Operativsystem ID2200/06

omtentamen

2017-08-21 8:00-12:00

Instruktioner

- Du får, förutom skrivmateriel, endast ha med dig en egenhändigt handskriven A4 med anteckningar.
- Svaren skall lämnas på dessa sidor, använd det utrymme som finns under varje uppgift för att skriva ner ditt svar.
- Svar skall skrivas på svenska eller engelska.
- Du skall lämna in hela denna tentamen.
- Inga ytterligare sidor skall lämnas in.

Versioner

Denna tentamen gäller för flera olika omgångar av kurserna ID2200/06. Beroende på vilken kursomgång du följer så skall olika delar av tentamensfrågorna besvaras.

För omtentander i ID2200 gäller följande:

- Registrerade för tentamen på 6hp, VT16 och HT16: besvara frågorna 1-9, inte fråga 10.
- Registrerade för tentamen på 3.8 hp, dvs före VT16: besvara frågorna 1-8, inte 9-10.
- För de som är registrearde för tentamen på 3.8hp men som ännu inte har lab-momentet avklarat kan man besvara även fråga 9 och då få det momentet tillgodoräknat. Fråga 9 hanteras separat så få poäng på fråga 9 kompensears inte av flera poäng i övriga delar.

För omtentander i ID2206 gäller följande:

- Registrerade för tentamen på 6hp, HT16: besvara frågorna 1-9, inte 10.
- Registrerade för tentamen på 4.5hp, före HT16: besvara frårna 1-8 och fråga 10
- För de som är registrearde för tentamen på 4.5hp men som ännu inte har lab-momentet avklarat kan man besvara även fråga 9 och då få det momentet delvis tillgodoräknat. Fråga 9 hanteras separat så få poäng på fråga 9 kompenseras inte av flera poäng i övriga delar.

Betyg för 6hp

Tentamen har ett antal uppgifter där några är lite svårare än andra. De svårare uppgifterna är markerade med en stjärna, *poäng**, och ger poäng för de högre betygen. Vi delar alltså upp tentamen i grundpoäng och högre poäng. Se först och främst till att klara grundpoängen innan du ger dig i kast med de högre poängen.

Notera att det av de 40 grundpoängen räknas bara som högst 36 och, att högre poäng inte kompenserar för avsaknad av grundpoäng. Gränserna för betyg är som följer:

- Fx: 21 grundpoäng
- E: 23 grundpoäng
- D: 28 grundpoäng
- C: 32 grundpoäng
- B: 36 grundpoäng och 12 högre poäng
- A: 36 grundpoäng och 18 högre poäng

Gränserna kan komma att justeras nedåt men inte uppåt.

Gränsen för E är för tentamen på 4.5hp 18 poäng och för 3.8hp tentamen 16 poäng. Gränsen för tillgodoräkning av lab-moment är 12 poäng på fråga 9.

Namn:_____ Persnr:_____

1 Operativsystem

1.1 vad händer här? [2 poäng]

Om vi ger kommandosekvensen nedan, vad kommer vi då få för resultat?

```
> echo "cd foo grep bar" | grep bar | wc -w
```

Svar: 4

1.2 kommandon i ett shell [2 poäng]

Ge en kort beskrivning av vad kommandona nedan gör.

- `mkdir`
- `cd`
- `cat`
- `ln`

Svar: Slå upp deras betydelse med hjälp av `man`.

Namn:_____ Persnr:_____

2 Processer

2.1 varför på heapen? [2 poäng]

I koden nedan har vi allokerat tre arrayer varav en på heapen, vilken array och varför är den allokerad på heapen och inte på stacken?.

```
#include <stdlib.h>
#include <stdio.h>

#define MAX 4

int h[MAX];

int *foo(int *a, int *b, int s) {

  int *r = malloc(s * sizeof(int));

  for(int i = 0; i < s; i++) {
     r[i] = a[i]+b[i];
  }
  return r;
}


int main() {
  int f[MAX];

  for(int i = 0; i < MAX; i++) {
    f[i] = i;
    h[i] = i*10;
  }

  int *g = foo(f, h, 4);

  printf("a[2] + b[2] is %d\n", g[2]);

  return 0;
}
```

Svar: Arrayen som pekas ut av `r` (och även av `g`) är på heapen och den måste vara det eftersom vi returnerar en pekare till arrayen. Om den hade varit allokerad på stacken så kommer den bli överskriven vid nästa proceduranrop.

