Tentamen i ID2206, ID2200 och IS1350 Operativsystem måndag 2012-01-09 kl 0900-1300

KTH 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, markera besvarade frågor på utdelat formulär.

Skriv svar bara på en sida av svarsbladen. Varje inlämnat blad skall förses med följande information:
- Namn och personnummer
- Nummer för behandlade uppgifter
- Sidnummer

ID2200: frågor markerade med ¤ ingår ej i tentamen
IS1350: frågor markerade med * ingår ej i tentamen

Rättning:
- ***Alla svar, även på delfrågor, måste ha åtminstone en kortfattad motivering för att ge poäng***
- Resultat beräknas anslås senast 120102

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

Lösningsförslag: beräknas anslås på kursens webbsida v2

Frågor:

1) Programmeringsspråket C är antagligen det mest använda språket för att programmera operativsystem och inbyggda system. Ge minst tre skäl till varför man ofta väljer att använda C framför assembler eller språk med högre abstraktionsnivå som objektorienterade språk för sådana tillämpningar. (1.5p)

Skäl att använda C före assembler: bättre portabilitet och högre abstraktionsnivå
Skäl att använda C framför andra högre nivå språk: tillåter explicita pekare, har bitvisa operatorer/ operander, effektivt (ingen GC etc.), enklare att kontrollera var i minnet data placeras.

2) När man skapar en ny process med systemanropet `fork()` returneras 0 till barnprocessen (den nya processen). Kalle föreslår att man istället skall returnera processId för föräldern till barnet. Red ut om detta är en bra eller dålig idé! (1p)

Kalles idé gör det mer komplicerat att direkt efter en fork() kunna se om man exekverar i barnprocessen eller i föräldraprocessen eftersom man i båda processerna måste antingen kontrollera sin egen processID eller förälderns för att kunna avgöra om man är i barnet eller föräldern. Dvs Kalles idé är inte bra.

3) a) Vilket är det främsta principiella problemet man måste ta hänsyn till då man inför prioritetsbaserad schemaläggning av processer/trådar? (1p)
b) Beskriv hur operativsystem brukar förhindra/mildra att problemet/en uppstår! (1p)

a) genom att införa prioriteter riskerar man att införa svältning, dvs. att lågt prioriterade processer aldrig kommer åt att få köra
b) genom att använda en temporär (tillfällig eller sk. "current") prioritet för schemaläggningen som kan avvika från basprioriteten och som kan förändras för att få en rättvis schemaläggning. Oftast hanterar man den temporära prioriteten genom sk. "aging" vilket betyder att man antingen sänker prioriteten för processer som fått exekvera mycket eller höjer prioriteten för processer som inte fått exekvera elelr som fått exekvera lite.

4) Kalle har hittat ett verktyg som i realtid kan visa tillståndet hos en process: READY, RUNNING, BLOCKED. Han kör på en 10 år gammal PC. Hans problem är att den process han följer går mellan tillstånden READY

och BLOCKED men han ser aldrig att den befinner sig i tillståndet RUNNING. Förklara vad Kalles problem beror på? (1p)

För att gå från READY till BLOCKED måste processen ha passerat tillståndet RUNNING, dvs ha körts. På en 10 år gammal PC kan man anta att man bara har en processorkärna alltså kan bara en process (eller OS:et) exekvera i taget. När applikationen exekverar och kan läsa av tillståndet för processen som Kalle monitorerar kan alltså inte den monitorerade processen exekvera. Applikationen ser alltså aldrig att "Kalles process" exekverar.

5) Vilka delar av den virtuella adressrymden är delade mellan trådar i en process? (1p)
   i) Stack
   ii) Heap
   iii) BSS
   iv) Data
   v) Text

Samtliga utom stacken som varje tråd har en egen privat. (BSS Block Started by Symbol, dataarea för globala variabler med odefinierat eller nollställt initialvärde)

6) En process har 37 KB stack, 13 KB data, 4KB BSS och 7 KB text.
   a) Om sidindelat minne används och en sida är 4KB stor - hur många sidor används då för processen? (1p)
   b) Hur mycket minne tar den del av en två-nivåers sidtabell som hålls i minne för processen ovan om en adress är 48 bitar, 18 bitar används för att adressera den yttre sidtabellen och en post i en yttre/inre sidtabell är 32 bitar lång? (1p)

a) Stack 10, data+BSS 5 och text 2 sidor, totalt 17 sidor
b) En virtuell adress delas in i: 18 bitar för att adressera inom den yttre sidtabellen, 18 bitar för att adressera inom inre sidtabell och 12 bitar för att adressera sig inom sidan. En inre sidtabell kan alltså innehålla 2^{18} översättningar från sidnr till ramnr. Det går alltså åt en inre sidtabell för stacken och en för data+BSS+text. Totalt går det åt en yttre och två inre sidtabeller: $2^{18}*4 + 2*2^{18}*4$ B = $3*2^{20}$ B = 3 MB.

