Laboration 1
Processhantering

Henrik Bäck
850611-6253
Mathias Andersson
850424-6292

Operating Systems DAVB01
VT 2005

Handledare: **Nils Dåverhög**
Hans Hedbom
Thijs Jan Holleboom

Inlämnad **2007-02-12**

Innehållsförteckning

1. Antaganden

Det antog att man skulle använda en *for-sats* som skulle köras så många gånger som antalet processer vi start. I denna *for-sats* antog vi att det givna kommandot *wait* skulle ligga.

2. Översikt

Programmet *CPU* skulle göra en oändlig loop som tog så mycket tid i processorn som möjligt. Programmet *mamma* skulle kalla på programmet *CPU* i det antalet instanser som valdes vid programstart. En observation av vad som händer i 'kön' av processer iakttogs under körning av programmen. Samt vad som händer vid modifiering av prioritet av processerna.

3. Detaljerad beskrivning

Programmen 'CPU' och 'mamma'

Programmet *CPU* skrevs enkelt i språket C och innehåller bara en while-sats som aldrig tar slut. Detta resulterar i att programmet kommer att 'skena' och ta så mycket systemresurser som möjligt.

```
int main()
{
      while(1){}

      return 0;
}
```

Programmet *mamma* skrevs även det i språket C och innehåller en fråga som tar emot ett värde från användaren om hur många olika processer som denne vill starta av *CPU*. Efter att detta värde matas in startar *mamma* upp, som barn till sig själv, så många processer av *CPU* som man angett. Detta program finns i två versioner, hädanefter kallat *mamma (1)* och *mamma (2)*.

Mamma (1) startar ett av användaren valt antal processer och låter dem köras. Därefter vilar programmet i 10 sekunder och avslutas därpå. *Mamma (2)* startar även här ett av användaren valt antal processer. Därpå väntar programmet på att dessa skall köras klart (avslutas/'dödas') och när alla dessa processer avslutats kommer även detta program att avslutas.

Nedan följer programmen *mamma (1)* och *mamma (2)*.

mamma (1)

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

```
#define PRG_CPU "./cpu.out"

int main()
{
	int antProc = 0;
	int i = 0;
	pid_t procID;

	printf("Ange antal processer: ");
	scanf("%d",&antProc);

	for(i=0;i<antProc;i=i+1)
	{
		if((procID = fork()) == 0)
		{
			printf("runNr: %d \n",&i);

			execlp(PRG_CPU,PRG_CPU,NULL);
		}
	}
	sleep(10)

	return 0;
}
```

mamma (2)

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

#define PRG_CPU "./cpu.out"

int main()
{
	int antProc = 0;
	int i = 0;
	pid_t procID;

	printf("Ange antal processer: ");
	scanf("%d",&antProc);

	for(i=0;i<antProc;i=i+1)
	{
		if((procID = fork()) == 0)
		{
			printf("runNr: %d \n",&i);

			execlp(PRG_CPU,PRG_CPU,NULL);
		}
	}
	for(i=0;i<antProc;i=i+1)
		wait();

	return 0;
}
```

Körning av programmen 'CPU' och 'mamma'

Vid körning av *mamma (1)* observerades att alla barn (instanser av *CPU*) som kördes fick samma prioritet och därmed samma 'del' av tiden i processorn. Processerna delade på den kapacitet som fanns tillgänglig. Genom att ge en av processerna en lägre prioritet (högre prioritetsvärde) iakttogs att denna process fick mindre tid i processorn. Det var inte möjligt att ändra så att en process fick högre prioritet.
Moderprocessen (*mamma (1)*) hade högre prioritet än sina 'barn' och efter att *mamma (1)* har avslutats blev hennes barn (alla instanser av *CPU*) zombies (föräldralösa). Dessa dog alltså inte med sin moderprocess *mamma (1)*.

Genom att modifiera programmet *mammma (1)* fås programmet *mamma (2)* vilkens enda skillnad är att denna väntar på att sina barnprocesser avslutas innan hon själv avslutas. Genom att nu starta detta program kan vi lätt observera att processen *mamma (2)* avslutas efter att alla 'barn' har avslutas. Genom att använda funktionen *kill* i Linux är det möjligt att stänga ner processer som ligger och kör. När alla processer tillhörande *mamma (2)* hade avslutas avslutades även *mamma (2)*. Detta beror på den modifiering som gjordes mellan *mamma (1)* och *mamma (2)*. Genom att reducera kommandot *sleep* och ersätta det med en *for-stas,* som körs lika många gånger som antalet processer vi startat, kan vi avgöra med hjälp av *wait-kommandot* om det är dags att avsluta. Kommandot *wait* väntar på att någon av moderprocessens barn skall avslutas, om detta sker går programmet vidare. Genom att upprepa detta det antalet processer som startas kan man vara säker på att alla startade processer avslutas.

4. Problem

När kommandot *fork()* skulle användas uppstod problem som inte kunde lösas. Genom att läsa i *man 3 fork* kunde inte den information vi ville ha erhållas. Enligt manualen för *fork* skulle funktionen returnera *pid* samt 0 om *fork()* lyckades. Hur detta skulle gå till förstods aldrig. Texten i *man 3 fork* var tvetydig och kunde inte tolkas. Problemet kvarstår. Dock är detta inte nödvändigt för uppgiften,

5. Sammanfattning

Uppgiften löstes utan större problem eftersom problemet beskrivet under punkt 4 kunde kringgås. För övrigt gav *man*-kommandot all den hjälp man skulle tänkas behöva. Då kurslitteratur saknades inför laborationen var alla uppgifter tvungna att lösas utan att läsa rekommenderade kapitel. Dock lyckades uppgiften att lösas trots detta.

6. Referenslista

Manual (*man*) för kommandon *wait, fork, exec* samt *sleep* användes. Denna manual fanns i RedHat Linux 9.