Tentamen i 2G1520 och 2G1504 Operativsystem tisdag 2006-12-19 kl 1500-1900

Institutionen för Mikroeletronik och Informationsteknik

Examinator: Robert Rönngren

Hjälpmedel: Inga
Tentamensfrågorna behöver inte återlämnas efter avslutad tentamen.

Ange på omslaget vilken kurs du tenterar (2G1520, 2G1504 eller äldre kurs) och vilken termin du läste kursen.

Varje inlämnat blad skall förses med följande information:
- Namn och personnummer
- Nummer för behandlade uppgifter
- Sidnummer

Rättning:
- ***Alla svar, även på delfrågor, måste ha åtminstonde en kortfattad motivering för att ge poäng***
- Resultat beräknas anslås senast 070116

Betygsgränser:
- Godkänt (3) garanteras från 50% av totala poängantalet på tentamen
- Student som ligger inom 1p från godkänt gränsen äger rätt att inom 14 dagar från det att tentamensresultaten anslagits via Email begära komplettering som om nöjaktigt utförd kan höja betyget till G

Lösningsförslag: anslås på kursens webbsida dagen efter tentamen

---

Frågor: (som normalt bara kräver kortfattade svar 1-10 rader)

1) Ge ett starkt skäl till varför Operativsystmet inte bör implementera `malloc()/free()/realloc()` utan kan/bör tillhandahålla ett så enkelt/primitivt gränssnitt för minnesallokering på heapen som `brk()`! (1p)

malloc/free/realloc implementerar alltid någon form av policy, t.ex optimering för snabbhet eller för minnesåtgång. vad man har behov av beror på tillämpningen och t.,ex hur stort primärminne som finns tillgängligt. Alltså - eftersom det är en bra strategi att skilja policy från mekanism bör OS:et bara implementera mekanismen - dvs tillhandahålla en mekanism för att göra mer minne tillgängligt på heapen med t.ex brk()

Det är också troligt att malloc/free/realloc anropas ofta från användarprogrammen för att allokera små enheter minne. Att då ha dessa som systemanrop med tillhörande kostbara context-switchar är inte bra ur prestandasynpunkt.

Bägge dessa faktorer talar starkt för att malloc/free/realloc bör implementeras som bibliotek på användarnivå.

2) I de flesta OS med schemaläggning baserad på prioriteter sätter användaren en grundprioritet för processen. Ändå kan samma process tidvis exekvera med annan prioritet. Förklara i vilka fall detta kan förekomma och varför det är så! (2p)

De två mest frekvent förekommande exemplen i normala operativsystem där användaren sätter en grundprioritet är (i realtids OS är det vanligt att prioriteten sätts direkt av schemaläggningsmekanismen utan att användaren sätter en grundprioritet t.ex med EDF eller RMS, vilket inte var vad som efterfrågas i uppgiften):

i) när en process övergår till att exekvera i kärnan, t.ex om den gör ett systemanrop, har den oftast högre prioritet för att den skall bli klar snabbare
ii) vissa OS prioriterar vissa typer av processer. vanligt är att man t.ex prioriterar interaktiva processer som kan få sin prioritet tillfälligt höjd i samband med I/O som t.ex i flera av Microsofts OS eller mer permanent höjd prioritet genom en form av aging som i t.ex LINUX

3) a) En process har 15 KB stack, 21 KB data och 6 KB text. Om sidinelat minne används och en sida är 4KB stor - hur många sidor används då för processen? (1p)

b) Hur mycket minne tar den del av en två-nivåers sidtabell som hålls i minne för processen ovan om en adress är 48 bitar, 18 bitar används för att adressera den yttre sidtabellen och en post i en yttre/innre sidtabell är 64 bitar lång? (1p)

a) 4 sidor för stacken, 6 sidor för data och 2 sidor för stacken = 12 sidor. Notera att heap och text segmenten inte delar på sidor då dessa har olika skydd (**rw** resp **rx**).
b) En yttre sidtabell om $2^{18}*2^3$ B + två inre sidtabeller som vardera kan adressera 2^{18}*4KB (dvs en för stack + en för data+text) $2*2^{18}*2^3$ B = $3*2^{21}$ = 6 MB

4) Kalle skall implementera sidtabellen i ett nytt operativsystem med en adressrymd om 48 bitar. Designalternativen han överväger är en flernivåerssidtabell med 3 nivåer eller en inverterad sidtabell implementerad med hjälp av hashning. Att beräkna hashfunktionen tar tid motsvarande en minnesaccess. Hur många sidor kan man tillåta mappas till samma värde i hashtabellen i medel för att den inverterade sidtabellen skall få bättre prestanda vad avser effektiv minnesaccesstid? (1p)

