Preliminära lösningsförslag till Tentamen i ID2206, ID2200, IS1350 Operativsystem
tisdag 2010-12-14

Examinator: Robert Rönngren

Rättning:
- ***Alla svar***, även på delfrågor, måste ha åtminstonde en kortfattad motivering för att ge poäng
- Resultat beräknas anslås senast 11/1

Betygsgränser:
Godkänt (E) garanteras från 50% av det totala poängantalet för den kurs som tenteras

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

1) Beskriv det som i UNIX-terminologi kallas ett "filter" ? (1p)

Ett filter är ett program som läser indata från fildeskriptor 0 (stdin), gör något med det den läser in och sedan skriver utdata till fildeskriptor 1 (stdout). (jfr lab 1 ID2200/ID2206). Exempel på filter är awk, sort, grep, less, more mfl. (man behöver inte ha med exempel för att få full poäng)

Vanliga fel: Filter ska inte förväxlas med inter-process kommunikationsmekanismen pipe(). pipes kan användas för att koppla ihop filter så att utdata från ett filter blir indata till nästa.

2) ¤ Kalle har skrivit en minimal kommandotolk (se kod nedan) som han provkör med enkla kommandon som "`ls`". Antag att han inkluderat nödvändiga `.h` filer.
   a) Exekveras kommandona i förgrunden eller bakgrunden? Varför? (1p)
   b) Vad händer om han försöker köra kommandot "`ls -la`"? (1p)

```
int main(int argc, char *argv[])
{ char command[81]; int status;
  while(1) {
    printf("next command>");
    fgets(command, 80, stdin);
    if(fork() == 0){
       execlp(command, command, NULL);
       perror(command);
       exit(1);
    }
    while(waitpid(-1, &status, WNOHANG));
}}
```

a) Kommandona exekveras i bakgrunden eftersom kommandotolken inte väntar till dess att det kommando den startat terminerar - den gör bara en mycket enkel pollning i waitpid(), där första parameterns process id= -1 indikerar att man kontrollerar om status förändrats för någon av samtliga barnprocesser. Parametern WNOHANG indikerar att waitpid() returnerar direkt om ingen barnprocess terminerat. (jfr lab 2 ID2200/ID2206)

b) Det fungerar inte eftersom det inte finns någon exekverbar fil eller inbyggt kommando som matchar strängen "ls -la\n". Faktum är att inga kommandon går att köra eftersom Kalle inte rensar bort nyradstecknet ur kommandoradssträngen (command) innan han försöker starta kommandot.

Vanliga fel: Om en process exekveras i bakgrunden eller förgrunden har inget att göra med om den exekveras som barn eller som förälder. Normalt exekveras kommandon i en kommandotolk i barnprocessen eftersom man annars skulle byta process ID på det som uppfattas som kommandotolk (om barnprocessen skulle fortsätta exekvera kommandotolken skulle den fortsätta på exakt samma ställe i koden, dvs. man skulle ha kvar en kommandotolk, men den skulle ha ett annat PID än det man startade med)

3) I Kernighan & Ritchies bok "The C-programming language" beskrivs en enkel implementation av en minneshanterare (i.e. `malloc(), free()`...) där information om blockstorlek, pekare till nästa block etc. läggs i minnesblocken omedelbart före den adress som lämnas ut av `malloc()`.
   a) `malloc(), free()` hanterar minne på heapen. Varför har man inte motsvarande minneshanterare för

minne på stacken? (1p)

b) ¤ Beskriv hur man på enklaste sätt utvidgar lösningar liknande Kernighan&Ritchies lösning till att med stor sannolikhet kunna upptäcka om användaren försöker anropa `free()` med en pekare till minne som inte allokerats via `malloc()`! (1p)

a) Stacken envänds av "run-time systemet" för att lägga upp aktiveringsposter på för funktioner/metoder/ procedurer som anropats. Det betyder att det är "systemet" som hanterar stacken och att användaren normalt inte skall hantera stacken (undantag då man indirekt hanterar stacken via systemanrop/biblioteksfunktioner är om man har funktioner med flera parametrar då man kan behöva inspektera stacken eller om man använder setjmp()/ longjmp()) Semantiken för stacken är alltså sådan att användaren inte skall kunna allokera minne där.

