

Mer om UNIX och IPC

Denna laboration gjordes i syfte i att lära sig om hur man kan kompilera filer via makefile och om konnektiviteten mellan en server och en klient via socketar.

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: 2015-10-09

Underskrift:

Özgun Mirtchev

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

Einar Larsson

Namn: *Einar Larsson*
Personnr: 930223-5677
E-post: *enar_larsson@hotmail.com*
Program: Dataingenjörsprogrammet

| |
|-----------------------|
| Lärarens anteckningar |
|-----------------------|

Innehållsförteckning

| | |
|----------------|---|
| Bakgrund | 2 |
| Resultat..... | 3 |
| Uppgift 1..... | 3 |
| Uppgift 2..... | 3 |
| Uppgift 3..... | 4 |
| Uppgift 4..... | 5 |

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 bounce.c (klient) och bounced.c (server), för att testa konnektiviteten mellan två socketar i samma lokala nätverk. Det gick ut på att bounced.c kördes för att aktivera server-porten som lyssnar på klienten. Därefter kördes klienten som kopplades till samma port för att lyssna på inmatning och dotter-processen som kördes från bounced.c tog emot sträng-inmatning i socket och utmatade samma sträng. I senare fall, fanns vissa karaktärer som skulle analyseras, nämligen "P", "p", och '\n'. När dessa karaktärer inmatades skulle istället dotter-processen skriva ut alla utskrivbara tecken på ASCII, som sträcker sig från 32-126. I fallet av karaktärerna "D" och "d" skulle siffror från 0-9 skrivas ut.

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

Resultat

Uppgift 1

Testkörning av `./demobounced.c` och `./demobounce.c` i Terminal påvisade att det tänkta programmen som ska kodas i denna laboration, ska vara en server och en klient. Klienten ska koppla upp sig till servern som körs i bakgrunden som i sin tur öppnar en dotter-process, som tar in en sträng och skriver ut samma sträng i nästa rad. Detta testades med två klienter för att säkerställa att moder-processen i `bounced.c` kan ta emot fler klienter till samma port.

Uppgift 2

I detta fall skulle en egen funktion, `take_call()`, skrivas klart för ändamålet att skapa en socket mellan klienten och servern för att ta emot data (sträng). Kodan som skrevs kan hittas i bilaga 1.

`Take_call()`-ligger i `bounced.c`-filen och kördes här mot port 9003.

```
datorkom@t002-client:~/IPC$ ./bounced 9003 &
[2] 18499
datorkom@t002-client:~/IPC$ Listening on port 9003
Connection 1 accepted, creating child process 1
Child process 1 is now running
```

Sedan kördes klienten mot samma port som servern innan öppnade. Filen `demobounce.c`-filen kördes då en klient-kod inte har programmerats ännu.

```
datorkom@t002-client:~/IPC$ ./demobounce 9003
Connected on port 9003, hopefully an echo server!
Type some text. Exit with Ctrl/D (end of file).
S: 
```

När klienten kopplades till servern, skickades ett meddelande till terminalen som accepterade anslutningen och startade dotter-processen som i sin tur öppnade socketen till klienten och bad om en sträng.

Uppgift 3

En egen version av bounce.c skulle skapas i denna uppgift. I den skulle vissa INET-funktioner ersättas med vanliga sys/socket-anrop. Därför ersattes många av funktionerna i main-funktionen i en halvklar bounce.c.

Som i den förra uppgiften så öppnas servern genom att köra ./bounced. I detta fall mot port 9005. Som det syns i bilden nedan, är den aktiverad och "lyssnar" efter klient-kopplingar.

```
datorkom@t002-client:~/IPC$ ./bounced 9005 &  
[1] 19327  
datorkom@t002-client:~/IPC$ Listening on port 9005
```

Klienten kopplades till servern, och tillbaka kom ett svar i form av en dotter-process som nu körs i klienten.

```
datorkom@t002-client:~/IPC$ make bounce  
cc -c bounce.c  
cc bounce.o -o bounce  
datorkom@t002-client:~/IPC$ ./bounce 9005  
Connected on port 9005, hopefully an echo server!  
Type some text. Exit with Ctrl/D (end of file).  
S: test  
R: test  
S: █
```

I samma tillfälle meddelar servern att en dotter-process har aktiverats.

```
datorkom@t002-client:~/IPC$ ./bounced 9005 &  
[1] 19848  
datorkom@t002-client:~/IPC$ Listening on port 9005  
Connection 1 accepted, creating child process 1  
Child process 1 is now running
```

Koden för bounce.c kan hittas i bilaga 2.

