

# UNIX

Denna laboration gjordes i syfte i att lära sig att kompilera och köra processer i en UNIX-miljö genom Linux-distributionen Ubuntu. Kompileringar och körningar av processer sköttes via operativsystemets terminal för kommandon.

Kurs: Datorkommunikation och nät (DT2017-0200/DT2022-0222)

*Härmed försäkrar jag/vi att jag/vi utan att ha erhållit eller lämnat någon hjälp utfört detta arbete.*

Datum: 2015-10-03

Underskrift:

Özgun Mirtchev

Namn: Özgun Mirtchev  
Personnr: 920321-2379  
E-post: ozziee@gmail.com  
Program: Dataingenjörsprogrammet

Einar Larsson

Namn: Einar Larsson  
Personnr: 930223-5677  
E-post: einar\_larsson@hotmail.com  
Program: Dataingenjörsprogrammet

Lärarens anteckningar
-----------------------

## Innehållsförteckning

Bakgrund .....	2
Resultat.....	3
Uppgift 2a.....	3
Uppgift 2b .....	4
Uppgift 3 .....	5
Uppgift 4 .....	6

## Bakgrund

Många av dem kommunikationsprotokollen som finns idag kommer från UNIX. Datorkommunikation är därför väldigt sammankopplad med UNIX och det finns många sätt att kunna utföra datakommunikationshandlingar genom det operativsystemet.

I denna laboration kompileras och exekveras program i Terminal från filen `unixfork.c` som utför två processer parallellt med varandra, en moder-process och en dotter-process. Moder-processen läser in data och skriver ut data medan dotter-processen skriver ut tid i kanten av samma process-fönster. I senare uppgifter utförs samma program fast med hjälp av pipes som överför data mellan de olika processerna.

Koderna för processerna/programmen hittas i Bilaga-sektionen, längst ned i detta dokument.

## Resultat

### Uppgift 2a

För att köra tiden i sitt eget program, klipptes kod-blocket ut och sattes in i en ny fil, show\_time.c. I Terminal kördes kommandot "cc show\_time.c -o stc" för kompilering av c-filen och sedan "./stc" för körning av exekveringsfilen från kompileringen.

```
ozziee@ozziee-X550LB:~/unix$ cc show_time.c -o stc 15:56:12
ozziee@ozziee-X550LB:~/unix$ ./stc
```

Som syns ovan befinner sig tid-beräkningen i det övre högra hörnet av fönstret.

I samma tillfälle öppnades ett annat Terminal-fönster och kommandot "ps a" kördes för att lista upp alla processer som var igång. I den listan syntes processen "./stc" som har PID = 2505.

```
ozziee@ozziee-X550LB: ~/unix
ozziee@ozziee-X550LB:~/unix$ ps a
  PID TTY          STAT       TIME COMMAND
   917 tty4      Ss+        0:00 /sbin/getty -8 38400 tty4
  2425 pts/13    Ss         0:00 bash
  2505 pts/13    S+         0:00 ./stc
  2618 pts/0      Ss         0:00 bash
  2698 pts/0      R+         0:00 ps a
ozziee@ozziee-X550LB:~/unix$
```

I samma Terminal-fönster kördes kommandot "kill 2505", för att döda processen med PID 2505 och som det syns i det övre fönstret i nedanstående bild stängdes processen ned i den första terminalen.

```
ozziee@ozziee-X550LB: ~/unix
ozziee@ozziee-X550LB:~/unix$ cc show_time.c -o stc 15:57:34
ozziee@ozziee-X550LB:~/unix$ ./stc
Terminated
ozziee@ozziee-X550LB:~/unix$

ozziee@ozziee-X550LB: ~/unix
ozziee@ozziee-X550LB:~/unix$ ps a
  PID TTY          STAT       TIME COMMAND
   917 tty4      Ss+        0:00 /sbin/getty -8 38400 tty4
  2425 pts/13    Ss         0:00 bash
  2505 pts/13    S+         0:00 ./stc
  2618 pts/0      Ss         0:00 bash
  2698 pts/0      R+         0:00 ps a
ozziee@ozziee-X550LB:~/unix$ kill 2505
ozziee@ozziee-X550LB:~/unix$
```

Koden för show\_time.c finns i Bilaga 1.

