Lösningsförslag till tentamen i IS1350, ID2200 och ID2206 Operativsystem (med speciella hänvisningar till IS1350 materialet då detta var ordinarie tentamen för IS1350)

lördag 2011-10-22 kl 1000-1400

ICT skolan

Examinator: Robert Rönngren

Hjälpmedel: ***Inga*** - observera att mobiltelefoner måste stängas av och inte får finnas i närheten av skrivplatsen, medföras eller användas vid något tillfälle under tentamen

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 åtminstone en kortfattad motivering för att ge poäng
- Resultat beräknas anslås senast 14/9

Betygsgränser:
- Godkänt (E) garanteras från 50% av det totala poängantalet för respektive kurs

Frågor markerade med ¤ besvaras inte av studenter som tenterar ID2200

Lösningsförslag: anslås på kursens webbsida efter tentamen under v43

---

1) Systemanrop utgör API:t mot operativsystemet. Antag att du anropar `read(fd,buf,nbytes)`. Beskriv hur assemblerkoden (generellt) ser ut som gör själva systemanropet. Skriv pseudo-kod med kommentarer! (1p)

   Systemanrop sker normalt genom att koden för systemanropet och parametrarna läggs i register, sedan utförs en TRAP-instruktion med en parameter som talar om att det är ett systemanrop som utförs. (Efter det att systemanropet är klart fångar man upp returvärdet från systemanropet som normalt finns i ett register.) Nedan ett exempel på systemanrop i Intel-assembler:

```
mov $3,%eax          # Lägg koden för systemanropet read i eax
mov $fd,%ebx         # Ange att vi vill skriva på stdout i ebx
mov $buf,%ecx        # Lägg adressen till bufferten i ecx
mov $nbytes,%edx     # Lägg max antal bytes som skall läsas i edx
int $0x80            # Gör systemanropet, 0x80 är koden för systemanrop
```

2) Systemanropet `write(fd, buf, nbytes)` skriver data till en ström (stream)
   a) Förklara vad en ström (stream) är! ge också exempel på olika "typer" av strömmar. (1p)
   b) Förklara i detalj vad som sker i operativsystemet när systemanropet görs och hur data skrivs till rätt ställe (2p)

   a) En ström i C (UNIX/LINUX) är en abstrakt datatyp som man kan läsa bytes från och/eller skriva bytes till. I vissa fall kan man också positionera sig i strömmen. Exempel på strömmar kan vara vanliga filer på en hårddisk, dataströmmar till/från I/O enheter som till ett fönster (`stdout`) eller från tangentbord (`stdin`)

   b) TRAP-handlern (avbrottshanteraren) kontrollerar via koden som skickades med till TRAP instruktionen att det var ett systemanrop. Koden för systemanropet kontrolleras (jfr uppg1 `%eax`). Eftersom det handlar om att läsa från en ström som indentifieras med fildeskriptorn `fd` så måste avbrottshanteraren hitta rätt drivrutin att använda. I den inod som finns inläst i minnet (in-core i-node) så finns information om vilken drivrutin som ska användas i form av ett "major device number". Med det indexerar man sig i drivrutinstabellen och får fram en datastruktur (`struct`) som innehåller en vektor med funktionspekare till de funktioner som implementerar systemanropen (`open, read, write ...`) för denna I/O enhet. Den vektorn indexeras med systemanropsnummret och man anropar funktionen för `write` i devicedrivern via funktionspekaren. (kompletteras med bild!)

3) I många moderna operativsystem kan avbrottshanterare vara "stackade". Förklara vad det innebär! (1p)

Det betyder att avbrottshanteraren har en lista med funktioner att anropa för varje avbrott. Den anropar funktionerna i tur och ordning till dess att den senast anropade funktionen returnerat ett värde som indikerar att avbrottet har hanterats. Detta har två fördelar: man kan installera en kedja med funktioner för att hantera avbrottet där olika funktioner kan återanvändas mellan olika avbrott; och man kan installera funktioner som bara loggar/kontrollerar de olika avbrott som sker utan att funktionen hos avbrottshanteringen ändras (se även nästa uppgift)

4) En "keyboard-logger" är ett program som loggar (spelar in) alla tangenttryckningar som sker på en dator. Beskriv hur man gör för att implementera ett sådant program. (1p)

Om man har ett OS med stackad avbrottshantering och man har root-rättigheter kan man enkelt installera en ytterligare avbrottshanteringsfunktion för tangentbordsavbrottet som körs innan den ordinarie funktionen för hantering av avbrottet. Dess funktion är att kontrollera vilken tangent som tryckts, logga denna på någon plats t.ex i en fil i filsystemet och sedan returnera att funktionen inte hanterat avbrottet. Då kommer avbrottshanteraren att köra nästa funktion i listan vilket är den ordinarie funktionen för att hantera tangentbordsavbrottet.

