KTH ICT

Framsida för labrapport

Operativsystem, ID2206

Period 2, läsår 2014

Fyll i alla uppgifter!

Labnr	Labnamn
1	DIGENV

Efternamn, förnamn	Personnummer	Tydlig datorpostadress
Björk, Kim	910702-4029	KimBjork@Kth.se
Olsson, Johannes	890728-0211	Johanneo@kth.se

Inlämningsdatum: 2014-12-08

Kommentarer

För internt bruk

Godkänd	Komplettera	Meddelad	Datum	Signatur

Registrerad	Ny	Gammal	G/B/U

KTH
Johannes Olsson johanneo@kth.se
Kim Björk kimbjork@kth.se
2014-12-07

Problembeskrivning

Program som använder sig av andra program skapar processer som används för att köra programmen. För att processer ska kunna kommunicera med varandra används pipelines. Problemet i denna laboration är att konstruera ett program som använder programmen *printenv, grep, sort* och en s.k. pager för att på ett smidigt sätt kunna filtrera och lista miljövariabler. Programmets funktion är som ett slags macro i terminalen. Programmet ska fungera genom att skapa 3 eller 4 barnprocesser som kommunicerar med föräldraprocessen via pipes.

Programbeskrivning

Programmet tar in en lista av argument. Det första argumentet är programmet som anropats, d.v.s. vårt program. De resterande är parametrarna som skickades med vid anropet. Inledningsvis byter vi ut det första argumentet i listan mot "grep", för att senare kunna anropa grep med samtliga parametrar som gjordes i anropet till vårt program. Sedan vill vi kontrollera om det finns en förvald pager som användaren vill använda för att visa utskriften. Om variabeln PAGER existerar används den pagern, i annat fall används less. Vidare lägger vi in alla kommandon (printenv, sort och grep) i en vektor. Vi valde att använda en vektor då den är väldigt enkel att traversera när den är NULL-terminerad. Varje element i listan, utom less/more, genererar således en iteration där vi genomför allt som genererar den output vi vill ha. En iterationen börjar med att kolla om kommandot vi vill genomföra är grep och om användaren har specificerat några speciella variabler vi ska söka efter, om inte fortsätter vi till nästa iteration. För varje loop skapar vi en ny barnprocess och en ny pipe. Varje barnprocess kopierar vår file descriptors läs och skriv ände till standard in- och output, så att det som annars skulle skrivas ut nu pipas till förändrarprocessen. För att inte lämna några öppna ändar till vår pipe stänger vi både läs- och skrivänden. Därefter exekverar vi det program vi just nu itererar över. När programmet terminerar avslutas barnprocessen.
Medan allt detta händer för barnprocessen väntar föräldern på att barnet ska avsluta. Då kopierar vi förälderns läs-ände till pipen så att vi kan använda den i nästa iteration. Vi stänger även föräldraprocessens läs-ände och sätter cmd-pekaren att peka på nästa element. När while-loopen är klar är det bara för oss att först kopiera vår file descriptors läs-ände till standard input, stänga kanalens läs-ände och skriva ut resultatet. Till utskriften använder vi den pager vi tidigare specificerat, och om den inte fungerar använder vi more.

Vi skulle enkelt kunna ändra beteendet av vårt program då vi endast behöver lägga till element i vår vektor med program som ska anropas. Den itererande strukturen är generisk.

Sökväg till program och källkod

Filerna ligger i */home/j/o/johanneo/os/pipes/sub*. De heter *digenv* samt *digenv.c.*

