaan kuin ne maksimissaan haluaisivat.

Aikaisin deadline ensin: Nämä skeduloijat toimivat hyvin koviin reaaliaikavaatimuksiin, sillä nämä löytävät sellaisen sovellusten suoritusjärjestyksen, jossa kaikki takuut pysyvät voimassa, jos sellainen on olemassa. Nämä vaativat pääsyvalvontaa, sillä ne toimivat huonosti ylikuormitustilanteessa.

Palautteeseen pohjautuvat: Palautteeseen pohjautuvat skeduloijat eivät toimi hyvin koviin reaaliaikavaatimuksiin, mutta hakevat palautteen perusteella parhainta mahdollista kompromissia korkean suorittimen suoritusasteen ja vähäisen saavuttamattomien sovellusten deadline-vaatimusten välille. Palautetta voidaan käyttää myös pääsyn valvonnassa. Takuut hieman huonot, sillä vaatimuksia kuitenkin rikotaan.

Hierarkinen skedulointi: Tähän tyyppiin kuuluvat suorittimen perintä skeduloija, jossa suorituksessa oleva prosessi voi valita seuraavaksi ajettavan prosessin, sekä useista eri skeduloijista koostuva järjestelmä. Suorittimen perintä antaa vastuun skeduloinnista käyttäjän prosesseille, jolloin käyttäjän prosessit ovat vastuussa takuiden säilymisestä. Tämän avulla voidaan kuitenkin rakentaa vanhan käyttöjärjestelmän päälle skeduloija, joka pystyy antamaan takuita. Useasta skeduloijasta koostuva hierarkinen skeduloija tukee hyvin taas erilaisia sovelluksia. Järjestelmän skeduloijat voivat antaa erilaisia takuita sovelluksille, ja sovellusten vaatimusten perusteella voidaan valita sopivin skeduloija.

4.1 Reaaliaikaskedulointi

Tässä kappaleessa esitellään kaksi perinteistä reaaliaikaskeduloijaa ja vertaillaan niitä hieman keskenään [StN95]:

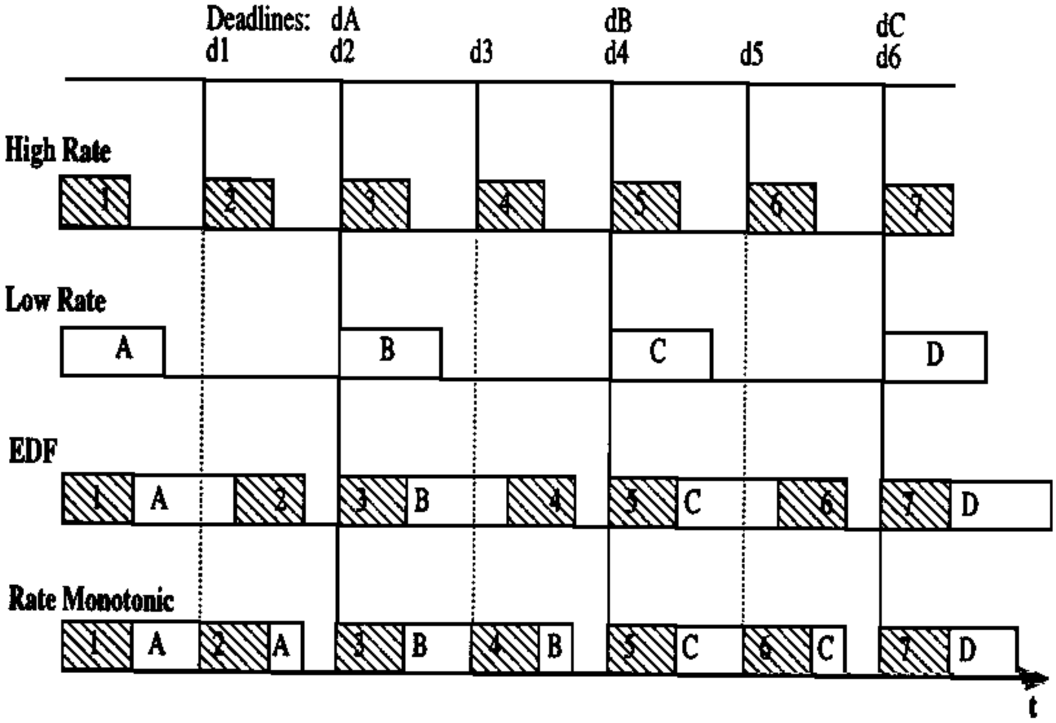
EDF (Earliest Deadline First) skeduloija toimii nimensä mukaisesti valitsemalla suoritukseen prosessin, jonka deadline umpeutuu seuraavaksi. Se on optimaalinen siinä mielessä että, jos sovellukset pystytään skeduloimaan, niin että vaatimukset toteutuu, niin tämä skeduloija onnistuu siinä. Ylikuormitustilanne rikkoo useita vaatimuksia, sillä skeduloija yrittää toteuttaa vaatimukset jokaisen prosessin kohdalla, vaikka se ei olisikaan mahdollista, ja tämä hidastaa myös muiden prosessien suoritusta.