En tre nivåers sidtabell genererar en overhead på tre extra minnesaccesser. För den hashade sidtabellen så kan man tillåta att fler än ett värde (sidnr(+pid)) hashar till samma värde (post i tabellen). Om mer än ett vårde hashar till samma post så får man en lten länkad lista att söka i vid den posten. Antar man att sökningarna är lika fördelade så får man i snitt två minnesreferenser extra om listan inte är längre än 3 poster i snitt ((1+2+3)/3) (I snitt får man traversera (N+1)/2 element i en lista med N element). I praktiken går det också åt lite tid för jämförelser etc. om man söker i listan. Så svaret blir <= 3.

5) a) Varför vill man i operativsystem med sidindelat minne hålla en lista med lediga ramar som inte tillåts bli tom? (1p)
b) Hur implementeras detta? (1p)

a) För att snabbt hitta en ledig ram att ladda in en sida i för processer som får sidfel (dvs som behöver ladda in en ny sida för sin fortsatta exekvering).
b) Den här uppgiften implementeras av en sk paging deamon som tar allokerade ramar från processer enligt någon algoritm/mekanism som kan innefatta en strategi för hur ofta - med vilken frekvens detta skall göras t.ex styrt av hur få/många lediga ramar som finns i listan och någon mekanism (jfr "page-replacement") för vilken sida som skall tas från vilken process (att svara att listan implementeras som en länkad lista eller bitmap ger inga poäng)

6) Vilka fördelar vinner man om sidstorlek och blockstorleken i filsystemet är desamma? (1p)

Den huvudsakliga vinsten är att man kan ha samma storlek på buffertar/cachar i filsystem och i minnessystem, dessutom kan man kan förenkla hanteringen av swap arean (men swap arean implementeras inte alltid i det/de vanliga filsystemet pga effektivitetsskäl).

7) a) Förklara hur allokering/implementation av filer mha FAT respektive i-noder fungerar (rita figurer) (2p)
b) Vilken teknik lämpar sig bäst för implementation av databaser? (0.5p)

a)

FAT: en fil implementeras som en länkad lista av block. länkarna ligger inte i diskblocken utan i en separat tabell i minnet.
i-node: är en datastruktur som dels innehåller information om filen själv, som t.ex accessrättigheter, skapande tid, ägare mm. men också en liten tabell med direkta länkar till till data block, och för större filer pekare till en tabell med pekare till tabeller med pekare till datablock, och för stora filer ytterligare en pekare till en tabell med pekare till tabeller med pakare till tabeller med pekare till data block (triple-indirect) jfr flernivåers sidtabeller.
(figurer som krävs för fullpoäng utelämnade)

b)
i-noder gör att man enklet kan indexera sig fram till godtycklig position i filen - vilket är överlägset jämfört med att behöva söka genom en länkad list av datablock - något som är viktigt vid för prestanda vid "random access" i stora filer som t.ex en stor databas. (hur stora filer som kan implementeras via FAT eller i-noder beror på den spcifika implementationen, teoretiskt finns inga begränsningar på filstorleken för endera tekniken)

8) När man gör konsistenskontroller på filsystem kan man göra "block konsistens" och/eller "filsystems konsistens" kontroller. Vad kontrollerar man i de två fallen och vilka typer av fel kan man upptäcka? (2p)

Block konsistens: ett block får bara förekomma en gång i antingen en fil, i listan med dåliga block eller i listan med fria block. Om ett block finns i fler än en av dessa så har man ett allvarligt fel i filsystemet - finns blocket både i en fil och i listan med dåliga block har man en fil som antagligen inte går att läsa, finns blocket i både dåliga block listan och i fri-listan bör blocket kontrolleras och föras till bara den ena listan beroende på om det är OK eller ej, finns blocket både i en fil och i fri-listan bör blocket tas bort från fri-listan.
Filsystemskonsistens: man kontrollerar att det finns så många refrenser till en fil från directoryna som räknaren i dess i-nod anger. Finns fler referenser än räknaren anger måste denna uppdateras pss om räknaren är högre än antalet referenser, finns inga referenser kan filen tas bort.