Uppgift 4

I denna uppgift ska en specifik inmatad karaktär, mata ut en specifik sträng. Karaktärerna i fråga är:

| Kommando | Utmatning |
|-------------------------------|--------------|
| \n (= [Return] eller [Enter]) | ASCII-tecken |
| p\n eller P\n | ASCII-tecken |
| d\n eller D\n | Siffror |

För att kunna upptäcka dessa karaktärer, måste funktionen `take_call()` ändras. För att kunna kontrollera den första positionen i arrayen `buf` som tar emot tecken genom socketen måste `buf[0]` användas i villkoret och jämföras mot respektive karaktär. Koden som används står nedan och kan även hittas i bilaga 3. Bounce.c kan hittas i bilaga 4 tillsammans med `clearBuf`.

```
if(buf[0] == 'P' || buf[0] == 'p' || (buf[0] == '\n')) {
    int i;
    for(i = 32; i <= 126; i++)
    {
        letters[i-32] = i;
    }
    letters[127-32] = '\n';
    write(sock, letters, sizeof(letters) - 4);
}
else if(buf[0] == 'D' || buf[0] == 'd')
{ write(sock, siffror, sizeof(siffror)); }
```

Dessutom deklarerades nya arrayer, `siffror[]` och `letters[]`. "0123456789\n" lagrades i arrayen `siffror[]` och ASCII-koderna för `letters[]` genererades automatiskt och sattes in i arrayen när villkoret uppfylldes. En rad med `letters[127-32] = '\n';` infördes för att retur skrivs ut när strängen avslutas så att allting skrivs ut per rad.

Ännu en sak som behövde åtgärdas var att strängen innan som skrevs stannade kvar i nästa rad och därför behövde den flushas ut, och därför användes en egen-skapad funktion `clearBuf`, som ersätter alla tecken i socket-arrayen med \0 som är samma som att radera arrayen från tecken. Detta utfördes på bounce.c efter den sista PROMPT-en. Resultatet:

```
datorkom@t002-client:~/IPC$ ./bounce 9000
Connected on port 9000, hopefully an echo server!
Type some text. Exit with Ctrl/D (end of file).
S: P
R: !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
S: p
R: !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
S:
R: !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
S: d
R: 0123456789
S: D
R: 0123456789
```

Bilagor

Bilaga 1 – Kod för Uppgift 2

```
/*bounced.c*/

int take_call(int sock, int n)
{
    int cc; /* Check code and character count */
    /*
    char buf[2048]; /* Send - receive (echo) buffer */

    printf("Child process %d is now running\n", n);

    while((cc = read(sock, buf, sizeof(buf))) > 0)
    {
        write(sock, buf, cc);
    }

    if(cc == 0)
    {
        printf("Connection %d closed by client\n", n);
    }
    else
    {
        fprintf(stderr, "Read call failed on connection %d. " ,n);
    }

    TCPclose(sock);
    printf("Child process %d is closing down\n", n);

    return cc;
}
```