RM (Rate Monotonic) skeduloija priorisoi prosessit niiden periodin mukaan siten, että lyhinperiodisin prosessi saa suurimman prioriteetin. Tämä skeduloija antaa mahdollisuuden prioriteettipohjaiselle järjestelmälle tukea periodisia reaaliaikasovelluksia. Ylikuormitustilanteessa RM skeduloija säilyttää takeet lyhytperiodisten prosessien kohdalla, mutta rikkoo pitkäperioidisten prosessien takeet toistuvasti.

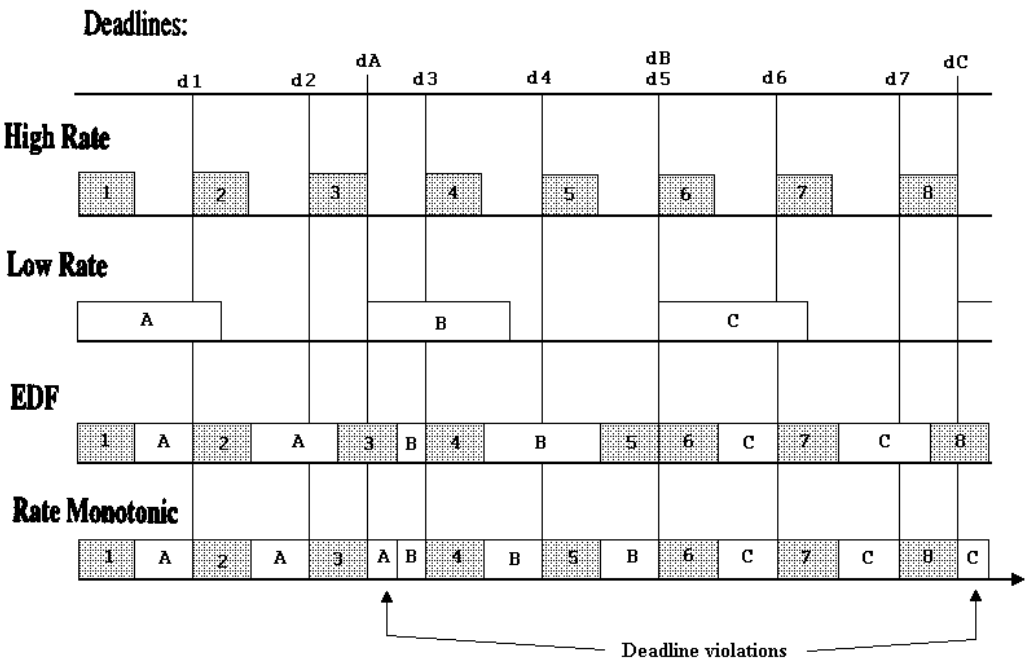
Kuvassa 3 vertaillaan näiden kahden skeduloijan toimintaa. RM skeduloija aiheuttaa keskimäärin hieman useampia prosessin vaihtoja, mikä hidastaa järjestelmää, sillä ne vievät jonkin verran suoritusaikaa.