KTH
Johannes Olsson johanneo@kth.se
Kim Björk kimbjork@kth.se
2014-12-07

Kompileringskommando

gcc -O3 -Wall -o digenv digenv.c

Testutskrifter

./digenv

```
HOME=/Users/johannes
JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.7.0_51.jdk/Contents/Home
LC_CTYPE=UTF-8
LOGNAME=johannes
OLDPWD=/Users/johannes/kth/os/pipes
PATH=/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin:/Applications/apache-maven-3.2.1/bin:/Applic
ations/Sublime
Text.app/Contents/SharedSupport/bin:/opt/X11/bin:/usr/local/git/bin:/Library/Java/JavaVirtual
Machines/jdk1.7.0_51.jdk/Contents/Home
PWD=/Users/johannes/kth/os/pipes/src
SECURITYSESSIONID=186a5
SHELL=/bin/bash
SHLVL=1
SSH_AUTH_SOCK=/private/tmp/com.apple.launchd.qMhoZCnLtO/Listeners
TERM=xterm-256color
TERM_PROGRAM=Apple_Terminal
TERM_PROGRAM_VERSION=343
TERM_SESSION_ID=909C47F6-38B1-45C4-85A5-A5F62D95A7EB
TMPDIR=/var/folders/s2/5qk7t0pd2c7bpwlb_yjt0q0c0000gn/T/
USER=johannes
XPC_FLAGS=0x0
XPC_SERVICE_NAME=0
_=./digenv
__CF_USER_TEXT_ENCODING=0x1F5:0x0:0x0
(END)
```

./digenv TERM

```
TERM=xterm-256color
TERM_PROGRAM=Apple_Terminal
TERM_PROGRAM_VERSION=343
TERM_SESSION_ID=909C47F6-38B1-45C4-85A5-A5F62D95A7EB
(END)
```

./digenv -n "HOME\\|SSH"

```
10:SSH_AUTH_SOCK=/private/tmp/com.apple.launchd.qMhoZCnLtO/Listeners
14:JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.7.0_51.jdk/Contents/Home
18:HOME=/Users/johannes
(END)
```

KTH
Johannes Olsson johanneo@kth.se
Kim Björk kimbjork@kth.se
2014-12-07

Utvärdering av laborationen

Det var vid många tillfällen man var var tvungen att läsa i manualer, etc, för att förstå hur man skulle gå tillväga. Detta gjorde att labben blev relativt tidskrävande. Det var också till stor nackdel att vi hade väldigt begränsade erfarenheter av programmering i C. Vi uppskattar av vi var tvungna att lägga ner ca 12 timmar på att skriva koden.
Ett speciellt ställe som vi fastnade på var när vi skulle börja implementera less/more. Vi kunde lista alla variabler samt söka efter specifika men fick problem när vi ville använda en pager till utskriften. Detta löste vi genom att skapa fler pipes där vi tidigare endast använt en och samma.
Vi fastnade också då vår kod varje iteration i while-loopen skrev över vår läs-ände, det fixade vi genom att skapa vår readbackup variabel.
I denna labb har vi fått mer kunskaper utav strukturerad programmering i språket C och även lärt oss hur man på ett korrekt sätt kommenterar denna. Vidare har vi också lärt oss hur man skapar och hanterar processer samt hur man kan kommunicera mellan dessa med hjälp av pipelines.
Det tillhörade labbPM:et tyckte vi var lite rörigt. Det skulle vara bättre att ha PM:et mer strukturerat och överskrådligt så att det blir enklare att bena ut hur all information hänger ihop. Då vi måste betygsätta labbPM:et ger vi det 3 av 5.

Förberedelsefrågor

1. När en maskin bootar med UNIX 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?

```
n134-p99:~ johannes$ ps -p 1
  PID TTY          TIME CMD
    1 ??         3:30.75 /sbin/launchd
```

2. Kan environmentvariabler användas för att kommunicera mellan föräldra- och barnprocess? Åt bägge hållen?

En barnprocess ärver den instans av förälderns miljövariabler som råder vid skapandetillfället.

4. Varför returnerar fork 0 till child-processen och child-PID till parent-processen, i stället för tvärtom?

Därför att hantering av processer bör ske genom att en förälder hanterar sina barnprocesser, snarare än en bottom-up approach där alla barn hanterar sin gemensamma förälder.

KTH
Johannes Olsson johanneo@kth.se
Kim Björk kimbjork@kth.se
2014-12-07

5. UNIX håller flera nivåer av tabeller för öppna filer, både en användarspecifik “File Descriptor Table” och en global “File Table”. Behövs egentligen File Table? Kan man ha offset i File Des- criptor Table istället?
File descriptor table är ett en mellanlagring som innehåller en entry för varje unik fil som är öppen hos någon process. Denna tabell är enklare att söka igenom än mängden av alla file descriptor tables, där samma pekare till inode skulle förekomma flera gånger.

