Tentamen i ID2206/ID2200/IS1350 Operativsystem
(fd 2G1520/2G1504/6B2019)

Onsdagen 2009-12-16 kl 0900-1300

Institution: ECS/ICT
Examinator: Robert Rönngren
Hjälpmedel: Inga
Tentamensfrågorna behöver inte återlämnas efter avslutad tentamen.
Ange på omslaget vilken kurs du tenterar och vilken termin du läste kursen.

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

ID2200 (fd 2G1504)
- Studenter som tenterar ID2200 besvarar inte frågor markerade med *****

IS1350 (fd 6B2019)
- Studenter som tenterar IS1350 besvarar inte frågor markerade med **¤**

Rättning:
- ***Alla svar***, även på delfrågor, måste ha åtminstonde en kortfattad motivering för att ge poäng
- Resultat beräknas finnas rapporterat i LADOK och tentorna tillgängliga via studentcentrum senast 18/1

Betygsgränser:
- Godkänt (E) garanteras från 50% av den maximala poängen för respektive kurs

Lösningsförslag: anslås på kursens webbsida efter tentamen

Frågor:

---

1) Du har blivit övningsassistent i OS kursen. En student har lämnat in en labb med koden nedan.
   a) förklara vad studenten försöker göra (1.5p)
   b) är detta en bra ide? (0.5p)

```
atexit(cleanup);  // En rad i main()

void cleanup(void)
{
    int i;
    for(i=0; i<NR_OPEN;i++) close(i); // förklara
    brk(_end); //förklara
    do waitpid(getpid(),&i,WNOHANG); while(!WIFEXITED(i)); //förklara
}
```

a) Studenten försöker uppenbarligen avallokera och rensa upp resurser knutna till processen. for-loopen stänger alla öppna filer - detta är den enda åtgärden som möjligen skulle kunna vara av nytta då innehållet i ev. buffrar skrivs tillbaka till disk - men detta sker ändå normalt vid en kontrollerad terminering av processen (dvs då _exit() körs). brk(_end) flyttar tillbaka brytpunkten för stacken till direkt efter datareorna "data" och "bss" - dvs det skulle "avallokera" saker allokerat på heapen - detta görs helt i onödan då processens adressrymd upphör att existera då processen avslutas då OS:et tar tillbaka både fysikt minne i primärminnet (ramar) och swapspace på disk. do-while satsen är uppenbarligen ett försök att läsa av status för processen då den terminerat för att undvika att processen blir en sk. zombie-process, dvs att den fortfarande tar upp plats i processtabellen. Den satsen blir en oändlig loop eftersom processen exekverar, dvs är aktiv, men samtidigt väntar på att den själv skall terminera.
b) Det är på det hela taget en riktigt dålig ide av skälen ovan

2) a) Vad innebär det att en schemaläggningsalgoritm är preemptiv? (0.5p)
   b) Vilka är fördelar och nackdelarna gentemot icke-preemptiva schemaläggningsmetoder? (1.5p)

a) I det generella fallet - vilket är det som efterfrågas - handlar detta om preemption på processorn. Dvs. att en exekverande process kan avbrytas av OS:et innan den självmant skulle ge upp processorn genom att den exekverat klart sitt tidskvanta eller att den blockerat på någon blockerande operation. Skälet att avbryta processen är normalt att en process med högre prioritet blivit körklar. OBS! detta ger ingen generell lösning på att bryta Coffmans villkor om "non-preemption"

b)Fördelen med preemptiv schemaläggning är framförallt att man kan ha processer med olika prioriteter och att man minskar risken att en högre prioriterad process får vänta någon längre tid på att få köra till följd av att lägre prioriterade processer exekverar. Nackdeln är en ökad komplexitet i schemaläggningen.

3) Som designer av operativsystem måste man alltid fundera på konsekvenserna av att införa olika lösningar. Anta att du inför följande i ett OS:
   i) prioriteter för processer
   ii) olika långa tidskvanta/timeslices per process
   Kan detta leda till problem? Beskriv för de två fallen om och vilka problem som kan uppstå och vilka åtgärder man bör vidta för att minimera problemen. (1p)

a) Som ovan - en nackdel är att man får mer komplex schemaläggning och ofta vill man implementera preemptiv schemaläggning. Dessutom kan man få problem med svältning om det alltid finns högprioriterade processer som tränger ut de lågprioriterade. Det leder till att man får lösa problemet med någon form av aging - antingen att lågprioriterade processer med tiden får ökande prioritet eller att processer som får exekvera mycket med tiden får lägre prioritet
b) om inte tidskvanta blir orimligt långa bör detta inte leda till problem. Möjligen skulle man kunna argumentera för att det t.ex skulle ge dåliga prestanda i interaktiva system om beräkningstunga processer skulle få långa tidskvanta. Detta får man i så fall lösa med att införa prioriteter och preemption.

