Lab Exercise 2

At the end of this lab exercise students should be able to:

- Write their own stream server using socket and its related system calls.
- Write their own stream client using socket and its related system calls.
- Write their own UDP server using socket and its related system calls.
- Write their own UDP client using socket and its related system calls.

Please complete this lab exercise and submit before the due date given by your instructor.

Introduction

For these laboratory exercises you are required to bring a LIVE CD of any Linux Distribution (Knoppix, Ubuntu etc). These exercises are designed for Linux GCC compiler. To compile, say a file called fork1.c, open up a terminal and execute:

1. gcc –o fork1 fork1.c

   gcc – gnu c compiler
   -o – is the option to name the output binary file, in this case fork1
   fork1.c – is the name of the input file to be compile

2. To run and see the output of the exercise, run:

   ./fork1

3. If you get an error, noted down the number of line at which the error occur. Edit the source file and compile again. Until there are no more error.

Please remember that all completed lab exercises MUST BE submitted to your instructor before the given dateline. Please email the completed exercise to:

ali@tmsk.uitm.edu.my

and make sure that subject is written as follow:

(ITT440) Laboratory 2: Name – Student ID

I have setup a filter in my email account which will sort all laboratory assignment into a folder. If your laboratory assignment is not in the folder automatically, that means you are not following the instruction properly.

All laboratory exercises are expected to be archive in a .tgz format which contains all the required exercises. Please DO NOT sent me multiple mails attaching one exercise per mail, and please DO NOT attach multiple files per email. Thank you.

Example 1: Stream Server

Compile the program and note the output:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <signal.h>

#define PORT "3490"  // the port users will be connecting to

#define BACKLOG 10   // how many pending connections queue will hold

void sigchld_handler(int s)
{
    while(waitpid(-1, NULL, WNOHANG) > 0);
}

// get sockaddr, IPv4 or IPv6:
void *get_in_addr(struct sockaddr *sa)
{
    if (sa->sa_family == AF_INET) {
        return &(((struct sockaddr_in*)sa)->sin_addr);
    }

    return &(((struct sockaddr_in6*)sa)->sin6_addr);
}

int main(void)
{
    int sockfd, new_fd;  // listen on sock_fd, new connection on new_fd
    struct addrinfo hints, *servinfo, *p;
    struct sockaddr_storage their_addr; // connector's address information
    socklen_t sin_size;
    struct sigaction sa;
    int yes=1;
    char s[INET6_ADDRSTRLEN];
    int rv;

    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE; // use my IP
```

continue next page ->

```
if ((rv = getaddrinfo(NULL, PORT, &hints, &servinfo)) != 0) {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
    return 1;
}

// loop through all the results and bind to the first we can
for(p = servinfo; p != NULL; p = p->ai_next) {
    if ((sockfd = socket(p->ai_family, p->ai_socktype,
            p->ai_protocol)) == -1) {
        perror("server: socket");
        continue;
    }

    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes,
            sizeof(int)) == -1) {
        perror("setsockopt");
        exit(1);
    }

    if (bind(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
        close(sockfd);
        perror("server: bind");
        continue;
    }

    break;
}

if (p == NULL)  {
    fprintf(stderr, "server: failed to bind\n");
    return 2;
}

freeaddrinfo(servinfo); // all done with this structure

if (listen(sockfd, BACKLOG) == -1) {
    perror("listen");
    exit(1);
}

sa.sa_handler = sigchld_handler; // reap all dead processes
sigemptyset(&sa.sa_mask);
sa.sa_flags = SA_RESTART;
if (sigaction(SIGCHLD, &sa, NULL) == -1) {
    perror("sigaction");
    exit(1);
}
```

continue next page ->

```
    printf("server: waiting for connections...\n");

    while(1) {  // main accept() loop
        sin_size = sizeof their_addr;
        new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);
        if (new_fd == -1) {
            perror("accept");
            continue;
        }

        inet_ntop(their_addr.ss_family,
            get_in_addr((struct sockaddr *)&their_addr),
            s, sizeof s);
        printf("server: got connection from %s\n", s);

        if (!fork()) { // this is the child process
            close(sockfd); // child doesn't need the listener
            if (send(new_fd, "Hello, world!", 13, 0) == -1)
                perror("send");
            close(new_fd);
            exit(0);
        }
        close(new_fd);  // parent doesn't need this
    }

    return 0;
}
```

After running the server, we should get the message:

"server: waiting for connection"

Other than this message, another way to verify that the server is listening on the correct port is by running the *netstat* command. Open up another terminal and run:

$*netstat –an | more*

Verify that we get output similar to the following:

Active Internet connections (including servers)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	(state)
tcp46	0	0	*.3490	*.*	LISTEN

Which shows that there are process listening on port 3490.

Example 2: Stream Client

Compile the program and note the output:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>

#include <arpa/inet.h>

#define PORT "3490" // the port client will be connecting to

#define MAXDATASIZE 100 // max number of bytes we can get at once

// get sockaddr, IPv4 or IPv6:
void *get_in_addr(struct sockaddr *sa)
{
    if (sa->sa_family == AF_INET) {
        return &(((struct sockaddr_in*)sa)->sin_addr);
    }

    return &(((struct sockaddr_in6*)sa)->sin6_addr);
}

int main(int argc, char *argv[])
{
    int sockfd, numbytes;
    char buf[MAXDATASIZE];
    struct addrinfo hints, *servinfo, *p;
    int rv;
    char s[INET6_ADDRSTRLEN];

    if (argc != 2) {
        fprintf(stderr,"usage: client hostname\n");
        exit(1);
    }

    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;

    if ((rv = getaddrinfo(argv[1], PORT, &hints, &servinfo)) != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
        return 1;
    }
```