9) Microsofts filsystem tillåter att en fil har flera strömmar av information. Varför? Ge exempel på när det kan vara användbart! (1p)

Det kan underlätta hanteringen av applikationer där man har behov av att spara logiskt skilda strömmar av information som hör till applikationen) Tydliga exempel är multimedia där t.ex. en dataström kan representera ljud och en annan bildiformation eller för dokument där en dataström kan representera informationen i dokumentet och en annan ändringar som är gjorda i dokumentet för att kunna implementera "undo"-funktionalitet på ändringarna

10) a) Beskriv hur I/O med DMA går till! (1p)
b) Vad betyder begreppet "pinning" och varför kan man behöva det vid DMA styrd I/O? (1p)

OS:et (drivrutinen) programmerar upp DMA enheten att hämta/skriva data till/från en area i minnet, normalt hos kärnan. DMA enheten för sedan över data parallelt med att CPUn arbetar med annat. När data är överfört genereras ett avbrott som talar om att överföringen är klar. Avbrottsrutinen tar hand om att uppdatera ev datastrukturer och se till att en ev. väntande process schemaläggs.

Pinning behövs för att låsa fast buffertar/ramar så att innehållet i dem inte slängs ut eller används för något annat medan DMA överföringen pågår vilket skulle kunna inträffa om OS:et t.ex behandlar ett sidfel..

11) Givet följande referenssträng, 1 7 2 4 3 2 7 5 6 2 5 8 7 , tre ramar och ren "demand paging", beräkna antalet sidfel för FIFO, LRU resp. OPT. (1.5p)

Räkningar måste redovisas! Fifo 11, LRU 11, OPT 9

12) Varför kan en flertrådad process ibland exekvera snabbare på en dator med en CPU än motsvarande process som bara utnyttjar en tråd? (1p)

Om man har *kärntrådar kan man få en överlappning av I/O och beräkning*, dvs om en tråd blockerar i väntan på I/O kan kärnan schemalägga en annan tråd som kan exekvera

13) I ett distribuerat filsystem kan man välja att implementera filservern med eller utan tillstånd. Jämför dessa två alternativ vad avser:
a) Storlek på meddelanden (0.5p)
b) Beteende vid serverkrasch (0.5p)
c) Beteende vid klientkrasch (0.5p)
d) Prestanda (0.5p)
e) Säkerhet (0.5p)

a) Större meddelanden om servern är tillståndslös eftersom man måste skicka med information om vilken fil som skall accessas, position i filen, accessrättigheter mm. eftersom den tillståndslösa servern t.ex öppnar och stänger filen för varje läsning/skrivning (den har ju inget tillstånd och därmed heller inte några öppna filer t.ex) Detta behövs inte skickas varje gång om servern håller information om öppna filer

b) Om servern kraschar är detta inget problem varken ur serverns eller klientens sida om servern är tillståndslös, just för att den inte har något tiullstånd, bortsett från att den tappar ev. meddelande kö och jobb den arbetar på. Har servern tillstånd så går det förlorat om den kraschar och efterföljande begäran från klienter med filer som var öppna i den gamla servern kan inte betjänas. Ev. skulle man i fallet att man har en server med tillstånd kunna försöka återskapa ett ev. tillstånd före krasch men det är normalt ingen större ide eftersom man kan ha tappat bort klienters begäran som var köade (och därmed ser tillståndet kanske ändå inkonsistent ut för klienten) och klienterna kanske hinner upptäcka att servern kraschat och försökt starta om och är ändå inte intresserade av ett gamalt tillstånd.

c) Med en tillståndslös server är det helt problemfritt om en klient kraschar eftersom den inte "kommer ihåg" något alls om klienterna. Har servern tillstånd måste den på något sätt upptäcka att t.ex filer som öppnats av en klient som kraschat inte längre används, dvs att dessa filer inte ligger kvar öpnna i servern vars tillstånd då kan bli felaktigt

d) En tillståndslös server måste öppna/stänga filerna vid varje access, meddelandena är längre och man kan inte göra optimeringar som t.ex read-ahead- den har därför oftast lite sämre prestanda än en server med tillstånd (att belastningen på nätet blir större med en tillståndslös server har redan klarats av i a) och ger inte poäng på den här deluppgiften)

e) En tillståndslös server bör normalt vara lite säkrare, under förutsättning att man kan skydda accessrättigheterna som skickas i meddelandena. Då kan den anses vara lite säkrare eftersom den alltid måste kontrollera accessrättigheterna vid varje operation och därmed minskas risken att man får göra operationer på filer där rättigheterna ändrats.

14) Moderna datorarkitekturer har ofta fler än en processorkärna på samma chip.
    a) Vad är motivet att man numera ofta väljer flera processorkärnor istället för en enda? (0.5p)
    b) Vad betyder begreppet false sharing? och hur påverkar det prestanda och programmerarens arbete? (1p)

Fysikaliska begränsningar gör att det är svårt att med dagens halvledarteknik bygga enskilda processorer som blir snabbare, bland annat kan man inte enkelt klocka kretsarna så mycket snabbare med dagens teknik än vad som görs idag. Däremot ökar möjligheterna att få in mer kretsar/grindar på ett och samma chip - något som kan utnyttjas till att bygga fler processorer på samma chip och därmed öka beräkningskapaciteten på samma chip. Dock bör man komma notera att det inte nödvändigtvis är en enkel väg att gå eftersom man får mer problem med schemaläggning och synkronisering - något som kan reducera en de vinster man kan nå.