4) Vad gäller för relationen mellan storleken på fysisk och virtuell adressrymd? (1p)
   i) fysisk <= virtuell
   ii) fysisk = virtuell
   iii) fysisk >= virtuell

Det finns inget samband mellan storleken på virtuell och fysisk adressrymd - det är en av grundideerna med att ha en virtuell adressrymd som är "frikopplad" från den fysiska. T.ex kan man köra ett 32 bitars OS på en maskin med 8 GB minne eller ett 64 bitars OS på en maskin med 1 GB minne, dvs i det ena fallet gäller fysisk > virtuell och i det senare fallet fysisk < virtuell adressrymd. (Bara för att man vanligtvis har en fysisk adressrymd som är mindre än den virtuella kan man alltså inte anta att svarsalternativ i) är rätt.) Att den fysiska minnesrymden kan vara större än den virtuella kan t.ex användas för att låta fler processer helt och hållet rymmas i den fysiska och kan också antas bli vanligt när man tar virtualiseringen ett steg längre och kör på virtuella maskiner. Att rätt svarsalternativ inte finns som ett enskilt alternativ bland de listade i frågan liknar verkligheten där en ingenjör måste våga göra egna bedömningar.

5) Antag att minnessaccesstiden i ett system är **m** tidsenheter för att accessa minnet direkt. Att systemet är sidindelat och använder en TLB där access/söktiden är **t** tidsenheter och att det tar **d** tidsenheter att överföra en sida från disk till minnet.
   a) Ange en formel för den minsta minnsesaccesstid vi kan förvänta oss vid en TLB miss. (1p)
   b) Ange en formel för den genomsnittsliga tiden för att ladda en variabel från en sida i minnet till ett register om vi har 95% träffsannolikhet i TLBn. (1p)

a) Missar vi i TLBn så kommer vi förutom att ha tiden för att accessa minnet för att hämta variabeln att ha söktiden i TLBn plus att vi måste göra en uppslagning i en sidtabell som minst kräver en extra minnesaccess (t.ex i fallet med en en-nivåers sidtabell, andra implementationer av sidtabellen kan ge längre accesstider)). Svar: t+2m
b) Sidan med variabeln finns i minnet - alltså krävs ingen diskaccess. Svar: t + m + 0.05*(uppslagningstid i sidtabellen) dvs minimal tid är t+1.05m

6) Vilken typ av datastruktur för implementationen av sidtabellen kan man förvänta sig hitta i ett 64 bitars operativsystem? En en-nivås, flernivås, inverterad eller hashad inverterad sidtabell? Förklara varför? (1p)

En-nivås är inte rimlig med tanke på storleken. Flernivåers är heller inte rimligt med tanke på antalet nivåer i tabellen. En inverterad sidtabell implementerad med hårdvarustöd i form av associativt minne (CAM) kan fungera - men för fallet att vi inte kan anta att hårdvaran har speciellt stöd för implementationen återstår en inverterad hashad sidtabell som kan implementeras i vanligt minne och mjukvara och ändå få rimlig accesstid.

7) Hur många sidfel får man om man har 4 ramar, ren demand paging och följande refrenssträng:
   0 2 4 7 2 6 4 1 8 4 6 3 2 7 4 6 4
   För algoritmerna OPT, LRU och FIFO. (1.5p)

OPT: 9, LRU: 12, FIFO: 13 (räkningarna måste redovisas)

8) Vilka datastrukturer kan man förvänta sig hitta på en partition i ett filsystem? (dvs. hur organiseras en partition) (1.5p)

En partition innehåller normalt sett:
Ett bootblock med info om hur ett OS på partitionen bootas
Ett superblock eller master file table som talar om filsystemstyp
En lista med lediga block
En lista med dåliga block
Ett root directory
I-noder eller FileAllocationTable

9) a) Vad är en RAID? (1p)
b) Vilka fördelar kan man åstadkomma med RAID system och hur gör man det? (2p)

a) En RAID är vad som för användaren logiskt sett ser ut som en disk men som i själva verket består av flera fysiska diskar styrda av en diskcontroller. Data skrivs/läses till/från en eller flera av diskarna
b) Man kan åstadkomma snabbare diskaccesser och/eller ökad säkerhet/tillgänglighet på data. Skriver man samma data till flera diskar parallellt eller delar upp data och skriver parallellt till flera diskar med felrättande koder kan man klara att återställa data även om en (eller flera) av de fysiska diskarna går sönder. Genom att dela upp data och skriva/läsa till/från flera diskar parallellt kan man få snabbare diskaccesser.

