Lösningsförslag (så kortfattade att de inte alltid ger fullpoäng) till Tentamen i ID2206/ID2200/IS1350 Operativsystem
(fd 2G1520/2G1504/6B2019)

Torsdagen 2008-05-22 kl 0900 - 1400

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

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

Lösningsförslag: anslås på kursens webbsida efter tentamen (antagligen inte förrän)

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

1) Beskriv och visa i koden vilka dataareor som accessas under exekveringen av följande lilla C-program på ett UNIX/LINUX-system? (2p)

```
int main(int argc, char **argv)
{
  static int four = 4;
  printf("four = %d\n", four);
}
```

Programmet innehåller bara en exekverbar sats - anropet till printf(). Under exekveringen hämtas intruktionerna för att utför anropet till printf() från *textarean*, parametrarna till printf() läggs upp på *stacken*, formatsträngen och variabeln four ligger i *dataarean* och pekaren respektive värdet 4 pekar (adresserar) in i dataarean resp. hämtas därifrån.

2) ¤ Semantiken för läsning från en pipe är att läsning från en tom pipe returnerar filslut endast då det inte längre finns någon process som har pipen öppen för skrivning. Den här semantiken kan leda till synkroniserings-problem som dead-lock om man glömmer stänga pipen i processer där den inte används. Kalle föreslår en enklare semantik: "Läsning från en tom pipe returnerar alltid filslut direkt". Disskutera Kalles förenklade semantik - fungerar den eller inte? (1p)

Kalles ide fungerar inte eftersom den förutsätter att processerna schemaläggs så att den som producerar indata till en pipe alltid måste få exekvera före den som läser - vilket man inte kan garantera i schemaläggningen - dvs. en läsande process kan i Kalles fall läsa filslut om pipen tillfälligt är tom fastän producenten fortfarande kommer att skriva data till pipen.

3) Relokerbarkod är ett begrepp som ofta förekommer i samband med operativsystem.
   a) Vad innebär begreppet (hur implementeras det)? (1p)
   b) Ungefär när uppfanns det och i vad var skälet till att man införde det? (1p)

Relokerbar kod = flyttbar kod, dvs kod som inte har absoluta adresser utan kod som har adresser som är relativa t.ex start av kodulen. De kan laddas på olika ställen i minnet genom att man modifoierar startadressen för där koden laddas.

Begreppet uppfanns omkring 1950 när man kom fram till att det kunde vara praktiskt att kunna återanvända kodmoduler för tex matematiska funktioner men att man då inte alltid kunde garantera att de alltid laddades på samma plats i minnet när man komponerade program av flera moduler (dvs när man länkade samman flera moduler)

4) Förklara skillnaden mellan policy och mekanism med ett exempel från processchemaläggning med prioriteter. (1p)

Mekansimen är schemaläggning baserad på prioriteter, dvs att den process som är klar att exekvera och har högst får exekvera först. Policyn bestäms av hur man väljer att sätta prioriteterna.

5) a) Vad betyder det att en sidutbytesalgoritm är global respektive lokal? (1p)
*b) Spelar det någon roll om sidutbytesalgoritmen är lokal eller global i ett OS med stöd för mjuka realtidsprocesser? (1p)

a) Vid sidfel kan en global algoritm välja att slänga ut en sida ur en ram oavsett vilken process sidan tillhör, en lokal algoritm byter bara ut sidor som hör till samma process.
b) För att kunna garantera realtidsaspelkterna är det viktigt att realtidsprocesserna har tillgång till de resurser de behöver - vilket talar för en lokal utbytesalgoritm där andra processer inte kan ta ramar från dem. Å andra sidan är det viktigt att snabbt kunna behandla ett sidfel och hitta en ledig ram - vilket talar för att man bör kunna håklla en ledig lista med tomma ramar och den hålls antagligen bäst om man kan ta ramar från även andra processer. Dvs en kombination av lokal och global algoritm är antagligen bäst där realtidsprocesserna kan garanteras tillgång till ett visst antal ramar- men där man kan ta ramar globalt från icke-realtidsprocesser

6) Ett datorsystem med vistuellminne använder en TLB och en hashad inverterad sidtabell för adressöversättning.
a) Givet följande parametrar ge en formel för den effektiva minnesaccesstiden!
Sannolikhet för träff i TLB = P, söktid i TLB = T, tid för att beräkna hashfunktionen = a, genomsnittligt antal element som mappar till samma hashvärde = b (1p)
b) Ge en formel för när den hashade sidtabellen är snabbare att använda en en vanlig två nivåerssidtabell. (1p)