b) I K&Rs lösning ligger en datastruktur omedelbart framför den adress som användaren har som startadress till blocket. Genom att lägga in ett specifikt, stort, heltal som alltid skall finnas i den datastrukturen (ett sk. magic number) kan man med stor sannolikhet konstatera om ett block allokerats via malloc()/realloc(). Om, blocket inte allokerats via malloc/realloc så är sannolikheten för att man skall hitta rätt magic number på rätt position före strtadressen till blocket mycket liten. En alternativ, men mer kostbar lösning, är att spara samtliga adresser till block som allokerats via malloc/realloc/calloc i en separat lista och kontrollera om den adress som lämnas åter via free() finns i listan.

Vanliga fel: Svarar man bara "för att det inte skall gå/man får inte göra det" utan motivering = 0p. Notera att hur man implementerar detta har att göra med att: Hur man tolkar innehållet i minne beror helt och hållet på programmet, dvs vilken kod programmeraren skrivit. Att kontrollera om en returnerad adress ligger i den adressrymd som hanteras av malloc/free räcker inte. Användaren kan råka returnera en adress som inte allokerats via malloc(). Då kommer man att tolka det minnesutrymme som ligger omedelbart före den adressen som en "Header" och antagligen få fel storlek på blocket etc. vilket leder till fel i minneshanteringen. Att bara sätta en flagga i Headern som skall indikera om blocket allokerats via malloc() räcker heller inte. En flagga kan i sin enklaste form vara en bit (normalt gör man en flagga så liten som möjligt) och sannolikheten för att en godtycklig bit, som man skulle hitta om man returnerade en godtycklig adress, har ett värde som indikerar att man allokerat via malloc() är i det fallet för hög, 50%.

4) I många OS används inte direkt den prioritet en process/tråd skapats med som prioritet för schemaläggningen. Istället baseras ofta schemaläggningen på en "aktuell" prioritet som OS:et beräknar och som kan förändras. Varför gör man så? (1p)

Genom att använda en aktuell prioritet kan man i många fall bättre tillgodose systemets och de enskilda processernas aktuella behov. I de exempel på schemaläggning som gåtts igenom i Windows, UNIX, LINUX och realtids OS finns i samtliga fall exempel på hur man använder andra prioriteter i schemaläggningen än de basprioriteter processerna har. Ett sådant exempel tillsammans med en motivering liknande den i första meningen i det här stycket räcker i svaret. Exemplen kan handla om aging i UNIX, goodness i LINUX, prioritering av processer som väntat på I/O i Windows eller arv av prioriteter i RTOS för att undvika prioritetsinversion.

Vanliga fel: En process som blockeras av I/O behöver inte få ändrad prioritet under det att den blockeras för att lämna plats åt någon annan process att exekvera. Är den blockerad exekverar den inte oavsett vilken prioritet den har.

5) De flesta interaktiva system har schemaläggning på två nivåer. Beskriv de repektive nivåerna, vad målen med dem är och om/hur de interagerar med andra delsystem i OS:et (2p)

Schemaläggning handlar i grund och botten om vilka processer eller trådar som skall få tillgång till en processor att exekvera på. Det frågan avsåg var: På den översta nivån har man swapping som flyttar hela processer till och från disk resp. primärminnet, dvs mellan den mängd processer som är aktiva på processorn och den mängd som är inaktiva på disk. Normalt flyttas processer från primärminnet till disken om systemet är överlastat. För de processer som finns i primärminnet, dvs de som är aktiva, sker schemaläggning på nivå två med den "vanliga" schemaläggaren som schemalägger processerna efter deras (tillfälliga) prioritet.

Svaret kan också innehålla: I system med flera processorer/kärnor tillkommer också en ytterligare nivå av schemaläggning där man inte bara måste ta hänsyn till när en process/tråd får exekvera men också var (på vilken processorkärna). Specifikt i fallet med multi-processorer kan man också behöva ta hänsyn till att schemalägga trådar inom en process samtidigt för att få bättre möjligheter för dessa att kommunicera med varandra.