Namn:_____ Persnr:_____

2.2 begränsad direkt exekvering [2 poäng*]

Vi implementering av ett operativsystem så använder man s.k. begränsad direkt exekvering (limited direct execution). Vilka är begränsningarna?

Svar: En process can exekvera alla instruktioner utom instruktioner som ändrar på speciella register (som ändrar på exekveringsnivå, placering av IDT mm) eller som läser eller skriver till s.k. kernel space. Den får inte exekvera under obegränsad tid utan blir avbruten efter en viss tid.

3 Schemaläggning

3.1 kortaste tid kvar först [2 poäng]

Antag att vi har en schemaläggare som implementerar *kortaste tid kvar först* (shortest time-to-completion first). Vi har tre jobb som anländer vid tidpunkterna 0, 10 och 30 ms och de är på 60, 30 och 10 ms. Hur blir då den den genomsnittliga omloppstiden (turnaround time) och svarstiden (respons time)?

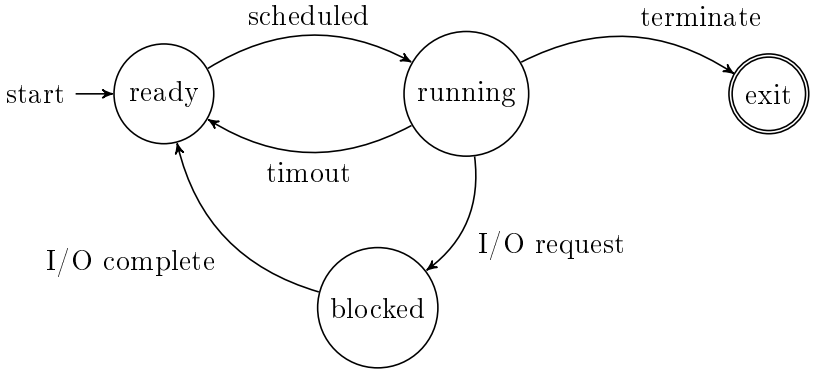
Svar: Den genomsnittliga omloppstiden blir $(100 + 30 + 20)/3 = 50ms$ (alternativt $100 + 40 + 10$ vilket ger samma resultat). Den genomsnittliga svarstiden blir $(0 + 0 + 0)/3 = 0ms$ eftersom de i detta fall kommer att schemaläggas direkt när de kommer (alternativt $(0 + 0 + 10)/3 = 10/3ms$ om vi låter den andra processen köra klart).

Namn:_____ Persnr:_____

3.2 tillståndsdiagram [2 poäng]

Här följer ett tillståndsdiagram för processer vid schemaläggning. Fyll i de markerad delarna så att man förstår vad tillstånden betyder och när en process förs mellan olika tillstånd.

Svar:



3.3 lotteri [2 poäng*]

Det finns schemaläggare som baseras på lotteri där man tilldelar ett antal lotter till varje process, drar ett vinnande nummer och låter den vinnade processen köra under en viss tid. I dessa schemaläggare så minimerar man varken omloppstid eller reaktionstid utan det är någonting annat man vill uppnå, vad är det man försöker uppnå?

Svar: En fördelning av resurser som är rättvis dvs. processer skall tilldelas tid i proportion till hur viktiga de är.

Namn:_____ Persnr:_____

4 Virtuellt minne

4.1 segmentering [2 poäng]

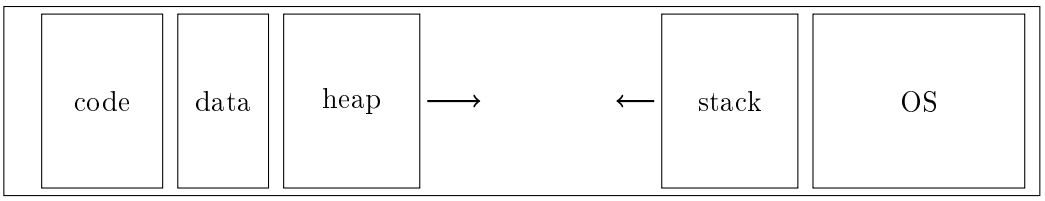
När man använder segmentering för att hantera fysiskt minne så kan man få problem med extern fragmentering. Detta undviks om man istället använder s.k. paging. Varför kan vi undvika extern fragmentering med hjälp av paging?