(b delen lyckades jag tappa bort på tentan när jag klippte bort svaren (skrev tenta och svar på en gång....))

a) PT + (1-P)(T + a + bM) + M
b) a+bM <= 2M

7) *¤ Förklara ideerna bakom quick-fit algoritmen för minneshantering på användarnivå (i.e. malloc, free...) (1p)

Jfr labPM lab 3

8) I virtuellminneshantering är "working set" ett centralt begrepp, förklara vad det är och hur det kan användas! (2p)

i) Working set är de sidor en process accessa under någon viss tidsrymd. ii) En sidutbytesalgoritm bör "uppskatta"/sträva efter att behålla processens working set i primärminnet när den väljer sidor att gör apge-out på vid sidfel. iii) Om en process inte kan hålla hela sitt working set i minnet ger det upphov till sk thrashing.

9) Filsystem kan i huvudsak implementeras med två metoder: antingen FAT eller index-noder. Beskriv hur de två metoderna fungerar och jämför deras för och nackdelar (2p)

För en fil behöver man hålla ordning på information om: i) administrativ data som skapande/modifierad tid, skapare, ägare, rättigheter mm, samt ii) vilka block på disken som används för att lagra filen

FAT: Håller ordning på diskblocken genom en tabell/vektor för alla diskblock i partitionen. Administrativ data lagras normalt på annat ställe i filsystemet (i directoryn) tillsammans med index för första diskblocket. Den som indexeras av detta i FAT innehåller i sin tur index för nästa diskblock osv. (rita figur!)

Fördelarna med FAT är att det är en förhållandevis enkel struktur. Hålls den i minnet kan det gå ganska fort att följa kedjan av index vid t.ex random access i en fil. Nackdelarna är i huvudsak att tabellen ofta har en viss förutbestämd storlek vilket gör att diskblocksstorleken inte alltid kan väljas oberoende av partitionens storlek, tabellens storlek kan i sig bli ett problem och man kan få snabbare random access till större filer med andra metoder/datastrukturer.

Indexnoder, som de implementeras i t.ex UFS. används på följande sätt: I directoryt finns en översättning från filnamn till indexnodnummer. Själva indexnoden är en datastruktur som ligger lagrad på disk men läses in i minnet vid behov. Den gförsta indexnoden innehåller administrativ data för filen och pekare till 10 direkta diskblock, pekare till indirekt block och dubbelindirekt block etc. (rita figur) Fördelarna med indexblock är att bara de indexblock som för tillfället behövs behöver hållas i minnet (till skillnad från FAT), man kan i princip enkelt utvidga iden till att klara hur stora filer som helst. bättre tidskomplexitet för att accessa godyckligt data block (log N istället för N som i fallet FAT), indexnoderna kan lagras nära med filens datablock på disken, gör det enklare att implementera virtuella filsystemslager, diskblockstorleken kan väljas obeoroende av implementationen (till skillnad från FAT)

10) I vilka sammanhang kan man tänka sig använda kontinuerlig allokering för ett filsystem? (1p)

Eftersom man i kontinuerlig allokering snabbt kan få stora problem med extern fragmentering vid omallokering av filerna används det i princip uteslutande för filsystem som endast skrivs en gång (i.e. read-only) som t.ex för vissa CD-rom filsystem etc.

11) Vad är en RAID-disk? Vad har den för fördelar och nackdelar? (2p)

En RAID disk är uppbyggd av en diskcontroller och två eller flera diskar som kontrolleras av controllern och som för användaren ser ut som en enda disk. Iden är att använda en av eller en kombination av följande tekniker för att öka prestandan, öka säkerheten eller åstadkomma en kombination av dessa: Teknoikerna är i) spegling - dvs data skrivs till mer än en disk vilket primärt ökar säkerheten, striping (strimling) av data där data delas upp ocjh skrivs till olika diskar vilket primärt ökar hastigheten, iii) felrättande koder vilket ökar säkerheten och kan göra det möjligt att återskapa tappad information. Kombinerar man alla tre teknikerna kan man få system där man vinner både prestanda och säkerhet så att man t.ex kan ersätta en skadad disk med en ny och återkapa informationen som fanns på den gamla från information som finns på de diskar som fanns kvar. Hur man kombinerat teknikerna och vad man på så sätt åstadkommer beskrivs i olika nivåer av RAID som benämns RAID 0, RAID 1 osv. där de lägre nivåerna är enklare implementationer

