Tentamen i IS1350/ID2200/ID2206 Operativsystem
måndag 2012-10-15 kl 1400-1800 (GMT+1)

Skolan för Informations och Kommunikationsteknik

Examinator: Robert Rönngren

Hjälpmedel: Inga. Mobiltelefoner skal vara avstängda och får inte finnas vid skrivplatsen.
Tentamensfrågorna behöver inte återlämnas efter avslutad tentamen.

Ange på omslaget vilken kurs du tenterar och vilken termin du läste kursen ffg.

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 inom 3 arbetsveckor
- Frågor markerade med ¤ ingår inte i tentamen för kursen ID2200

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

Lösningsförslag: anslås på kursens webbsida efter tentamen

Frågor:

1) När man standardiserar operativsystem i t.ex POSIX väljer man oftast att göra det genom att standardisera gränssnittet i form av systemanrop implementerade, i t.ex C, på biblioteksnivå. Ge minst två bra skäl till varför man väljer att göra så! (1p)

   1) Man vill inte detaljstyra exakt hur funktionaliteten skall implementeras
   2) Det går inte att standardisera det hela på assemblernivå eftersom hur man gör t.ex TRAP, var parametrarna skall läggas etc. kan skilja åt mellan olika datorarkitekturer och assemblers

2) I C kan man deklarera funktioner och variabler som `static`. Förklara följande:
   i) I vilken del av den virtuella minnesrymden läggs en lokal variabel deklarerad som
   ```
   static int lokal;
   ```
   (0.5p)
   ii) I vilken del av den virtuella minnesrymden läggs en global variabel deklarerad som
   ```
   static float pi = 3.14;
   ```
   (0.5p)
   iii) I vilken del av den virtuella minnesrymden läggs en funktion deklarerad som
   ```
   static void fun(void);
   ```
   (0.5p)

   i) Det är en variabel som skall initieras till 0 och skall instansieras en gång (dvs finnas under hela körningen av processen) och inte instansieras vid varje funktionsanrop vilket gör att den läggs i dataarean (eller möjligen i BSS om man nollställer BSS)
   ii) Globala variabler som är instansierade till ett värde skiljt från 0 läggs alltid i dataarean
   iii) En funktions kod finns alltid i textarean

3) Vad används stacken till när man exekverar en process? (1p)

   På stacken skapas aktiveringsposter för de funktioner som anropas. De viktigaste delarna i aktiveringsposterna är lokala variabler, parametrar och returadress.

4) Följande lilla program kan lasta ner en dator så att den blir oanvändbar. Förklara till vad processortiden kommer att spenderas! (1p)

```
int main(int argc, char **argv)
{
    while(1) if(fork() == 0) execlp("date", "date", NULL);
}
```

Programmet skapar processer som kör kommandot date. Själva exekveringen av date tar förhållandvis liten tid, det handlar om ett par systemanrop. Mesta delen av exekveringstiden går åt för operativsystemet ska skapa, schemalägga, dvs. utför context switchning och terminerar processerna som skapas.

5) Vilket är det viktigaste målet för en sidutbytesalgoritm? (1p)

Det viktigaste målet är att hålla de sidor som ingår i processernas Working Set i primärminnet så man minimerar antalet sidfel och minskar arbetet med paging. Dvs att få primärminnet att fungera som en typ av cache för processerna.

6) Resonera kring vilken av sidutbytesalgoritmerna FIFO, Clock och LRU som är att föredra att använda i ett riktigt operativsystem. (2p)

Även om LRU är den algoritm som antagligen bäst kan approximera den optimala sidutbytesalgoritmen är den i praktiken för kostsam att implementera. Clock är något mer komplicerad att implementera än FIFO och kräver att det finns en referensbit, men den ger en bättre approximation av Working Set än FIFO. Clock är därför oftast att föredra. Men om man har tillgång till relativt mycket minne och vill ha en snabb, dvs enklast möjliga algoritm att implementera, kan FIFO användas.

7) En rättfram (naiv) implementation av en inverterad sidtabell är i praktiken omöjlig att använda medan en hashad inverterad sidtabell används i många system. Förklara vad det är som gör skillnaden i praktisk användbarhet. (1p)

En naiv implementation av en inverterad sidtabell håller bara, i en vektor med en post per ram, reda på vilken process sida som finns i varje ram. En sådan implementation kräver att man söker genom vektorn varje gång man skall göra en adressöversättning. Det ger en oacceptabel kostnad. I en hashad sidtabell ersätts huvuddelen av sökningen med att beräkna en hashfunktion som leder till att man direkt kan accessa en kort, medellängden ligger ofta när 1 men bör i regel inte överstiga 2, länkad lista med information om vilken ram som innehåller vilken sida. I korthet betyder det här att söktiden (eller uppslagstiden) i en hashad inverterad sidtabell kan bli acceptabel medan den är oacceptabelt lång i den naiva inverterade sidtabellen.