7) När kan man anta att adressöversättningen sker till fysiska adresser i (förklara varför!):
   i) en persondator (1p)
   ii) ett hårt realtidssystem (1p)

i) man kan anta att en PC kör ett OS med sidindelat virtuellminne, dvs adressöversättning sker under exekveringen
ii) ett hårt realtids OS använder normalt fysiska adresser, dvs adressöversättningen sker innan exekveringen vid kompilering, länkning eller som senast vid laddning av processen/programmet

8) Vilken datastruktur är troligast att man använder för att implementera sidtabellen i ett modernt OS för persondatorer eller stordatorer? Förklara varför denna datastruktur är att föredra jämfört med alternativen! (2p)

Moderna OS använder normalt stora virtuella adressrymder (>= 2^{64} B) Normalt använder man i sådana OS hashade inverterade sidtabeller. En en-nivås vanlig sidtabell skulle bli för stor (proportionell mot storleken på den virtuella adressrymden). Fler-nivåers vanliga sidtabeller kan göras proportionella mot storleken på den del av den virtuella adressrymden processen utnyttjar men skulle antagligen få väldigt många nivåer och därmed blir det (för) många extra minnesaccesser vid TLB-miss/sidfel. En inverterad sidtabell är proportionell i storlek mot det fysiska minnet men kräver i sin enklaste implementation en sökning i tabellen för att se om en viss sida finns inladdad i det fysiska minnet. Det enda alternativ som blir rimligt i storlek och som kan ge (nära) konstant uppslagstid är inverterade hashade sidtabeller.

9) a) Förklara hur datastrukturerna för att hålla reda på vilka block som ingår i en fil ser ut för implementation med FAT respektive i-noder (1p)
   b) Vad är den förväntade tidskomplexiteten för att, med hjälp av FAT respektive i-noder, identifiera vilket fysiskt diskblock som innehåller ett sökt block i en fil. (1p)

a) (RITA FIGURER) FAT implementeras via en tabell (vektor) i vilken man implementerar en enkellänkad lista med index till diskblocken. i-noder innehåller en lista med direkta länkar till de första (~10) diskblocken i filen, en pekare till ett block med pekare till nästa (~10) diskblock, en

pekare till block med pekare till block med index till diskblock, en pekare med pekare till block med pekare till block med pekare till block med index till diskblock (triple indirection). Dvs en typ av trädstruktur med index till filens diskblock.
b) FAT: söktiden i en enkellänkad lista är av komplexitet O(n) där n är antalet diskblock i filen. i-node: söktiden i ett träd är av komplexitet O(log(n))

10) Lokalitet är viktig i de flesta sammanhang i datorsystem. Beskriv minst två sätt på vilka man kan utnyttja lokalitet i implementation av filsystem och vad de ger för fördelar. (2p)

i) Användaren har oftast ett "working directory" som är den "lokalitet" i filsystemet man för tillfället använder, informationen i working directory läses ofta in och hålls tillgänglig i OS:et. ii) för implementationen av en enskild fil på hårddisken försöker man ofta utnyttja diskblock som ligger fysiskt nära varandra på disken. iii) i system som använder i-noder kan man försöka lägga dessa fysiskt nära de block som används för filerna, t.ex i samma eller närliggande spår.

11) Vad är det viktigaste förutom filer, kataloger och länkar som man normalt kan hitta i och som görs synligt i filsystemen? (1p)

Drivrutiner för fysiska och virtuella enheter installeras så att de syns i filsystemen, antingen i /dev katalogen i UNIX/LINUX eller i Registryt på Windows maskiner.