Kuvassa 4 on vertailtu skeduloijien toimintaa, kun suorittimen käyttöaste on tasan 100%. Kuvassa oletetaan, että skeduloijan toiminta ja prosessin vaihto eivät kuluta suoritinaikaa. Kuvasta nähdään, että RM aiheuttaa tällöin prosessille annettujen takeiden rikkoutumiseen, kun taas EDF suoriutuu skeduloimisesta kunnialla ilman virheitä. RM skeduloijalla ei voidakaan saavuttaa niin korkeaa suorittimen käyttöastetta kuin EDF skeduloijalla ilman takeiden rikkoutumista.

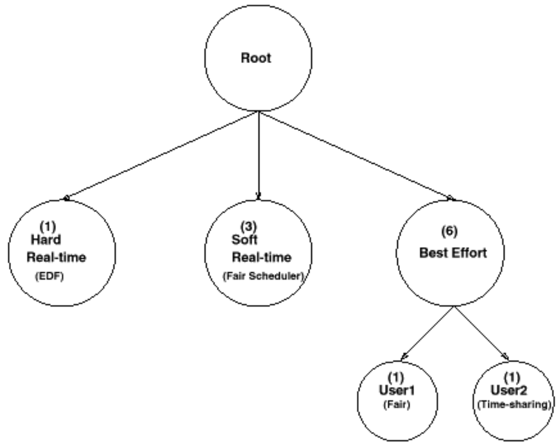
Kuva 3: EDF:n ja RM:n toiminta [StN95].



Kuva 4: EDF ja RM korkealla suorittimen käyttöasteella [StN95].



Kuva 5: Skedulointipuu [GGV96].



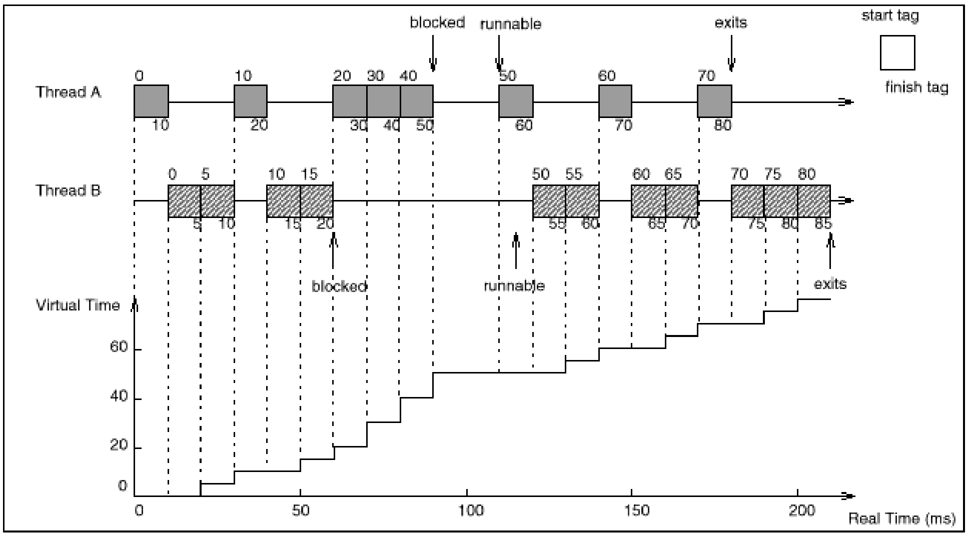
Pehmeiden reaaliaikasovellusten tarpeisiin molemmista skeduloijasta on versiot, jotka sallivat prosessien sisältää taatun pakollisen osion lisäksi vapaaehtoisen lisäosan, jonka suoritus voidaan jättää ylikuormitustilanteessa väliin.

4.2 Hierarkinen skedulointi

Hierarkinen skedulointi [GGV96] sopii hyvin erilaisten multimediasovellusten skeduloimiseen. Siinä skedulointirakenne on puu, jossa lehtisolmuina ovat varsinaiset prosessit. Sisäsolmuina on erilaisia skeduloijia, jotka saavat vanhemmiltaan painoarvonsa verran suoritinaikaa. Kuvassa 5 on esimerkki skedulointipuusta ilman lehtisolmuina olevia prosesseja.