Continue next page ->

Credits: Beej's Guide to Unix Socket
http://beej.us/guide/bgnet/

```
    // loop through all the results and connect to the first we can
    for(p = servinfo; p != NULL; p = p->ai_next) {
        if ((sockfd = socket(p->ai_family, p->ai_socktype,
                p->ai_protocol)) == -1) {
            perror("client: socket");
            continue;
        }

        if (connect(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
            close(sockfd);
            perror("client: connect");
            continue;
        }

        break;
    }

    if (p == NULL) {
        fprintf(stderr, "client: failed to connect\n");
        return 2;
    }

    inet_ntop(p->ai_family, get_in_addr((struct sockaddr *)p->ai_addr),
            s, sizeof s);
    printf("client: connecting to %s\n", s);

    freeaddrinfo(servinfo); // all done with this structure

    if ((numbytes = recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1) {
        perror("recv");
        exit(1);
    }

    buf[numbytes] = '\0';

    printf("client: received '%s'\n",buf);

    close(sockfd);

    return 0;
}
```

To test the client, run the stream server in example 1. While the server is waiting for connection, open up a terminal and run the client. Note the output.

Programming Exercise 1:

1. Based on example 1 and 2, write a server that will wait for connection from client, once a client is connected, it will wait for message from the client and print back the message to the client.

   Example:

   Client: Hi there!
   Server: Hi there!
   Client: I am Abu.
   Server: I am Abu.

   (Basically what you are creating is an echo server)

2. Based on example 1 and 2, write a server that will wait for connection from client, once a client is connected, it will send back the current time to the server.

   (Basically what you are creating is a time server)

3. Based on example 1 and 2, write a server that will wait for connection from client, once a client is connected. It will wait for further command from the client.

   The server should be able to accept the following command:

   a. ADD
   b. SUBTRACT

   Followed by 2 numbers. If the command is ADD, server will add these 2 numbers. If the command is SUBTRACT, server will subtract these 2 numbers.

   Example:

   Server: waiting for connection

   Client: ADD 2 2

   Server: 4

4. Example 3: UDP Server

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#define MYPORT "4950"   // the port users will be connecting to

#define MAXBUFLEN 100

// get sockaddr, IPv4 or IPv6:
void *get_in_addr(struct sockaddr *sa)
{
    if (sa->sa_family == AF_INET) {
        return &(((struct sockaddr_in*)sa)->sin_addr);
    }

    return &(((struct sockaddr_in6*)sa)->sin6_addr);
}

int main(void)
{
    int sockfd;
    struct addrinfo hints, *servinfo, *p;
    int rv;
    int numbytes;
    struct sockaddr_storage their_addr;
    char buf[MAXBUFLEN];
    size_t addr_len;
    char s[INET6_ADDRSTRLEN];

    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_UNSPEC; // set to AF_INET to force IPv4
    hints.ai_socktype = SOCK_DGRAM;
    hints.ai_flags = AI_PASSIVE; // use my IP

    if ((rv = getaddrinfo(NULL, MYPORT, &hints, &servinfo)) != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
        return 1;
    }
```

continue next page ->

```
    // loop through all the results and bind to the first we can
    for(p = servinfo; p != NULL; p = p->ai_next) {
        if ((sockfd = socket(p->ai_family, p->ai_socktype,
                p->ai_protocol)) == -1) {
            perror("listener: socket");
            continue;
        }

        if (bind(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
            close(sockfd);
            perror("listener: bind");
            continue;
        }

        break;
    }

    if (p == NULL) {
        fprintf(stderr, "listener: failed to bind socket\n");
        return 2;
    }

    freeaddrinfo(servinfo);

    printf("listener: waiting to recvfrom...\n");

    addr_len = sizeof their_addr;
    if ((numbytes = recvfrom(sockfd, buf, MAXBUFLEN-1 , 0,
        (struct sockaddr *)&their_addr, &addr_len)) == -1) {
        perror("recvfrom");
        exit(1);
    }

    printf("listener: got packet from %s\n",
        inet_ntop(their_addr.ss_family,
            get_in_addr((struct sockaddr *)&their_addr),
            s, sizeof s));
    printf("listener: packet is %d bytes long\n", numbytes);
    buf[numbytes] = '\0';
    printf("listener: packet contains \"%s\"\n", buf);

    close(sockfd);

    return 0;
}
```

Example 4: UDP Client

Compile the program and note the output:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#define SERVERPORT "4950"   // the port users will be connecting to

int main(int argc, char *argv[])
{
    int sockfd;
    struct addrinfo hints, *servinfo, *p;
    int rv;
    int numbytes;

    if (argc != 3) {
        fprintf(stderr,"usage: talker hostname message\n");
        exit(1);
    }

    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_DGRAM;

    if ((rv = getaddrinfo(argv[1], SERVERPORT, &hints, &servinfo)) != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
        return 1;
    }

    // loop through all the results and make a socket
    for(p = servinfo; p != NULL; p = p->ai_next) {
        if ((sockfd = socket(p->ai_family, p->ai_socktype,
                p->ai_protocol)) == -1) {
            perror("talker: socket");
            continue;
        }

        break;
    }

    if (p == NULL) {
        fprintf(stderr, "talker: failed to bind socket\n");
        return 2;
    }
```

Continue next page ->

```
    if ((numbytes = sendto(sockfd, argv[2], strlen(argv[2]), 0,
             p->ai_addr, p->ai_addrlen)) == -1) {
        perror("talker: sendto");
        exit(1);
    }

    freeaddrinfo(servinfo);

    printf("talker: sent %d bytes to %s\n", numbytes, argv[1]);
    close(sockfd);

    return 0;
}
```

- end of laboratory session -