8) Beräkna det minnesutrymmet som krävs för att hålla ordning på en process sidor med en två-nivåers sidtabell givet en virtuell adressrymd om 2^{32}, en fysisk adressrymd om 1GB, sidstorlek på 4KB, en inre sidtabell är lika stor som en sida, posterna (entry) i den yttre och inre sidtabellerna är 32 bytes långa. Processen använder 128KB för stack, 160KB för data (heap, data och bss) och 50 KB för textarean. (2p)

Varje inre sidtabell kan adressera 4K/32 = 128 sidor, dvs 512KB. Det gör att vi behöver en yttre och två inre sidtabeller. För att adressera i den inre sidtabellen går det åt 7 bitar. Kvar är 13 bitar som används för att adressera i den yttre sidtabellen som då blir $2^{13}*2^5$ = 256 KB stor. Totalt går det alltså åt 264 KB för att hålla sidtabellen för processen. (Egentligen borde frågan formulerats som att det går åt 32 bytes för en post i den inre sidtabellen. I den yttre räcker det att spara en pekare per post dvs 32 bitar eller 4 bytes, men nu var frågan formulerad som den var....)

9) i) Beskriv begreppen spår, sektor och cylinder. (1p)
ii) Vad är skillnaden mellan fysik och virtuell geometri för en elektromagnetisk hårddisk? (1p)

Enklast är att rita en figur....
Spår: den del av en skiva i en elektromekanisk hårddisk som roterar under ett läs/skrivhuvud
Cylinder: om man har flera skivor så utgör alla spår för en given position för läs/skrivhuvudet en cylinder
Sektor: varje spår delas in i "Block" som rymmer lika mycket data, dessa kallas sektorer

10) a) Beskriv kortfattat hur man implementerar ett filsystem baserat på FAT respektive i-noder? (1p)
b) Vad är fördelar och nackdelar med respektive teknik? (1p)

i) Båda varianterna av filer förutsätter att man har blockindelad allokering av diskarna. I FAT läggs adresserna till diskblocken i en separat tabell, övrig information om filen som namn, accessrättigheter, tid för skapande, tid för senaste modifiering, ägare etc. läggs i katalogerna (directoryna). För varje fil håller man en länkad lista över vilka block som ingår i filen, listan implementeras i tabellen.

Om i-noder används så implementeras varje fil med en liten datastruktur, i-nod, som lagras på disken. i i-noden finns information om ägare, skapare, accessrättigheter, tider för skapande, senast access och modifiering samt pekare till diskblocken. I directoryna/katalogerna finns en koppling mellan filnamn och i-nodsnummer.

ii) FAT är enklare att implementera men har flera nackdelar som att filens attribut inte finns lagrade tillsammans med filen, vilket gör att man t.ex kan få olika accessrättigheter beroende på från vilken katalog filen accessas. Genom att pekarna till

diskblocken ligger i en länkad lista i FAT går det aningen långsammare att hitta till en godtycklig position i filen än om man använder i-noder som har organiserat diskblockspekarna i en trädstruktur. En ytterligare nackdel med FAT är att storleken på diskblocken kan komma att bestämmas av storkleken på partitionen och det faktum att FAT tabellen är av fix storlek vilket inte nödvändigtvis ger en i andra aspekter optimal blockstorlek.

11) Kalle arbetar på ett datalagringsföretag och har upptäckt att de filer som lagras på servrarna i allmänhet antingen är väldigt stora eller ganska små. Han föreslår därför att man skall implementera "ett nytt effektivare filsystem baserat på variabel blockstorlek".
i) Diskutera om det är en bra eller dålig idé! (1p)
ii) Föreslå ett alternativ som kan ge de egenskaper Kalle troligtvis eftersträvar. (1p)

i) Generellt sett är variabel blockstorlek inte en bra idé när man implementerar filsystem annat än om filsystemet bara skall skrivas en gång. Det beror på att man med variabel blockstorlek får extern fragmentering som kan kräva att filer flyttas eller att man kompakterar hårdisken när filerna storlekar förändras. Problemet är att detta blir både komplicerat att implementera och allt för tidskrävande. Så Kalles idé är inte bra.
ii) Vad Kalle observerat är att man gärna vill ha större block för stora filer för att minska antalet block, enklare kunna lägga blocken nära varandra på disken för att snabba upp accesserna och för att minska den relativa overheaden vid skrivning/läsning. För mindre filer vill man gärna ha mindre diskblock för att minska den interna fragmenteringen. Man kan komma nära dessa egenskaper genom att t.ex. ha en stor grundstorlek på block som sedan kan delas upp i mindre block - dvs. att man i parktiken har två blockstorlekar. I Microsofts filsystem används en liknande idé där filer om möjligt lagras i konsekutiva "runs" av diskblock på disken. Det ger snabbare access och de sk. "runs" kan ses som en form av större diskblock.