Suoritinaika jaetaan sisäsolmuissa oleville skeduloijille reilulla SFQ (Start-time Fair Queuing) -algoritmilla. SFQ on reilu siinä mielessä, että sen skeduloimat prosessit (tai toiset skeduloijat) saavat suoritinaikaa mahdollisimman tarkasti niiden painoarvojen suhteessa. Tavalliset prioriteetteihin perustuvat algoritmit eivät ole yleensä näin tarkkoja reiluuden suhteen. Hierarkisessa skeduloinnissa tämä reiluus on erittäin tärkeää, jotta voidaan taata esim. kovien reaaliaikasovellusten skeduloijalle

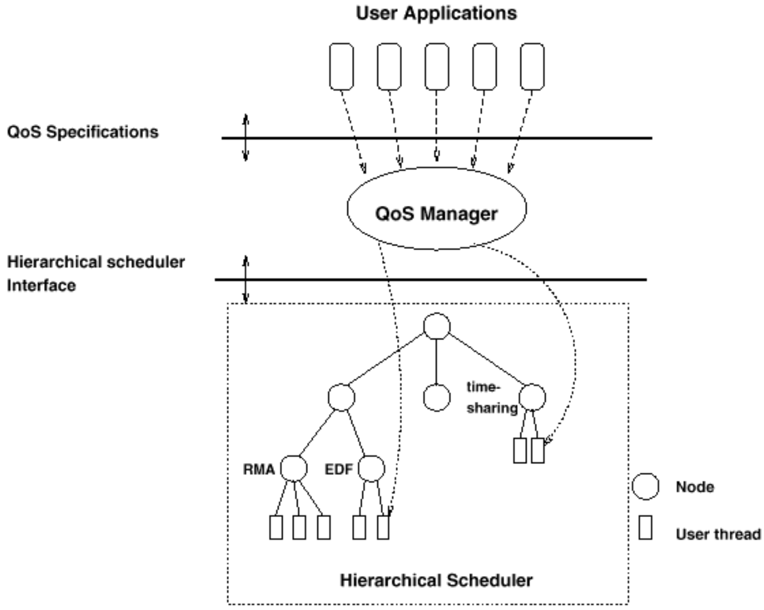
Kuva 6: SFQ-algoritmin toiminta [GGV96].



tietty osuus kokonaissuoritinajasta ja sitä kautta sitä skeduloijaa käyttäville sovelluksille heille luvattu suoritinaika. SFQ skeduloija pitää myös huolen siitä, ettei yksikään järjestelmän skeduloija pysty käyttämään enempää suoritinaikaa kuin sille kuuluu, jos muillakin skeduloijilla on suoritettavia prosesseja. Jos jollain skeduloijalla ei ole yhtään prosessia, niin sille varattu suoritinaika jakautuu muille saman tason skeduloijille niiden painoarvojen suhteessa.

Kuvassa 6 on kuvattuna esimerkki SFQ-algoritmin toiminnasta. Sen toiminta perustuu prosessien (tai sisänoodin skeduloijien) suorituksen alku- ja loppuaikaleimoihin, sekä virtuaaliseen aikaan, jota kasvatetaan pitämään kirjaa järjestelmän kulunutta aikaa yksittäisen prosessin kannalta. Suoritinaika jaetaan sopivan pituisiin paloihin, jotka jaetaan prosesseille. Kuvassa käytetään 10 ms mittaisia suoritusjaksoja. Prosessien alkuaikaleimojen perusteella valitaan suoritukseen prosessi, joka sillä hetkellä on edennyt vähiten. Virtuaalinen aika mittaa sitä, kuinka pitkälle kaikki suorituskelpoiset (runnable) prosessit ovat edenneet. Jos joku prosessi odottaa jotain ulkopuolista signaalia blocked-tilassa, niin virtuaalisen ajan kasvamiseen riittää muiden prosessien eteneminen. Prosessin alkuaikaleima on aina maksimi virtuaalisesta

