

# Operativsystem ID2200/06

omtentamen

2017-12-18 14:00-18:00

## Instruktioner

- Du får, förutom skrivmateriel, endast ha med dig en egenhändigt handskrivna 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 registrerade 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 kompenseras 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 frågorna 1-8 och fråga 10
- För de som är registrerade 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]

Vad är foo och bar?

```
$ ls -l
total 8
drwxrwxr-x 2 kalle admin 4096 dec 1 11:53 bar
-rw-rw-r-- 1 jonny angels 58 dec 1 11:53 foo
```

### 1.2 kommandon i ett shell [2 poäng]

Ge en kort beskrivning av vad kommandona nedan gör.

- ln
- chown
- less
- pwd

Namn: \_\_\_\_\_ Persnr: \_\_\_\_\_

## 2 Processer

### 2.1 vad är problemet? [2 poäng]

Koden nedan kanske fungerar att kompilera men vi gör ett allvarligt misstag. Vilket fel gör vi och vad skulle kunna hända?

```
#include <stdlib.h>

#define SOME 100 // should be > 2

int *some_fibs() {

    int buffer[SOME];

    buffer[0] = 0;
    buffer[1] = 1;

    for(int i = 2; i < SOME; i++) {
        buffer[i] = buffer[i-1] + buffer[i-2];
    }
    // buffer contains SOME Fibonacci numbers
    return buffer;
}
```

### 2.2 Intel 80286 [2 poäng\*]

I processorn 80286 som lanserades 1982, hade Intel lagt till en privilegierad instruktion LIDT (Load Interrupt Descriptor Table). Vad menas det med att instruktionen är privilegierad och varför behöver denna instruktion vara privilegierad?

Namn: \_\_\_\_\_ Persnr: \_\_\_\_\_

### 3 Schemaläggning

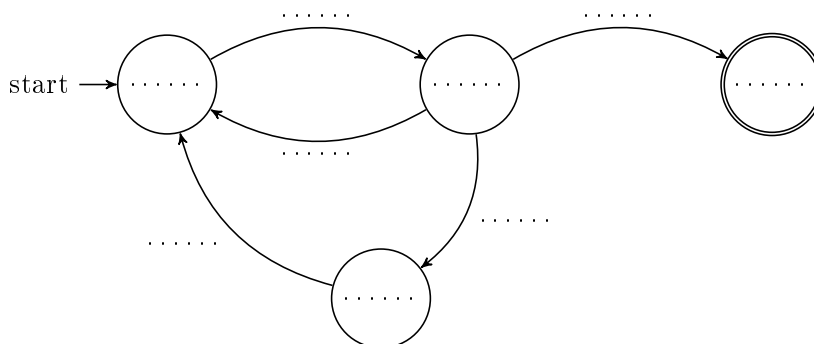
#### 3.1 interaktiva processer [2 poäng]

Antag att vi vill implementera en schemaläggare med bra responstid för interaktiva processer men att vi inte har kunskap om processerna är interaktiva eller inte. Vad är en bra heuristik för att avgöra vilka processer som är interaktiva och hur skulle vi kunna ge dem bättre responstid?

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.



### 3.3 stride scheduling [2 poäng\*]

Man skulle kunna implementera en *stride scheduler* genom att ha alla processerna i en lista ordnad efter ett s.k. *pass value*. Den process som är först i listan väljs som den process som skall exekverar. När processen har exekverats så skall den läggas in i listan igen, på vilken position skall den läggas in?

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?

### 4.2 ett träd [2 poäng]

Vid representation av en *sidtabell* (page table) så används en trädstruktur. Varför har man en trädstruktur, det skulle gå betydligt snabbare att slå upp ett värde om man representerade tabellen som en *array* med direkt åtkomst till elementen. Ett träd ger oss bara en eller flera indirektionssteg så varför använda ett träd?

Namn: \_\_\_\_\_ Persnr: \_\_\_\_\_

### 4.3 x86\_64 adressering [2 poäng\*]

I en x86-processor i 64-bitarsmode så innehåller ett PTE en ramadress på 40 bitar. Denna kombineras med den virtuella adressens offset på 12 bitar till en fysisk adress. Detta blir 52 bitar men en process har endast 48-bitars virtuellt minne. Vilken fördel får vi genom att ha en 52-bitars fysisk adress?

## 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?



Namn: \_\_\_\_\_ Persnr: \_\_\_\_\_

## 5.2 vem är din buddy [2 poäng]

Antag att vi använder buddy-allokering och har minsta block på 16 bytes. Om vi frigör ett block på 32 bytes som har nummer 0b001010, vilket block är då vår buddy?

