Operativsystem ID1200/06 och ID2200/06

Tentamen TENA 6 hp

2018-04-03 14:00-18:00

Omtentander på ID2200 TEN1 3.8 hp och ID2206 TEN1 4.5 hp skall inte skriva denna tentamen!

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.

Betyg

Tentamen har ett antal uppgifter där några är lite svårare än andra. De svårare uppgifterna är markerade med en stjärna, *[p*]*, 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 24 grundpoängen räknas bara som högst 22 och, att högre poäng inte kompenserar för avsaknad av grundpoäng. Gränserna för betyg är som följer:

- Fx: 12 grundpoäng
- E: 13 grundpoäng
- D: 16 grundpoäng
- C: 20 grundpoäng
- B: 22 grundpoäng och 6 högre poäng
- A: 22 grundpoäng och 10 högre poäng

Gränserna kan komma att justeras nedåt men inte uppåt.

Namn:_____ Persnr:_____

1 Processer

1.1 fork() [2p]

När man skapar en ny process genom att använda `fork()` så kommer de två processerna att **dela** vissa saker. Vilka, om några, av följande saker kommer de två processerna att dela?

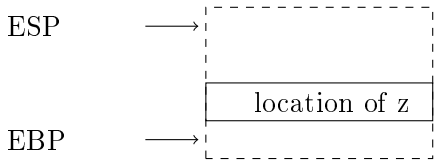
- Stack
- Heap
- Globalt minne
- Kod-area
- Öppna filer

1.2 vad finns på stacken [2p]

Antag att vi gör ett proceduranrop till `foo()` som i exemplet nedan och att all informations läggs på stacken dvs vi använder inte register för argument och resultat, vad finns då på stacken när vi kommer in i `foo`? Rita upp en stack och beskriv vad som finns där. Visa även var stackpekaren och baspekaren pekar.

```
int foo(int x, int y) {
  return x + y;
}

int bar() {
  int z;
  z = foo(3, 4)
  return z;
}
```



Namn:_____ Persnr:_____

1.3 vad är felet [2p]

I programmet nedan har vi en procedur `print_fibs/0` som med hjälp av `some_fibs/1` skriver ut en sekvens med Fibonacci-tal från 0 till 46. Allt fungerar utmärkt, eller.... vad har vi glömt och vilka konsekvenser kan det få?

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

#define MAX 46

int *some_fibs(int n) {

  if (n <= MAX) {
    int *buffer =  malloc((n + 1) * sizeof(int));
    buffer[0] = 0;
    buffer[1] = 1;
    for(int i = 2; i <= n; i++) {
      buffer[i] = buffer[i-1] + buffer[i-2];
    }
    return buffer;
  } else {
    return NULL;
  }
}

void print_fibs() {

  int *fibs;
  int n = 46;

  fibs = some_fibs(n);
  for(int i = 0; i <= n; i++) {
    printf("fib(%d) = %d\n", i, fibs[i]);
  }
}
```

Namn:_____ Persnr:_____

1.4 varför malloc [2p]

När man programmerar i C så måste man explicit allokera en del datastrukturer på heapen med hjälp av malloc. I språk som har s.k. automatisk minneshantering, som till exempel Java, Erlang och Python, så behöver man inte tänka på var någonstans datastrukturer allokeras. Hur fungerar dessa språk? Allokeras allt på stacken finns det ingen heap? Om det finns en heap måste man då inte ta bort datastrukturer som inte används? Förklara hur dessa språk hanterar minnet.

1.5 lika-lika [2p*]

När man implementerar malloc/free är ett alternativ att bara dela ut block som är en jämn exponent av två. Man kanske har någon nedre gräns så man kanske delar ut block av storlek: 16 byte, 32 byte, 64 byte, 128 byte osv. Detta har vissa fördelar men även vissa nackdelar. Beskriv hur systemet skulle kunna implementeras och dess för och nackdelar.

Namn:_____ Persnr:_____

1.6 yield() eller futex_wait() [2p*]

Om vi implementerar ett spin-lock så kan vi anropa `sched_yield()` eller `futex_wait()` som i exemplen nedan, om vi inte får låset på första försöket. Varför vill vi göra det och vilken skillnad finns mellan de båda lösningarna.

```
int try(int *lock) {
  __sync_val_compare_and_swap(lock, 0, 1);
}

int futex_wait(volatile int *futexp) {
  return syscall(SYS_futex, futexp, FUTEX_WAIT, 1, NULL, NULL, 0);
}
```

Så här:

```
void lock(volatile int *lock) {
  while(try(lock) != 0) {
    sched_yield();
  }
}
```

eller så här:

```
void lock(volatile int *lock) {
  while(try(lock) != 0) {
    futex_wait(lock, 1);
  }
}
```

Namn:_____ Persnr:_____

2 Kommunikation

2.1 count [2p]

Om vi exekverar programmet nedan så kommer vi skriva ut det slutgiltiga värdet på variabeln `count`. Vilka möjliga utskrifter kan vi få och varför?

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

int count = 0;

int main() {

  int *status;

  int pid = fork();

  if( pid == 0) {
    for (int i = 0; i < 10; i++) {
      count += 1;
    }
    return 0;
  } else {
    for (int i = 0; i < 10; i++) {
      count += 1;
    }
    wait(status);
  }
  printf("count = %d\n", count);
  return 0;
}
```

Namn:_____ Persnr:_____

2.2 pipes [2p]

Vilket av följande påståenden, om något, är sant?