Svar: Eftersom alla ramar är lika stora och en process kan tilldelas vilken ram som helst så kan en ram alltid användas. Det finns med andra ord inga luckor som är för små för att användas.

4.2 minnet [2 poäng]

I Linux, och många andra operativ system, har varje process en virtuell minnesrymd. Inom denna minnesrymd ligger processens egna areor: stack, kod, globala data och heap. Inom adressrymden finns även operativsystemet. Rita upp en schematisk bild för hur man brukar lägga ut dessa segment. Ange även för stacken och heapen åt vilket håll dessa växer.

Svar:



Namn:_____ Persnr:_____

4.3 mer minne [2 poäng*]

Om en virtuell adress är 32 bitar så kan vi adressera maximalt 4 GiByte minne. Antag att minne är billigt och att vi utan vidare kan bygga maskiner med 64 GiByte minne. Hur skulle vi kunna utnyttja en sådan maskin utan att ändra storleken på den virtuella adressen?

Svar: Vi har ett sidindelat minne och en omvandlingstabell som till exempel omvandlar ett 20 bitars sidnummer till ett 24 bitars ramnummer. Varje enskild process kan fortfarande bara adressera 4 GiByte men vi kan få rum med flera processer i minnet; som maximalt kan vara 64 GiByte. Vi måste naturligtvis ha hårdvara som stödjer, i detta fall, en fysisk adress på 36 bitar. Processorn måste även ha en TLB som är anpassad till det ändrade formatet på ramnummer.

5 Minneshantering

5.1 malloc() [2 poäng]

I Linux (och alla Unix dialekter) så är malloc en biblioteksrutin och inte ett systemanrop. Varför är det en biblioteksrutin? Vore det inte snabbare om vi anropade systemanropet direkt, det är ju operativsystemet som hanterar allt minne i alla fall?

Svar: Ett systemanrop är dyrt. Vi kan göra en betydligt effektivare minneshanterare om vi gör några få systemanrop där vi begär mer minne än vi behöver för stunden och hanterar allokering av detta internt i processen.

5.2 best-fit vs first-fit [2 poäng]

En strategi för att hitta ett lämpligt minnesblock är att hitta det block som bäst motsvara den storlek som vi behöver (utan att vara för litet); det måste ju ur alla aspekter vara en bra strategi. En annan är att ta första bästa block man hittar även om det är betydligt större än vad vi behöver. Vad skulle fördelen vara med den senare strategin och vad är eventuellt nackdelen?

Svar: Den uppenbara fördelen är att vi inte behöver söka igenom alla block för att hitta ett som passar. Om vi i de båda strategierna delar upp det funna blocket och lägger tillbaks den del som vi inte behöver så får vi bättre extern fragmentering jämfört med best-fit. Om vi inte delar upp det funna blocket så kommer vi får en högre intern fragmentering.

Namn:_____ Persnr:_____

5.3 intern paging [2 poäng*]

När vi implementerar minneshantering internt för en process (till exempel med malloc()) så använder vi en form av segmentering. Det är därför vi kan få problem med extern fragmentering. Om det är bättre med så kallad paging, varför använder vi inte paging då vi implementerar intern minneshantering?

Svar: Vi skulle behöva implementera en adressöversättare som i varje referens delade upp en adress i sida och offset. Adressen till sidan skulle sen behöva omvandlas till en ram-adress med hjälp av en omvandlingtabell. Vi skulle med all säkerhet behöva arbete med väldigt små sidor för att inte få för stor intern fragmentering. Detta skulle ge en stor omvandlingtabell som skulle vara svår att hantera. Att göra detta i mjukvara utan stöd från hårdvara i processorn är alldeles för kostsamt.

Ett alternativ skulle vara att bara dela ut block av en mycket liten storlek, säg 16 byte (stor nog för två pekare), och låta processen representera alla objekt med hjälp av dess. Inte omöjligt och det är nästan så en del listbaserad programmeringsspråk hanterar sitt minne.