## Uppgift 2b

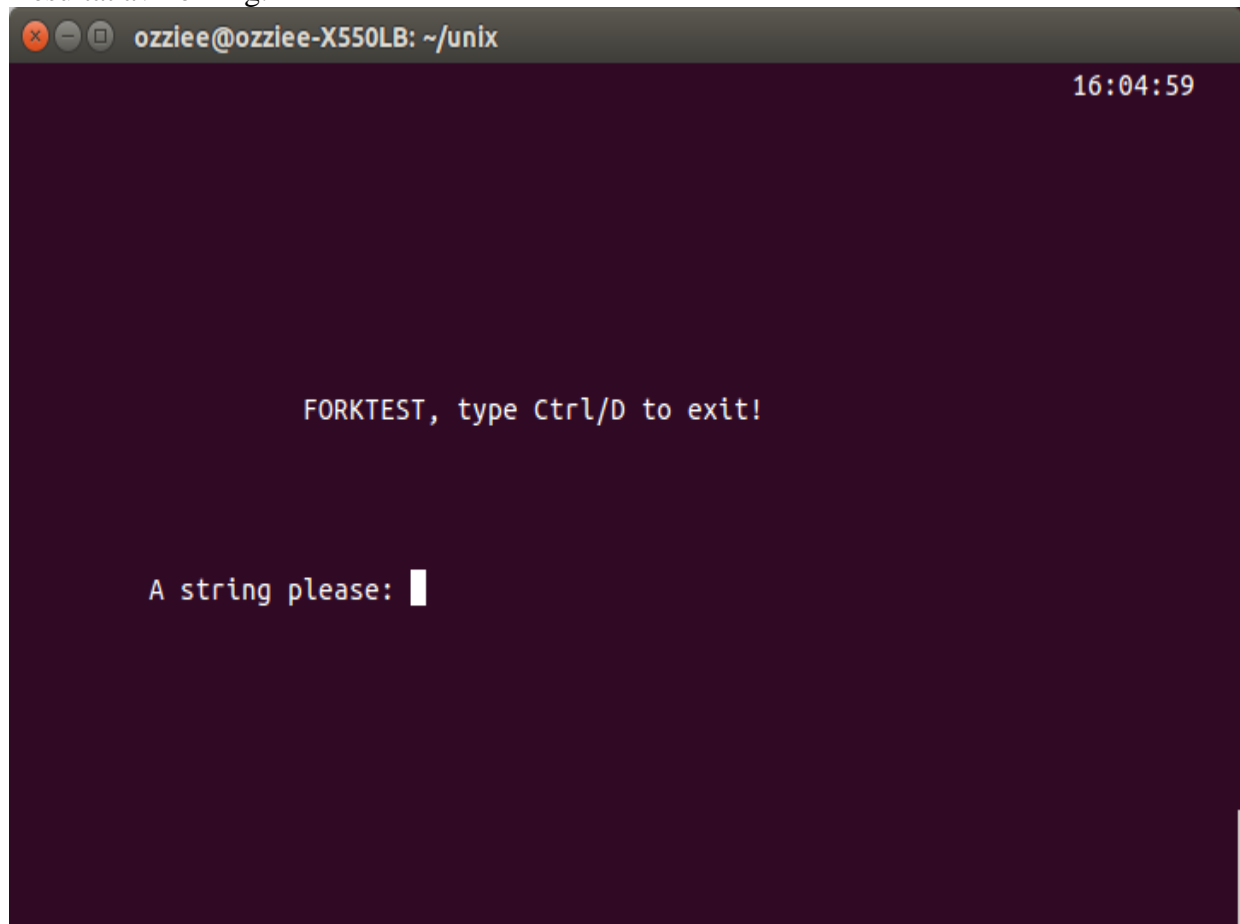
För att köra `show_time.c` på ett annat sätt kan man i `unixfork.c` använda sig utav systemanropet `exec`, istället för att anropa själva `show_time`-funktionen i samma fil. Istället för `show_time()`;, användes i detta fall kod-radens:

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

Denna funktions-anrop gör samma sak som `show_time()`. Dock så behöver man ange adressen till exekveringsfilen från kompileringen av den specifika filen. I detta fall `show_time.c`. Exekveringsfilen befann i samma katalog som `unixfork.c` och därför behövdes inte hela adressen anges. Det är dock viktigt att man anger samma namn i parametern för `execlp` som exekveringsfilens namn. Nedan kan man se att `cc show_time.c` kompilerades som `show_time`.

```
ozziee@ozziee-X550LB:~/unix$ cc unixfork.c -o execuf
ozziee@ozziee-X550LB:~/unix$ cc show_time.c -o show_time
ozziee@ozziee-X550LB:~/unix$ ./execuf
```

Resultat av körning:



```
ozziee@ozziee-X550LB: ~/unix 16:04:59

FORKTEST, type Ctrl/D to exit!

A string please: 
```

Hela koden för `unixfork.c` befinner i bilaga 2.