Bilaga 2 – Kod för Uppgift 3

```
/* bounce.c */

#include <string.h>
#include <stdio.h>
#include <errno.h>
#include "INETutils.h"

#define PROMPT(s) { printf(s); fflush(stdout); }

int main(int argc, char *argv[])
{
    int sock; /* Socket file descriptor */
    int rcc, wcc; /* Check code and character count */
    int port; /* TCP port in host byte order */
    char sbuf[2048]; /* Send buffer */
    char rbuf[2048]; /* Receive buffer */
    char *host = NULL; /* Remote host name or IP address */

    if (argc < 2) {
        fprintf(stderr, "TCP port number is missing\n");
        return 1;
    }
    if (sscanf(argv[1], "%d", &port) != 1) {
        fprintf(stderr, "TCP port number is invalid\n");
        return 1;
    }
    if (argc > 2)
        host = argv[2];

    /*
-----
-
Connect to bounce echo service on host, write command and read answer.
-----
*/
    /**
    *****
    /** Make your changes here. ***/
    /** Replace all INETutils calls by /sys/socket calls! ***/
    *****
    */

    struct sockaddr_in sin;
    struct hostent *hp; /* Pointer to host data structure */
    struct servent *sp; /* Pointer to service data structure */

    char buf[100];

    bzero(&sin, sizeof(sin)); /* Clear Internet socket address structure */

    gethostname(rbuf, sizeof(rbuf)); /* Get local host name */
    host = rbuf; /* Default host name = local host name */

    hp = gethostbyname(host);
    bcopy(hp->h_addr, &sin.sin_addr, hp->h_length);

    sin.sin_port = htons(port);
    sin.sin_family = AF_INET;
```

```
server */
sock = socket(AF_INET, SOCK_STREAM, 0);
connect(sock, (struct sockaddr *) &sin, sizeof(sin)); /* Connection to

printf("Connected on port %d, hopefully an echo server!\n", port);
printf("Type some text. Exit with Ctrl/D (end of file).\n");
PROMPT("S: ");
while ((wcc = read(0, sbuf, sizeof(sbuf))) > 0)    /* Read from stdin */
{
    write(sock, sbuf, wcc    /* Write to socket */
    rcc = read(sock, rbuf, sizeof(rbuf)); // Read echo from socket
    PROMPT("R: ");          /* Received */
    write(1, rbuf, rcc);     /* Write echo to stdout */
    PROMPT("S: ");          /* Sent */
}

shutdown(sock, 2);          /* Disallow both send and receive */
close(sock);                /* Close socket */
}
```

Bilaga 3 – Kod för Uppgift 4

```
/* bounced.c */

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <signal.h>
#include "INETutils.h"

int take_call(int sock, int n)
{
    int cc; /* Check code and character count */
    char buf[2048]; /* Send - receive (echo) buffer */
    char siffror[] = "0123456789\n";
    char letters[100];

    printf("Child process %d is now running\n", n);

    while((cc = read(sock, buf, sizeof(buf))) > 0)
    {
        if(buf[0] == 'P' || buf[0] == 'p' || (buf[0] == '\n'))
        {
            int i;
            for(i = 32; i <= 126; i++)
            {
                letters[i-32] = i;
            }
            letters[127-32] = '\n';
            write(sock, letters, sizeof(letters) - 4);
        }
        else if(buf[0] == 'D' || buf[0] == 'd')
        {
            write(sock, siffror, sizeof(siffror));
        }
        else
        {
            write(sock, buf, cc);
        }
    }

    if(cc = 0)
    {
        printf("Connection %d closed by client\n", n);
    }
    else
    {
        fprintf(stderr, "Read call failed on connection %d. ", n);
    }

    TCPclose(sock);
    printf("Child process %d is closing down\n", n);

    return cc;
}
```

Bilaga 4 – Kod för Uppgift 4

```
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include "INETutils.h"

#define PROMPT(s) { printf(s); fflush(stdout); }

clearbuf(char* buf, int size)          /* För rensning av buf */
{
    int i = 0;
    for(i; i < size; i++)
    {
        buf[i] = '\0';
    }
}

int main(int argc, char *argv[])
{
    int sock;                          /* Socket file descriptor */
    int rcc, wcc;                       /* Check code and character count */
    int port;                           /* TCP port in host byte order */
    char sbuf[2048];                    /* Send buffer */
    char rbuf[2048];                    /* Receive buffer */
    char *host = NULL;                  /* Remote host name or IP address */

    /*
    -----
    Parse command line for TCP port number and remote host name.
    -----
    */
    if (argc < 2) {
        fprintf(stderr, "TCP port number is missing\n");
        return 1;
    }
    if (sscanf(argv[1], "%d", &port) != 1) {
        fprintf(stderr, "TCP port number is invalid\n");
        return 1;
    }
    if (argc > 2)
        host = argv[2];

    /*
    -----
    Connect to bounce echo service on host, write command and read answer.
    -----
    */
    /******
    /*** Make your changes here.                ***
    /*** Replace all INETutils calls by /sys/socket calls! ***
    *****/

    struct sockaddr_in sin;
    struct hostent *hp;                  /* Pointer to host data structure */
    struct servent *sp;                  /* Pointer to service data structure */

    char buf[100];

    bzero(&sin, sizeof(sin));

    gethostname(rbuf, sizeof(rbuf));
    host = rbuf;
```