Namn:_____ Persnr:_____

6 Flertrådad programmering

6.1 saker på högen [2 poäng]

Om vi har ett flertrådat program så kan naturligtvis trådarna läsa och skriva till globala variabler och därmed arbeta med gemensamma datastrukturer. Hur är det med datastrukturer som en tråd allokerar på heapen, kan dessa läsas och skrivas från andra trådar?

Svar: Ja, det enda som krävs är att tråden som allokerat strukturerna på något sätt ger en pekare till dessa. Det kan med lätthet göras via en global variabel.

6.2 deadlock, nästan [2 poäng]

Vad är skillnaden mellan s.k. deadlock och livelock?

Svar: Vid en deadlock så gör ingen process några framsteg. I en livelock så har vi hamnat i en loop där varje process rör på sig men beräkningen som sådan inte gör några framsteg.

Namn:_____ Persnr:_____

6.3 världen är inte enkel [2 poäng*]

I programmet nedan har vi två procedurer som båda uppdaterar en global variabel `count` med 1000 steg. För att kunna köra dessa procedurer i två trådar så skyddas uppdateringen med hjälp av två globala flaggor, `a` och `b`. Varje process kommer börja med att sätt sin flagga och sedan fortsätt endast om den andra processen inte har satt sin flagga. Nu är världen inte alltid så enkel och om vi har en processor som enbart garanterar "total store order" så kan konstiga saker hända - förklara vad som kan hända.

```
void *ping(void *arg) {

 int i;

 for(i = 0; i < 1000; i++) {
   while(1){
     a = 1;
     if(b != 1) {
      count++;
      a = 0;
      break;
     } else {
      a = 0;
     }
   }
 }
}
```

```
void *pong(void *arg) {

 int i;

 for(i = 0; i < 1000; i++){
   while(1){
     b = 1;
     if(a != 1) {
      count++;
      b = 0;
      break;
     } else {
      b = 0;
     }
   }
 }
}
```

Svar: Total store order garanterat inte att skrivningen av en flagga kommer att ske innan läsningen av den andra flaggan. Vi kan läsa den andra procedurens flagga (b) innan vår flagga (a) har blivit satt. Detta gör att båda trådarna kan läsa ett gammalt värde för den andra trådens flagga och fortsätta med uppdatering parallet. Eftersom `count++` inte är en atomär operation så kan vi nu "förlora" uppdateringar.

Namn:_____ Persnr:_____

7 Filsystem och lagring

7.1 ta bort en fil [2 poäng]

Om vi använder kommandot `rm` så tar vi inte bort en fil utan bara en hård länk till en fil. Hur tar man bort själva filen?

Svar: Varje inode inhehåller information om hur många hårda länkar som finns till filen. När vi tar bort den sista länken så kommer filen att att tas bort (dess data kommer ligga kvar på disk men är inte åtkomligt via filsystemet).

7.2 två av tre [2 poäng]

Antag att vi har ett enkelt filsystem utan journal där vi skriver direkt till inoder, bitmappar och datablock. Antag att vi får en crash och att vi vid skapandet av en fil endast hinner göra två av de tre skrivningar som krävs. I vart och ett av fallen nedan, beskriv vilket problem vi kommer att stå inför.

- inode och bitmappar

  Svar: Filen kommer peka på datablock som innehåller skräp.

- bitmappar och datablock

  Svar: Den allokerade inoden kommer innehålla skräp, de datablock som allokerades kommer att gå förlorade.

- inode och datablock

  Svar: Filen finns men de block som har andvänts är inte markerade som taga. Om det inte upptäcks så kan de användas för andra ändamål.

7.3 loggbaserade fs [2 poäng*]

I ett loggbaserar filsystem skriver vi alla förändringar i en kontinuerlig logg utan att göra förandringar i de redan existerande block som en fil har. Vad är poängen med att hela tiden skriva nya modifierade kopior av datablock istället för att gå in och göra de små förändringar som vi vill göra? Om det är bättre, är det något som blir sämre?

Svar: Genom att hela tiden skriva i slutet på loggen behöver vi inte röra skrivhuvudet. Om vi skall skriva på alla enskilda block så måste vi föra skrivhuvudet fram och tillbaks vilket kommer vara en stor nackdel. Vi betalar priset när vi skall läsa en fil som i värsta fall nu är utspridd över hela disken.