False sharing betyder att data som accessas av olika processorer hamnar på samma cachelinje i en delad minnesarkitektur med cachning. Cache-konsistensprotokollen kan då i onödan invalidera cachad data vid skrivningar - något som påverkar programmets exekveringstid negativt och som kan göra att det blir svårare att programmera om man måste kunna kontrollera var i minnesrymden data läggs ut.

15) Dining philosophers är ett problem som kortfattat går ut på att det runt ett bord finns N sittplatser och en gaffel mellan varje par av sittplatser (dvs N gafflar) För att en filosof skall kunna äta spagetti krävs två gafflar.
    a) Beskriv varför/när/hur deadlock kan uppstå i exemplet! (1p)
    b) Ge två lösningar på hur deadlock kan undvikas och bevisa att de är korrekta (2p)

Om N filosofer samtidigt sätter sig till bords och tar upp sin högra gaffel *uppfylls samtliga coffmans villkor* för deeadlock. De: i) äger sin gaffel exklusivt, de håller en gaffel medan de väntar på den andra (vänstra), de väntar samtliga på någon annan filosof (cirkulär väntan) och det finns ingen som kan ta gafflar från dem (no preemption)

Lösningar: numrera alla gafflar och ta dem i nummerordning, bryter cirkulär väntan, tillåty bara N-1 filosofer att sätta sig samtidigt (cirkulär väntan), inför en hovmästare som kan ta gafflar från filosofer vid behov för att bryta deadlock (no-preemption).....

Lösningar som onödigt reducerar parallellismen, som att bara tillåta en filosof att äta åt gången, ger inte full poäng!

16) Varför bör man om möjligt använda "analytiska metoder" snarare än vanlig debuggning när man letar efter synkroniseringsproblem? (1p)

Synkriniseringsproblem är oftast schemaläggningsberoende och därmed svåra att återskapa. Speciellt kan man notera att en debugger nästan oundvikligen kommer att påverka schemaläggningen. Därför bör man använda analytiska metoder för att i) kunna undvika synkroniseringsproblem genom att designa systemen så att de inte kan uppstå; ii) om de ändå uppstår är det bättre att analysera koden än att försöka återupprepa ett synkroniseringsproblem som kanske inte går att upprepa inom rimlig tid

17) Vad gör det mer riskabelt att ladda hem och installera en drivrutin än ett vanligt program? (1p)

Den laddas in i kärnan och får därmed "kernel-mode" rättigheter automatiskt vilket inte andra "vanliga" program får

18) Kan en virusscanner garantera att ett system inte drabbas av virus? Varför? (1p)

Nej, den kan aldrig känna till/upptäcka alla nya varianter på virus och den kan inte varna för alla potentiella virus eftersom en del legitim kod liknar viruskod.

19) Vilken typ av realtidssystem (hårda eller mjuka) kan stödas/stöds av vanliga operativsystem som Microsofts eller LINUX/UNIX? Varför? (1p)

Mjuka eftersom man normalt inte kan garantera att deadlines möts i schemaläggningen då man har t.ex virtuellminne, oftast inte möjligheter att undvika prioritetsinversion, avbrott, filsystem, schemaläggning som känner till och kan ta hänsyn till deadlines mm.

20) Hur många rader kod (C-kod) kan man förvänta sig hitta i ett Microsoft OS respektive ett UNIX/LINUX OS? (0.5p)
Vad beror skillnaden på? (0.5p)

C:a 30 Miljoner rader kod i ett MS OS respektive 2-3 miljoner rader kod i vanliga LINUX/UNIX implementationer

Skillnaden beror till stor del på att Microsoft valt att integrera många delar i OS:et, som i UNIX/LINUX är separata servrar/processer i user-mode, som t.ex grafik, webbläsare mm.

21) Kalle hävdar att alla IPC-problem som kan lösas med hjälp av signaler/signalhanterare lika väl kan lösas med hjälp av `wait()/waitpid()`. Stämmer detta? (1p)

Nej, det är två skilda saker. Med wait() kan man läsa av status för en process men inte skicka signaler eller ta emot signaler - vilket man kan göra med signaler/signalhanterare. som däremot inte kan läsa av status för en process.....

"Alla morgondagens blommor gömmer sig i de frön som sås idag"
Kinesiskt ordspråk