5) I processschemaläggning förekommer två begrepp: "time slice" och "context switch".
a) förklara dessa begrepp! (1p)
b) hur bör "time slice" förhålla sig till tiden för att göra en "context switch"? Varför? (1p)

a) timeslice: den maximala tid en process får köra innan den plockas bort från processorn (om den inte tidigare blockerats av I/O eller terminerat) på ett system med avbrytande (preemptiv) schemaläggning. Kan vara fix, dvs lika för alla processer, eller individuellt anpassad per process.
context switch: byte av den omgivning man exekverar i, oftast liktydigt med processbyte men kan också användas som begrepp för att för att en process t.ex. går mellan user och kernel mode.

b) timeslice bör vara relativt lång i förhållande till tiden för context switch för att datorsystemet skall utnyttjas effektivt (dvs för att en stor andel av CPUns arbete skall gå åt till att köra användarprocesser och inte åt administrativa uppgifter som context switching)

6) När man går från att ha processer med en tråd till system med möjlighet att ha flera trådar i en process kan minneshanteringen bli mer komplicerad för programmeraren. Förklara varför! (1p)

Varje tråd behöver en egen stack. I det singeltrådade fallet kan man utnyttja den virtuella adressrymden maximalt (åtminstone i teorin) genom att låta stack och heap växa mot varandra från var sitt håll. Om man måste ha utrymme för flera stackar i den virtuella adressrymden måste man reservera plats för dessa och de kommer också att vara av fixerad storlek. För programmeraren kan det betyda att man måste kunna förutsäga hur många och hur stora stackar trådarna kommer att behöva - vilket kan vara besvärligt.

7) I processer med prioritetsbaserad schemaläggning finns ofta mekanismer som förändrar den prioritet som processerna/trådarna schemaläggs efter (sk. "current priority").
Förklara vilka huvudsyften dessa mekanismer har! (2p)

Schemaläggningen skall generellt sett vara rättvis och åstadkomma en effektiv schemaläggning för det syfte systemet har. Rättvisekravet betyder att alla processer bör få exekvera inom en rimlig tidshorisont och en effektiv schemaläggning bör prioritera processer som är viktiga för systemet. Därav finns i de flesta operativsystem mekanismer som förändrar den basprioritet processen (tråden) fick när den skapades. Dessa mekanismer kan antingen syfta till A) att alla processer får exekvera. Processer med låg prioritet riskerar att aldrig få exekvera om det hela tiden finns körklara processer med högre prioritet. Då kan mekanismerna antingen temporärt höja prioriteten för processer med låg prioritet eller sänka prioriteten för processer som fått exekvera mycket. Mekanismerna kan också syfte till att B) processer som anses viktiga tillfälligt kan få höjd prioritet. Ett exempel kan vara i interaktiva system där processer som väntat på I/O kan få höjd prioritet när I/O är klar för att förbättra responstiden.

8) Förklara hur adressöversättningen går till i den enklaste formen av Virtuellt sidindelat minne (dvs utan TLB). (2p)

Programmeraren ser en virtuell adressrymd som han/hon uppfattar som kontinuerlig. Den virtuella adressrymden delas in i block som kallas sidor av fix storlek. Sidstorleken är normalt en jämn två-potens. Det fysiska minnet delas in i block av samma storlek som sidorna. Blocken i det fysiska minnet kallas ramar. När

en process startas ligger dess sidor normalt sett på hårddisken i an area som kallas SWAP-area. När processen exekveras lägger CPUn ut adresser i den virtuella adressrymden. Dessa delas in i två fält: de n lägsta bitarna (med sidstorlek 2^n) används för att adressera sig inom en sida, övriga bitar utgör sidnummer. När en virtuell adress skall accessas tar MMUn (memory management unit) hand om adressen. Den tar sidnumret och slår upp motsvarande post i processens sidtabell. I sidtabellen finns indikerat om sidan finns i en ram i primärminet eller om den ligger på disk. Om sidan ligger på disk letar MMUn upp en ledig ram i primärminnet att lägga in sidan i, finns ingen ledig ram slängs en sida som finns i minnet ut till disken, sedan läses sidan in till primärminnet och sidtabellen uppdateras. När sidan ligger i primärminnet innehåller posten för sidan i sidtabellen information om vilket ramnummer sidan ligger i. Den virtuella dressen översätts till en fysisk adress genom att sidnumret ersätts med ramnumret. (kompletteras med bild)

9) Vad är en TLB, till vad används den, vad innehåller den och hur uppdateras den? (2p)