Namn:_____ Persnr:_____

8 Virtualisering

8.1 hur långsamt [2 poäng]

När vi kör ett helt operativsystem virtualiserat så kommer exekveringen ta längre tid eftersom en vi då har en virtualisering i två nivåer. Ungefär hur mycket långsammare kommer ett beräkningsintensivt program att gå: nästan lika snabbt, halva hastigheter eller typiskt en faktor tio långsammare? Motivera ditt svar.

Svar: Nästan lika snabbt. Processen kommer att köra med full hastighet med hjälp av begrändas direkt execvering. Endast då det virtualiserade operativsysteet kopplas in, vid systemanrop eller vid sidfel etc, så kommer vi betala en liten kostnad. Det virtualiserade operativsystemet kommer även det att köra i full fart men själva bytet tar lite längre tid.

8.2 sätta IDT [2 poäng*]

När en hypervisor startar ett virtualiserat operativsystem så kommer det virtualiserade systemet med all säkerhet vilja sätta det register som pekar ut den IDT som den vill använda. Vad är problemt och hur man kan lösa det?.

Svar: Om det virtualiserade operativsystemet kör i user mode så kommer hypervisorn få ett avbrott när IDT skall sättas eftersom det är en privikligerad instruktion. Hypervisorn sparar en pekare till den IDT som systemet vill använda så att den vet vilka rutiner som det virtualiserade systemet vill kör. När vi sedan får avbrott så kan vi låta det virtualiserade operativsystem köra sina rutiner i user mode.

Namn:_____ Persnr:_____

9 Implementering

9.1 minnesmappning [2 poäng]

Nedan följer en, något förkortad, utskrift av en minnesmappning av en körande process. Beskriv kortfattat vad varje segment markerat med `???` fyller för roll.

```
> cat /proc/13896/maps

00400000-00401000 r-xp 00000000 08:01 1723260            .../gurka ???
00600000-00601000 r--p 00000000 08:01 1723260            .../gurka ???
00601000-00602000 rw-p 00001000 08:01 1723260            .../gurka ???
022fa000-0231b000 rw-p 00000000 00:00 0                  [???]
7f6683423000-7f66835e2000 r-xp 00000000 08:01 3149003    .../libc-2.23.so ???
    :
7ffd60600000-7ffd60621000 rw-p 00000000 00:00 0          [???]
7ffd60648000-7ffd6064a000 r--p 00000000 00:00 0          [vvar]
7ffd6064a000-7ffd6064c000 r-xp 00000000 00:00 0          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0  [vsyscall]
```

Svar: De första tre segmenten är: kod, read-only data och global data för processen `gurka`. Efter det har vi ett segment för processens *heap*. Segmentet markerat med `libc-2.23.so` är ett delat bibliotek. I den övre regionen hittar vi processens stack.

Namn:_____ Persnr:_____

9.2 fork [2 poäng]

Om vi kör prgrammet nedan, vad kommer att skrivas ut på skärmen? Var noga med i vilken ordning utskrifterna kommer.

```
int x = 0;

int main() {

  int pid;

  pid = fork();

  if(pid == 0) {
    x = x + 10;
    sleep(1);
  } else {
    sleep(1);
    x = x + 2;
    wait(NULL);
  }
  printf("x is %d\n", x);

  return 0;
}
```

Svar: vi kommer får `x is 10` följt av `x is 2`. Moderprocessen kommer att vänta på att barnprocessen är klar. De har varsin kopia av variablen `x` så vi får inte ett resultat `x is 12`.

Namn:_____ Persnr:_____

9.3 Boba [2 poäng]

Antag att vi har ett program `boba` som skriver "Don't get in my way" på `stdout`. Vad kommer resultatet bli om vi kör programmet nedan och varför blir det så? (proceduren `dprintf()` tar en fildescriptor som argument)

```
int main() {

  int pid = fork();

  int fd = open("quotes.txt", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);

  if(pid == 0) {
    dup2(fd, 1);
    close(fd);
    execl("boba", "boba", NULL);
  } else {
    dprintf(fd, "Arghhh!");
    close(fd);
    wait(NULL);
  }
  return 0;
}
```

Svar: I `dup2(fd,1)` sätter vi om `stdout` till den öppnade filen. Boba kommer då att skriva sin rad till filen `quotes.txt`. Samtidigt kommer moderprocessen att skriva "Arghhh!" på samma fil. Eftersom filen öppnades efter det att vi gjorde `fork()` så har de olika processerna olika representationer av filen och skriva över varandra. Vi kommer därför att få antinge "Don't get in my Way" eller "Arghhh!et in my Way" beorende på vilken process som hinner skriva först.