Kuva 7: Hierarkinen skedulointi ja palvelun laatu [GGV96].



ajasta ja prosessin edellisestä loppuaikaleimasta. Tämä johtaa siihen, että kun prosessi tulee jälleen suorituskelpoiseksi, sen alkuaika onkin sen hetkinen virtuaaliaika, eikä prosessi saa odottaessaan menetettyä aikaa takaisin (kun taas useissa prioriteettipohjaisissa järjestelmissä suositaan saman prioriteetin prosesseista sitä, joka on saanut vähiten suoritinaikaa viime aikoina). Loppuaikaleima lasketaan lisäämällä alkuaikaleimaan suoritusajan pituus jaettuna prosessin painoarvolla. Kuvassa prosessien A ja B painoarvot ovat 1 ja 2. Kuvasta nähdään, että silloin kun molemmat prosessit ovat suorituskelpoisia, prosessi B saa kaksi kertaa enemmän suoritinaikaa kuin prosessi A.

SFQ skeduloijaa käytetään siis jakamaan suoritinaikaa alemman tason skeduloijille, mutta sitä voidaan käyttää myös eräänä varsinaisena prosessien skeduloijana. Hierarkinen skedulointijärjestelmä sijoittaa prosesseja skeduloijille, jotka vastaa niiden tarpeisiin. Katso kuva 7. Samalla voidaan suorittaa mm. pääsyn valvontaa estämään järjestelmän ylikuormittuminen. Hierarkinen skeduloija on erittäin joustava skedulointimenetelmä multimediakäyttöjärjestelmän tarpeisiin.

5 Levypyyntöjen skedulointi

Tässä luvussa esitellään levypyyntöjen skedulointimenetelmiä. Useimmat näistä on tehty ajatellen magneettisia levyjä, kuten kovalevyä varten. Samoja tekniikoita voi kuitenkin käyttää myös optisten levyjen kanssa, mutta usein paljon hitaammat hakuajat johtavat siihen, ettei optisilta levyiltä haeta useita reaaliaikaisia datavoita yhtäaikaa. Optisten levyjen kanssa tuetaan usein ainoastaan limittäin talletettujen datavoiden (kuten esimerkiksi kuvadatan ja halutun äänidatan) yhtäaikaista lukua DVD-levyltä, jossa on limittäin talletettuina useita ääni-, kuva- ja tekstitysvaihtoehtoja.

Seuraavassa listassa kuvataan lyhyesti perinteiset levypyyntöjen skedulointimenetelmät: [StN95]:

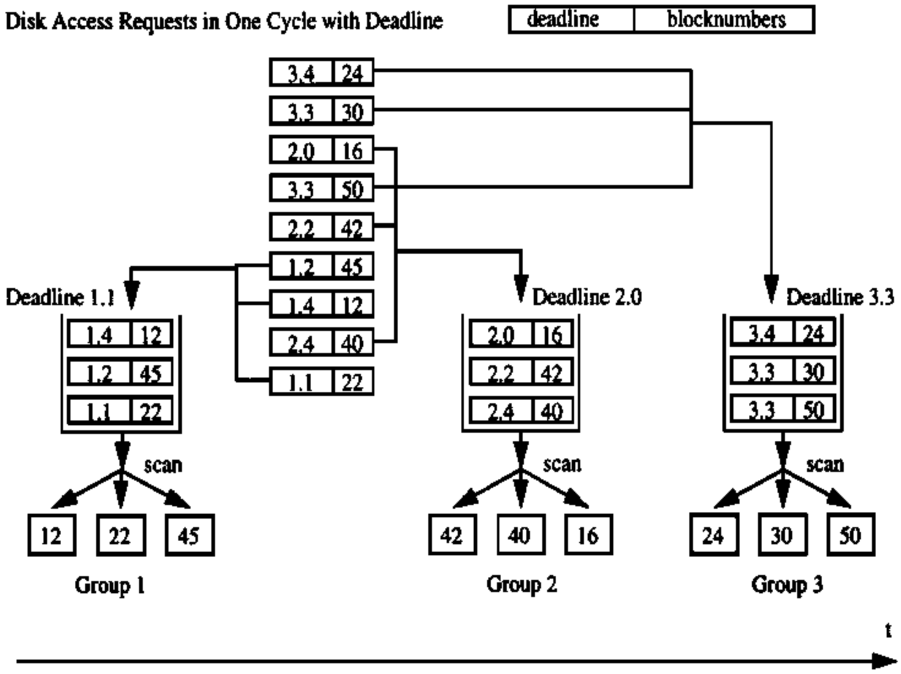
FCFS (First-Come-First-Served) suorittaa pyynnöt siinä järjestyksessä, missä ne saapuvat. Keskimääräinen hakuaika on aika korkea, joten FCFS ei ole kovin tehokas menetelmä.

