Lösningsförslag till tentamen i IS1350, ID2206, ID2200 mfl Operativsystem
måndag 2008-10-20 kl 0800-1300

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

Rättning:
- ***Alla svar***, även på delfrågor, måste ha åtminstonde en kortfattad motivering för att ge poäng
- Resultat beräknas anslås senast mån 10/11

OBS! Till följd av en oförutsedd tragisk händelse i mitten av oktober som jag var tvungen att prioritera att hantera, ligger jag c:a 3 veckor efter med allt arbete, inkluderat rättningen av tentan /RR

Betygsgränser:
- Godkänt (E) garanteras från 50% av totalt antal poäng på tentamen

Lösningsförslag: anslås på kursens webbsida efter tentamen (antagligen inte förrän 22/10)

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

1) Förklara vad som händer om man, inloggad som root, startar följande lilla program: (1p)

```
int main(int argc, char ** argv)
{ while() fork(); }
```

Programmet kommer att rekursivt anropa sig självt och skapa nya processer. Antalet processer i systemet växer snabbt och processtabellen kommer att fyllas och till slut går det inte längre att skapa nya processer. Det betyder att systemet blir oåtkomligt och antagligen behöver startas om.

2) Rita en schematisk bild av adressöversättningen i ett system med sidindelning , TLB och en-nivås sidtabell. (1p)

Se föreläsningsanteckningarna

3) I ett system finns både korta jobb som i medeltal exekverar 0.1 sek innan de blockerar på I/O eller är klara, och långa jobb som i medeltal exekverar 10sek-10min. Diskutera för- och nackdelar med att använda schemaläggningsstrategierna FirstComeFirstServe, ShortestJobFirst respektive round-robin för systemet (3p)

Man kan utgå från att de korta jobben till stor del består av interaktiva processer.
FCFS: kommer att ge oacceptabla fördröjningar för de korta I/O intensiva (korta) jobben om de hamnar efter ett eller flera långa CPU-tunga jobb.
SJF: riskerar att svälta ut de CPU-tunga jobben
Round-Robin: Fungerar heller inte bra av samma anledning som FCFS

För att systemet ska fungera bra bör man införa olika prioriteter för jobben och preemption. Då kan Round-Robin fungera bra per prioritetsklass.

4) Kalle håller på att designa ett eget litet OS. Han planerar att ha två prioritetsnivåer för användarprocesser där interaktiva processer har hög prioritet och beräkningstunga processer har låg prioritet. Schemaläggningen skall vara preemptiv. Tilldelningen av CPU tid ska inte vara baserad på fixa timeslice utan baserad på individuella tidskvanta (quantum) liknande schemaläggningen i LINUX.
Frågan är hur systemet påverkas om quantum tillåts bli väldigt lång för enskilda processer? (2p)

Systemet bör fungera ganska bra under de flesta laster. Problem kan uppstå om det hela tiden finns många högprioriterade processer som inte är blockerade då lågprioriterade processer kan få vänta länge på att få exekvera och kanske svältas. För lågprioriterade jobb kommer schemaläggningen att likna FCFS.

5) Förklara varför TEXT och DATA normalt inte kan ligga på samma sida i ett system med sidindelat virtuell minne! (1p)

Därför att kontroll av accessrättigheter till olika delar av den virtuella adressrymden, dvs read-write-execute rätttigheter, sker per sida. Då TEXT i de allra flesta system endas kan accessas för RX och DATA normalt endast kan accessas som RW kan de inte dela sida.

6) Givet ett system med virtuellt minne, TLB, en-nivås sidtabell, access tid till minne som är ***M***, söktid i TLBn som är ***h*** och träffsannolikhet i TLBn som är ***P***. Antag att vi vill modifiera systemet och införa en två-nivåers sidtabell. Hur stor måste träffsannolikheten i TLBn (kalla den t.ex ***P'***) vara för att den upplevda (genomsnittsliga) minnessaccesstiden skall vara densamma med två-nivåers sidtabellen? (2p)

Ställ upp den genomsnittsliga accesstiden med en-nivås respektive två-nivåers sidtabell och räkna fram P' för när accesstiderna är lika.

$P(h+M) + (1-P)(h+M+M) = P'(h+M) + (1-P')(h+M+2M)$

Vilket ger: $P' = (P+1)/2$

Att man räknat rätt kontrolleras enkelt genom att t.ex kontrollera P' för värdena P=1 samt P=0. För fallet P=1 måste P' också vara 1, och för fallet P=0 måste rimligtvis P' > 0 gälla.