Namn:_____ Persnr:_____

9.4 pipes [2 poäng]

I koden nedan har vi ett program som skapar en pipe och som sedan använder denna till att skicka meddelande (inte med i koden). Hur skulle motsvarande skelettkod se ut för ett program som öppnar den skapade pipe:en för att läsa de skickade meddelenden?

```
int main() {
  int mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
  mkfifo("sesame", mode);

  int flag = O_WRONLY;
  int fd = open("sesame", flag);
   :
   :
}
```

Svar:

```
int main() {
  int flag = O_RDONLY;
  int fd = open("sesame", flag);
    :
    :
}
```

Namn:_____ Persnr:_____

9.5 en mapp [2 poäng]

En mapp i Linux representeras på samma sätt som en fil dvs med en inode som pekar ut ett datablock där datablocket innehåller namn och referenser till filer som ligger i mappen. Det betyder att vi använder samma systemanrop när vi vill läsa en mapp som när vi läser en fil - sant eller falskt? Motivera.

Svar: Falskt - även om mappar representeras med hjälp av samma structurer som en fil så är det konceptuellt en anna sak. Vi använder anrop som `opendir()` och `readdir()` för att läsa innehållet i en map.

9.6 sbrk() och sen då [2 poäng*]

Man kan använda systemanropet `sbrk()` för att allokera mer utrymmer för heapen men hur kan en process lämna tillbaks minne?

Svar: Genom att sätta toppen av heapen explicit genom att använd systemanropet `brk()`.

Namn:_____ Persnr:_____

9.7 en dyr operation [2 poäng]

Nedan är ett utsnitt från ett program som implementerar *Least Recently Used* (LRU). Koden visar på varför LRU är dyr att implementera och att man kanske istället väljer att approximera denna strategi. Vad är det koden gör och när används den?

```
    :
if(entry->present == 1) {

    if(entry->next != NULL) {

      if(first == entry) {
        first = entry->next;
      } else {
        entry->prev->next = entry->next;
      }
      entry->next->prev = entry->prev;

      entry->prev = last;
      entry->next = NULL;

      last->next = entry;
      last = entry;
    }
} else {

  :
}
```

Svar: Koden länkar ut ett entry och lägger det sist i en länkad lista som skall vara uppdaterad med de minst använda sidorna först. Operationen måste göras varje gång en sida refereras.

Namn:_____ Persnr:_____

9.8 volitile [2 poäng*]

I koden nedan ser man att `count` och `turn` har deklarerats som `volatile`. Varför är det viktigt att deklarera vissa variabler som `volitile` när vi arbetar med multitrådad kod?

```
int loop = 100000;
volatile int count = 0;
volatile int turn = 0;

void *toggle(void *args) {

  int id = ((struct ids*)args)->id;
  int od = ((struct ids*)args)->od;

  for(int i = 0; i < loop; i++) {
    while(turn != id) {}
    count++;
    turn = od;
  }
}
```

Svar: Deklareringen `volatile` är ett signal att variablen kan komma att ändras av någon anna tråd och att kompilatorn därför inte kan effektivisera koden genom att göra en läsning och sen lägga värdet i ett register. I koden ovan så är det meningen att en en annan process skall skriva till `turn` om den inte redan är satt till `id`. Om vi bara gör en läsning så kommer tråden inte att se förändringar som görs.

Namn:_____ Persnr:_____

9.9 character device [2 poäng*]

Vi kan skapa en s.k. *character device* och interagera med den med hjälp av `ioctl`. I koden nedan, beskriv vad `fd`, `JOSHUA_GET_QUOTE` och `buffer` är och hur vårt *device* kan tänkas fungera.

```
if (ioctl(fd, JOSHUA_GET_QUOTE, &buffer) == -1) {
  perror("Hmm, not so good");
} else {
  printf("Quote - %s\n", buffer);
}
```

Svar: Parametern `fd` är en fildeskriptor som vi får när vi öppnar den fil som modulen (*devicet*) har registrerat sig på. Parametern `JOSHUA_GET_QUOTE` är en kodad intruktion som beskriver vad vi vill ha gjort, om vi har givit en buffer och om den skall användas för läsning eller skrivnig. Den tredje parametern `buffer` är en minnesarea där modulen kan läsa från eller skriva till. I detta exempel kan man tänka sig att vi begär ett citat från modulen och denne skriver citatet i den angivna buffern.