10) Kalle hävdar att DMA alltid är att föredra framför pollning och avbrottstyrd I/O. Stämmer det? (1p)

Nej det stämmer inte. Om man t.ex bara skall överföra enstaka tecken eller inte vill öka komplexiteten i implementationen med avbrottshantering (som ger komplexitet både i hårdvaran och i mjukvaran i form av context switchar) så är avbrottstyrning eller pollning att föredra. Andra exempel kan t.ex vara I/O mot klockor som oftast inte innebär överföring av stora datamängder - snarare vill man ha klockavbrott än klock-DMA.

11) a) Vad är en "hårddiskcontroller"? (1p)
b) En hårddiskcontroller innehåller ofta ett ett ganska stort RAM minne. Vad används det till? (1p)

a) En hårdvaruenhet som används för att styra och kontrollera en eller flera fysiska diskar
b) Minnet kan användas som buffer/cache för read-ahead av data från en fil på disken, för att mellanlagra information som skrivs till disken men som inte fysiskt hinner skrivas ner på diskens lagringsmedia i den takt det kommer till disken, eller för att mellanlagra skrivningar och läsbegäran för att kunna åstakomma en bättre schemaläggningsalgoritm för disken än FCFS

12) ¤På KTH används ett distribuerat filsystem AFS. Beskriv kortfattat hur klient resp. server implementeras och hur cachening i AFS går till. (2p)

Klienten implementeras i kärnan eftersom en användare som vill använda ett filsystem gör detta genom systemanrop (som för över kontrollen till kärnan). Vid ett systemanrop på klientsidan så identifierar man att filen ligger i AFS när man går genom det virtuella filsystemslagret (VFS med v-noder). Servern implementeras som en vanlig server i användarläge baserad på socket-kommunikation. Filerna lagras i det lokala filsystemet på den maskin som kör servern.
På klientsidan cachas filer i det lokala filsystemet. Som konsistenssemantik använder AFS i huvudsak sessionssemantik, dvs att ändringar skrivs tillbaka då filen stängs,

13) Måste man ta hänsyn till synkronisering i ett system där man vet säkert att Coffmans villkor inte kan uppfyllas? (1p)

Ja om något i systemet går parallellt och kan kommunicera via delade resurser som t.ex delade variabler eller meddelandesändning så måste man normalt ta hänsyn till synkroniseringsproblematiken. Tag t.ex ett system där två processer kan accessa samma variabel i minnet - kan bägge både läsa, uppdatera och skriva variabeln måste dessa läsningar och skrivningar synkroniseras för att slutresultatet skall bli korrekt. Gör vi ingen sådan synkronisering kan visserligen inte deadlock uppstå..... dvs om vi inte synkroniserar kan inte Coffmans villkor uppfyllas men systemet beter sig knappast som avsett.

14) I schemaläggning baserad på prioriteter kan i vissa fall svältning uppstå. Beskriv den vanligaste mekanismen man implementerar för att undvika problemet! (1p)

Man implementerar aging dvs att processers prioritet kan minskas om de exekverar (för) mycket eller ökas om de inte får exekvera

15) ¤I dagarna har några av de stora hårdvarutillverkarna annonserat nya processorchip med upp mot 64 kärnor per chip. Detta skapar nya utmaningar för tillverkarna av Operativsystem.
Vilka är de viktigaste problemen att lösa och vilka speciella hänsynstaganden måste göras när man skall schemalägga trådar/processer på chip med många tiotals eller ännu flera processorkärnor? (2p)

För schemaläggning på multi-core system måste man ta hänsyn till när en process skall schemaläggas och var den skall schemaläggas (på vilken processorkärna). Processer som ofta kommunicerar med varandra bör schemaläggas samtidigt och i närheten av varandra (kommunikationsmässigt), processer bör också schemaläggas så att de ligger nära resurser de utnyttjar ofta (I/O, minnesbankar) och om möjligt på samma processorkärna som tidigare för att slippa flytta data och för att ev. kunna återutnyttja cache och TLB information.

16) a) Vad är en trojan? (1p)
b) Varför har det betydelse om det är användaren `root` eller användaren `kalle` som exekverar en trojan? (1p)
c) Skriv i C med POSIX-standard systemanrop en enkel trojan för kommandot `rm` som skall ha den fulla funktionaliteten hos `rm` men som också skall anropa en funktion `fuffens()` (2p)

a) Om vi bortser från att en trojan är en medborgare i i den antika stadsstaten Troja eller referenser till hästen från Troja så handlar det i det här samanhanget om ett program som kan klassas som "malware". Dvs ett program som synbarligen gör vad det skall göra men som också dolt kan göra annat med mer eller mindre onda avsikter
b) ett program som exekveras, dvs en process, körs normalt med samma identitet och rättigheter som den som startade processen, dvs körs det av root kan man anta att det körs med root-rättigheter och har full tillgång till systemet, medan om kalle kör programmet så har det kalles mer begränsade rättigheter vilket naturligtvis har betydelse för vad trojanen kan åstadkomma på systemet.

