Lösningsförslag till Tentamen i ID2206, ID2200, IS1350
(2G1520, 2G1504, 2G1113)
Operativsystem
torsdag 2008-12-18 kl 0900-1300

Institutionen för Mikroeletronik och Informationsteknik

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 finnas tillgängligt via Daisy/LADOK senast under v3
- Studenter som tenterar ID2200 besvarar inte frågor märkta med ¤
- Studenter som tenterar IS1350 besvarar inte frågor märkta med *

Betygsgränser:
- Godkänt (E) garanteras från 50% av maximala poängen på tentan

---

Lösningsförslag: anslås på kursens webbsida senast v3

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

1) Kalle har fått sitt första jobb som konsult och ska bygga en enkel client-server lösning baserad på sockets. Hur många datorer, IP-adresser och processer (antag att han inte har tillgång till trådar) behöver han minst ha tillgång till och/eller skapa för att kunna utföra de enklaste testerna av sin lösning? (1p)

En dator på vilken man exekverar två processer (klient & server) som kommunicerar via en socket över loopbackinterfacet 127.0.0.1

2) i) Beskriv kortfattat vad som händer då ett systemanrop exekveras under UNIX. (1p)
ii) Förklara vilken process som exekverar koden som utför själva systemanropet. (skilj på anropet och exekveringen). (1p)

i) När man exekverar ett systemanrop anropas normalt en biblioteksrutin (i t.ex C). Biblioteksrutinen lägger parametrar och systemanropsnummret på på platser där systemanropskoden i kärnan förväntar sig hitta dessa, t.ex i register eller på förutbestämda adresser. Sedan exekverar biblioteksrutinen en trap-instruktion och kontrollen överförs till kärnan. Där kontrolleras vilket systemanrop som skall utföras och rätt kod i kärnan anropas. När systemanropet är färdigexekverat i kärnan återgår kontrollen till biblioteksfunktione feter trap-instruktionen. Biblioteksfunktionen tar hand om returvärdet och returenerar det till den funktion som anropade systemanropet.

ii) Normalt fortsätter den användarprocess (tråd) som anropade systemanropet att exekvera i kärnan - dock byter man stack till en kärnstack under det att man exekverar i kärnan.

3) i) Vad innebär det att kod är relokerbar? (1p)
ii) Vid vilken tidpunkt: kompilering, laddning, eller exekvering binds adresser i ett system med sidindelat minne? Varför? (0.5p)

i) Att koden inte behöver laddas på förutbestämda adresser i minnet utan är flyttbar (relokerbar), vilket åstadkoms genom att adresser offsett inte är absoluta utan relativa.

ii) Vid exekvering eftersom det är först då som man vet var och om det man adressar finns i minnet.

4) Antag att vi har en sidstorlek på 200 bytes.
i) Ge refrensströngen för följande minnesaccesser:

512, 614, 728, 398, 373, 328, 14, 18, 22, 354, 407, 302, 276, 865, 423, 496 (1p)
ii) Givet ett primärminne om 800 byte hur många sidfel fås vid ren demand paging för respektive: FIFO, OPT och LRU sidutbytesalgoritmerna. (1.5p)

i) Referenssträngen innehåller aldrig upprepningar av referenser till samma sida!!

Referenssträng: 2, 3, 1, 0, 1, 2, 1, 4, 2

ii) Utan att redovisa räkningarna vilket krävs för full poäng: FIFO 6, OPT 5, LRU 5

5) Antag att vi har ett sidindelat virtuellminnessystem på en dator med 4GB virtuell adressrymd, 2GB primärminne, 256 KB cache och 4KB sidstorlek. Antag vidare att ett process id är 16 bitar, att vi använder 5 bitar för information om tillståndet hos en sida, 3 bitar för accessrättigheter och att vi har en virtuellminneslayout för en process liknande den som används i t.ex UNIX. Hur stor blir då för en minimal process:
i) En två-nivåers sidtabell i det fall vi använder 10 bitar för att adressera den yttre sidtabellen (2p)
ii) En inverterad sidtabell (1p)