12) När man gör I/O finns i princip tre sätt att vänta på att I/O enheten skall bli klar.
Beskriv dessa kortfattat! (1.5p)

Programstyrd I/O sk pollning. Man kontrollerar och styr I/O enheten genom att läsa av och sätta ptar i t.ex ett kontrollregister. Normalt loopar man och läser av kontrollbitarna vilket drar processortid

Avbrottstyrd I/O: I/O enheten programmeras att utföra en uppgift. Att I/O enheten är klar signaleras genom ett avbrott vilket gör att processorn kan användas för andra ändamål mellan det att operationen initieras och till dess den är klar.

DMA: Direct Memory Access man har en slavprocessor som programmeras upp att sköta överföringen och flytta in/ut ett block av data till /från rimärminnet och en I/O enhet. När DMA överföringen är klar genereras ett avbrott. Skillnaden mot avbrottstyrd I/O är att DMA enheten kan programeras att läsa/skriva större mängder med data.

13) I/O delarna av ett OS interagerar oftast med andra delar ov OS:et. Ge i ett modernt OS för en PC exempel på tre delar som I/O systemet normalt interagerar med och beskriv interaktionen kortfattat! (1.5p)

De delar man i första hand tänker på är: Filsystem - läsa och skriva filer kräver I/O, Virtuellminnessystem - att flytta sidor till/från disk kräver I/O, Processhantering - läsa in kod kräver I/O (via fuílsystemet), Power Managment - interaktion för att spara energi genom att stänga av I/O enheter, (i princip alla andra delar av OS:et är på något sätt beroende av I/O systemet)

14) I praktiskt taget alla OS som implementerar kärntrådar behåller man ändå processbegreppet.
Förklara varför! (1p)

Om man skulle ta bort processbegreppet tvingades man knyta hanteringen av resuresr som filer, minne etc. till en tråd som då i princip skulle bli en process - man skulle också ha problem med att veta vem av trådarna som skule vara ägare till dessa resuresr om man vill ha flera trådar i en "context" - (contexten i säg kan ju sägas vara processen)

15) Vid kommunikation i system baserade på TCP/IP protokollen anger man normalt ett portnummer förutom den IP-adress man vill kommunicera med. Varför? (1p)

För att kunna adressera den tillämning man vill kommunicera med. IP-adressen identifierar bara ett nätverksinterface (oac alltså inte en tillämpng), ProcessID kan inte användas eftersom man normalt inte vet vilket processID en tillämpning har och då detta kan variera om mna tvingas starta om en process. Portnummer kan däremot standardiseras eller överenskommas om mellan maskiner. DVS man har nytta av indirektionslager!

16) ¤ Dining philosophers är ett klassikt synkroniseringsproblem. Beskriv problemet och ge två lösningar som är deadlock fria (högre grad av parallellism ger mer poäng än mer restriktiva lösningar) (2p)

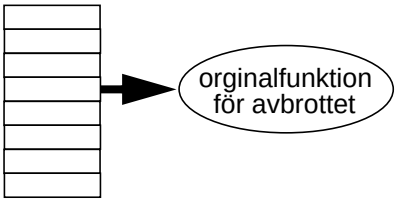
I problemet finns ett runt bord med N stolar och en gaffel mellan varje par av stolar. På bordet står en skål med spagetti. En filsoof som vill äta sätter sig på en ledig stol och tar upp en gaffel från varje sida om sin plats eftersom han/hon behöver två gafflar för att klara av att äta den hala spagettin. Systemet kan ekelt hamna i deadlock då en filosof aldrig lägger ner en gaffel han/hon tagit förrän dess att han/hon ätit klart. Ett enkelt exempel är ett bord med två stolar och två gafflar där två filosofer sätter sig samtidigt och tar upp höger gaffel - när de sedan försöker ta vänster gaffel är den redan tagen av den adra filosofen - dvs cirkulär väntan, ingen preemption, hold&wait och ömsesidig uteslutning = Deadlock (eftersom vi bara har en resurs av typen gaffel mellan stol n och n+1)

