Tentamen i 2G1504 (fd 2G1113) Operativsystem
tisdag 2004-06-01 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 (2G1113 eller 2G1504) 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 anslås senast 23/6

Betygsgränser:
- Totalt antal poäng på tentamen är 31
- Godkänt (3) garanteras från 16 poäng

Lösningsförslag: anslås på kursens webbsida efter tentamen (antagligen inte förrän 2/6)

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

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

2) Kalle har skrivit en minimal kommandotolk (se kod nedan) som han provkör med enkla kommandon som "`ls`". Tyvärr verkar kommandotolken vara behäftad med fel eftersom den kan utföra ett kommando mer än en gång och i bland hamna i loopar där den utför samma kommando godtyckligt många gånger trots att man inte gett kommandot mer än en gång. Antag att han inkluderat nödvändiga `.h` filer.
   a) Exekveras kommandona i förgrunden eller bakgrunden? (0.5)
   b) Förklara problemet som gör att kommandon kan exekveras mer än en gång! (1p)

```
void handler(int par) {
  int status;
  wait(&status);
}

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

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 datablocken omedelbart före den adress som lämnas ut av `malloc()`.
   a) `malloc(), free()` hanterar minne på heapen (data arean). 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)

4) Ge minst en för- respektive nackdel med att använda lång respektive kort time-slice i system med schemaläggning baserad på round-robin! (1p)

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

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

7) Hur bör man dela upp en virtuell adress för att minimera utrymmet som krävs för en två-nivåers sidtabell givet en virtuell adressrymd om 2^{32}, en fysisk adressrymd om 1GB, sidstorlek på 4KB, lika många bitar (β) används för en post (entry) i den yttre som inre sidtabellen, och att en genomsnittslig process använder 128KB vardera för stack, data och textareaorna? (2p)

8) Beskriv den viktigaste fördelen resp. nackdelen med att använda ren segmentering i förhållande till sidindelning vid implementation av virtuellminne. (1p)

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

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)

11) Om man lägger information om accessrättigheter för filer direkt i directoryt där filen finns så skulle man vinna på att man enklare kan accessa informationen vid t.ex en lång listning av directoryt. I praktiken lägger man emellertid oftast denna information tillsammans med övrig information knutet till filen.Varför? (1p)

12) Vilken specifik information kan man förvänta sig att det finns knutet till en kärntråd? (2p)

13) Kalle har skrivit ett multitrådat program där en tråd försöker öppna en fil, `fd = open("FILEN", O_RDONLY),` men misslyckas. När tråden kontrollerar `errno` indikerar dess värde "segmentation violation". Vad kan ha hänt? (1p)

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

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

16) Bankir algoritmen kan användas för att undvika deadlock genom att hålla ett system i "säkert" tillstånd.
a) Vad avses med ett säkert tillstånd? (1p)
b) Givet följande system, vad är det minsta värdet på X som gör att systemet är i ett säkert tillstånd? (1p)

Process	Allokerat			Max behov			Tillgängligt		
A	1	2	3	6	9	4	2	X	2
B	0	4	2	3	4	5			
C	3	5	6	5	6	8			
D	2	2	3	4	8	9			

17) Ge ett skäl till att det kan vara fördelaktigt ur säkerhetssynpunkt att ha alla kommandon en användare förknippar med ett OS hårdlödda in i OS:et! (1p)

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

19) Även om EDF är optimal som schemaläggningsmetod i hårda realtidsystem har den en egenskap som gör den mindre lämpad för mjuka realtidsystem. Förklara vad det är! (1p)

20) Beskriv, på en övergripande nivå, vilka datastrukturer man går igenom när man anropar en funktion i en device driver i ett system som tillåter att man lägger till/tar bort device drivers dynamiskt som t.ex LINUX. Och hur används minor/major device number? (1p)

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

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

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

De program ni kopplat ihop med hjälp av "pipes" i årets lab1, som `sort, grep, less` är exempel på filter. Ett filter är ett program som läser från `stdin` (fildeskriptor 0), gör någon operation på strömmen av tecken det läser och skriver resultatet till `stdout` (fildeskriptor 1).

2) *Kalle har skrivit en minimal kommandotolk (se kod nedan) som han provkör med enkla kommandon som "`ls`". Tyvärr verkar kommandotolken vara behäftad med fel eftersom den kan utföra ett kommando mer än en gång och i bland hamna i loopar där den utför samma kommando godtyckligt många gånger trots att man inte gett kommandot mer än en gång. Antag att han inkluderat nödvändiga `.h` filer.*
a) Exekveras kommandona i förgrunden eller bakgrunden? *(0.5)*
b) Förklara problemet som gör att kommandon kan exekveras mer än en gång! *(1p)*