i) (mycket kortfattat) en adress delas upp i <10bitar yttre, 10 bitar inre, 12 bitar offset>
det krävs en ytter och två inre sidtabeller där de inre används för stack resp data+text. Den yttre sidtabellen innehåller 2^{10} poster där varje post är en pekare till en inre sidtabell, en pekare är 32 bitar eller 4 bytes, dvs storleken på den yttre sidtabellen är 4KB. En inre sidtabell innehåller också 2^{10} poster där varje post består av <5 bitar tillstånd, 3 bitar accessrättigheter, 20 bitar ramnr> = 26 bitar (vi ska ersätta de 20 bitarna i den virtuella adressen som vi använder för att indexera sidtabellerna med ett ramnr när man gör adressöversättningen från den virtuella till den fysiska adressrymden) Eftersom det är rimligt att anta att datastrukturer består av ett helt antal bytes så kan man anta att en post i en inre sidtabell också tar upp 32 bitare eller 4 bytes. Total storlek: 1 yttre + 2 inre sidtabeller blir då 12 KB.

ii) Antal poster i en inverterad sidtabell = antal ramar i det fysiska minnet = $2^{31}/2^{12} = 2^{19}$ = 0.5 M Varje post består av <16 bitar PID, 20 bitar sidnr, 5 bitar tillstånd, 3 bitar accessrättigheter> = 44 bitar. Återigen är det rimligt att anta att detta de facto tar upp ett jämnt antal bytes vilket ger en postlängd på 48 bitar eller 6 bytes. Hela den inverterade sidtabellen tar då upp 3 MB

6) Kalle har fått i uppdrag att förändra en operativsystemskärna så att den ska tåla högre last innan thrashing inträffar. Kalle föreslår följande åtgärder:
i) Om systemet börjar bli överlastat bör "paging demonen" köras mer sällan för att minska lasten på systemet.
ii) Sidstorleken bör ökas drastiskt
Är Kalles idéer bra eller dåliga (resonera, motivera)? (2p)

Kalles ideer är inte alltid lysande....
i) genom att minska hur ofta paging demonen körs så kan man vinna på att de andra processerna kan få mer exekveringstid och mer minne om paging demonen inte "ligger i minnet". Men om mna får thrashing genereras väldigt många sidfel i systemet och för att snabbt kunna hantera dessa så behövs en lista med lediga ramar som kan användas. Den listan genereras av paging demonen. Så om inte paging demonen kördes alldeles för ofta från början så förlorar man antagligen på Kalles förslag.

ii) Att öka sidstorleken är definitivt en dålig ide. Om man är utsatt för thrashing så betyder det att processerna inte samtidigt kan hålla det de behöver för sin exekvering i minnet. Att öka sidstorleken minskar då ytterligare sannolikhetehn för att det processerna behöver kan hållas i minnet då "upplösningen" minskar (dvs man får antagligen med sånt man inte behöver i de större sidorna) och då ökar sidfelsfrekvenserna ytterligare

7) ¤* Antag ett system med blandat vanliga processer och processer med mjuka realtidskrav. För att systemet skall fungera bra bör då sidutbytesalgoritmen/erna vara lokal, global eller en blandning av dessa typer? (1p)

En global sidutbytes algoritm ger normalt bärst prestanda globalt sett, men fungerar inte nödvändigtvis bra för mjuka realtidsprocesser. För att bättre skydda det de behöver ha i minnet är det oftast bättre att ha en lokal sidutbytesalgoritm för dessa. Dvs en kombination av global sidutbytes algoritm för processer utan realtidskrav och en lokal algoritm för de mjuka realtidsprocesserna fungerar antagligen bäst.

8) I en del moderna filsystem försöker man lagra filer i en eller flera "runs" av på disken konsekutiva diskblock? Vad är fördelarna och vad är nackdelarna? (2p)

Genom att lagra filer i konsekutiva (direkt på varandra följande) block på disken kan förbättra prestanda genom t.ex möjlighet till read-ahead och färre diskarmsrörelser, men det gör samtidigt allokeringsproblematiken svårare då de lediga blocken kan komma att ligga utspridda så att man vid nyallokering/omallokering av "runs" kan få svårt att hitta tillräckligt långa runs - dvs något som kan kallas en typ av fragmentering kan uppstå. Dessutom kan man behöva inte bara hålla ordning på de lediga blocken utan också hur de hänger ihop i "runs".