En TLB är en typ av cacheminne för de mest frekventa översättningarna från sidnummer till ramnummer och finns i MMUn. Avsikten är att snabba upp adressöversättningar från virtuellt till fysiskt minne. Den implementeras med hjälp av så kallat CAM minne, en typ av minne där man kan söka igenom hela minnet efter ett nyckelvärde (i detta fallet sidnummer och PID). Den innehåller normalt sett ett begränsat antal poster i storleksordningen 32-256 poster. TLBn innehåller normalt översättningar för fler än en process varför den innehåller <söknyckel: PID, Sidnr : information: motsvarande post ur sidtabellen för processen med PID>, posten ur sidtabellen innehåller normalt <ramnr, ref-bit, modify-bit, accessrättigheter, cachening tillåten bit>. Om man får en miss i TLBn görs normalt en TRAP till OS:et som går till sidtabellen för att hitta informationen (eventuellt behandlas också ett ev. sidfel) och som uppdaterar TLBn med informationen.

10) Beskriv vad en länkare (linker) gör? (1p)

En länkare tar ett antal objektkodsmoduler och sätter samman dessa tillsammans med ev. biblioteksfunktioner som används till en exekverbar modul. De olika objektkodsmodulerna har normalt relativ adresserad kod, dvs alla adresser i koden anges relativt start för modulen, dvs adress 0. För att bygga en modul behöver dessa adresser modifieras beroende på var, relativt starten på den sammansatta modulens start, som de hamnar.

11) En student har svarat på en fråga där det gällde att räkna ut antalet sidfel som genereras med ren "demand paging", tre ramar och följande referenssträng: 1,4,3,8,3,6,4,2,1,3,2,7,6,1,8 Studenten redovisade räkningarna nedan som visade på att man fick 10 sidfel (markerade med x).
a) Vilken var sidutbytesalgoritmen och hur kan man se det? (1p)
b) Hur många sidfel hade genererats med samma sidutbytesalgoritm och 4 ramar? (1p)

```
1 4 3 8 3 6 4 2 1 3 2 7 6 1 8
x x x x   x   x x     x x   x
1 4 3 3 4 4 3 3 3 2 1 1 1 1 8
  1 4 4 3 3 6 2 2 1 2 2 6 6 1
    1 8 8 6 4 6 1 3 3 7 2 2 6
```

a) sidutbytesalgoritmen var den optimala vilket man ser på att om en sida byts ut är det den sida som det dröjer längst innan den används som byts ut

b) 8 sidfel - räkningarna måste redovisas för att få poäng

12) Vilken information kan man förvänta sig att hitta för en vanlig fil i en katalog ("directory") i ett UNIX filsystem? (1p)

filnamn, inodsnr - dvs en översättning från det filnamn användaren ser till numret för den inod som implementerar filen

13) Kalle hävdar att det i moderna filsystem är bättre att lägga informationen om en fils accessrättigheter i den datastruktur som implementerar filen än i katalogen (directoryt). Har Kalle rätt eller fel? Varför? (1p)

Kalle har rätt då moderna filsystem ofta tillåter att man har flera mjuka och ofta också hårda länkar till en och samma fil. Om accessrättigheterna ligger i datastrukturen som implementerar filen säkerställer man att man alltid ser samma accessrättigheter oavsett vilken hård eller mjuk länk man använder för att accessa filen. Om accessrättigheterna låg i katalogerna skulle en och samma fil kunna ha olika accessrättigheter beroende på genom vilken katalog den accessades vilket inte är konsistent.

14) Vad vinner man på att göra mer intelligent schemaläggning av diskaccesser än FCFS och var implementeras detta? (1p)

Genom att välja en annan schemaläggning än First Come First Serve så kan man minska diskarmsrörelserna och på så sätt minska overheaden för att accessa disken. Detta kan antingen implementeras i filsystemet i OS:et - men det kräver att OS:et vet hur disken är organiserad i spår och sektorer. Moderna diskar visar ofta en logisk struktur för OS:et, dvs den fysiska disklayouten kan avvika från vad OS:et ser. I dessa fall kan en bättre diskarmsschemaläggning implementeras i drivrutinen för hårddisken - som oftast tillhandahålls av leverantören av hårddisken.

15) Kalle är systemadministratör för en maskin som används av studenter på en mindre högskola. Han har upptäckt att en mindre grupp studenter som kör laborationer med stora datamängder, t.ex numerisk analys, hållfasthetsproblem och databaser, ofta belastar maskinen hårt. Han har också upptäckt att genom att öka sidstorleken för dessa studenters processer från ~2^{10}B till ~2^{20}B så minskar deras belastning på maskinen väsentligt.
a) Förklara vad den troliga orsaken till detta är! (1p)
b) Är det en bra ide för Kalle att öka sidstorleken för alla studenter till ~2^{20}B? (1p)

a) Att ändra sidstorleken ändrar inte på hur datat accessas, dvs det förändrar inte lokaliteten. Däremot kommer man att behöva färre sidor för samma datamängd vilket kan göra att man kan få bättre träffsannolikhet i TLBn om man har applikationer med stora datamängder. Detta påverkar i sin tur hur mycket arbete som OS:et måste göra vid TLB missar för att uppdatera TLBn. Det kan minska den upplevda minnessaccesstiden för processen dramatiskt och minskar naturligtvis processens last på systemet.