```
void handler(int par) {
  int status;
  wait(&status);
}

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

a) Kommandona exekveras i bakgrunden eftersom kommandotolken inte väntar direkt på dem.
b) För att ett kommando skall exekveras flera gånger måste man komma tillbaka upprepade gånger till `execlp(command, command, NULL)` utan att värdet på texten `command` ändrats. Det betyder i sin tur att inläsningen med `fgets()` måste ha misslyckats. `fgets()` kan avbrytas utan att ha läst in något om det kommer en signal från en process som kör ett tidigare kommando, i vilket fall signalhanteraren `handler()` körs. Har man otur kommer man att kunna få en schemalägganing där ett och samma kommando upprepas i all oändlighet.

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 datablocken omedelbart före den adress som lämnas ut av `malloc()`.*
a) `malloc(), free()` hanterar minne på heapen (data arean). 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 används för att dynamiskt lägga upp sk. aktiveringsposter för funktions / procedur / metodanrop på. Den kan alltså inte användas för att hantera mer långvarig lagring av data som skall "överleva" sådana anrop.
b) Lösningen bygger på att man lägger kontrollinformation i direkt anslutning till de datablock användaren allokerar via `malloc()`... Kontrollinformationen utgörs av en "`struct`" (där man också vidtagit åtgärder för att den skall bli korrekt alignad). Genom att utvidga kontrollinformationen så att den också innehåller fält med data som alltid skall ha kända och konstanta värden kan man kontrollera att dessa har de förväntrade värden a då ett block återlämnas genom att startadressen skickas som parameter till `free()`. Är den adressen en adress som inte direkt före sig har ett block med kontrollinformkation (dvs. den är inte allokerad via `malloc()`) kan man med stor sannolikhet uptäcka detta.

```
struct control {
  int magic1;             /* always equal to X if correct */
  struct control *next;   /* next free block if this block is in free list */
  int size;               /* size of block in bytes */
  int magic2;             /* always equal to X if correct */
};
```

4) *Ge minst en för- respektive nackdel med att använda lång respektive kort time-slice i system med schemaläggning baserad på round-robin!* *(1p)*

De mest uppenbara för nackdelarna är:
lång: + mindre relativ overhead för context switch, bättre för CPU-tunga jobb
- dålig responstid, sämre för interaktiva jobb
kort: + bättre responstid, bra för interaktiva jobb
- mer relativ overhead för context switch, sämre för CPU-tunga jobb

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

Låg nivå: De algoritmer / processer som väljer nästa process att köra från ready-kön (ev. basert på någon prioritet. Och som flyttar processer till / från ready-kön då de t.ex blir blockerade pga. I / O. Interagerar med I / O systemet och (virtuell)minneshanteringssystemet.

Hög nivå: Det som i många system kallas "swapper". En process / algoritm som flyttar hela processer till / från disk och på så sätt gör dem tillgängliga / icke tillgängliga för schemaläggning av lågnivå schemaläggaren. Kan t.ex användas för att minska lasten vid thrashing situationer. Interagerar framförallt med (virtuell) minnehanterings systemet.