### Uppgift 3


I UNIX-processer kan man använda sig utav pipes (rör) mellan varandra för att överföra data som kan användas av båda processerna. I detta fall skulle moderprocessen läsa in en sträng från användaren och en dotter-process skulle skriva ut den strängen. Data överförs till dotter-processen från moder-processen med hjälp av just dessa rör.

I bilagan för koden förklaras vilken som är dotter-processen och vilken som är moder-processen. I uppgiften användes `pipe()`-funktionen för att öppna röret mellan processerna och `read()`- och `write()`-funktionerna som läser från röret respektive skriver till röret. Strängen som läses in från användaren kommer att skrivas in i röret av moder-processen och därför används `write()`, medan strängen som skrivs ut av dotter-processen kommer att använda sig utav `read()`-funktionen. När moder-processen har läst in en sträng i en evighets-loop körs redan dotter-processen i bakgrunden, också i en evighets-loop, och läser av datan från röret.

Kompilering och körning av kod:

```
ozziee@ozziee-X550LB:~/unix$ cc rw_pipe1.c -o rwp1
ozziee@ozziee-X550LB:~/unix$ ./rwp1
```

Producerat resultat:

A screenshot of a terminal window with a dark background. The window title is 'ozziee@ozziee-X550LB: ~/unix'. The terminal output shows the text 'FORKTEST, type Ctrl/D to exit!' followed by a blank line. Then, it displays 'The string was: hej' and 'New string please: hej' with a cursor at the end of the second line.

```
ozziee@ozziee-X550LB: ~/unix
FORKTEST, type Ctrl/D to exit!

The string was: hej
New string please: hej
```

Kod-filen, `rw_pipe1.c`, som skrevs för denna uppgift kan hittas i Bilaga 3.

## Uppgift 4

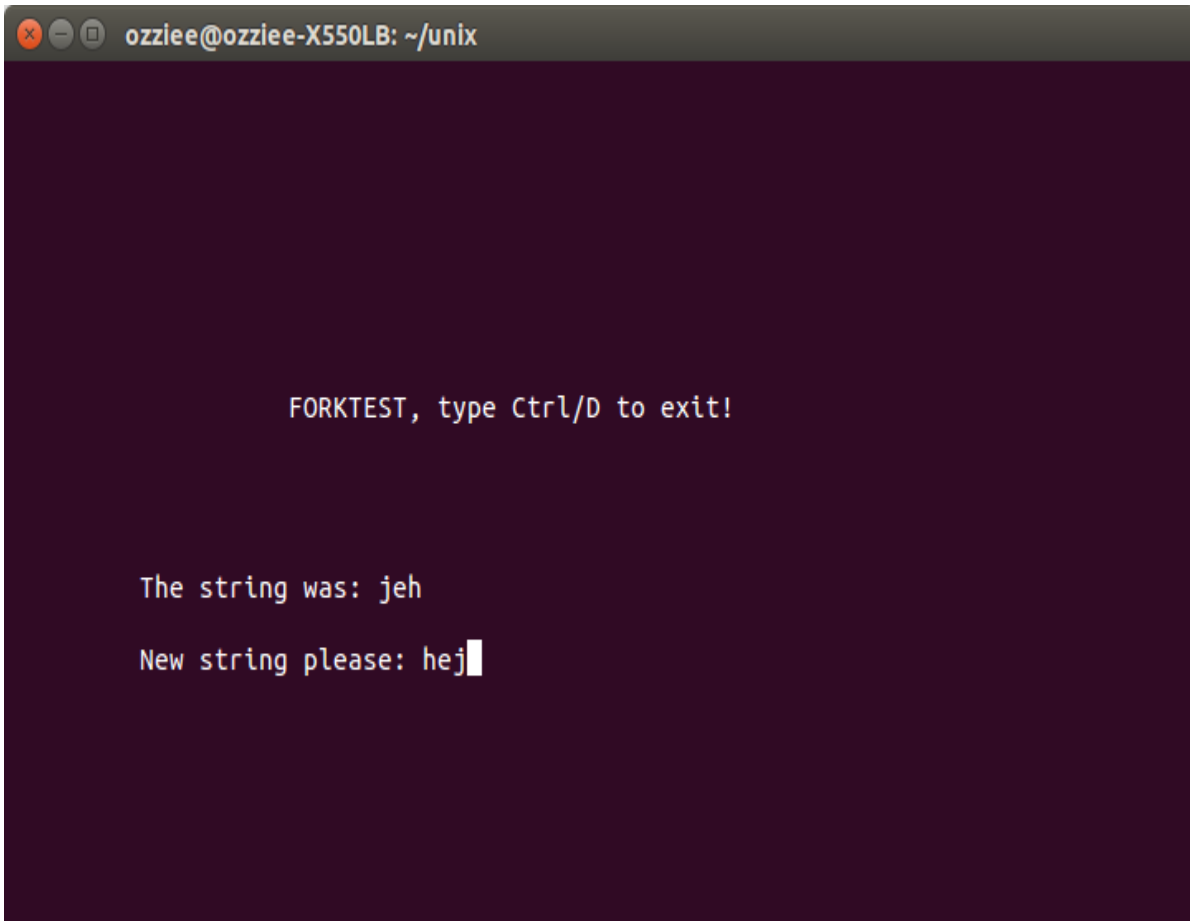
För denna uppgift skulle en till dotter-process införas från moder-processen från föregående uppgift. Denna dotter-process skulle omvända strängen som lästes in och sedan skicka vidare strängen till den första dotter-processen som skriver ut strängen på skärmen.

