Studieplan del 2

Det förutsätts att man gör förberedelser inför kursmötena. Mötena i kursen är av tre slag, föreläsningar, Peer-instruction-möten och Peer-assessment-möten. *PI = Peer Instruction* och *S = Seminarium*. Vid en *FL* eller ett *PI*-möte går vi igenom nytt kursinnehåll, för att hänga med ska man ha gjort vissa förberedande självstudier och övningar innan (det förutsätts dock inte inför föreläsningarna). Dessa förberedelser beskrivs i studieplanerna. När man gjort dessa kan man ta till sig det nya kursinnehållet bättre vid ett *PI*-kursmöte. Vid ett *S*-möte – Seminarium - (det finns två stycken totalt) ska ska vi titta på gamla tentafrågor och exempelfrågor som skulle kunna komma på datortentan. Vi ser på rättningen av de gamla tentafrågorna gemensamt på mötet och diskuterar utfallet av rättningen och det bakomliggande kursmaterialet. Ett *S*-möte fungerar då alltså som ett slags repetition av en del av kursen med fokus på hur man ska klara tentan. Två buffertmöten finns som vi använder till att repetera begrepp och svara på olika frågor, se till att komma med frågor till dessa tillfällen.

Det finns i allmänhet övningar hörande till alla kursmötena. Dessa övningar hänvisas till i det förberedelsearbete och efterarbete som beskrivs nedan. I allmänhet bör du också göra övningarna inför och efter varje kursmöte, men det är kanske lite svårare att göra dem och följa med i kursens takt. För att följa med i kursens takt är denna studieplan framtagen så satsa på att följa den och arbeta med övningarna i din egen takt.

Del I av kursen: *Systemprogrammering*

Kursmöte 6 (30 jan) (*FL*): *Introduktion till kursen och systemprogrammering.*

Efter föreläsningen: Skriv in en del av exempelprogrammen och provkör dem (gör så många du behöver). Lägg gärna in `sleep()` direkt efter `fork()`-anropen för att lättare kunna studera strukturen på de körande processerna med `ps`. Du kan också göra detta efterarbete uppblandat med förarbetet inför nästa kursmöte.

Exempel: Om man skriver in programmet

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

main()
{
  pid_t pid;
  pid = fork();
  sleep(10);
}
```

i filen `testafork1.c` och sedan kör "`gcc testafork1.c; ./a.out & ps -l`" ("`-l`" är ett litet "l", inte en etta.) så kan man få utskriften

```
[1] 5607
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  5540  5538  0  80   0 -  4462 wait   pts/1    00:00:00 bash
0 S  1000  5607  5540  0  80   0 -   932 hrtime pts/1    00:00:00 a.out
0 R  1000  5608  5540  0  80   0 -  1598 -      pts/1    00:00:00 ps
1 S  1000  5609  5607  0  80   0 -   932 hrtime pts/1    00:00:00 a.out
```

vilket ska tolkas så här: Programmet `a.out` kör och får processid 5607. Detta program skapar en

barnprocess som också kallas `a.out` (`fork` skapar ju kopior av processer) och denna process har processid 5609. Kommandot `ps` självt (som skriver ut denna information) är också en process med id 5608 och skapas tydligen innan barnprocessen 5609 skapas. Tolken (*bash*) som vi använder för att köra kompilering och `ps` i har tydligen processid 5540. Kör `ps -l` ett par gånger tills du ser att barnprocessen försvinner. Arbeta med liknande sätt med de andra exempelprogrammen.

Kursmöte 7 (1 februari) (*PI*): *Uppföljning av föreläsningen.*

Innan mötet: Läs början av OSTEP om processer och hela kapitel 3 i ALP, utom avsnittet om signaler. Man behöver inte läsa och förstå allting, men läs så gott det går. Provkör samtliga exempelprogram från föreläsningen (eller i alla fall de du känner du behöver/vill prova). På kursmötet kommer vi att ställa frågor om parallella processer och förtydliga det som varit svårt att förstå. Här finns en viktig sak att komma ihåg: Kursmötet är inte en föreläsning! Kursmötet är primärt till för att stödja er inlärning av saker ni redan stött på – för att få tid till detta är det viktigt att du läser på egen hand, vi kommer alltså inte att gå igenom det som du läst från litteraturen men vi kommer att ställa frågor på det på kursmötet och era frågor är som vanligt mycket välkomna.

Studera följande kommandorad:

```
sleep 10 & ps -o pid,ppid,pgid,sid,comm | cat2 | cat
```

Här är `cat2` bara en kopia av `cat`, skapas genom kommandot `cp /bin/cat /bin/cat2` som man kan göra om man är inloggad som `root`. Denna kommandorad illustrerar hur Stevens och Rago (i referenslitteraturen) använder en skräddarsydd variant av `ps`-kommandot för att se olika attribut hos processer. Experimentera med denna kommandorad (genom att förlänga pipen, med flera `cat`, eller lägga till flera `sleep` eller liknande) och se hur resultatet förändras. Ta också reda på vad `ps -o stat,pid,comm` gör och testa det. (Använd `man ps` för att få veta vad `stat` betyder.) Skriv ett program i *C* som utför kommandot ”`ps -o pid,ppid,pgid,sid,comm | cat`” med hjälp av anropet `system()`, du kan först testa ett enkelt *C*-program som bara gör `system("ls");` för att förstå vad som ska ske.

Kursmötet: Denna gång kommer vi inte att presentera så mycket nytt material alls. Det är meningen att vi ska använda det vi sett på föreläsningen och övningarna som är beskrivna ovan för att behandla dessa frågeställningar. Så gör ett bra försök på alla övningar och experiementera med testprogrammen på föreläsningen så blir kursmötet mer givande.

Efter mötet: Använd kommandot `kill` för att avsluta de processer du har sövt med `sleep` (alltså det som gjordes omdelbart efter föreläsningen).

Kursmöte 8 (3 feb) *(S)*: Seminarium. Vi studerar gamla tentafrågor tittar särskilt på hur studenter svarat på tentafrågor som inte lyckats få maximal poäng på tentan.

Kursmöte 9 (6 feb) (*PI*): *Processhierarkier, processgrupper, sessioner och signaler.*

Innan mötet: Repetera kursmöte 5. Vår attityd här ska bli inte att plugga in en massa teori utan att skriva systemprogram som fungerar som det är tänkte genom att meka med dessa program.. Läs också avsnittet om signaler i ALP i kapitel 3. Skriv in och testa programmet `sigurs1.c` ur ALP

på sidan 54, där det står `/*Do some lengthy stuff*/`, skriv in koden