Vanliga fel: OS:et känner bara till schemaläggning som sker i kärnan, schemaläggning av ev. trådar på användarnivå känner det inte till och kan därför inte hantera. Normalt schemalägger ett OS antingen

processer eller kärntrådar - väldigt sällan både och.... och i sådant fall bara under speciella omständigheter.

6) a) Beskriv sidutbytesalgoritmerna LRU och Clock! (2p)
b) Vilken är att föredra i ett verkligt system? Varför? (1p)

a) LRU Least Recently Used byter ut den sida som gått längst tid sedan den användes (refererades)
Clock ordnar informationen om sidorna i primärminnet i en cirkulär lista, initialt i FIFO ordning. Man har en post per ram i primärminnet med information om den (ev.) sidan i ramen. Till varje sida finns en refererad bit (R-bit) som sätts till 1 varje gång sidan refereras. När en ram behöver frigöras flyttar man en pekare runt i listan. Om den pekar ut en sida där R-biten är satt nollställs den och pekaren flyttas till nästa sida. När pekaren pekar ut en sida för vilken R-biten är nollställd kastas den sidan ut ur primärminnet och den nya sidan läses in i den frigjorda ramen och informationen i den cirkulära listan uppdateras. R-biten för den nyinlästa sidan sätts också direkt (för att undvika att den direkt skulle kunna kastas ut)

b) LRU är i de flesta fall för komplex att implementera för att vara ett bra val. Den kräver normalt både en tidsstämpel som sätt varje gång en sida refereras och en sökning/sortering av sidorna för att hitta den som det var längst sedan den refererades vid ett ev.sidutbyte. Clock kräver inga tidsstämplar och/eller avancerad sökning/ sortering och är alltså betydligt enklare och effektivare i termer av komplexitet att implementera men ger en hygglig approximation av working set. Clock är därför att föredra framför LRU i de flesta fall.

Vanliga fel: Att beskriva LRU som att man byter ut den sida som man refererade till **sist** är att beskriva LRU som om den hade LIFO egenskaper map. tiden för sidreferenser.

7) Beskriv vad en TLB är och hur den används! (2p)

En TLB kan liknas vid ett cache-minne där man håller de mest aktuella översättningarna från sidnr till ramnr, dvs en cache för delar av sidtabellerna för de aktiva processerna. Den implementeras i associativt minne vilket gör att man kan söka parallellt i hela TLBn på en eller ett fåtal clockcykler. TLBn är normalt del av MMUn. När en adressöversättning från virtuell till fysisk adress sker kontrolleras TLBn först. Vid träff i TLBn går adressöversättningen betydligt snabbare än om man tvingas gå via sidtabellen. Om man får en miss i TLBn gör man uppslagningen i sidtabellen och uppdaterar TLBn (detta fall kan också involvera ett sidfel, dvs att sidan inte fanns i primärminnet och måste läsas in från disken).

Vanliga fel: TLBn är inte att förväxla med de vanliga cachearna för minne. TLBn innehåller alltså inte sidor (data).

8) Virtuellminneshantering. Antag att en process använder bara en liten del, X %, av den virtuella adressrymden.
a) Ungefär hur mycket minne kan vi maximalt spara för implementationen av sidtabellen genom att införa en flernivåers sidtabell istället för en en-nivås sidtabell. Uttryck svaret som procent av minnet som sparas jämfört med en-nivås sidtabellen! Förklara ditt svar! (1p)
b) Antag att minnesaccesstiden i systemet är M, att söktiden i TLBn är t << M och att träffsannolikheten i TLBn är T. Hur mycket längre blir den upplevda minnesaccesstiden med en två-nivåers sidtabell jämfört med om man använder en en-nivås sidtabell? (1p)

a) En en-nivås sidtabell innehåller poster för alla sidor i den virtuella adressrymden och dess storlek är alltså direkt proportionell mot den virtuella adressrymdens storlek. En fler-nivåers sidtabell innehåller en yttre sidtabell och ett antal inre sidtabeller för de områden av den virtuella adressrymden som processen använder - dess storlek är alltså i huvudsak proportionell mot den delen av den virtuella adressrymden som processen utnyttjar. Minnesbesparingen för implementationen av sidtabellen skulle alltså maximalt kunna bli (något mindre än) 100-X% av storleken av en-nivås sidtabellen.