6. Kan man strunta i att stänga en pipe om man inte använder den? Hur skulle programbeteendet påverkas? Testa själv. Läs mer i pipe(2).

Programbetéendet måste inte nödvändigtvis påverkas om en pipe lämnas öppen. Om inte alla skrivändar av pipen stängs skickas ingen EOF, vilket gör det omöjligt för mottagarprocessen att garantera att inte mer data kommer skickas över pipen, och kommer således inte sluta att komma ur läsläget.

7. Vad händer om en av processerna plötsligt dör? Kan den andra processen upptäcka detta?
Om en process försöker skriva i en pipe som inte har några lyssnare skickas en SIGPIPE-signal till sändaren.

8. Hur kan du i ditt program ta reda på om grep misslyckades? Dvs om grep inte hittade någon förekomst av det den skulle söka efter eller om du gett felaktiga parametrar till grep?

Grep returnerar 1 och inga rader hittades. Om någon kod efter anropet till grep körs har något gått fel.

KTH
Johannes Olsson johanneo@kth.se
Kim Björk kimbjork@kth.se
2014-12-07

```
 7//macros for readability
 8#define READ fd[0]                      //macro for file descriptor's read end
 9#define WRITE fd[1]                     //macro for file descriptor's write end
10
11/*
12If this program is called with no arguments, it
13will execute sort on the output from printenv and forward to a pager.
14If the PAGER environment variable is set, the program will use that.
15Otherwise less. If less fails, the program will use more.
16If this program is called with one or more arguments,
17they will all be passed to grep which is executed before
18sort in the above sequence when program is called with no arguments.
19*/
20
21//argc: the number of arguments in the program call
22//argv: a string array with parameters
23int main(int argc, char **argv)
24{
25        argv[0] = (char *) "grep";      //changing the first argument which is something like ./digenv to grep in preparation for call to grep
26        const char **grep = (const char**) argv;        //string array for call to grep
27
28        const char *pager[] = {NULL!=getenv("PAGER") ? getenv("PAGER") : "less", NULL}; //string array for call to the pager (user defined or less)
29
30        const char *printenv[] = {"printenv", NULL}; //string array for call to printenv
31        const char *sort[] = {"sort", NULL};    //string array for call to sort
32
33        const char **commands[] = {printenv, grep, sort, NULL}; //list of programs to run before print-out to pager
34        const char ***cmd = commands;   //pointer used to traverse the list of programs
35
36        int fd[2], readbackup;  //two integers for the filedescriptors read and write ends respectively, one to pass read end between pipes
37
38        /* Loop to iterate over the programs to call. The list is null-terminated */
39        while(NULL != cmd[0])
40        {
41                if(1==argc && !strcmp("grep", cmd[0][0]))       //if program for this iteration is grep and no arguments have been passed to this program, grep is not run
42                //if(1==argc && "grep" == cmd[0][0])
43                {
44                        cmd++;
45                        continue;
46                }
47
48                pipe(fd);       //creating new pipe with our file descriptors
49                switch(fork())  //forking this process
50                {
51                        case 0:         //only the child process runs this case
52                                dup2(readbackup, 0);    //copying our saved file descriptor's read end to standard input
53                                dup2(WRITE, 1); //copying our file descriptor's write end to standard output
54                                close(WRITE);   //closing our channel to the file descriptor's write end
55                                close(READ);    //closing our channel to the file descriptor's read end
56                                execvp(cmd[0][0],(char *const*) cmd[0]); //executing a command from the command's list
57                        default:        //only the parent process runs this case
58                                wait(NULL);     //waiting for the child process to terminate
59                                readbackup = READ; //backing up the read end of our file descriptor
60                                close(WRITE); //closing our channel to the file descriptor's write end
61                                cmd++; //incrementing the pointer to the command's list
62                }
63        }
64        dup2(readbackup, 0);    //copying our saved file descriptor's read end to standard input
65        close(READ);    //closing our channel to the file descriptor's read end
66        execvp(pager[0], (char *const*) pager); //executing call to the pager
67        pager[0]="more"; //if user defined pager was not defined or less did not work, change pager to more
68        execvp(pager[0], (char *const*) pager); //retry executing call to the pager
69}
```