Tentamen i ID2200, ID2206, IS1350 Operativsystem

(fd 2G1520, 2G1504, 2G1113, 6B2019)

Tisdagen 2009-05-26 kl 0900-1300
Institution: C&C/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, 2G1504
- Studenter som tenterar ID2200, 2G1504 besvarar inte frågor markerade med *****

IS1350, 6B2019
- Studenter som tenterar IS1350, 6B2019 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 STEX senast mån 15/6

Betygsgränser:
- Godkänt (E) garanteras från 50% av maximalt poängantal

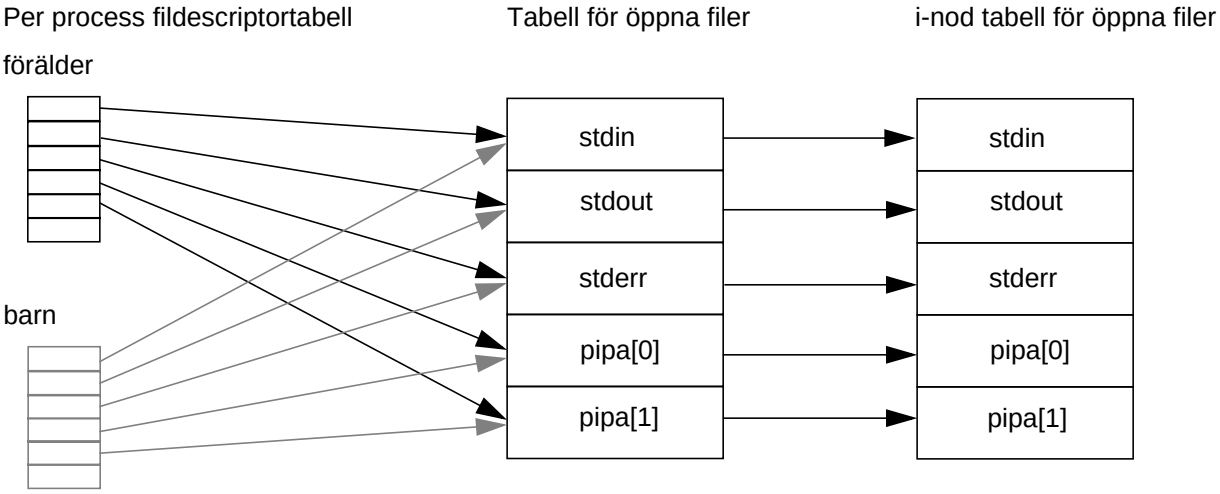
Lösningsförslag: anslås på kursens webbsida efter tentamen (antagligen inte förrän 28/5)

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

---

1) Antag att nedanstående program exekveras på ett UNIX/LINUX system
   a) Rita en bild över systemets datastrukturer för öppna filer som det ser ut när programmet exekverats till LabelA (2p)

(grått - efter fork())



b) I vilken ordning terminerar processerna? (1p)

```
int main(int a, char** c)
{
  pid_t pid;
  int pipa[2], status;
  char tkn;
  pipe(pipa);
```

```
LabelA:
 if(pid = fork()) {write(pipa[1], "HELLO\n", 6); waitpid(pid,&status,0);}
  else while(read(pipa[0], &tkn, 1) == 1 && tkn != EOF) putchar(tkn);
}
```

Ingen av processerna terminerar. Barnprocessen blockerar på read() på en tom pipe där både barnet och föräldern fortfarande potentiell skulle kunna skriva. Föräldern blockerar i waitpid() eftersom barnet inte terminerar.

2) Beskriv vilken data/dataareor en tråd måste ha privata (dvs som inte kan delas med andra trådar inom processen) och förklara varför dessa måste vara privata! (2p)

En tråd måste ha en privat stack för att kunna lägga upp aktiveringsposter på för procedur/funktionsanrop som den gör. Den måste också ha egna kopior av alla register inkluderat stackpekare och programräknare för att den ska kunna exekvera sin egen väg genom koden.