12) När man gör konsistenskontroller på filsystem kan man göra "block konsistens" och/eller "filsystems konsistens" kontroller. Vad kontrollerar man i de två fallen och vilka typer av fel kan man upptäcka? (2p)

Block-konsistens: man kontrollerar att varje diskblock bara finns en gång i antingen en fil, i listan med lediga block eller i listan med dåliga block. Finns ett block både i en fil och i listan med lediga block tas det bort från listan med lediga block. Finns blocket inte i någon lista är det ett "missing block" och bör läggas till i listan med lediga block (eller om det inte går att läsa/skriva i listan med dåliga block). Finns blocket i två filer bör man kopiera blocket och se till att filerna använder olika block (en fil är/blir antagligen trasig men filsystemet blir korrekt)
Filsystems-konsistens: Man kontrollerar att antalet hårda länkar från katalogstrukturen stämmer överens med länkräknaren i implementationen av varje fil/katalog (t.ex i-noden). Om länkräknaren i implementationen av filen/katalogen avviker från antalet länkar från katalogstrukturen uppdaterar man länkräknaren i implementationen av filen/katalogen (t.ex. i-noden) till att avspegla det faktiska antalet hårda länkar till filen/katalogen.

13) a) Beskriv hur I/O med pollning fungerar. (1p)
b) När kan det vara fördelaktigt att använda pollning jämfört med avbrottstyrd eller DMA orienterad I/O? (1p)

a) Vid pollning (programstyrd I/O) läser man och skriver i I/O-enhetens data- och kontrollregister direkt från ett program utan att använda avbrott eller DMA. Ex. En utskrift med pollning sker genom att programmet kontrollerar enhetens kontrollregister för att se om enheten är redo att skriva ut nästa data. När enheten är redo skriver man datat till dataregistret och uppdaterar kontrollregistret/n med information om att data finns för utskrift (och ev. också vart data skall skrivas t.ex vilken sektor på en hårddisk)). Sedan läser programmet av kontrollregistret för att se att data skrivs ut och fortsätter ev. med nästa data som skall skrivas ut.
b) pollning kan vara att föredra om: man inte har råd med avbrott eller DMA, väntan på att en operation skall bli klar är kort och i vissa fall kan det t.ex vara enklare att implementera en rättvis fördelning av I/O mellan flera enheter genom pollning

14) a) Beskriv sidutbytesalgoritmen Clock! (1p)
b) Hur uppskattar Clock processens "working set"? (1p)

a) Clock håller en cirkulär länkad lista i FIFO-ordning med information om de sidor som finns inladdade i det fysiska minnet, specifikt finns en "referensbit" som ett-ställs varje gång en sida refereras. När en sida behöver bytas ut i minnet (dvs när man behöver frigöra en ram) så skannar man igenom listan och kontrollerar referensbiten för varje sidan man tittar på. Är referensbiten satt, nollställer man den och går vidare i listan. Den första sida som har sin referensbit nollställd är den som byts ut (victim page).
b) Normalt kan man anta att man inte behöver skanna alla sidor för att hitta en sida med nollställd referensbit och/eller att man inte får sidfel i varje instruktion. Det betyder att sidor som fått sin referensbit nollställd i skanningen och som ligger i working set (dvs som används av processen) normalt har en god chans att hinna användas igen, och därmed få referensbiten satt, innan den skannas igen av Clock. Det ger en hygglig approximation av working set.

15) Varför kan en flertrådad process ibland exekvera snabbare även på en dator med en enda processorkärna än motsvarande process som bara utnyttjar en tråd? (1p)

Omm man har kärntrådar kan man överlappa I/O och beräkningar. I en enkeltrådad process blockeras hela processen om man gör blockerande I/O. I en flertrådad process blockeras bara den tråd som gör I/O om man har kärntrådar.

16) I distribuerade filsystem måste man ta hänsyn till konsistenproblematiken.
*a) Förklara begreppet konsistensproblematik? (1p)
*b) Vad innebär UFS-semantik och sessionssemantik? (1p)
*c) Vilken av ovanstående två semantiker är enklast och/eller bäst att implementera i ett distribuerat filsystem? (förklara varför) (1p)

a) I ett distribuerat filsystem kan man anta att det finns flera olika kopior (versioner) av en fil. Konsistensproblematiken är att se till att alla användare helst skall se samma version.
b) Unix File System semantik betyder att alla användare omedelbart ser alla ändringar som sker till en fil. Sessionssemantik innebär att en användare arbetar med en kopia av en fil och att ändringar blir synliga för andra användare först då användaren avslutar användadet av filen (sessionen).
c) Sessionssemantik är enklare att implementera. UFS-semantik är svårare att implementera, svårt att göra på ett effektivt sätt och ofta finns andra mekanismer som block(buffer) cache där man cachear diskblock på ett sätt som gör att man ändå inte får ren UFS semantik.