9) Vilken information hittar man i en i-nod i ett UNIX-filsystem (1p)

Se boken/föreläsningsanteckningar: hård länkräknare, id för ägare, id för skapare, tider för skapande/senaste access/senaste uppdatering, accessrättigheter rwx för ägare/grupp/alla användare, pekare till 10 direkta diskblock, pekare till single/double/triple indirekta diskblock

10) Antag att tre processer, A, B, och C i den ordningen har blockerats i väntan på att läsa olika filer från en och samma disk. I vilken ordning kan man anta att det blir körklara (dvs. i vilken ordning kan man anta att de går från tillståndet BLOCKED till READY)? Varför? (1p)

Man vet i normalfallet inget alls om i vilken ordning processerna blir klara eftersom I/O operationerna inte behöver bli klara i samma ordning som de startats eftersom en hårddisk ofta omordnar, dvs schemalägger operationerna i en annan ordning öän de kom in, av effektivitetsskäl t.ex för att får en bättre diskarmsschemaläggning.

11) Beskriv kortfattat "the Morris Internet worm"! (1p)

The Morris Internet worm är något av ett datorhistoriskt landmärke eftersom det var första gången Internet utsattes för en allvarlig och storskalig attack. Så det hör till allmänbildningen för en ingenjör inom ICT området att känna till basfakta om den.

Morris Internet Worm (MIW) bestod av två delar ett huvudprogram som upptäckte om systemet redan var infekterat och startade om MIW bara på var 7:e redan infekterat system. Huvudprogrammet upptäckte också vilka möjliga adresser som kunde infekteras när masken spred sig vidare. MIW utnyttjade säkerhetsbrister i sendmail, finger och rsh. Till ett system med säkerhetsläckor skickades ett litet program - the grappling hook (änterhaken) - som fick fotfäste på det nya systemet och sedan kopierade dit och startade huvudprogrammet. MIW var den första attacken som på mycket kort tid lyckades slå ut stora delar av Internet.

12) Pollning anses av många vara en mindre effektivt metod att styra I/O än avbrottstyrning eller DMA. Så är inte alltid fallet. Ge två exempel på där det kan vara effektivare/bättre att använda pollning än avbrott eller DMA! (2p)

Pollning kan vara att föredra på: i) Små system som av kostnads/komplexitetsskäl inte kan implementera avbrott eller DMA, ii) Vara effektivare än avbrott/DMA om den enhet man styr kan förväntas bli klar på kortare tid än det tar att hantera avbrott/DMA, iii) Om man har flera enheter att styra och man vill kunna garantera en rättvis behandling av alla enheter

13) I en del datorsystem har man infört skilda TLB:er för data repektive intruktioner. Varför gör man det och vad hoppas man vinna för på det? (1p)

Data och instruktioner behöver inte visa upp samma lokalitet. Man kan t.ex ha bra instruktionslokalitet men usel datalokalitet i ett program som arbetar med matriser (jfr datorteknik gk). Om man blandar data- och instruktionssidsöversättningar i en TLB kan man då riskera att datasidsöversättningar ersätter instruktionssidsöversättnigar i TLBn trots att de senare används oftare. Därför kan man ofta vinna prestanda på att ha skilda TLB:er för data och instruktioner.

14) Kalle är rädd att hans dator är utsatt för attacker från hackare som försöker tränga sig in på hans LINUX maskin när den är uppkopplad via Ethernet-porten till Internet. Han bygger därför in monitorering av Ethernetaktiviteten så att då ett PING-meddelande kommer in vi hans Ethernet så skickar Ethernetdrivrutinen omedelbart en signal till hans kommandotolk. I kommandotolken lägger Kalle till en signalhanterare som utan fördröjning skriver ut ett varningsmeddelande på skärmen när en signal från Ethernetdrivrutinen kommer in. När Kalle testar sitt nya varningsystem upptäcker han till sin förskräckelse att kommandotolken ofta beter sig konstigt i samband med att signaler om PING-meddelanden kommer in. Kalle tror att han är utsatt för hacker-attacker och vågar inte koppla upp sig mot Internet mer. Lisa tror däremot att problemen beror på något helt annat. Vad tror du är den troligaste orsaken till problemen? (1p)

Kalle har antagligen inte förstått att systemanrop som han gör riskerar att avbrytas utan att vara klara då signaler från Ethernet-porten kommer in, vilket kan resultera i konstiga utskrifter och att kommandotolken inte fungerar som normalt. (ska man ge Kalle ett råd - vilket inte ingick i uppgiften- så är det att skaffa en bra brandvägg som förhoppningsvis stänger ute eventuella attacker)

15) Antag att vi har ett operativsystem som har trådar som schemaläggningsenhet. Förklara varför det kan vara nödvändigt att ta hänsyn också till vilken process tråden tillhör för att få en bra schemaläggning och när den här kännedomen kan vara extra viktig? (2p)

Trådar inom samma kommunicerar antagligen med varandra och delar resurser som data, filer etc. inom en och samma process. Att schemalägga trådar som hör till samma process efter varandra på en en-processors maskin kan därför minska antalet context-switchar och bättra på utnyttjandet av cachear, TLB:er och minska antalet sidfel. Ännu mer kan vinnas på multi-core processorer där man kan anta att kommunikationen mellan trådarna i en och samma process kan förenklas och effektiviseras om de får exekvera samtidigt.