## 5.3 adressorterad [2 poäng\*]

Antag att vi implementerar en minneshanterare och länkar de fria blocken i en enkellänkad lista. Om vi håller denna lista sorterad på adressordning så kanske vi kan hitta block som skall slås ihop när ett block frigörs. Hur skulle implementationen se ut, hur hittar vi block som kan slås ihop?

Namn: \_\_\_\_\_ Persnr: \_\_\_\_\_

## 6 Flertrådad programmering

### 6.1 count [2 poäng]

Om vi exekverar proceduren `hello()` nedan samtidigt i två trådar så blir resultatet - ja vad blir resultatet? Hur kan det komma sig?

```
int loop = 10;

void *hello () {
    int count = 0;

    for (int i = 0; i < loop; i++) {
        count++;
    }
    printf("count is: %d", count);
}
```

### 6.2 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?

Namn: \_\_\_\_\_ Persnr: \_\_\_\_\_

### 6.3 thread local storage [2 poäng\*]

Antag att vi implementerar en minnes hanterare (alloc/free) där vi använder nedanstående konstruktion för att hantera frilistan. Vilka fördelar och eventuell nackdelar skulle detta medföra?

```
__thread chunk *free = NULL;

void free(void *memory) {
    if(memory != NULL) {
        struct chunk *cnk = (struct chunk*)((struct chunk*)memory - 1);
        cnk->next = free;
        free = cnk;
    }
    return;
}
```

Namn: \_\_\_\_\_ Persnr: \_\_\_\_\_

## 7 Filsystem och lagring

### 7.1 rpm [2 poäng]

De finns hårddiskar med olika prestanda, en sak som skiljer dem är hur snabbt de roterar. Vad har rotationshastigheten för betydelse, vad förbättras och vad kan komma att försämrats?

### 7.2 vad var [2 poäng]

Antag att vi har ett enkelt filsystem utan journal där vi skriver direkt till inoder, bitmappar och datablock. Antag att vi skall skriva till en fil och behöver ytterligare ett datablock. Vilka strukturer kommer att ändras och vilken förändring skall genomföras.

Namn: \_\_\_\_\_ Persnr: \_\_\_\_\_

### 7.3 loggbaserade fs [2 poäng\*]

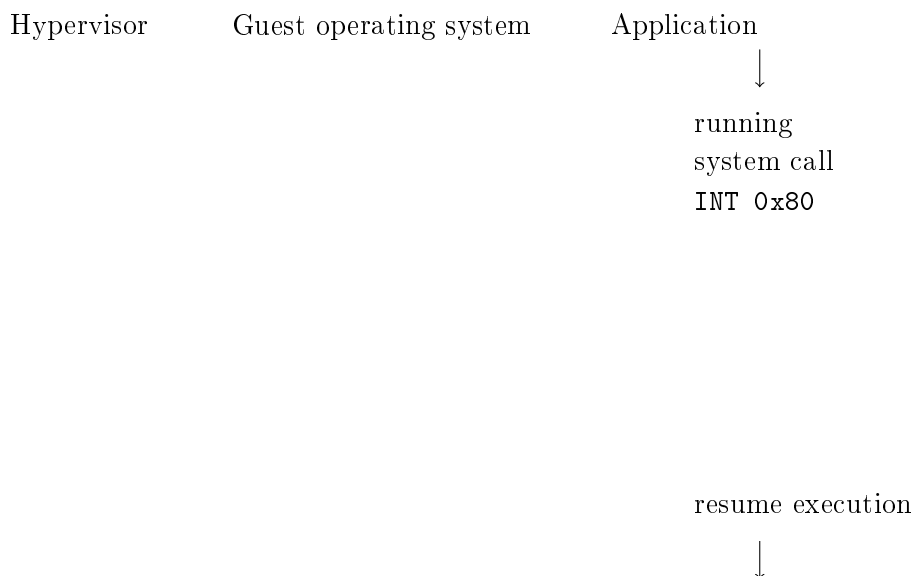
I ett loggbaserat filsystem skriver vi alla förändringar i en kontinuerlig logg utan att göra förändringar i de redan existerande block som en fil har. Vad är poängen med att hela tiden skriva nya modifierade kopior av datablock istället för att gå in och göra de små förändringar som vi vill göra? Om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

Namn: \_\_\_\_\_ Persnr: \_\_\_\_\_

## 8 Virtualisering

### 8.1 ett system anrop [2 poäng]

När vi gör ett systemanrop i en användarprocess i ett virtualiserat operativsystem så kommer kontroll föras mellan: användarprocessen, det virtualiserade operativsystemet och hypervisorn. Visa i ett sekvensdiagram vad som händer från det att användarprocessen exekverar `INT 0x80` tills det att den får tillbaks kontrollen.