b) Nej det är en dålig ide att öka sidstorleken generellt. Detta av två skäl: det viktigaste är man kan anta att de flesta processer har relativt god lokalitet och/eller små datamängder. Ökar man sidstorleken för dessa fyller man primärminnet inte bara med data som används utan även med data som inte används. Dvs man minskar precisionen i vad man laddar in i primärminnet (som fungerar som en typ av cache). Det gör att man inte längre kan köra lika många processer parallellt på ett effektivt sätt. Det andra skälet är att man också riskerar att få mer intern fragmentering i sidorna med ökande sidstorlekar.

16) Filer kan ses som en datatyp. Vilken är den största skillnaden mellan filer som datatyp i UNIX filsystem och i Windows filsystem från och med NT? och vad är användningen? (1p)

Den viktigaste skillnaden som datatyp är att filer i UFS bara innehåller en dataström medan filer i WINDOWS kan innehålla flera dataströmmar. Det senare är intressant för olika typer av applikationer som t.ex för ordbehandlare där en ström kan vara tecknen i filen och en annan kan vara ändringarna som gjorts, eller för multimedia där man kan ha en ström för bildinformation och flera strömmar för ljud på olika språk.

17) ¤ Ge ett tillräckligt villkor för att en realtidsschemaläggning på en en-kärnig processor skall vara optimal! (1p)

Att schemaläggningen sker i strikt ökande ordning efter deadlines.

18) ¤ Schemaläggning av hårda realtidssystem på flerkärniga processorer medför problem.
a) vilka nya problem uppstår på flerkärniga maskiner relativt om man kör på en en-kärnig processor? (1p)
b) vilken/vilka typer av schemaläggningsmetoder - statiska och/eller dynamiska - är att föredra för hårda realtidssystem på flerkärniga processorer? (1p)

a) Schemaläggningen måste ta hänsyn inte bara till när ett jobb skall få köra utan även också på vilken kärna det skall få köra vilket leder till att schemaläggningsproblemet generellt sett blir NP-komplett.
b) Idag är statiska metoder att föredra då man i förväg räknar ut om en schemaläggning möter alla deadlines.

19) Vilket av Coffmans villkor bryter man normalt mot i operativsystem som Windows, OS-X, Linux vid:
a) processschemaläggning (1p)
b) utskrift till skrivare (1p)

a) Dessa OS använder preemptiv (avbrytande) schemaläggning och bryter därför mot villkoret om "No pre-emption"
b) Utskrift sker normalt via sk. spoolning vilket gör att skrivaren upplevs som en delad resurs. Det bryter alltså mot villkoret ömsesidig uteslutning

20) a) vad betyder det att man static-deklarerar en funktion i C (semantiken) (1p)
b) varför static-deklareras normalt funktioner som ingår i drivrutiner och för avbrottshantering (1p)
c) med vilken mekanism/hur anropas funktioner i drivrutiner från kärnan (1p)

a) att namnet på funktionen bara är synligt inom den moduls om kompileras (översätts), dvs bara inom källkodsfilen (jfr private i objektorienterade språk)
b) detta är för att undvika namnkollisioner, drivrutiner och avbrottsrutiner har ofta gränssnitt med i vilket väldefinierade och förutbestämda funktioner skall ingå som t.ex open, read, write... Om man då laddar in flera moduler i kärnan med som har samma namn på funktionerna skulle man inte kunna reda ut (resolve) vilken funktion som skulle anropas när, t.ex om man laddar in flera funktioner med namnet write som tar samma parametrar.
c) funktioner i drivrutiner och för avbrottshantering installeras i kärnan genom att pekare till funktionerna görs tillgängliga i kärnan, dvs de anropas av kärnan via funktionspekare

21) I Unix/Linux finns verktyg som heter truss/strace/Dtrace.
a) Vad gör dessa verktyg? (1p)
b) Vad är likheten mellan hur dessa verktyg fungerar och hur många virus fungerar? (1p)

a) alla verktygen kan användas för att se vilka systemanrop som en process anropar (Dtrace kan göra en hel del mer också)

b) Verktygen installerar sig så att de körs då TRAP-handlern hanterar systemanrop. Många virus har också behov av att se vad som görs på en dator, antingen för att hitta exekverbara filer att infektera eller för att spionera. Dessa installerar sig på motsvarande sätt som strace/truss/Dtrace så att de körs då TRAP-handlern skall hantera systemanrop.

"Bara den okunnige tror han kan allt"