12) Varför kan det vara enklare att implementera trådar på ett 64-bitars system än på ett 16-bitars system? (1p)

Ett av de större problemen med att implementera system som stödjer att trådar skapas dynamiskt under körningen är att varje tråd behöver en egen stack. Grundproblematiken är att kunna allokera plats för dessa stackar så att de också har möjlighet att växa vid behov. Även om man gör antagandet att hela processen inklusive trådar ryms inom 2^{16} bytes är det enklare att implementera det i ett 64 bitars system. På ett 64-bitars system är den virtuella adressrymden i de flesta praktiska sammanhang att betrakta som oändlig vilket gör det betydligt enklare att placera ut och reservera plats för trådarnas stackar med plats för dem att växa i den virtuella minnesrymden än i ett tydligt minnesbegränsat system som ett 16-bitars system.

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

Det är ett abstraktions/indirektionslager som gör att man inte direkt accessar en fils i-nod utan en v-nod som i sin tur talar om i vilket filsystem filen lagras. Dvs. det är ett abstraktionslager som gör det möjligt att enkelt kunna stödja flera olika typer av lokala och distribuerade filsystem på en maskin.

14) ¤ Beskriv grundprinciperna för hur koden i kärnan i UNIX, LINUX respektive WINDOWS är uppbyggd för att hantera I/O. Utgå från en beskrivning av vad som händer vid ett systemanrop till write(2). (2p)

Vid ett systemanrop läggs ett heltal som indikerar vilket systemanrop som skall utföras samt parametrarna till systemanropet på "rätt" ställen så de kan kommas åt från kärnan, oftast läggs de i register. Sedan utförs en TRAP-instruktion med en parameter som indikerar att det rör sig om ett systemanrop. Kontrollen förs över till kärnan och man exekverar i en avbrotts- eller TRAP-hanterare. I fallet med systemanropet write() så kan man via fildeskriptorn som är första parameter identifiera på vilken I/O-enhet filen finns, dvs man hittar ett deviceId. DeviceId används för att hitta/slå upp en datastruktur (struct) som implementerar gränssnittet mot device drivern. I datastrukturen finns en lista (vektor) med pekare till funktioner som implementerar gränssnittet mot devicedrivern, oftast funktionerna för att operera på en fil, dvs. open(), read(), write() etc. Numret på systemanropet används för att indexera in i listan och hitta rätt funktion (funktionspekare) att anropa. Principerna skiljer sig inte åt för UNIX, LINUX och Windows.

15) i) Vilka säkerhetsmekanismer fanns inbyggda i MS-DOS från början? (1p)
ii) Resonera kring om de designval man gjorde när man implementerade säkerhetsmekanismerna i MS-DOS var väl avvägda när MS-DOS implementerades respektive deras giltighet i dag. (1p)

i) Det fanns inga säkerhetsmekanismer alls, ingen accesskontroll, man skiljde inet på olika användare, skiljde inte på user och kernel mode och man hade rättigeheter att skriva var som helst i minnet och exekvera kod var som helst i det fysiska minnet även i den del som användes för OS:et.
ii) När MS-DOS designades hade man att förhåll sig till att datorerna hade mycket begränsad kapacitet, det fanns inga nätverk, malware var ett okänt begrepp och man förutsatte att PCn var en dator som användes av en person som enbart nyttjade programvara köpt från pålitliga leverantörer. I den kontexten var valet att inte implementera säkerhetsmekanismer ett val som kunde motiveras. Att förutse den sabba utveckling som initierades via PCn var också i väldigt svårt. Till del förvärrades problematiken av att man hade bråttom att få ut produkterna, PCn, på marknaden och därför mer var inriktade

på att snabbt få fram ett OS som fungerade än att det skulle vara "perfekt" från början. Idag skulle det vara svårt att motivera designvalen (men jämför med säkerheten i OS på smartphones....)

16) I `libc` finns en funktion `system(char *)` som tar en textsträng som argument. Dess funktionalitet är att skicka textsträngen som ett kommando till kommandotolken (shell) som skall utföras (samma funktionalitet kan fås genom `strcpy(3)`). Kan man använda det här som en väg för att åstadkomma en "buffer-overflow" attack även på system som inte tillåter att man exekverar data på stacken? Och i så fall hur? (2p)

Om funktionen system() finns inlänkad och programmet i övrigt har en öppning för en buffer-overflow attack, dvs att det finns en inläsning där man kan skriva över inmatningsbufferten på stacken så öppnar det här möjligheten att göra allt som den användare som exekverar processen får göra (system() i sig öppnar inte direkt för en attack) . Allt man behöver göra är att mata in data som skapar en stackpost som ser ut som den skulle göra vid ett anrop till system() och se till att man returnerar till startadressen för system. På det viset kan man starta ett nytt skal eller vilken process man vill som användaren som startat processen har rätt att köra.