### 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 problemet och hur man kan lösa det?.

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]
ffffffff600000-ffffffff601000 r-xp 00000000 00:00 0          [vsyscall]
```

Namn: \_\_\_\_\_ Persnr: \_\_\_\_\_

## 9.2 fork [2 poäng]

Om vi kör programmet nedan, vad kommer att skrivas ut på skärmen? Varför?

```
int x = 0;

int main() {

    int pid;

    pid = fork();

    if(pid == 0) {
        printf("child: x is at %p\n", &x);
    } else {
        printf("mother: x is at %p\n", &x);
        wait(NULL);
    }

    return 0;
}
```



Namn: \_\_\_\_\_ Persnr: \_\_\_\_\_

### 9.3 pipes [2 poäng]

Vi kan enkelt (typ enkelt) göra en `fork()` och sen sätt upp `stdin` och `stdout` för de båda processerna kommunicerar via en s.k. *pipe*. Hur kan vi få två processer att göra det samma om det inte är vi som skapar den andra processen via `fork()`. Hur kan en process skapa en pipe som en annan process skall läsa från?

Namn: \_\_\_\_\_ Persnr: \_\_\_\_\_

#### **9.4 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.

#### **9.5 sbrk() och sen då [2 poäng]**

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

Namn: \_\_\_\_\_ Persnr: \_\_\_\_\_

### 9.6 en billig operation [2 poäng]

Nedan är ett utsnitt från ett program som implementerar *klock-algoritmen*.  
Koden visar på varför klock-algoritmen är billigare än implementationen av  
LRU. Vad skulle vi gjort i motsvarande fall om vi implementerat LRU?

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

Namn: \_\_\_\_\_ Persnr: \_\_\_\_\_

## 9.7 execlp() [2 poäng\*]

I programmet nedan anropar vi biblioteksproceduren `execlp()` som ersätter den körande processens programkod. Här använder vi det för att anropa `/bin/ls` och skriva ut dels aktuell mapp (*current directory*), dels användarens hemma-mapp. Varför kommer detta inte att fungera?

```
int main() {

    int pid = fork();

    if(pid == 0) {

        char cwd[1024];
        getcwd(cwd, sizeof(cwd));
        printf(" This is the current directory: \n");
        execlp("/bin/ls", "/bin/ls", &cwd, NULL);

        printf(" This is your home directory: \n");
        execlp("/bin/ls", "/bin/ls", getenv("HOME"), NULL);

    } else {
        wait(NULL);
    }
    return 0;
}
```

Namn: \_\_\_\_\_ Persnr: \_\_\_\_\_

### 9.8 sockets [2 poäng\*]

När vi vill kommunicera mellan två processer kan vi använda så kallade *sockets*. Dessa finns i flera versioner bland annat `SOCK_STREAM` och `SOCK_DGRAM`. Beskriv skillnaden mellan dessa olika varianter.

### 9.9 delat minne [2 poäng\*]

Processer i ett Unix-system kan kommunicera med varandra via flera olika kanaler. De kan även dela minnesareor på liknande sätt som två trådar i en process kan dela *heap* och global data. Hur kan vi få två processer att dela minne?

Namn: \_\_\_\_\_ Persnr: \_\_\_\_\_

### 9.10 context [2 poäng\*]

Med hjälp av biblioteksanropet `getContext()` kan en process spara undan sitt eget så kallade *context*. Vi skulle kunna bygga upp ett bibliotek som lät oss skapa nya exekverande trådar och växla mellan dessa manuellt genom att en exekverande tråd anropade en schemaläggare.

Varför skulle vi vilja bygga upp ett liknande bibliotek, finns det några fördelar? Vad skulle nackdelarna vara?

Namn: \_\_\_\_\_ Personr: \_\_\_\_\_

## **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?

### **10.2 saltade lösenord [2 poäng]**

När lösenord lagras i krypterad form på en server så används ofta ett så kallat *salt*. Vad har saltet för funktion?

Namn: \_\_\_\_\_ Persnr: \_\_\_\_\_

### **10.3 multiprocessor [2 poäng\*]**

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

### **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?