Tentamen i Operativsystem, HI1025:TEN1 - 23 mars, 2016

Allmänna instruktioner. Tentamen innehåller 10 frågor/uppgifter med totalt 48 poäng. För lägsta godkända betyg (E) krävs ungefär 22-23 poäng. För att få komplettera (Fx) krävs ungefär 20-21 poäng.
Viktigt: När du svarar på en fråga ska det i allmänhet framgå varför du vet svaret frågan, det uppnås enklast genom att du sätter in termer och begrepp som du hanterar i deras sammanhang på ett korrekt sätt. Men det är också viktigt att inte bli för mångordig, i uppgifter med 1 poäng krävs ett kort svar, i uppgifter med mer poäng behöver svaret utvecklas mer. **OBS: Svara *inte* i tentan, *bara* på svarspapper.**

Anmärkning: Av layoutskäl saknas nödvändiga `#include`-direktiv i källkoden i vissa av uppgifterna, men uppgifterna ska behandlas som om direktiven fanns där.

Lycka till!
Johnny

Uppgifter

(1) Förklara vad *time-sharing* är och hur det går till. I din redogörelse ska nyckelorden *process*, *CPU*, *context switch* ingå och vara insatta i sammanhanget på ett korrekt sätt. Det ska klart framgå hur *time-sharing* virtualiserar CPU:n. **(4p)**

(2) Definiera termerna *process* och *tråd* på ett noggrannt sätt som inte bara är ”en process är ett program som kör”. Definitionen av process ska också klart och tydligt bygga på definitionen av tråd. Följande termer ska ingå och förklaras i definitionen av tråd: *programräknare*, *stack*. Följande ord ska ingå och förklaras i definitionen av process: *processbild*, *processid*, *tråd*. Av de båda defintionerna ska det klart framgå varför man inte kan betrakta trådar och processer i opposition till varandra. (Ibland kan man höra ord som ”trådar är bättre än processer”, men rent tekniskt kan vi inte säga så.) **(6p)**

(3) Förklara följande systemanrop vad de gör, varför de är nödvändiga. Förklara hur de samverkar **(1p)**.
   (a) `fork()` **(2p)**
   (b) `exec()` (principen bakom `execve()`/`execlp()`/`execl()` etc.) **(2p)**
   (c) `wait()` **(2p)**

(4) Förklara skillnaden mellan ett systemanrop och ett anrop av en vanlig funktion, vad menas med att man måste kontrollera returvärdet från ett systemanrop och varför måste vi det? Ge ett konkret exempel på hur du gjort då du arbetat med övningar och laborationer där detta förekommer och motivera just varför du kontrollerat returvärdet av det systemanrop du exemplifierar. **(4p)**

(5) Förklara vad en DMA är och hur den ökar användningen av CPU:n i ett operativsystem jämfört med om systemet inte skulle ha en DMA. **(2p)**

(6) Förklara skillnaden mellan hårda och symboliska (mjuka) länkar och ange deras användningsområden. Se till att det användningsområde du anger för symboliska länkar inte fungerar med hårda länkar. **(4p)**

(7) Du arbetar som systemadministratör på ett företag. Det system du arbetar med har olika partitioner monterade i en traditionell *UNIX*-filhierarki. (`/dev/sdaX`, `X=1,2,3,...`.) Katalogen `/var` är separat monterad på `/dev/sda6`. Du vill nu byta filsystemstyp till `XFS` på den partitionen men förstås inte förstöra innehållet. Hur gör du detta? Du har tillgång till en reservhårddisk (`/dev/sdb1`) för att mellanlagra data när du formaterar om. Ange i detalj, steg för steg, hur du skulle använda kommandona `mkdir`, `cp` (eller `mv`), `mkfs.xfs` (skapar ett XFS-filsystem), och liknande kommandon för att åstadkomma detta. *Ledning:* Ett av delstegen är `mkfs.xfs /dev/sda6` som ju formaterar om partionen som `/var` är monterad på men det måste göras på rätt sätt så att data inte förstörs. **(3p)**

(8) Då en process innehåller parallella trådar som kör med gemensamma delade resurser måste dessa trådar ibland synkroniseras. Beskriv vad synkronisering innebär, varför det är viktigt och hur det går till. I din redogörelse ska begreppen *kapplöpning/race condition*, *mutual exclusion/ömsesidig uteslutning*, *mutex*, *kritisk sektion/critical section* förekomma och vara behandlade på ett korrekt sätt. **(7p)**

(9) Studera nedanstående program:

```
main()
{
  int x=10,y=20,z=0, p1[2], p2[2]; pipe(p1); pipe(p2);
  close(p2[1]); close(p2[0]);
  if(!fork()) {
    close(p1[1]); close(p2[0]);
    while(read(p1[0],&z,sizeof(int))){z=z*3;write(p2[1],&z,sizeof(int));}
    close(p2[1]); close(p1[0]);
    if(!fork()) {
      close(p2[1]); close(p1[0]); close(p1[1]);
      while(read(p2[0],&z,sizeof(int))){z=z/2;printf("%d\n",z);}
      close(p2[0]); exit(0);
    }
    wait(0);
    exit(0);
  }
  close(p1[0]);
  write(p1[1],&x,sizeof(int)); write(p1[1],&y,sizeof(int));
  close(p1[1]);
  wait(0);
}
```

(a) Programmet fungerar inte som det ska, beskriv programmets funktion och rita ett tidsdiagram som illustrerar de ingående processernas relationer men rita inte in de två piparna. Förklara vad som händer när programmet kör och ange dess utmatning. **(3p)**

(b) Programmet är tänkt att ta talen 10 och 20 och skicka in det i en barnprocess som multiplicerar talen med 3 och sedan skickar vidare till en barnbarnprocess som delar med 2 och skriver ut talen. Rätta programmet så att det fungerar som det ska och förklara din rättning. **OBS:** Rättningen ska göras genom att du flyttar om kod, inte lägger till ny kod eller tar bort kod. Rita ett nytt tidsdiagram som hör till den rättade versionen, nu med piparna inritade. **(4p)**

(10) Studera nedanstående program:

```
int file1, file2; pthread_mutex_t lock1, lock2;
void * tf1 (void * p){
  pthread_mutex_lock(&lock1); pthread_mutex_lock(&lock2);
  work_with_1(file1,file2);
  pthread_mutex_unlock(&lock1); pthread_mutex_unlock(&lock2);
}
void * tf2 (void * p){
  pthread_mutex_lock(&lock2); pthread_mutex_lock(&lock1);
  work_with_2(file1,file2);
  pthread_mutex_unlock(&lock1); pthread_mutex_unlock(&lock2);
}
main(){
  pthread_t t1, t2; init (&lock1,&file1,&lock2,&file2);
  pthread_create(&t1,NULL,&tf1,NULL); pthread_create(&t2,NULL,&tf2,NULL);
  while(1);
}
```

Två trådar arbetar med två delade resurser i form av två fildeskriptorer `file1` och `file2`. Dessa resurser vaktas av `lock1` respektive `lock2`. Det finns ett allvarligt problem med ovanstående, förklara detta problem och föreslå en modifikation av koden som löser problemet. Förklara också varför ditt förslag löser problemet. (Du behöver inte ta bort eller lägga till kod för att lösa uppgiften och problemet finns inte i `main()`. `init()` gör allt som behövs för att programmet ska fungera, förutom en sak.) **(4p)**