b) Accesstid (träff i TLB + TLBmiss med söktid i sidtabell + minnesaccess):
En-nivås sidtabell: T*t + (1-T)*(t+M) + M
Två-nivås sidtabell: T*t + (1-T)(t+2M) + M
Dvs. med en två nivåers sidtabell blir den upplevda minnesaccesstiden (1-T)*M längre

Kommentar: I varken uppgift a eller b handlar det om formler som vi gått igenom i alla detaljer eller som man skall lära sig utantill. Frågan handlar om att man med en ingenjörsmässig förståelse för algoritmerna och datastrukturerna skall kunna göra ingenjörsmässiga enkla matematiska modeller av systemen. För uppgift b kan man också enkelt validera att den formel man kommer fram till kan vara korrekt genom att observera att omm man har 100% träffsannolikhet i TLBn bör den upplevda minnesaccesstiden inte påverkas alls, medan omm man får 0% träff i TLBn bör den upplevda

minnesaccesstiden öka med hela den extra accesstiden för den mer komplexa implementationen av sidtabellen som ökning av accesstiden.

9) Vad är/innehåller en fysisk backup (dump) respektive en logisk backup av ett filsystem? (1p)

En fysisk dump innehåller kopior av samtliga diskblock (utom möjligen de som ligger i fri-listan och de som finns i listan med dåliga block). En logisk backup av ett filsystem innehåller kopior av samtliga kataloger och filer som förändrats sedan den senaste backupen gjordes (kallas ibland också incrementell dump/backup)

10) Beskriv kortfattat hur man implementerar ett filsystem baserat på FAT ? (1p)

Partitionen delas in i block av fix storlek. Man har en tabell med en post för varje block på partitionen (File Allocation Table). En fil implementeras som en lista av disk-block. För filen anger man i directoryinformationen vilket block som är startblock för filen. På den positionen i tabellen för diskblocken finns antingen index för nästa block i filen eller, om det var sista blocket i filen, en slutmarkör. Dvs istället för att ha pekaren till nästa diskblock i diskblocket har man samlat in dessa och lagt dem i "file allocation table". (se boken och föreläsningsanteckningarna för mer detaljer)

11) I UNIX filsystem lagrar man accessrättigheterna för en fil i i-noden.
Varför lagrar man dem inte i katalogerna (mapparna)? (1p)

Eftersom man tillåter hårda länkar till en fil från fler än ett directory skulle man kunna hamna i en situation där filen hade olika accessrättigheter för en och samma användare beroende på via vilken hård länk (vilket directory) den accessades om accessrättigheterna lagrades i directoryt. Dvs resonemanget för UNIX filsystem är att filens accessrättigheter är en egenskap hos filen och bör knytas till denna. För system utan hårda länkar skulle däremot accessrättigheterna mycket väl kunna ligga i directoryt.

Vanliga fel: Man kan lägga information om accessrättigheter till en fil i den post som knyter filnamnet till i-node i directoryt, dvs. informationen i directoryt för en fil skulle innehålla <filnamn, accessrättigheter, i-nodsnr>. Det går utmärkt bra att på så sätt ha individuella accessrättigheter per fil (jfr. Microsofts enklaste filsystem som FAT) - speciellt om filsystemet saknar möjlighet till hårda länkar. Dvs att ha accessrättigheterna för filerna i directoryn innebär inte att alla filer i ett directory får samma accessrättigheter.