```
printf("PID: %d.\n", getpid());
sleep (50);
```

för att låta programmet ange sitt processid. Efter du startat programmet, vid kommandoprompten, ge kommandot "`kill -USR1 1234`" (om processid:t var 1234) och se att processen verkligen fångar upp signalen `USR1`. Programmet kommer troligen att avbrytas efter signalen är uppfångad. Kan du få programmet att fånga `USR1` flera gånger? (Gräv inte för mycket i det, men det kan vara en intressant frågeställning.)

Studera följande kommandorad:

```
sleep 10 & ps -o pid,ppid,pgid,sid,comm | cat2 | cat
```

återvänd till experimenten med kommandona av typen `sleep 10& ps -o pid,ppid,pgid,sid,comm | cat2 | cat` och rita skisser över relationerna mellan de körande processerna.

Efter mötet: Följande program startar ett antal barnprocesser som inte gör något annat än att sova i 1 sekund:

```
int main(int argc, char *argv[])
{
  int i, n = atoi(argv[1]);
  for(i=0;i<n;i++)
    if(fork()); else {sleep(1); exit(0);}
  system("ps -o pid,ppid,pgid,sid,comm");
}
```

Studera kommandot `killpg()` och använd det för att skriva ett annat program som samverkar med detta program på följande sätt: program 1 skapar ett antal barnprocesser (alla barn till en förälder) som alla endast sover i 30 sekunder och inte gör någonting annat. (Ändra 1 till 30 i argumentet till `sleep()` ovan.) Program 2 tar sedan ett processgruppsid som indata och skickar signalen –9, (alltså *process termination*) till alla processer i en speciell processgrupp. Testa program 2 på program 1 genom att med program 1 skapa 10 barnprocesser i en viss processgrupp och avsluta alla dessa genom att köra program 2. Testa också program 2 på exempel liknande de från kurmöte 5. Skriv också ett program som startar X barnprocesser där barnprocess nr Y ska skapar ett barnbarn. X och Y ska anges på kommandoraden som startar programmet.

<u>Kursmöte 10 (10 feb) (*PI*)</u>: *Kommunikation mellan processer, pipes.*

Innan mötet: Gör kommunikationsövningen.

Efter mötet: Rita fildeskriptortabeller för de ingående processerna i kommunikationsövningen. Stora kommunikationsövningen kommer att vara mycket nyttig för att ni ska kunna lösa laboration 2, vi kommer även att ställa frågor på innehållet i stora kommunikationsövningen på tentamen så denna övning är troligen en av de bästa saker du kan göra i denna kurs. Och: om du inte knäcker den, fråga på kursmöten, det är säkert väldigt många som också har samma fråga som du.

Kursmöte 11 Buffert (13 feb): Detta är en buffert så ta med många frågor. Viss tid kan ägnas åt gamla tentor.

Kursmöte 12 (15 feb) (*PI*): *Kommunikation mellan processer och demoner via UNIX-domain sockets, client-serverbegreppet.*

Det här kursmötet kommer att bli lite annorlunda jämfört med de andra kursmötena. Nu har vi väldigt mycket material att arbeta med, det betyder att det inte är så meningsfullt att presentera mer innan ni hunnit arbeta med det som hittills presenterats. Efter detta kursmöte kan vi dessutom formulera laboration 2 och kursens laborationer är förstås bland det viktigaste som finns. Därför kommer detta möte att vara fokuserat på att behandla det ni tidigare stött på men också förtydliga det här med kommunikation via sockets samt client-serverbegreppet.

Innan mötet: Läs kapitel 5 i ALP samt fortsätte läsa ur OSTEP enligt läsanvisningarna. Läs också `http://www.itp.uzh.ch/~dpotter/howto/daemonize`. Fundera över vad följande kod betyder:

```
main()
{
  pid_t pid;
  pid = fork();
  if(pid)exit(0);
  sleep(10);
}
```

Studera processrelationerna med `ps -l`. Det skapas en barnprocess, men vart tar föräldraprocessen vägen? När föräldraprocessen är borta, vem är då förälder till barnprocessen? (Allt detta vet man genom att använda kommandot `ps -l`.) **Studera nu dokumentet som innehåller anteckningarna inför kursmötet, här finns mer mycket bra övningar som hjälper er med laboration 2.**

Efter mötet: Gör så mycket av övningarna från kursmötet som du vill/behöver. Nu kan du också börja med laboration 2. (Om du gjort övningarna tidigare så är det lättare att arbeta med laboration 2, den innebär att vi kombinerar innehållet i de olika övningarna.)