17) Moderna datorarkitekturer har ofta flera processorkärnor på samma chip.
*a) Vad är det viktigaste problemet ett OS för flerkärniga system måste lösa? (1p)
*b) Vilket är det viktigaste problemet att hantera för en programmerare som parallelliserar en applikation?(1p)

a) schemaläggning: dvs var (på vilken kärna) och när en process/tråd skall få exekvera.
b) synkronisering av de parallella aktiviteterna i applikationen.

18) a) Förklara vad det innebär att ett villkor för något (t.ex X) är "nödvändigt" respektive "tillräckligt". (1p)
¤b) Ge ett tillräckligt villkor för att en realtidsschemaläggning på ett en-processorssystem skall vara optimal. (1p)

a) Ett nödvändigt villkor för X måste vara uppfyllt för att X skall kunna inträffa (X skall kunna vara sant). Att ett nödvändigt villkor är uppfyllt betyder inte att X inträffat (eller att X är sant). Ex. ett nödvändigt villkor för att en glödlampa skall lysa är att det ligger en tillräckligt stor spänning över glödlampan. Ett tillräckligt villkor är ett villkor som om det är uppfyllt innebär att X inträffat (eller att X är sant). Ex. ett tillräckligt villkor för att en glödlampa skall lysa är att glödlampan här hel och att det ligger en tillräckligt hög spänning över den.
b) Jackssons regel: att jobben schemaläggs i strikt ökande deadline ordning

19) I Sverige finns två typer av fyrvägskorsningar mellan vägar av samma status: Antingen tillämpas högerregeln eller så finns STOP-skyltar på alla fyra tillfartsvägar. I det första fallet skall man alltid lämna företräde för trafik som kommer från höger. I det andra fallet så gäller att alla bilar stannar vid sin STOP-skylt och att den som först stannat vid sin STOP-skylt också får köra först.
Analysera de två olika modellerna för fyrvägskorsningar med avseende på baklås och svältning. (3p)

I brist på enkla möjligheter för mig att rita: kalla tillfartsvägarna **N**orr, **Ö**ster, **S**yd och **V**äster. Högerregeln: Kan råka ut för både svältning och baklås (deadlock). Svältning: Antag ett det hela tiden kommer bilar från **Ö** som kör mot **V**, så kommer en bil som vill köra från **S** mot **N** att bli svält. Baklås: Om det står bilar vid alla tillfarter (NÖSV) så får vi baklås.
Fyrvägs-STOP: Kan inte få svältning eller baklås. Baklås: Under försutsättning att två fordon på olika tillfartsvägar inte kan komma fram exakt samtidigt till sin respektive STOP-skylt så kommer vi att få en strikt ordning i vilken fordonen får passera korsningen. Dvs vi bryter mot Coffmans villkor om cirkulär väntan. Svältning: en bil som står i kö för att köra igenom en korsning och har N bilar framför sig i sin kö vet att det som mest kommer att vara 4N bilar som kommer att få köra genom korsningen före.

20) Beskriv vad ett virus kan göra om det lyckas ta kontroll över:
a) Hårdvaruavbrottshanteraren (interrupt handler) (1p)
b) Mjukvaruavbrottshanteraren (TRAP handler) (1p)

a) Man kan följa den mesta I/O som görs på maskinen. Specifikt kan man logga alla tangentbords och pekdons (mus, plattor) tryckningar som sker.
b) Man kan se alla systemanrop som sker, dvs man kan få en bild av i princip allt som sker på datorn.

21) Microsoft har för flera av sina operativsystem använt en objektorienterad konvention för programmering i vanlig C. Beskriv vad ett "objekt" i denhär konventionen är och hur det implementeras i C. (1p)

Ett objekt i det här fallet är den enklaste formen av objekt med instansvariabler och funktioner utan stöd för arv och polymorfism i språket. Ett objekt implementeras som en "struct" med data- och funktionspekarfält.

"Alla morgondagens blommor gömmer sig i de frön som sås idag"
Kinesiskt ordspråk