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

NRU: Not Recently Used. Använder två kontrollbitar per sida: R "referenced" som sätts varje gång en byte i en sida refererats och som normalt nollställs vid jämna intervall, t.ex vid varje klockavbrott. M modifierad, sätts varje gång en byte i en sida modifieras, nollställs om / då en sida skrivs tillbaka till disk från primärminnet. När en sida behöver bytas ut väljs en sida ur den lägsta klass med sidor som existerar enligt följande (allt för att undvika att sidor som anbänds slängs ut):

Table 1:

klass	R	M
0	0	0
1	0	1
2	1	0
3	1	1

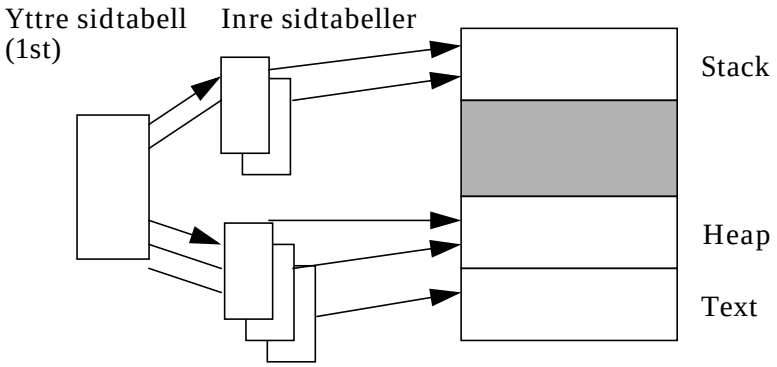
Clock: Använder enbart R biten för att avgöra om en sida skall bytas ut eller ej. Man ordnar samtliga sidor i en cirkulär lista, med en pekare till senast inspekterade sida. När en sida behöver bytas ut kontrolleras R biten för den sida som pekas ut. Är den nollställd byts sidan ut och den nya sidan läggs in på dess plats i den cirkulära listan. Om R biten är satt nollställs den och pekar flyttas fram ett steg.

b) NRU ger en ganska gro approximation av en process working set, Clock ger en bättre approximation och är enkel att implementera. Clock är alltså att föredra.

7) *Hur bör man dela upp en virtuell adress för att minimera utrymmet som krävs för en två-nivåers sidtabell givet en virtuell adressrymd om 2^{32}, en fysisk adressrymd om 1GB, sidstorlek på 4KB, lika många bitar (β) används för en post (entry) i den yttre som inre sidtabellen, och att en genomsnittslig process använder 128KB vardera för stack, data och textareaorna?* *(2p)*

Sidstorleken 4KB=2^{12} B ger att vi har 20 bitar att använda till att adressera yttre resp. inre sidtabell. Eftersom en post i såväl yttre som inre sidtabell upptar lika många bitar (β) kan vi räkna på storleken i "enheten" antal poster. Vidare kan man konstatera att stacken (128KB = 2^{17}B, dvs 2^5 sidor a 4KB) ligger

i en ände av den virtuella adressrymden och heap och text (128KB + 128KB = 256 KB) intill varandra i en annan ände av adressrymden. En schematisk bild av sidtabeller och adressrymd blir då:



Minnesutrymmet som krävs för sidtabellerna är en yttre sidtabell + det antal inre sidtabeller som går åt för att adressera stack respektive heap+text. Notera att så länge som en inre sidtabell kan hålla minst 2^6 poster, dvs adressera 2^6 sidor, krävs det bara två inre sidtabeller, en för stack och en för heap+text. Vi kan tabellera minnesutrymmet för olika uppdelningar av den virtuella adressen:

Table 2:

yttre	inre	#inre för stack	#inre för heap+text	Storlek på sidtabeller i antal poster
2^7	2^{13}	1	1	$2^7 + 2*2^{13} =$ 16.125K
2^8	2^{12}	1	1	$2^8 + 2*2^{12} =$ 8.25K
2^9	2^{11}	1	1	$2^9 + 2*2^{11} =$ 4.5K
2^{10}	2^{10}	1	1	$2^{10} + 2*2^{10} =$ 3K
2^{11}	2^9	1	1	$2^{11} + 2*2^9 =$ 3K
2^{12}	2^8	1	1	$2^{12} + 2*2^8 =$ 4.5K
2^{13}	2^7	1	1	$2^{13} + 2*2^7 =$ 8.25K

Optimalt är alltså att använda 10 eller 11 bitar till att adressera den yttre sidtabellen och 10 resp 9 bitar för de inre sidtabellerna. (intuitivt kan man inse att man bör hamna kring en uppdelning i ungefär lika stort antal bitar för den yttre resp. inre eftersom man får maximal storlek för sidtabellerna om man driver det åt endera ytterligheten: yttre / inre 20 / 0 resp 0 / 20. Minimum bör hamna någonstans mellan dessa värden och givet hur funktionen ser ut någonstans mittimellan)

8) Beskriv den viktigaste fördelen resp. nackdelen med att använda ren segmentering i förhållande till sidindelning vid implementation av virtuellminne. *(1p)*

Fördelen är att man får en tydlig logisk uppdelning i de olika segment ett program under exekvering består av och kan ha skydd anpassat för dessa delar. Nackdelarna är att adressöversättningen blir något mer komplicerad och att man framförallt kan får problem med att det mest rättframma sättet att implementera segmentering bygger på kontinuerlig allokering av segmenten och de problem detta medför.

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

En fysisk dump innehåller alla block på en disk skrivna i sekvens på backupmediat. En logisk dump innehåller alla directoryn med start i något / några directoryn och de filer som ändrats i dessa sedan något givet bas / start datum.

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)*

FAT: File Allocation Table är en minnesresident tabell över hur samtliga filer i filsystemet allokerats.
i-nodes baserat: index-noder är en länkad trådliknande struktur för att peka ut disk blocken. Systemet håller en tabell över aktiva i-noder i minnet.

FAT: + enkel
- onödigt minneskrävande, diskblockens storlek kan bero på diskens storlek om man har fix storlek på tabellen

i-node:+ flexibel, kräver minimalt med minnesresident data
- något mer komplex implementation

11) Om man lägger information om accessrättigheter för filer direkt i directoryt där filen finns så skulle man vinna på att man enklare kan accessa informationen vid t.ex en lång listning av directoryt. I praktiken lägger man emellertid oftast denna information tillsammans med övrig information knutet till filen.Varför? (1p)

Accessrättoigheter är knutet till filen. Om man skulle ha denna information i directoryt kan man få olika accessrättigheter för en och samma fil beroende på genom vilket directory den accessas. Dvs. man kan få säkerhetsproblem med att uåårätthålla konsistenta accessrättigheter.

12) Vilken specifik information kan man förvänta sig att det finns knutet till en kärntråd? (2p)

En tråd behöver åtminstonde egna resuresr för att kunna exekvera "självständigt" inom en process. En minimal uppsättning sådana resurser är:
- Programräknare
- Register
- Stack
- Tillstånd (t.ex blockerad, running etc.)

13) *Kalle har skrivit ett multitrådat program där en tråd försöker öppna en fil,* `fd = open("FILEN", O_RDONLY),` *men misslyckas. När tråden kontrollerar* `errno` *indikerar dess värde "segmentation violation". Vad kan ha hänt? (1p)*

Segmentation violation indikerar att man t.ex försökt skriva i ett skrivskyddat segment eller liknande. Det är högst osannolikt att biblioteksfunktionen `open()` skulle vara behäftad med sådana fel. Rimligt i detta fall är att filen "FILEN" inte existerar. Felmeddelande i `errno` kommer dock troligen från en annan tråd som exekverat efter det att den aktuella tråden anropat `open()` men innan den hunnit kontrollera felet i `errno`. `errno` är normalt en global, delad variabel vilket kan ställa till problem i multitrådade tillämpningar.