7) Beskriv begreppen dynamisk laddning respektive dynamisk länkning! (2p)

Se föreläsningsanteckningar

8) Hur brukar man säkerställa att en användare inte kan exekvera en fil med t.ex data eller text även om den har rättigheter i filsystemet (protection) som medger exekvering? (1p)

Genom att se till att en exekverbar fil har ett internt format där man alltid inleder filen med ett unikt värde, ett så kallat magic number, där alla exekverbara filer ska starta med samma magic number. När man förösker lada en fil för exekvering kontrolleras detta värde. Väljs detta från en tillräckligt stor talrymd blir sannolikheten för att man av misstag skall kunna tolka en icke exekverbar fil som exekverbar försumbart liten.

9) Beskriv kortfattat hur ett directory (katalog eller mapp) implementeras i ett UNIX-filsystem och vilken information man kan förväntas hitta i directoryt för en vanlig fil (som katalogen har en hård länk till) (1p)

Ett directory implementeras som en fil taggad så att den kan indentifieras som ett directory. Directoryt i sig innehåller en enkel databas där posterna består av "översättningar" som t.ex: filnamn till i-nod, mjuk länk till sökväg, monteringspunkt till filsystem etc.

10) Kalle hävdar att context switch tiden att byta mellan två kärntrådar är betydligt mindre än att byta mellan två processer - har han rätt eller fel? Förklara någorlunda utförligt varför? (2p)

Kalle har rätt. Ett processbyte mellan två processer involverar ett helt context byte (en hel context-switch) och en schemaläggning av en ny process. Ett byte mellan två kärntrådar innebär bara att man byter registeruppsättning och stack inom en och samma context / process och innebär normalt inte nödvändigtvis ett schemaläggningsbeslut för en hel process.

11) Många operativsystem använder något som kallas "copy-on-write" när nya processer skapas med fork(). Vad innebär begreppet och vad hoppas man vinna på att använda det? (2p)

I samband med fork() innebär copy-on-write att man inte gör en kopia av föräldraprocessens hela adressrymd för barnprocessen. Istället kopieras bara sidtabellen och man markerar i bägge sidtabellerna att om någon av processernaförsöker skriva i en sida skall barnproceseen får en egen kopia av sidan i fråga. Eftersom de flesta nyskapade processer oftast direkt byter ut innehållet i den virtuella adressrymden genom att anropa exec*() utan att skriva i sidorna som den ärvt från föräldern så kan man vinna tid på att inte kopiera adressrymder i onödan.

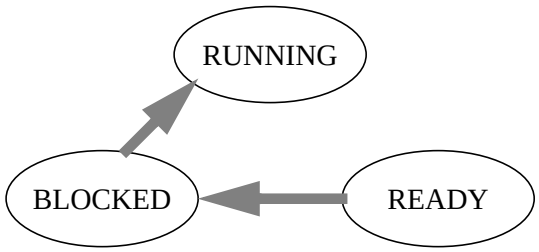
12) Beskriv kortfattat vilken metod man använder i Microsofts operativsystem för att på bästa sätt schemalägga trådar på multi-core processorer! (1p)

MS nya operativsystem som VISTA försöker om möjligt schemalägga trådar på samma processorkärna som de tidigare exekverat på för att om möjligt kunna återanvända cache-innehåll mm.

13) Vilka av följande funktioner kan man använda inifrån kärnan på ett LINUX system (förklara varför): stat(), scanf(), printf(), read(), listen(), write() (1.5p)

Ingen - alla är systemanrop eller biblioteksfunktioner på användarnivå som i sin tur använder systemanrop. Och systemanrop kan bara användas för att accessa kärnan utifrån - inte innifrån.

14) Processchemaläggning involverar normalt tillstånden READY, RUNNING samt BLOCKED. Förklara varför man (i princip) aldrig implementerar de gråmarkerade tillståndsövergångarna i tillståndsgrafen nedan i schemaläggningsalgoritmer?



(2p)

För att en process skulle kunna flyttas direkt från tillståndet READY till BLOCKED skulle krävas att den exekverande processen eller OS:et skulle upptäcka att något den exekverande processen gjort eller något som hänt i en annan del av systemet förorsakat att en process i ready-kön blivit blockerad. Detta är i princip omöjligt att implemntera dels av skälet att man inte vet när kontrollerna skall utföras och dels för att det skulle innebära mycket onödigt arbete och bli onödigt tidskrävande. Enklare är att låta processen upptäcka detta själv när den får exekvera.

Att flytta en process direkt från BLOCKED till READY skulle innebära att man kringgick schemaläggaren (och eventuella prioriteter) alternativt att man var tvungen att göra en mer komplex schemaläggare. Betydligt enklare, mer korrekt och antagligen mer effektivt är att låta schemaläggaren bestämma vilken process som skall exekvera genom att flytta en tidigare blockerad process till ready-kön när processen inte längre är blockerad.