Namn:_____ Persnr:_____

9.10 AC/DC [2 poäng*]

Om vi har allokerat en två-dimensionell array `table` och som vi sedan vill summera alla värden ur så kan vi göra det enligt koden nedan. Vad i koden nedan får ett oönskat beteende och hur skulle vi kunna förbättra körtiden? Motivera.

```
#define ROWS 4000
#define COLS 1000

int table[ROWS][COLS];

int main() {

  :
  long sum = 0;

  for(int c = 0; c < COLS; c++) {
      for(int r = 0; r < ROWS; r++) {
        sum += table[r][c];
      }
  }
  :

}
```

Svar: Problemet är att vi kommer att gå igenom matrisen kolumn för kolumn men den ligger rad för rad i minnet. Varje rad är 4 KByte (1000 int) så vi kommer att hoppa från sida till sida för när vi går från en rad till en annan. Om vi istället traverserar matrisen rad för rad så kommer vi att stanna inom samma sida och gå igenom alla element i raden innan vi går till nästa sida. Detta kommer ha inverkan på TLB:n och ge oss en bättre körtid. Prova gärna det är rätt så stor skillnad.

Namn:_____ Persnr:_____

10 Bara för omtentamen i ID2206 reggade före HT16 (4.5hp tentamen)

10.1 NFS och AFS [2 poäng]

NFS och AFS två exempel på distribuerade filsystem. Vad gör dessa system för att effektivisera skrivning och läsning av filer och vilka problem medför det?

Svar: Båda systemen håller lokala kopior av öppna filer på den klient som har öppnat filen. Läsningar och skrivningar kan då göras lokalt. Problem uppstår då flera klienter öppnar samma fil och skrivingar som gör av den ene inte omedlbart blir synliga för den adre eller ännu värre då skrivoperationer uppfattas som gjorda i olika ordning. Det blir ett problem att upprätthålla den sekvensiella semantik som vi är vana vid.

10.2 krypering med publika nyckar [2 poäng]

Om du får ett okryperat email där Alice ber dig välja ett av hundra alternativ och skicka tillbaks svaret krypterat med personens publika nyckel så att bara hon kan läsa svaret så kanske det kanske inte är det så säkert. Förklara varför det inte är säkert och att det i det hör fallet skulle vara betydligt säkrare om ni hade en hemlig symetrisk nyckel som du kunde använda.

Svar: Någon som läser breven kan kryptera alla de hundra alternativen med Alice publika nyckel (eftersom den är publik) och sen jämföra ditt svar med de hundra lternativen. Om du hade en symetriskt nyckel som du delade med Alice så skulle läsaren inte bli klokare.

10.3 multiprocessor [2 poäng*]

Om vi har en multiprocessor så måste en schemaläggare naturligtvis kunna låta processer köra på de olika processornerna. Vi kan dock göra bättre eller sämre schemalägging, beskriv en aspekt som vi måste ta hansyn till och hur vi anpassar schemaläggaren.

Svar: Vi måste till exempel ta hänsyn till vilken kärna en process körde på senast och i möjligaste mån låta processen köra på samma kärna får att förbättra cache-prestanda. Schemaläggaren kan ha en kö per kärna och endast flytta processer mellan kärnor om balansen blir alltför snedfördelad.

Namn:_____ Persnr:_____

10.4 at-least-once [2 poäng*]

Om vi har en implementering av RPC som erbjuder "at-least-once" så kan vi få problem när vi implementerar en tjänst. Vad är problemet och hur löser man det?

Svar: LANGProblemet är att anrop til tjänsten kan dubbleras och utför a mer än en gång. Man löser det genom att enbart ha s.k. idepotenta operationer dvs operationer som kan processas flera gånger utan att tillståndet förändras mer än vid första anropet.