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

Det tjänar som indirektionsnivå - en v-nod beskriver var och i vilken typ av filsystem en viss fil ligger. Dvs det underlättar att ha flera olika typer av filsystem (både lokala och distribuerade).

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

Tillståndslös: + inga problem med att återställa ett tillstånd om servern kraschar och måste återstartas
- mer data måste skickas i meddelanden från klient till server, något mer komplex implementation både på server och klientsida, kan vara något mindre effektiv än server med tillstånd

Med tillstånd: + går att implementera effektivare än utan tillstånd
- mindre kommunikation(data) mellan klient och server, svårare att återställa tillstånd vid krasch

16) *Bankir algoritmen kan användas för att undvika deadlock genom att hålla ett system i "säkert" tillstånd.*
a) Vad avses med ett säkert tillstånd? (1p)
*b) Givet följande system, vad är det minsta värdet på X som gör att systemet är i ett säkert tillstånd? (1*p)

Process	Allokerat			Max behov			Tillgängligt		
A	1	2	3	6	9	4	2	X	2
B	0	4	2	3	4	5			
C	3	5	6	5	6	8			
D	2	2	3	4	8	9			

a) ett säkert tillstånd är ett tillstånd från vilket det finns en sekvens av allokeringar där varje process kan få sina maximala resursbehov uppfyllda
b) X=1, C kan få sina behov uppfyllda -> tillgängligt = (5, 6, 8) -> B kan få sina behov uppfyllda -> tillgängligt = (5, 10, 10) och så kan t.ex C få sina behov uppfyllda och sedan A....

17) Ge ett skäl till att det kan vara fördelaktigt ur säkerhetssynpunkt att ha alla kommandon en användare förknippar med ett OS hårdlödda in i OS:et! (1p)

Det minskar risken att någon kan smyga in trojaner, t.ex som en annan implementation av `ls`

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

Virset kommer att exekveras ofta, t.ex vid varje systemanrop. Det kan också kontrollera vad som händer vid varje systemanrop / avbrott och på så sätt enkelt hitta filer att infektera eller känslig information.

19) Även om EDF är optimal som schemaläggningsmetod i hårda realtidsystem har den en egenskap som gör den mindre lämpad för mjuka realtidsystem. Förklara vad det är! (1p)

EDF kan om man missar en deadline ge upphov till dominoeffekter där flera efterföljande jobb missar sina deadline. I ett mjukt realtidsystem kan det vara viktigare att försöka hålla många deadlines snarare än samtliga. Detta implicerar att det kan vara tillåtet att missa enstaka deadlines och därmed kan det finnas bättre schemaläggningsmetoder än EDF.

20) Beskriv, på en övergripande nivå, vilka datastrukturer man går igenom när man anropar en funktion i en device driver i ett system som tillåter att man lägger till/tar bort device drivers dynamiskt som t.ex LINUX. Och hur används minor/major device number? (1p)

Major device number används för att indexera i en tabell / vektor med drivrutiner. En post i en tabell innehåller en pekare till en datastruktur som innhåller pekare till funtioner som implementerar gränssnittet för drivrutinen (definerat av OS:et). Minor devive number används för att identifiera en specifik enhet om en och samma drivrutin hanterar flera enheter, t.ex en viss disk.

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

Windows har i huvudsak en lokal sidutbytesalgoritm. Varje process får gränser för hur många ramar den minimalt / maximalt får allokera. Det skall återspegla storleken på processens working set. Gränserna bestäms bland annat av hur stort primärminnet är. Normalt sett byter en process sut en av sina egna sidor vid sidfel. Det finns globala delar i algoritmen också som bland annat gör att man kan ta ramar från processer om det inte fins ett tillräckligt stort antal lediga ramar på global nivå.

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