SSTF (Shortest-Seek-Time-First) palvelee lyhimmän hakuajan pyynnöt ensin. Keskimääräinen hakuaika jää pieneksi, mutta pyynnöt levyn reunoilta saattaa jäädä kokonaan palvelematta.

SCAN järjestää pyynnöt siten, että hakuvarsi liikkuu kerrallaan vain toiseen suuntaan, kunnes siltä levyn puolelta ei ole enää pyyntöjä. Tämän jälkeen varsi vaihtaa suuntansa kohti toista levyn reunaa. SCAN palvelee myös levyn reunat varmasti, mutta silti levyn keskiosa tulee palvelluksi keskimäärin nopeammin.

C-SCAN toimii kuten SCAN, mutta ei palvele pyyntöjä varren liikkuessa takaisin päin. Keskimääräinen hakuaika kasvaa verrattuna SCAN-menetelmään, mutta keskimäärin pyynnöt mistä tahansa levyltä saavat yhtä nopean palvelun.

Kuva 8: Group sweeping —skedulointi [StN95].



Seuraavat menetelmät sopivat multimediapyynnöille, joilla on aikarajavaatimuksia [StN95]:

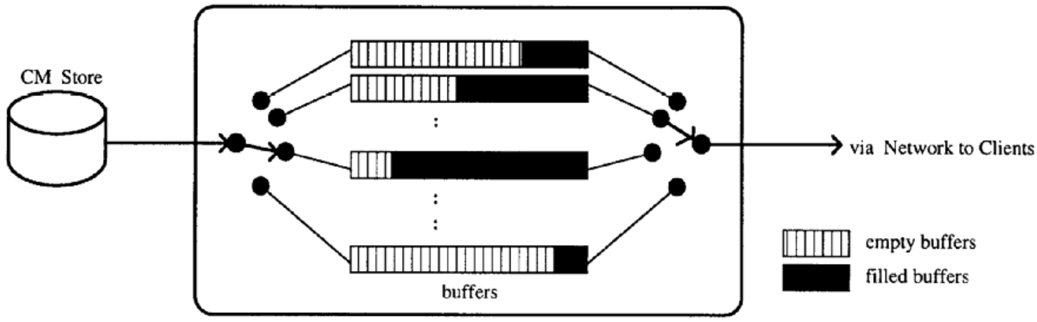
EDF palvelee pyynnöt deadline järjestyksessä, eikä ole kovin tehokas.

SCAN-EDF suorittaa pyynnöt deadline järjestyksessä, mutta suorittaen SCAN-menetelmän mukaista hakuvarren optimointia pyynnöille, joilla on sama deadline.

GSS (Group Sweeping Scheduling) järjestää pyynnöt deadlinen mukaan ja ryhmittelee ne mahdollisuuksien mukaisesti. Ryhmän deadline on siellä olevien pyyntöjen minimi. Ryhmän sisällä noudot suoritetaan SCAN-menetelmää käyttäen. Katso kuva 8.