15) Beskriv vad "login spoofing" är och hur man kan implementera det (2p)

Login-spoofing innebär att man härmar beteendet och utseendet hos en tilllämpning som kräver att användaren ger sina inloggningsuppgifter. Syftet är att komma åt inloggningsuppgifterna. Det sker enklast genom att man skriver ett program och sparar den exekverbara filen med ett namn som liknar den riktiga tillämpningens och lägger det på ett ställe där användaren av misstag enkelt kör det, dvs. man gör en trojan. När man fått inloggningsuppgifterna kan man meddela användaren att autenticeringen misslyckades och avsluta eller ev. starta den applikation användaren tänkt köra.

16) Vilka är Coffmans fyra villkor för deadlock? (2p)

Se föreläsningsanteckningarna.

17) Beskriv hur gränssnittet mellan OS-kärnan och en drivrutin ser ut i LINUX! (2p)

Det består av tre delar:

OS-kärnan har en vektor (tabell) med strukturer (struct) som indexeras med major-device number för drivrutinen. Den struct som indexeras innehåller funktionspekare till funktioner i drivrutinen. Dessa funktionspekare har namn, och skall peka ut funktioner, som motsvarar de vanliga operationerna på en fil, dvs open(), read(), write(), close() mfl. Drivrutinsmodulen skall dessutom tillhandahålla två funktioner med förutbestämda namn som anropas av OS:et vid installation av drivrutinen (då bland annat funktionspekarna i drivrutinsvektorn sätts upp) och vid avinstallation av drivrutinen.

18) Ett företag som säljer en anti-virus programvara hävdar att det aldrig kan komma in virus på en dator som använder deras anti-virus scanners. Är påståendet sant eller falskt? (1p)

Nej, det är omöjligt att upptäcka alla virus genom att kontrollera virussignaturer eftersom det kan dyka upp nya okända signatuer men också för att virussignaturer kan vara så lika legal kod att de inte kan stoppas.

19) Varför är det enklare att implementera ett drivrutinsvirus än ett bootsektorvirus? (1p)

Drivrutinsviruset installeras i kärnan av användaren själv och behöver alltså inte i sig försöka nästla sig in i kärnan. Drivrutinsviruset installeras också normalt efter det att operativsystemet startats och behöver därför inte implementera åtgärder för att inte skrivas över av OS:et när det startar.

20) Hur många sidfel får man givet ett primärminne om 4 ramar som är tomt från början, ren demand-paging och följande referenssträng: 1, 4, 2, 6, 4, 5, 8, 2, 5, 3, 2, 5, 1, 6, 3
För sidutbytesalgoritmerna FIFO, LRU och OPT (1.5p)

```
FIFO   1 4 2 6 4 5 8 2 5 3 2 5 1 6 3
       x x x x   x x     x x   x x
       1 4 2 6 6 5 8 8 8 3 2 2 1 6 6
         1 4 2 2 6 5 5 5 8 3 3 2 1 1
           1 4 4 2 6 6 6 5 8 8 3 2 2
             1 1 4 2 2 2 6 5 5 8 3 3

LRU    1 4 2 6 4 5 8 2 5 3 2 5 1 6 3
       x x x x   x x x   x     x x x
       1 4 2 6 4 5 8 2 5 3 2 5 1 6 3
         1 4 2 6 4 5 8 2 5 3 2 5 1 6
           1 4 2 6 4 5 8 2 5 3 2 5 1
             1 1 2 6 4 4 8 8 8 3 2 5

OPT   1 4 2 6 4 5 8 2 5 3 2 5 1 6 3
      x x x x   x x     x       x
      1 4 4 4 2 2 2 5 2 2 5 1 3 3 3
        1 2 2 1 5 5 2 5 5 1 3 1 6 6
          1 1 6 1 1 1 1 1 3 5 5 1 1
            6 4 6 8 8 8 3 2 2 2 5 5
Fifo 10, LRU 11, OPT 8
```

21) Virus och säkerhetsprogram som vill kontrollera vilka filer/program som exekveras fungerar på samma sätt - men hur gör de för att se (upptäcka) vilka filer/program som exekveras? (1p)

I OS-kärnan finns en vektor (tabell) med funktionspekare som pekar ut de funktioner som skall köras i kärnan när ett visst systemanrop görs. Man indexerar sig i tabellen med det systemanropsnummer som förs över till kärnan när man gör systemanropet (dvs som läggs på en plats åtkomlig för kärnan innan TRAP görs). Virus / säkerhetsprogrammet tar en kopia av pekaren för systemanropet för att exekvera ett nytt program (_execve) och ersätter pekaren i tabellen med en pekare till sin egen funktion. När virusets / säkerhetsprogrammets funktion anropas i systemanropet ser den alla försök at exekvera nya program i processer, funktionen avslutar normalt med att anropa den ursprungliga funktionen som pekades ut via systemanropstabellen.

"Hur mycket vore möjligt om ingen försökte det omöjliga?"
Arne Hirdman