För att lyckas med detta, behövdes ett ytterligare rör, som datan lästes in i av moder-processen, som sedan den andra dotter-processen, som omvänder datan, skulle läsa och skriva in i ett annat rör. Det andra röret, som den första dotter-processen använde sig utav, läste den omvända datan och skrev ut det på skärmen.

Kompilering och körning av kod:

```
ozziee@ozziee-X550LB:~/unix$ cc rw_pipe2.c -o rwp2
ozziee@ozziee-X550LB:~/unix$ ./rwp2
```

Producerat resultat:



```
ozziee@ozziee-X550LB: ~/unix

FORKTEST, type Ctrl/D to exit!

The string was: jeh
New string please: hej
```

För att vända på strängen, användes funktionen `char *strrev(char *str)`, som kan hittas i Bilaga 5 i mer detalj.

**Hela koden, `rw_pipe2.c`, för denna uppgift befinns i Bilaga 4.**

## Bilagor

### Bilaga 1 – Kod för Uppgift 2a

```
/* show_time.c */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <time.h>
#include "vt200.h"

void show_time()          /* This function shows actual time in   */
{                          /* upper right corner on screen.   */
    long sec;
    struct tm *now;

    sleep(2);
    for(;;)               /* Repeat forever! */
    {
        time(&sec);          /* Time in seconds since January 1, 1970 */
        now = localtime(&sec); /* Time in hours, minutes, seconds, ... */
        CURSOR_OFF;
        SAVE_CURSOR;
        POS(1,70);
        printf("%02d:%02d:%02d", now->tm_hour, now->tm_min, now->tm_sec);
        RESTORE_CURSOR;
        CURSOR_ON;
        fflush(stdout);
        sleep(1);
    }
}

int main()
{
    show_time();
}
```