3) Kalle ska designa en schemaläggningsalgoritm för ett batch-system där processerna (jobben) ibland läser/skriver till disk. Kalle hävdar att man antagligen får högre genomströmning om man höjer prioriteten för en process då den får börja exekvera och om processen behåller den förhöjda prioriteten till dess den terminerar. Dvs vad Kalle förelår är ett system med två prioriteter (för användarprocesser): Låg - för processer som ännu inte fått exekvera och Hög - för processer som påbörjat sin exekvering.
Diskutera om Kalles ide är bra eller dålig och motivera varför! (1p)

Kalles ide är bra om det gäller ett Batch-system (i de flesta andra typer av system är iden usel). En process som påbörjat sin exekvering bör få avsluta den med högre prioritet än de som ännu inte påbörjats - dels för att det minskar risken för att för många processer blir aktiva samtidigt (vilket skulle kunna leda till fler sidfel, thrashing och ev. swapping) och för att man antagligen bättre kan utnyttja lokalitet i cachear, TLB och i fall med sidindelat minne.

4) Moderna OS använder olika tekniker för att snabba upp filsystemen. Det här kan ibland leda till problem i samband med t.ex systemkrascher. I ett OS hittar man nedanstående beskrivning av ett systemanrop sync(). Vad är det för mekanism som gör att man tvingats införa systemanropet och varför har man infört det? (2p)

SYNOPSIS
```
#include <unistd.h>

void sync(void);
```

DESCRIPTION
The sync() function writes all information in memory that should be on disk, including modified super blocks, modified inodes, and delayed block I/O.

För att snabba upp filsystemen cachar man ofta information. Det kan handla om öppna inoder, eller block i en fil som cachas i primärminnet. Den här informationen skrivs inte till disk särskilt frekvent. Modifierade filblock och inoder skrivs t.ex normalt inte förrän filen stängs. Det här gör att man kan få problem med inkonsistens i t.ex en databas om systemet kraschar under det att databasen var i drift. Med sync(2) systemanropet kan man tvinga systemet att skriva tillbaka all information på disken. På samma sätt bör man anropa sync(1) innan man stänger ner ett system som kör LINUX/UNIX för att säkerställa att filsystemet är intakt på disken innan strömmen bryts.OBS! att om ett system kraschar hinner man oftast inte, om inte systemet spoecifikt är designat för att klara just detta, av att anropa sync(2)

5) I nära nog alla typer av filsystem är det första blocket i varje partition reserverat för ett visst ändamål. Vilket?Och måste blocket vara reserverat för detta ändamål? (1p)

Det är normalt reserverat som boot-block - dvs om partitionen innehåller ett OS innehåller boot-blocket det program som systemet kör om man vill boota OS:et på partitionen. Alla partitioner innehåller inte ett OS, men om man vill kunna lägga ett OS i partitionen måste det finnas ett boot-block. Om man inte reserverat ett boot-block och senare kommer på att man vill lägga ett OS på partitionen kan man tvingas att flytta all data på partitionen om inte första blocket reserverats som boot-block - vilket innebär onödigt arbete. Därför reserveras normalt första blocket i en partition som boot-block oavsett om partitionen kommer att innehålla ett OS eller ej.

6) Var bör man placera i-noder på en hårddisk - i början, slutet eller i mitten av disken? (1p)

Det generella svaret är att inoder bör placeras så att de är enkla att hitta och så att diskarmsrörelserna minimeras. För det senare gäller att inoderna bör ligga nära datablocken för de filer de implementerar. Med de svarsalternativ som gavs i frågetällningen är *i mitten* av disken att föredra eftersom det bör minimera diskarmsrörelserna.

7) a) Rita en figur och beskriv med exempel hur en File Allocation Table fungerar! (1p)

Se boken (upplaga 3) figur 4-12

b) I vilka typer av moderna system används fortfarande FAT? (1p)

FAT är utvecklat för små system med begränsade processorkapaciteter och små diskar. Alltså används det fortfarande i sådana system t.ex för minneskort som används i kameror och liknande.

8) Vilka fördelar ger sidindelad minneshantering jämfört med ren segmentering? (1p)

Ren segmentering kräver kontinuerlig allokering av segmenten. Det ger upphov till problem med extern fragmentering, att hålla ordning på lediga/upptagna segment av olika storlekar och att klara av att segmenten växer. Alla dessa problem avhjälps med sidindelat minne.

9) Kalle hävdar att i ett system med sidindelat minne bör operativsystemet alltid nollställa en sida/ram som en process allokerar till följd av att stacken eller heapen utvidgas. Varför? (1p)

För att undvika säkerhetshål där en process som allokerar en ny sida/ram kan få tillgång till data från en annan process som skulle kunna ligga kvar i ramen den får tillgång till.

