Blekinge Tekniska Högskola

DV1492, DV1556 – (Realtids- och) Operativsystem

Laboration: Processer i Linux

Författare: Carina Nilsson

Datum: september, 2017

Innehåll

1 Inledning

Syftet med en här laborationsuppgiften är att bekanta sig med med processer i ett operativsystem. Vi ska titta på exempel på processer som skapar andra processer, föräldralösa processer, terminerade processer som ligger kvar (zombie-processer) och på fördelning av exekveringsresurser (CPU-tid). Vid laborerande ställs alla kommandon från ett terminalfönster. En terminal i Ubuntu kan öppnas med snabbkommandot Ctrl+Alt+ t . Sitter man i en Debian miljö, så kan ett terminalfönster öppnas med **Application Menu→Terminal emulator**.

I laborationen ska ett användbart verktyg för processövervakning användas, kommandot:

```
top
```

Vid laborerande bör `man` -kommandot användas flitigt. Manualsidorna som visas vid detta kommando är uppdelade i numrerade sektioner: 1 för kommandon, 2 för lågnivåfunktioner, 3 för ANSI-funktioner etc. Om man vill komma till en specifik sektion skriver man:

```
man 1 printf   # visar manualsidan for kommandot printf
man 3 printf  # visar manualsidan for funktionen printf
```

I laborationsuppgiften behöver du använda kommandot `top` . För att se manualsidan för kommandot skriver man `man top` , eller `man 1 top` om man vill vara tydlig.

För att skapa processer och exekvera program ska `fork` (sektion 2) och `exec` (sektion 3) användas. `exec` finns i flera varianter. Använd t.ex. `execlp` . Om du har en exekverbar fil i samma mapp som du arbetar i som heter `Program1` kan exekveringen av den startas i aktuell process med följande funktionsanrop i din kod:

```
execlp("./Program1", "./Program1", NULL);
```

Observera att nätverksenheten ”studentfiles” i Linux-miljön inte kan läsas från ett Windowssystem. Om du vill kunna komma åt dina filer från Windows bör du spara dem på en USB-sticka, It’s Learning eller liknande. Observera också att Desktop på dessa datorer är lokal på varje maskin till skillnad från Windowsinstallationerna på skolan där skrivbordet tillhör er nätverksprofil.

2 Redovisning

Laborationsuppgiften görs lämpligen i grupper om två personer. Annan gruppstorlek ska beviljas av labbhandledaren. Grupper med fler än tre deltagare godtas inte.

Redovisningen sker med en skriftlig rapport per grupp (se dokumenten under ”Introduktion till rapportskrivning” och ”Checklista - rapport”) tillsammans med den programkod som skrivits. Rapportens text ska besvara alla frågor som ställs i labbhandledningen och presentera alla resultat ni kommit fram till under laborationen. Redovisningen ska laddas upp som en arkivfil (.zip, .tar eller dylikt). Rapporten ska vara i PDF-format,

Laborationsuppgiften går bra att köra på vilket Linux/Unix-baserat system som helst. *Det ska framgå i rapporten vilken miljö som använts vid experimenten!* Med miljö menas vilken version av operativsystem som använts och hur många CPU-kärnor datorsystemet har, eftersom dessa faktorer påverkar resultaten i laborationen.

3 Förberedande uppgift

Läs kapitel 2.1 och 2.4 i boken, som handlar om processer och schemaläggning.

4 Laborationsuppgift

4.1 Första processen

När en maskin startar upp med Unix/Linux skapas en process som har `PID=1` och den lever så länge maskinen är uppe. Från den här processen skapas alla andra processer med `fork()`. Vad heter denna process?

Tips: Använd kommandot `ps` eller kommandot `top`

4.2 Systemanropet `fork()`

Vad gör systemanropet `fork()`?
Varför ger `fork()` 0 som returvärde till barnprocessen och barnprocessens processidentitet som returvärde till föräldraprocessen istället för tvärtom?

4.3 Litet girigt program

Skriv ett program som går i en oändlig loop, och därmed använder all CPU-tid som det kan få.

4.4 Skapa flera processer

Skriv ett ”moderprogram” som skapar ett antal processer (med `fork()`) och ser till samtliga barnprocesser kör programmet du skrev i förra deluppgiften (med hjälp av `execlp()`). Antalet processer ska vara valbart direkt i programmet genom en fråga till användaren alternativt som argument till programmet.

4.5 Utvärdera schemaläggningsprinciper

Kör ”moderprogrammet” och låt det skapa ett antal processer. Det är inte meningsfullt att ha så få processer att varje process kan ha en egen CPU-kärna om man ska kunna se hur systemet delar ut exekveringsresurser.

- Prova både med att låta barnprocesserna ha en terminalutskrift i sin loop och att inte ha det. Vad blir det för skillnad? Varför är det så?
- Notera med hjälp av `top` hur mycket CPU-tid var och en av processerna får (utan utskrifter i processerna). Är det en rättvis fördelning? Notera även processernas prioritet.
- Har alla barnprocesserna samma prioritet?
- Har barnprocesserna samma prioritet som moderprocessen? (Det är bra att ha en oändlig loop i moderprogrammet också efter att processerna skapats så att den inte terminerar eller sover om den ska synas bra i `top` .)

4.6 Ändra prioritet

Ställ om prioriteten hos någon process med terminalkommandot `renice` och försök komma på hur prioriteten styr CPU-tilldelningen.

4.7 Terminerad föräldraprocess

Vad händer om moderprocessen dör (terminerar)?

4.8 Terminerade barnprocesser

Skriv om programmet för processerna så att det inte lägre är en oändlig loop. Låt processerna sova i 20 sekunder och sedan terminera. Låt istället föräldraprogrammet avslutas med en oändlig loop så att det inte terminerar. Vad händer nu när programmet körs? Försvinner barnprocesserna ur systemet?

4.9 Systemanropet `wait()`

Om föräldraprocessen avslutas med `wait()` istället för en oändlig loop, vad händer då?

4.10 Processers vetskap om varandra

Vad händer om en process plötsligt dör? Kan andra processer se det och vilka i så fall? På vilket sätt kan de i så fall få den informationen?