16) * Beskriv och motivera var, i kärnan eller som användarprocess, man kan implementera klient- repektive serverdelen av ett distribuerat filsystem om det skall vara transparent för användare och applikationer? (1p)

Access till filer går alltid via systemanrop - vilket gör att det normalt är enklast att implementera klient-delen som en del av kärnan. På serversidan spelar det ingen roll ur transparenssynpunkt om serberna är implementerad som en användarprocess (jfr. AFS) eller som en del av kärnan (jfr NFS)

17) Kalle hävdar att om en metod för synkronisering är bevisbart fri från deadlock (baklås) så är den därmed också fri från starvation (svältning). Har Kalle rätt eller fel? (bevisa/motbevisa påståendet) (1p)

Kalle har fel: Antag att vi har ett system där cirkulär väntan aldrig kan uppstå. Det systemet är bevisbart fritt från deadlock då ett av Coffmans nåvändiga villkor för deadlock är brutet. I systemet kan vi ändå ha processer som är utsatta för svältning då de väntar på att få resurser men aldrig får dem eftersom andra procerre hela tiden prioriteras före dessa.

18) Skriv i C main() funktionen för en minimal "ls"-trojan, som när användaren exekverar den beter sig exakt som UNIX/LINUX kommandot "ls" men som också anropar funktionen "badThingsHappen()" (1p)

```
int main(int argc, char *argv[])
{
    badThingsHappen();
    argc[0] = "ls";
    execvp("ls", argv);
}
```

19) (¤) Varför föredrar man i många fall att i realtidsystem använda schemaläggningsmetoden Rate Monotonic Scheduling (RMS) framför Earliest Deadline First (EDF) även om RMS inte klarar lika höga laster som EDF? (1p)

RMS har fixa prioriteter medan prioriteterna i EDF hela tiden ändras då job kommer närmare sin deadline. EDF är därför mer komplex att implementera eftersom man måste räkna om alla jobs prioriteter vid varje schemaläggningstillfälle.

20) ¤ En devicedriver består normalt av två delar. Vilka är det och vad är deras funktion? (1p)

Om vi använder UNIX trminologi och antar att enheten som styrs av drivrutinen kan generera avbrott (vilket är vanligast) så består drivrutinen av en top-half device driver som är den funktion som anropas av kärnan när den vill att drivrutinen skall utföra något. Bottom-half devicedriver är avbrottsrutinen som skall anropas då enheten signalerar via ett avbrott att den är klar med sin uppgift.

21) I flera varianter av LINUX schemaläggning kan motsvarigheten till UNIX-schemaläggningens time-slice påverka processens faktiska prioritet (goodness). Det kan tyckas vara en sammanblandning av två saker som borde vara oberoende. Beskriv hur detta sker och varför det fungerar utan större problem? (1p)

Se föreläsningsanteckningarna: Den schemaläggningsvariant vi studerat i LINUX bygger på att processer får en aktuell prioritet "goodness" som bland annat blir högre om en process/tråd har lång tid kvar att exekvera på sin individuella tidskvanta "quantum". quantum i sin tur beräknas om för alla processer när det inte finns några processer som kan exekvera längre då alla är blockerade eller har kört slut på sin quantum. quantum blir längre för processer med hög prioritet och för processer som är blockerade men inte har exekverat klart hela sin tidigare

quantum. Principen är nyttQuantum = kvarvarandeGammaltQuantum/2 + prioritet . Det gör att framförallt högprioriterade interaktiva processer som inte kör slut på sina quantum kan få både långa quantum och höga godness. Schemaläggningen fungerar ändå bra eftersom sådana processer med stor sannolikhet ändå snart blockeras och de kommer därför inte att svälta ut lägre priorioterade beräkningstunga processer.

"Att dra nytta av goda råd kräver mer förnuft än att ge dem"
Churton Collins