Usein multimediajärjestelmissä kohdattu ongelma on suorittaa levynoutoja pitämään useiden jatkuvien mediavirtojen puskurit täynnä. Katso kuva 9. Ongelma

Kuva 9: Mediavirtojen puskuroiminen [ToH99].



on lukea jokaista mediavirtaa kerrallaan niin paljon, että kaikkia muita virtoja ehditään palvelemaan ennen kuin juuri palveltu mediavirta tarvitsee uutta noutoa. CMFS (Continuous Media File System) skedulointi on kehitetty tämän kaltaiseen ongelmaan. Se suorittaa mediavirtojen palvelua siinä järjestyksessä, että sitä mediavirtaa palvellaan, jonka puskuri tyhjenisi nopeiten ilman palvelua. Varsinainen datan määrä puskureissa ei ole tässä ratkaiseva tekijä, vaan päätökseen vaikuttaa siis myös se, kuinka nopeasti dataa viedään puskureista eteenpäin.

CMFS skeduloinnissa on pakollisten siirtojen välissä olevan luppoajan käytölle pari eri vaihtoehtoa [StN95]. Luppoajalla tarkoitetaan aikaa, jonka ajan levy voi suorittaa muita kuin varattujen mediavirtojen palvelua ilman, että yhdenkään mediavirran puskuri ehtyy ennen kuin sitä ehditään palvella. Nämä vaihtoehdot ovat:

Ahne-menetelmä laskee luppoajan ennen jokaista mediavirran palvelua. Tämä ylimääräinen aika käytetään kyseisen mediavirran noudon pidentämiseen, kunhan vain sen puskuritila antaa myöten.

Syklinen suunnitelma —menetelmä jakaa luppoajan tasaisesti jokaiselle mediavirralle yrittäen maksimoida koko järjestelmän luppoajan seuraavalla palvelukierroksella.

6 Yhteenveto

Tässä tutkielmassa käytiin läpi erilaisten multimediasovellusten sisältämät reaaliaikavaatimukset. Havaittiin, että useimmat multimediasovellukset sisältävät periodisia toimintoja, mutta että kaikki sovellukset eivät vaadi, että joka periodilla saadaan koko toiminto suoritettua ajoissa.

Multimediakäyttöjärjestelmien keinot sopivien takuiden antamiseksi eri sovelluksille käytiin myös läpi. Sopivimmaksi suorittimen skedulointimenetelmäksi havaittiin hierarkinen skedulointi, jonka avulla voidaan antaa eri sovelluksille erilaisia takuita niiden tarpeiden mukaan. Levypyyntöjen skeduloimisessa oli hieman useampia vaihtoehtoja. Sekä SCAN-EDF ja GSS skedulointimenetelmät olivat hyviä yleisille multimediajärjestelmille, mutta CMFS sopi taas multimediapalvelimelle, jonka lukee levyiltään useita eri mediavirtoja lähettäen niitä asiakkailleen.

Lähteet

BlS98 Gordon Blair, Jean-Bernard Stefani, *Open distributed processing and multimedia*, Addison-Wesley, 1998, Ch 9.

GGV96 P. Goyal, X. Guo, and H.M. Vin, A Hierarchical CPU Scheduler for Multimedia Operating Systems. *Proceedings of Operating System Design and Implementation (OSDI'96)*, Seattle, October 1996, 107–122.

KHS95 Deepak R. Kenchammana-Hosekote, Jaideep Srivastava, I/O scheduling for digital continuous media. *Multimedia Systems*, 5, 4, 1997, 213–237.

PGH00 Thomas Plagemann, Vera Goebel, Pål Halvorsen, Operating system support for multimedia systems. *The Computer Communications Journal*, Elsevier, 23, 3, 2000, 267-289.

RJS00 John Regehr, Michael B. Jones, John A. Stankovic, Operating system support for multimedia: the programming model matters. *Microsoft Research*, MSR-TR-2000-89, September 2000.

Sch96 Henning Schulzrinne, Operating system issues for continuous media. *Multimedia Systems*, 4, 5, 1996, 269–280.

StN95 Ralf Steinmetz, Klara Nahnrstedt, *Multimedia: computing, communications and applications*, Prestice Hall, 1995, Ch. 9.

ToH99 Tsun-Ping J. To, Babak Hamidzadeh, Dynamic real-time scheduling strategies for interactive continuous media servers. *Multimedia Systems*, 7, 2, 1999, 91–106.