12) Vilken specifik information och datastrukturer kan man förvänta sig att det finns knutet till en kärntråd? (1p)

Ett unikt trådID (trådID i sig kan vara unikt eller kombinationen PID, trådID måste unikt kunna identifiera tråden)
En egen stack
Egen kopia av registren inklusive programräknare och stackpekare

13) Kalle jobbar som konsult och skall hjälpa en resebyrå som har problem med att deras populära resebokningsystem upplevs som segt av kunderna. Systemet körs på ett UNIX -system. Han föreslår att man skall minska time-slicen för processerna till några få millisekunder för att öka interaktiviteten.
Är det en bra eller dålig lösning? Motivera varför! (1p)

Om kunderna upplever att systemet är segt/långsamt beror det antingen på att systemet i sig är överlastat av för många kunder eller att bokningsystemet i sig är för långsamt. Genom att korta time-slicen för processerna så ökar man bara lasten på processorn genom fler context-switchar. Det gör att systemet antagligen kommer att upplevas som ännu segare....

Vanliga fel: En process behöver inte exekvera klart det den skall göra inom en time-slice - den kan bli avbruten om time-slicen tar slut och sedan fortsätta exekvera under nästa time-slice. En process som blir klar innan den förbrukat hela sin time-slice eller en process som blockerar pga av I/O blockerar inte processorn för återstoden (resten) av sin time-slice. Om systemet är överlastat så kommer inte nödvändigtvis en kortare time-slice att leda till fler sidfel - men det kan vara en del i problematiken.

14) ¤ Vad fyller det virtuella filsystemslagret och v-noder för funktion i UNIX/LINUX? (1p)

v-noderna identifierar i vilket filsystem en fil ligger. Dvs de/det utgör ett extra abstraktionslager som tillåter att man kan ha flera filsystem på en och samma maskin, t.ex distribuerade eller olika typer av lokala filsystem (där alla t.ex inte behöver vara i-nods baserade)

15) Beskriv för och nackdelar med att implementera en server som tillståndslös respektive med tillstånd! (2p)

Se boken och föreläsningsanteckningarna

Notera: servrar med eller utan tillstånd handlar inte nödvändigtvis om filservrar

16) Beskriv hur Bankir algoritmen avgör om ett system är säkert eller inte! (1p)

Se boken och föreläsningsanteckningarna

Vanliga fel: Att säga att ett säkert tillstånd är ett tillstånd som inte leder till deadlock ger inga poäng i sig. Det intressanta, och det som efterfrågas, är hur man kan avgöra om ett tillstånd kan leda till deadlock eller ej.

17) Förklara varför vissa bedömare tror att vi kan få än mer säkerhetsproblem när vi går från enkla mobiltelefoner till mobilsystem med fler tjänster och sk. smartphones? (1p)

Smartphones är att betrakta som datorer med möjlighet att exekvera kod som kan laddas in på enheten t.ex genom att hämtas via nätet. Det är den huvudsakliga källan till säkerhetsproblem speciellt då de i allmänhet har mer begränsade beräkningsresureser än vanliga datorer och man kör ofta inte lika bra (om ens någon) anti-virus eller annan säkerhetsprogramvara på dem.

Vanliga fel: Att en mobiltelefon kommunicerar vi olika nät behöver i sig inte innebära säkerhetsproblem. En "gammal" mobiltelefon laddar också hem data (digitaliserat ljud) men det innebär inget säkerhetsproblem. Att trafiken på näten kan avlyssnas är heller inget unikt problem för smartphones - det problemet finns i alla tillämpningar som kommunicerar via ett nätverk (modem, gamla telefoner på fasta linjer etc.)

18) Varför försöker ett virus ofta ta kontroll över avbrotts- och TRAP-vektorerna? (1p)

Det som är speciellt med att ha kontroll över avbrotts- och TRAP-vektorer mot att viruset kan exekvera på andra ställen i kärnan är: Genom att kontrollera avbrottsvektorn kan man i princip se all I/O som sker mot maskinen och genom att kontrollera TRAP-vektorn/handlern kan man följa samtliga systemanrop som sker vilket ger full insikt i allt som sker på maskinen men också full möjlighet att kontrollera och styra vad som sker.