10) I sidindelade system kan sidutbytesalgoritmen antingen vara lokal eller global. Vad betyder detta? Och vilka för- och nackdelar har de? (2p)

En lokal sidutbytesalgoritm kan bara byta innehållet i ramarna som är allokerade för den process som genererat sidfelet (om man inte begär en ny oallokerad ram från systemet). En global sidutbytesalgoritm kan välja att byta ut vilken sida som helst i minnet för att frigöra en ram. Fördelen med en lokal algoritm är att ett sidfel i en process i huvudsak bara påverkar processen själv. Men globala algoritmer har generellt bättre prestanda och klarar ofta höga laster bättre eftersom man helt enkelt kan göra bättre globala val av sida att kasta ut. (PS för den som läst in sig för mycket på gamla tentor så har valet av en lokal eller global sidutbytesalgoritm inte så mycket med realtidssystem att göra. vad gäller det och sidindelat minne kan man konstatera att man i hårda realtidsystem aldrig har virtuellt sidindelat minne och därmed har man inte sidutbytesalgoritmer - frö mjuka realtidsystem kan man antingen välja att ha lokala algoritmer för relatidsprocesserna eller att ha en global algoritm med möjlighet att "pinna" (låsa) sidor i minnet för realtidsprocesserna och få liknande prestanda)

11) Beskriv vad som händer om man får en TLB-miss. (1p)

Den sökta översättningen sida till ram finns inte i cachad i TLBn - adressöversättningen måste istället hämtas från sidtabellen i minne. Därefter skall TLBn uppdateras genom att en ledig post hittas eller skapas genom att en post slängs ut och den saknade översättningen sida-ram lagras i TLBn. Detta kan göras antingen av hårdvaran eller i moderna system ofta i mjukvara.(OBS! många verkar ha missuppfattat vad TLBn gör. Man lagrar inte hela sidor i TLBn utan man cachar bara översättningar sida->ram)

12) * Drivrutiner är en viktig del av ett operativsystem
a) beskriv vilka huvudsakliga logiska delar en drivrutin består av och vad deras uppgifter är! (2p)

En drivrutin består logiskat oftast av två delar: en del för att anropa funktioner, ofta motsvarande filoperationer read(), write() etc., för att starta en operation mot enheten drivrutinen styr. Eftersom de flesta i/o enheter och datorsystem brukar stöda avbrott finns också oftast en (eller flera) avbrottsrutiner för att ta hand om avbrottet som kommer när operationen är klar. I Unix terminologi brukar dessa delar kallas top-half resp. bottom-half device driver.

b) beskriv hur gränssnittet för att anropa funktioner i en drivrutin från operativsystemet brukar organiseras.(1p)

Gränssnittet består normalt av ett antal funktioner motsvarande systemanropen på en fil (om en fil är abstraktionen för en i/o-enhet). Dessa kommer man åt via drivrutinsvektorn där en post normalt består av en struktur i programmeringsspråket C med funktionspekare till funktionerna i drivrutinen.

13) Så kallade en-vägsfunktioner används för att implementera en del säkerhetssystem i datorsystem.
a) vad är en en-vägsfunktion? (1p)

En funktion för vilken det är mycket svårt eller omöjligt att beräkna inversen.

b) ge ett exempel på och förklara hur något säkerhetssystem fungerar som bygger på en-vägsfunktioner. (1p)

Det finns flera exempel och varje svar bedöms individuellt: Exempel är hur lösenord hanteras i UNIX eller hur man kan generera engångslösenord

14) Systemet för accesskontroll i AFS (som bland annat KTH använder) är på många sätt utvidgat och förbättrat jämfört med accesskontrollmekanismerna i UFS. Beskriv och förklara de viktigaste skillnaderna! (2p)

I UFS kan användaren sätta accessrättigheterna läs (r), skriv (w) och exekvering (x) för respektive användare (user), grupp (group) och alla användare (other) per fil. Grupper och grupptillhörighet administreras av sytemadministratör. I AFS sätter man istället accessrättigheter per katalog (directory). Man kan bland annat sätta rättigheter för: listning (l), läsning (r), skrivning (w), borttagnig (d), exekvering (x) men också rättigheter att administrera rättigheter (a). Rättigheterna kan sättas för enskild användare eller grupper av användare där en enskild användare kan skapa grupper och lägga till användare i grupperna.