Bra lösningar som ger mycket parallelism, dbsv tillåter N-1 filosofer att samtidigt försöka äta är:
- en vaktmästare tillåter bara N-1 filosofer att försöka äta samtidigt, bryter cirkulär väntan
- på alla platser utom plats N tar man upp höger gaffel först, på plats N tar man vänster gaffel först, bryter cirkulär väntan
- numrera gafflarna från 1 till N, och inför en regel som säger att man måste ta upp gafflarna i nummerordning på varje plats (amma sak som lösningen ovan)

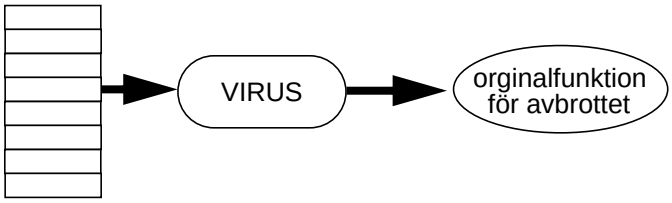
17) a) Varför försöker ofta virus ta kontroll över avbrotts/TRAP-vektorerna? (1p)
b) Skissa en enkel schematisk bild över hur funtionsanropen ser ut i en avbrottsvektor före och efter det att ett virus tagit kontroll över avbrottsvektorn. (1p)

a) Eftersom om man där t.ex kan se alla anrop till exec() och på så sätt enkelt hitta vilka exekverbara filer som finns i systemet som man kan infektera, dessutom ser man allt som sker på systemet vilket kan vara bra om man vill spionera

b) avbrottsvektorn är en vektor av pekare till funktioner. När viruset tar kontroll över avbrottsvektorn ser det till att avbrottsvektron pekar ut virusets kod/funktion och kommer i sin tur ihåg vad som pekades ut från början så viruset kan anropa den funktionen efter att det själv körts



Utan virus



Med virus

18) Varför kan det vara sämre ur säkerhetssynpunkt att inte behöva ge hela programnamnet när man vill starta exekveringen av ett nytt program än att behöva ge hela programnamnet med alla extensions? Ge också ett konkret exempel på när det här kan bli ett problem! (1p)

Om man inte behöver ge extension förutsätts att OS eller kommandotolk letar upp en körbar fil med rätt namn och en extension som talar om att den (borde vara) är exekverbar. Ett ydligt exmpel på att det kan bli problem är att i en del av Microsofts OS letas först en exekverbar fil med extension .com upp och bara om man inte hittar en sådan provar man att hitta och exekvera en fil med extension .exe Den senare extension är vanligare och öppnar för att man inför trojaner med samma namn men med extension .com istället för .exe - dessa kommer då att köras istället för det man (kanske) egentligen ville köra

19) * ¤ I realtidsystem händer det att lågprioriterade processer utför uppgifter åt högprioriterade processer. Ett problem i det här sammanhanget är att den högprioriterade processen kan få vänta onödigt länge på svaret från den lågprioriterade procesen. Beskriv vad problemet kan vara och hur man åtgärdar det! (1p)

Problemet är att när den lågprioriterade processen utför uppgifter åt den högprioriterade så kan man se det som om den högprioriterade exekverar med den lägres prioritet - den kan då avbrytas av och behöva vänta på en process med mellanhögprioritet, Lösninge är att låta den lågprioriterade processen exekvera med den högprioriterades priorited medan den utför uppgifter åt den högprioriterade.

20) * ¤ I ett OS för hårda realtidsystem vill man införa en generell möjlighet till synkronisering via lås. Är det här en bra eller dålig ide? Varför? (1p)

En dålig ide eftersom schemaläggningsproblemet då blir NP-komplett i det generella fallet

21) * UNIX/LINUX och Microsofts OS har olika syn på hur användargrafik skall implementeras. Beskriv i grova drag skillnaderna och dess för- och nackdelar! (2p)

Microsoft har valt att integrera grafiken i kärnan. Fördelarna är att det (kan) blir effektivare och att förhoppningsvis alla applikationer får samma "look&feel" - nackdelen är att det är svårt att byta gränssnittet om man skulle vilja det. I UNIX/LINUX ligger grafiken som användarprocesser (server-client) fördelarna är att man enkelt kan byta/ uppdatera gränssnittet obeeroende av kärnan - nackdlear kan vara möjligen sämre prestanda och att man kan få olika look&feel på applikationer