---



## Bilaga 2 – Kod för Uppgift 2b

```
/* unixfork.c */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <time.h>
#include "vt200.h"
#include <unistd.h>

void show_time(){} /* Se bilaga 1 */

void interaction() /* This function asks for a string an */
{ /* prints it out again, until Ctrl/D is */
    char str[60]; /* pressed (EOF). */

    POS(15,10); printf("A string please: "); CLRLINE;
    fflush(stdout);
    while(scanf("%s", str) != EOF) {
        POS(15,10); printf("The string was: %s", str); CLRLINE;
        POS(17,10); printf("New string please: "); CLRLINE;
        fflush(stdout);
    }
}

main()
{
    int pid;

    CLRSCR; HOME;
    POS(10,20); printf("FORKTEST, type Ctrl/D to exit!");
    fflush(stdout);

    switch (pid = fork()) /* Try to fork! */
    {
        case -1:
            perror("Forking"); /* Couldn't fork! */
            exit(1);

        case 0: /* This is child process. */
            execlp("./show_time", "show_time", NULL); // show_time();
            break;

        default: /* This is parent process. */
            interaction(); /* Child has process id = pid */
            kill(pid, SIGTERM); /* Kill child process. */
            break;
    }

    CLRSCR; HOME; CURSOR_ON;
    fflush(stdout);
}
```

---

## Bilaga 3 – Kod för Uppgift 3

```
/* rw_pipe1.c (1 child-process) */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <time.h>
#include "vt200.h"
#include <unistd.h>
#include <string.h>

main()
{
    int pid, fd[2];
    char buf[100], str[100];

    CLRSCR; HOME;
    POS(10, 20); printf("FORKTEST, type Ctrl/D to exit!");
    fflush(stdout);

    pipe(fd);

    pid = fork(); /* Try to fork! */
    if (pid == -1)
    {
        perror("Forking");          /* Couldn't fork! */
        exit(1);
    }
    else if (pid == 0)               //1st child process
    {
        while (1)
        {
            read(fd[0], buf, sizeof(buf));
            SAVE_CURSOR;
            POS(15, 10); printf("The string was: %s", buf);
            CLRLINE;
            RESTORE_CURSOR;
            fflush(stdout);
        }
    }

    /* This is parent process. */
    POS(15, 10); printf("A string please: "); CLRLINE;
    fflush(stdout);
    while (scanf("%s", str) != EOF)
    {
        POS(17, 10); printf("New string please: "); CLRLINE;
        fflush(stdout);
        write(fd[1], str, strlen(str) + 1);
    }

    kill(pid, SIGTERM);
    CLRSCR; HOME; CURSOR_ON;
    fflush(stdout);
}
```

---

## Bilaga 4 – Kod för Uppgift 4

```
/* rw_pipe2.c (2 child processes) */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <time.h>
#include "vt200.h"
#include <unistd.h>
#include <string.h>

char *strrev(char *str){} /* Se Bilaga 5 */

main()
{
    int pid, pid2, fd[2], fd1[2], n;
    char buf[100], str[100];

    CLRSCR; HOME;
    POS(10,20); printf("FORKTEST, type Ctrl/D to exit!");
    fflush(stdout);

    pipe(fd1);
    pipe(fd);

    pid = fork(); /* Try to fork! */
    if (pid == -1)
    {
        perror("Forking"); /* Couldn't fork! */
        exit(1);
    }
    else if (pid == 0) //1st child process
    {
        while(1)
        {
            read(fd[0], buf, sizeof(buf));
            SAVE_CURSOR;
            POS(15,10); printf("The string was: %s", buf); CLRLINE;
            RESTORE_CURSOR;
            fflush(stdout);
        }
    }

    pid2 = fork();

    if (pid2 == -1)
    {
        perror("Forking"); /* Couldn't fork! */
        exit(1);
    }
    else if (pid2 == 0) //2nd Child process
    {
        while(1)
        {
            n = read(fd1[0], buf, sizeof(buf));
            strrev(buf);
        }
    }
}
```

---

```
        strcpy(str,buf);
        write(fd[1], str, strlen(str) +1);
    }

}

/* This is parent process. */
    POS(15,10); printf("A string please: "); CLRLINE;
    fflush(stdout);
while(scanf("%s", str) != EOF)
{
    POS(17,10); printf("New string please: "); CLRLINE;
    fflush(stdout);
    write(fd1[1], str, strlen(str) +1);
}

    kill(pid, SIGTERM);
    CLRSCR; HOME; CURSOR_ON;
    fflush(stdout);
}
```

## Bilaga 5 – StringReverse-funktion för Linux

```
char *strrev(char *str)
{
    char *p1, *p2;

    if (!str || !*str)
        return str;
    for (p1 = str, p2 = str + strlen(str) - 1; p2 > p1; ++p1, --p2)
    {
        *p1 ^= *p2;
        *p2 ^= *p1;
        *p1 ^= *p2;
    }

    return str;
}
```