c)
```
int main(int argc, char **argv)
{
    fuffens();
    execv("/usr/bin/rm", argv);
}
```

Två saker är viktiga att tänka på: i) att parametrarna som angetts på komandoraden skickas med till den riktiga "rm" annars avslöjas direkt att trojanen är en trojan genom ett annat beteende; ii) om man anropar execvp("rm", argv) dvs gör execanropet så att man inte anger hela sökvägen till den korrekta "rm" utan går via användarens PATH så kommer man att anropa den "rm" man hittat - dvs man anropar trojanen rekursivt vilket också avslöjar trojanen direkt

17) Kalle har fått ett virus på sin PC. Han har tänkt städa bort det genom att ta bort alla filer med kommandot `rm -R *` som han kör från roten av filträdet. Efter det skall han lägga tillbaka OS:et och starta om datorn. Är detta ett säkert sätt att bli av med viruset på? (1p)

Att ta bort alla filer som Kalle försöker räcker inte - ett virus kan finnas kvar i form av t.ex ett boot-sektorvirus som inte försvinner om inte boot-sektorn formateras/skrivs över. Dessutom kan viruset finnas kvar i OS:et om inte Kalle stänger av maskinen och bootar från ett säkert media (Dvs ett media som inte innehåller ett virus som t.ex en orginal CD för OS:et)

18) *a) Hur definieras att en realtidsschemaläggning är optimal? (1p)
*b) Om en realtidsschemaläggningsalgoritm är optimal på en dator med endast en processorskärna är den då optimal på en multicore maskin (1p)

a) en realtidsschemaläggningsalgoritm är optimal om den endast misslyckas med att schemalägga ett givet system om det inte finns någon annan algoritm som skulle klarat uppgiften
b) Nej det är inte sant. Visas t.ex med ett exempel där EDF (Earliest Deadline First) är optimal på en processor men inte nödvändigtvis på två (jfr föreläsningsanteckningarna - motbeviset måste redovisas för full poäng)

19) *Om man från ett användarprogram använder en fil som ligger på en hårddisk eller på ett USB-minne så öppnar, läser och skriver man bägge filerna med hjälp av samma systemanrop. Men eftersom filerna ligger på olika I/O enheter så accessas de via olika drivrutiner (device drivers). Förklara hur mekanismerna fungerar som gör att olika drivrutiner används transparent ur användarens perspektiv. (2p)

Det som gör detta transparent är att: **Drivrutinen för en I/O-enhet "installeras"/monteras normalt i filsystemet, dvs när man passerar en monteringspunkt så kopplas man också till vilken drivrutin som**

skall användas. Interfacet mot drivrutinen i sig består av en struktur (i C) som innehåller funktionspekare till funktioner i drivrutinen. Dessa funktionspekare svarar mot systemanropen för att acessa filer (open, read, write.....) och anropas av operativsystemet när systemanropet utförs

20) För att ett system med sidindelat virtuellminne skall fungera bra bör systemet hålla en lista med fria ramar.
a) vad är syftet med listan med fria ramar? (1p)
b) beskriv mekanismerna och ideerna bakom hur listan med fria ramar hanteras i SOLARIS (2p)

a) För att snabbt kunna allokera en ram åt en process som fått ett sidfel och alltså behöver läsa in en sida till minnet - dvs för att snabbare kunna hantera sidfel
b) Solaris använder en paging demon (pageout) och en två-visares clock-algoritm för att bestämma vilka ramar som skall tas från en process. I clock-algoritmen finns en cirkulär lista med de sidor som finns i ramar i primärminnet. Med en variabel frekvens så scannas ett visst antal sidor per sekund. När den första visaren når en sida nollställs dess referensbit och om referensbiten fortfarande är nollställd då den andra visaren når sidan så tas ramen sidan ligger i från processen och sidan pageas ut. Avståndet mellan visarna är ett visst antal sidor (fixt antal, typiskt 1024). Hastigheten med vilken man skannar sidorna i minnet bestäms av genomsnittsligt antal lediga ramar i fri-listan de senaste 30 sekunderna. Har man mycket lediga ramar kan frekvensen gå ned till noll och har man få lediga ramar kan den gå upp till 8K sidor/sekund

21) Vilken är den enskilt viktigaste skillnaden, som ökar säkerhet och tillförlitlighet, i modernare versioner av Windows från de baserade på NT och framåt gentemot MS-DOS? (1p)

Viktigast är att man skiljer på user- och kernel-mode, viktigt är också att olika processers adressrymder (och OS:ets) blir skyddade från varandra och att en process inte får skriva i hela sin adressrymd (inte i text-arean)

"Alla såg äpplen falla men det var bara Newton som frågade sig varför"
B. M. Baruch