19) *Kalle har uppfunnit en ny avbrytande prioritesbaserad schemaläggningsalgoritm för periodiska realtidsystem. Antag att starttiden för ett jobb, dvs tiden när ett jobb blir klart att köras, är S, att jobbets kritiska väg är C och att dess deadline är D. Kalle föreslår att jobben skall få fasta prioriteter P som skall sättas som: $P = 1/(D - (C + S))$ (högt numeriskt värde betyder hög prioritet). Är schemaläggningsmetoden optimal på en-processors system? (1p)

Nej den är inte optimal eftersom den inte schemalägger jobben efter strikt ökande deadlines. Det visas genom att man t.ex kan konstruera ett exempel där JobbA: D=10, C = 9.5, S=0 och JobbB: D=1, C=0.01, S=0. JobbA får prioritet 2 och JobbB får prioritet c:a 1. Dvs JobbA tillåts exekvera först och JobbB missar sin deadline. Däremot hade JobbB kunnat exekvera först utan att JobbA missat sin deadline.

20) *Beskriv vilka datastrukturer man går igenom när man anropar en funktion i en device driver i ett UNIX/ LINUX system. (1p)
Hur används minor/major device number? (1p)

a) En device-driver anropas indirekt via ett systemanrop, t.ex read(), write(), när man skriver/läser till en enhet eller en fil. Abstraktionen för en enhet i UNIX/LINUX är en fil med tillhörande i-nod. Det betyder att oavsett om man gör I/O mot en fil eller en specifik I/O enhet har den en i-nod och i den finns information om device-id (major device number) Major device number används för att i device-driver vektorn slå upp en datastruktur med pekarna till gränssnittsfunktionerna för device-drivern. Gränssnittsfunktionerna implementerar funktionerna motsvarande systemanropen för att manipulera filer (read(), write() etc.)

b) Major device number identifierar drivrutinen medan minor device number identifierar den fysiska enhet det handlar om i det fall en drivrutin hanterar flera fysiska enheter t.ex flera diskar.

21) Har Windows en lokal eller global sidutbytesalgoritm? Beskriv huvuddragen i algoritmen! (1.5p)

Windows har i de flesta versioner primärt en lokal sidutbytesalgoritm men arbetar också globalt. En process får använda ett antal ramar som ligger i intervallet WSmin - WSmax. Gränserna för working set sätts bland annat baserat på primärminnets storlek men också på egenskaper hos processen. Om ramar behöver frigöras eller

sidor bytas ut tar man i första hand ramar från processen som fått sidfel (lokal). Man kan också ta ramar från processer i allmänhet (globalt) om systemet är högt lastat. Se boken och föreläsningsanteckningar för detaljer.

Vanliga fel: Att man svarat "global" utan vidare motivering = 0p

22) ¤ Kalle föreslår att man i ett distribuerat transaktionsystem skall ersätta de logiska tidsstämplar man implementerat för att kunna ordna händelser kausalt med GPS-tidsstämplar.
a) Vad betyder att tidsstämplarna ger en "kausal ordning"? (1p)
b) Kan GPS-tidsstämplar användas för att skapa en kausal ordning i systemet? (1p)

a) Kausal ordning - orsaks-händelse ordning: Om en händelse A orsakat en annan händelse B, indirekt eller direkt, så är tidsstämpeln för A strikt mindre än tidsstämpeln för B. En kausal ordning är alltså normalt en partiell ordning (dvs den ordnar inte parallella eller obeorende händelser)
b) En strikt kausal ordning, en ordning som alltid uppfyller egenskaperna i a), kan bara implementeras med logiska tidsstämplar. En fysisk klocka har alltid en viss inneboende felmarginal och kan inte användas för att skapa samma typer av kausala ordningar som en logisk klocka.

Vanliga fel: Man har inte förstått att GPS-systemet kan ge väldigt tillförlitlig tidsinformation - inte bara positioner. Dock inte tillräckligt exakt för att skapa en strikt kausal ordning. En kausal ordning har inget med någon plats/lokalitets baserad ordning att göra (vet inte hur en sådan ordning skulle definieras...)

"Vi visste inte att det var omöjligt så vi klarade av det"
Okänd