- En pipe är en dubbelriktad kanal för kommunikation mellan processer inom samma operativsystem.
- När man läser från en pipe läser man ett meddelande åt gången. Man får alltid ett helt meddelande, aldrig ett halvt.
- När man läser från eller skriver till en pipe så använder man de vanliga biblioteksrutinerna som används för filer.

2.3 typer av socket [2p*]

När vi skapar en socket så anger vi vilken typ av socket vi vill ha; nedan är tre typer listade. Ange för varje typ vilka egenskaper som den socket som skapas har. Notera egenskaper såsom: byte- eller meddelandeorienterad, enkel- eller dubbelriktad, tillförlitlighet, ordnad och flödeskontroll.

- `SOCK_DGRAM`

- `SOCK_STREAM`

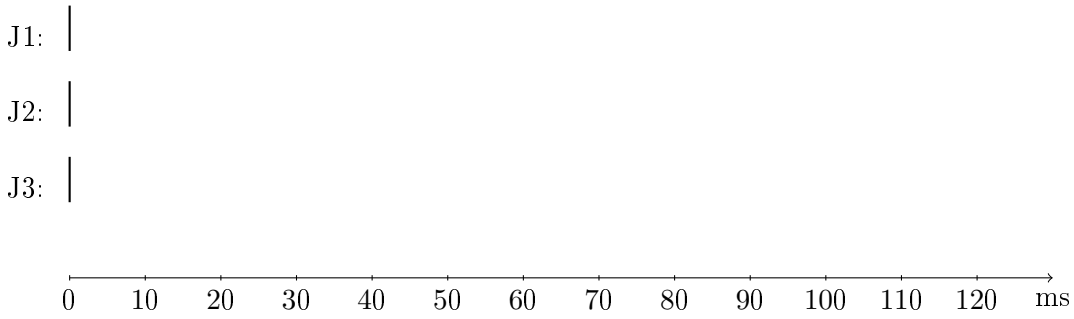
- `SOCK_SEQPACKET`

Namn:_____ Persnr:_____

3 Schemaläggning

3.1 sämre omloppstid [2p]

Visa med ett tidsdiagram hur vi kan få sämre omloppstid (turnaround time) om vi istället för att köra tre jobb på 30 ms var efter varandra, implementerar round-robin där varje jobb får en tidslucka på 10 ms.



Namn:_____ Persnr:_____

3.2 MLFQ - WTF! [2p]

Antag att vi implementerar en MLFQ för vår schemaläggning av jobb enligt reglerna nedan. Vad kommer problemet bli och hur skulle vi kunna lösa det?

- Regel 1: om Prioritet(A) $>$ Prioritet(B) så schemalägg A.
- Regel 2: om Prioritet(A) $=$ Prioritet(B) så schemalägg A och B i *round-robin*.
- Regel 3: ett nytt jobb startar med högsta prioritet.
- Regel 4a: ett jobb som måste avbrytas (tidsluckan förbrukad) flyttas till en lägre prioritet.
- Regel 4b: ett jobb som initiaserat en I/O-operation (eller *yield*) stannar kvar på sin nivå.

3.3 en zombie [2p*]

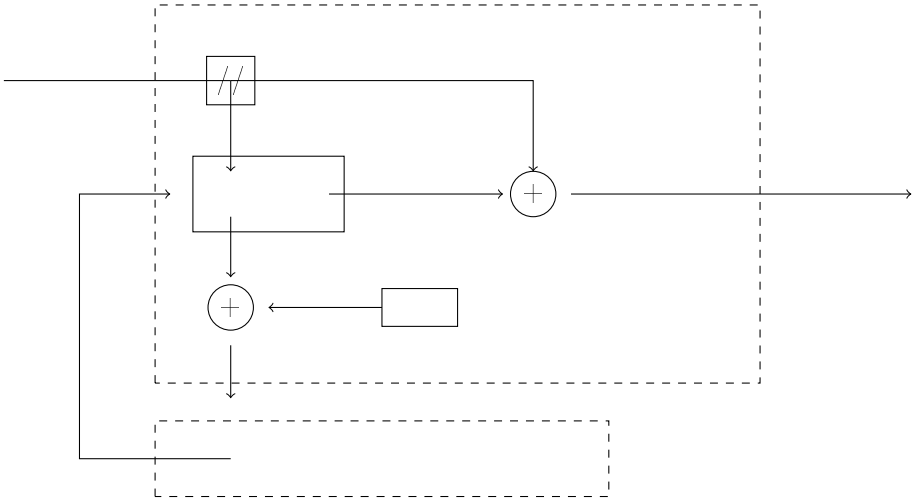
En process kan befinna sig i ett s.k. zombie-tillstånd. Vad betyder detta, varför är processen i det tillståndet, när tas den därifrån och vad händer sedan med processen?