17) Kalle har ett nytt jobb igen. Nu skall han utveckla en ny processorarkitektur för PC som skall stödja mjuk realtid på ett bättre sätt. Hans förslag är att dela upp TLBn så att det finns instruktions- respektive data-TLB för vanliga processer och dessutom separata instruktions- respektive data-TLB för realtids-processer.
i) Resonera kring om det är en bra eller dålig idé att skilja på data- och instruktions-TLB (1p)
¤ ii) Resonera kring om det är en bra eller dålig idé att skilja på TLB:er för vanliga resp. realtids-processer. Resonera också kring vilka saker Kalle måste tänka på vid implementationen. (2p)

i) Det är normalt en bra idé eftersom data- och instruktionsaccesser kan ha olika lokalitet.
ii) Fördelen med att ha separata TLB:er för realtids - och vanliga processer är att realtidsprocesserna skyddas från att de vanliga processerna orsakar att realtidsprocessernas översättningar i TLB:erna kastas ut. Problemet är att de vanliga processerna fortfarande kan orsaka att realtidsprocessernas sidor kastas ut från primärminnet. För att idén skall fungera fullt ut måste Kalle antagligen förändra OS:et så att man kan skydda (del av) realtidsprocessernas sidor i minnet, t.ex. genom att man kan reservera en del av ramarna att användas för realtidsprocesser. Om man tjänar något på idén eller komplexiteten överväger fördelarna måste nog utvärderas närmare innan Kalle implementerar idén. En annan fråga är om Kalle kan påverka implementationen av OS:et.

18) Förklara begreppet "kritisk sektion" som förekommer när man talar om synkronisering! (1p)

Kritisk sektion: De delar av koden där olika processer/trådar kan accessa och modifiera delad data eller delade resurser.

19) i) Beskriv Coffmans villkor! (2p)
ii) Kalle hävdar att man genom att införa prioriteter alltid kan bryta minst ett av Coffmans villkor. Stämmer det? (1p)

i) se boken - en kort beskrivning av varje villkor ger 0.5p per villkor
ii) Coffmans villkor har att göra med baklåsproblematik som uppstår när processer/trådar synkroniserar för att undvika konflikter när man delar resurser. Att införa prioriteter utan koppling till hur processer synkroniserar eller använder resurser bryter inte något av Coffmans villkor. Kalle har fel. Prioriteter kan användas för att i vissa fall bryta Coffmans villkor t.ex. genom att högre prioriterade processer kan ta resurser från lägre (jfr. avbrytande schemaläggning av processer) eller bryta villkoret om cirkulär väntan genom att ordna ordningen i vilken man får allokera resurser. Men det går inte alltid att bryta Coffmans villkor genom att införa prioriteter.

20) Beskriv översiktligt hur användarens inställningar för sin miljö (environment) implementeras i UNIX/LINUX respektive WINDOWS. (1p)

I UNIX/LINUX implementeras miljövariabler som variabler som finns i varje process adressrymd åtkomliga via ett tredje argument till main(). De initieras vid inloggning från filer som finns i användarens hemkatalog. De ärvs mellan föräldra- och barnprocesser och kan sättas vid skapande av en ny process eller via systemanrop. I WINDOWS lagras motsvarande inställningar i registryt där varje användare har ett delträd.

21) Förklara varför LINUX och inte UNIX kom att bli det dominerande alternativet till Microsofts operativsystem för PC under andra hälften av 1990-talet (1p)

UNIX var vid den här tidpunkten det mest avancerade och antagligen bästa OS:et som fanns fritt tillgängligt med fri källkod och som utvecklades av en relativt stor grupp erfarna utvecklare världen över. Men när UNIX släpptes fritt av UC Berkeley i form av BSD så kom UC i en legal tvist med AT&T om upphovsrättigheter (sk. IPR). det gjorde att de flesta inte vågade

använda UNIX eftersom man var rädd att hamna i konflikt med AT&T. Ungefär samtidigt släpptes LINUX, som iofs. inte var lika beprövat eller avancerat, men som inte var behäftat med några legala problem för användarna och där det snabbt växte fram företag som specialiserade sig på att paketera LINUX distributioner och ge användarna support.

"Det viktigaste är inte varifrån man kommer, utan vart man är på väg"
Bernie Rhodes