15) Kalle hävdar att man i design av ett operativsystem inte behöver bry sig om synkroniseringsproblematik om man inte designar ett system med både processer och trådar. Har Kalle rätt eller fel? Förklara! (1p)

Kalle har fel. Oavsett om man har enbart processer eller både processer och trådar i ett OS måste man alltid när man kan ha flera aktiviteter pågående (pseudo)-parallelt som kan dela resurser i systemet bry sig om synkroniseringsproblematiken

16) a) Beskriv kortfattat begreppen livelock respektive deadlock (baklås). (1p)

Livelock: två eller flera processer exekverar men håller hela tiden på att synkronisera med varandra så att de inte kommer förbi synkroniseringspunkten.
Deadlock: en mängd processer där alla väntar på en händelse som bara kan orsakas av någon av de väntande processerna i mängden. Ingen process i mängden exekverar och processerna kan inte komma vidare i sin exekvering.

b) Vilket/vilka problem kan man räkna med att stöta på om man försöker hitta ett deadlock i ett system med hjälp av en debugger? (1.5p)

Det kan vara svårt att återskapa den schemaläggning som ledde till deadlock så att man hamnar i samma problem. Debuggern kan i sig också påverka schemaläggningen vilket kan göra det än svårare att få samma ordning på händelserna som ledde till deadlock. Debuggern hjälper heller inte till att förstå vilken händelseordning som ledde till deadlock.

17) * För periodiska realtidsystem används ofta Rate Monotonic Scheduling som schemaläggningsmetod. Beskriv kortfattat hur metoden fungerar och vilka för- och nackdlear den har jämfört med Earliest Deadline First. (2p)

RMS kan användas för system med periodiska jobb. RMS tilldelar fasta prioriteter till jobben som är omvänt proportionella mot jobbens period. RMS kan schemalägga system med laster upp till c:a 0.69. EDF klarar högre laster, upp till 1, men har inte fixa prioriteter vilket gör schemaläggningen mer resurskrävande.

18) ¤ I ett distribuerat filsystem som tillåter cachning eller replikering av filer måste man hantera problem med konsistens och konsistenssemantik.
a) förklara vad problemen innebär/betyder? (1p)

Konsistens: att alla ser en och samma version av en fil. Konsistenssemantik: handlar om i vilken grad man kan upprätthålla att det bara finns en version av en fil och hur detta implementeras

b) ger "write-through" cachesemantik i ett distribuerat filystem sekvensiell (sk. UNIX-) semantik (1p)

Nej, det ger inte sekvensiell semantik. Även om man skriver alla ändringar direkt till serverna så kommer de att komma med varierande grad av fördröjning till servern.

19) ¤ I multiprocessor och multi-core system använder man ibland så kallad gäng-schemaläggning (gang scheduling). Vad innebär begreppet och vad hoppas man vinna på det? (1p)

Det innebär att man försöker schemalägga (tillåta schemaläggning av)alla processer/trådar för exekvering samtidigt. På så sätt hoppas man vinna att processer/trådar som synkroniserar med varandra snabbare klarar av synkroniseringen istället för att kanske tvingas vänta i synkroniseringen mot en process/tråd som inte tillåtits exekvera.

20) I Microsofts NT filsystem så implementerar man i) att filer kan ha flera strömmar av data knutna till sig (från början en ide lånad från MAC för kompatibilitet) och ii) man lagrar filer i "runs" av konsekutiva diskblock.
a) Ge minst två exempel på hur en applikation kan dra nytta av att en fil kan ha flera dataströmmar (1p)

Ordbehandlare med en ström innehållande dokumentet och en ström med ändringar gjorda i dokumentet som tillåter "undo" möjligheter. Multimedia med olika strömmar för bild- och ljudinformation.

b) Vad ger det för fördelar och nackdelar att lagra datat i "runs" (1p)

Fördelen med att lagra filer i "runs" är att skrivning och läsning av flera på varandra följande block i filen kan gå snabbare med mindre diskarmsrörelser. Nackdelen är att allokeringen av diskblock blir mer komplicerad. Man får problem liknande de man har vid kontinuerlig allokering där det kan vara svårt att hitta tillräckligt många konsekutiva diskblock för ett run och det kan vara svårt att utvidga ett run om en fil växer.

21) Sockets är en abstraktion/mekanism för kommunikation mellan processer över ett nätverk. På klientsidan används normalt bara en fildeskriptor men på serversidan används/skapas flera. Varför används normalt minst två, men ibland flera, fildeskriptorer på serversidan? (1p)

Man har normalt en fildeskriptor som man lyssnar på inkommande uppkopplingar på. När en uppkoppling mot socketen kommer så skapas oftast en unik fildeskriptor for just den kommunikationen med den specifika klienten, vilket tillåter att man kan ha flera parallella uppkopplingar med olika klienter mot en och samma socket på serversidan.