Namn:_____ Persnr:_____

4 Virtuellt minne

4.1 En sidad MMU med TLB [2p]

Nedan är en schematisk bild av en MMU som använder sig av en TLB för att översätta en virtuell adress till en fysisk adress. Identifiera följande enheter och adresser: virtuell adress, fysisk adress, register för basadress till sidtabell (PTBR), offset i sida, sidnummer (VPN), ramnummer (PFN), TLB, sidtabell, sidtabellspost (page table entry, PTE).



Namn:_____ Persnr:_____

4.2 den gömda koden [2p]

Vid implementering av malloc/free så kan man göra ett litet trick och "gömma" information om ett block som man delar ut. Vilken information är det som man gömmer och hur göms den? Förklara varför det är nödvändigt, hur det görs och var i koden nedan det sker.

```
void *malloc(size_t size) {
  if( size == 0 ){
    return NULL;
  }
  struct chunk *next = flist;
  struct chunk *prev = NULL;

  while(next != NULL) {
    if (next->size >= size) {
      if(prev != NULL) {
        prev->next = next->next;
      } else {
        flist = next->next;
      }
      return  (void*)(next + 1);
    } else {
      prev = next;
      next = next->next;
    }
  }

  void *memory = sbrk(size + sizeof(struct chunk));
  if(memory == (void *)-1) {
    return NULL;
  } else {
    struct chunk *cnk = (struct chunk*)memory;
    cnk->size = size;
    return  (void*)(cnk + 1);
  }
}
```

Namn:_____ Persnr:_____

4.3 inverterad sidtabell [2p*]

En inverterad sidtabell är en tabell som givet processidentifierare och sidnummer returnerar vilken ram som sidan finns i (om den finns i minnet). Tabellen har ett element per ram i minnet och kan inte indexeras med hjälp av sidnummret, man måste på något sätt söka igenom tabellen för att hitta rätt ram.

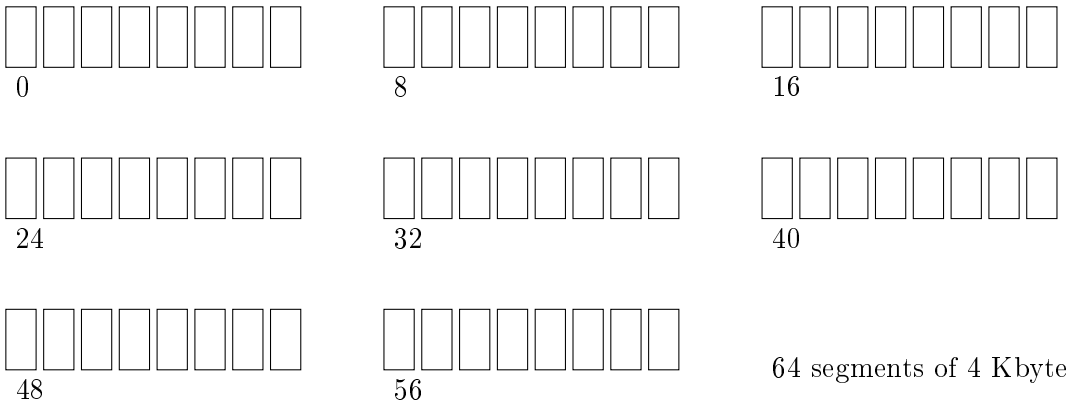
Vad är fördelen med att ha en inverterad sidtabell om uppslagningen kommer att ta längre tid? Förklara vilka fördelar vi får och när det är mest fördelaktigt att använda en inverterad tabell.

Namn:_____ Persnr:_____

5 Filsystem och lagring

5.1 ett enkelt filsystem [2p]

Beskriv hur ett enkelt filsystem skulle kunna organiseras på en hårddisk som är uppdelat i segment om 4 Kbyte. Beskriv vilka datastrukturer som behövs och rita in hur dessa kan organiseras på hårddisken (vi tar en minimal disk som exempel).



Namn:_____ Persnr:_____

5.2 innehållet i en map [2p]

Antag att du har skrivit ditt eget program för att lista innehållet i en map enligt nedan. Du har testat ditt program på olika mappar och allt tycks fungera. Plötsligt får du ett segmenteringsfel, vad har gått fel?

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

int main(int argc, char *argv[]) {

  if( argc < 2 ) {
    perror("usage: myls <dir>\n");
    return -1;
  }

  char *path = argv[1];

  DIR *dirp = opendir(path);

  struct dirent *entry;

  while((entry = readdir(dirp)) != NULL) {
    printf("\tinode: %8lu",  entry->d_ino);
    printf("\tname: %s\n", entry->d_name);
  }

  return 0;
}
```

Namn:_____ Persnr:_____

5.3 loggbaserade fs [2p*]

I ett log-baserat filsystem så har inoderna inte någon bestämd plats utan skrivs i segment tillsammans med datablocken. Problemet är då att hitta de inoder man söker efter: hur vet man var olika inoder ligger? Beskriv och rita in i figuren nedan hur man hittar de inoder man söker efter.