# System Programming Lecture - Report 6

Student Number: 201520740

Name: LIN WEI

## Exercise 1

### Program

**main.c**

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

#include "logutil.h"

#define DEFAULT_SERVER_PORT	10000
#ifdef SOMAXCONN
#define LISTEN_BACKLOG SOMAXCONN
#else
#define LISTEN_BACKLOG 5
#endif

char *program_name = "sp6-server";

int
open_accepting_socket(int port)
```

```
{
	struct sockaddr_in self_addr;
	socklen_t self_addr_size;
	int sock, sockopt;

	memset(&self_addr, 0, sizeof(self_addr));
	self_addr.sin_family = AF_INET;
	self_addr.sin_addr.s_addr = INADDR_ANY;
	self_addr.sin_port = htons(port);
	self_addr_size = sizeof(self_addr);
	sock = socket(PF_INET, SOCK_STREAM, 0);
	if (sock < 0)
		logutil_fatal("accepting socket: %d", errno);
	sockopt = 1;
	if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR,
	    &sockopt, sizeof(sockopt)) == -1)
		logutil_warning("SO_REUSEADDR: %d", errno);
	if (bind(sock, (struct sockaddr *)&self_addr, self_addr_size) < 0)
		logutil_fatal("bind accepting socket: %d", errno);
	if (listen(sock, LISTEN_BACKLOG) < 0)
		logutil_fatal("listen: %d", errno);
	return (sock);
}

void
usage(void)
{
	fprintf(stderr, "Usage: %s [option]\n", program_name);
	fprintf(stderr, "option:\n");
	fprintf(stderr, "\t-d\t\t\t\t... debug mode\n");
	fprintf(stderr, "\t-p <port>\n");
	exit(1);
```

```
}

void *
process(void* sock)
{
        int s = *((int *) sock);
        char c;
        FILE * in;

        if((in = fdopen(s, "r")) == NULL){
                fprintf(stderr, "Failed in fdopen(read).\n");
                exit(EXIT_FAILURE);
        }

        while ((c = getc(in)) != EOF)
        {
                printf("%c",c);
        }

        fprintf(stderr, "Disconnected from client %d.\n", s);
        fclose(in);
        free(sock);
}

void
main_loop(int sock_fd)
{
        struct sockaddr_in client_addr;
        int addr_len = sizeof(client_addr);
  int client_sock_fd;
        pthread_t processor;
  printf("Waiting for connection...\n");
        while(1){
                if((client_sock_fd = accept(sock_fd, (struct sockaddr *)&client_addr, &addr_len)) <
0){
                        fprintf(stderr,"Failed in accept.\n");
                        exit(EXIT_FAILURE);
                }
                printf("Accepted connection from %d, port: %d\n", inet_ntoa(client_addr.sin_addr),
                                        ntohs(client_addr.sin_port));
                int *arg = malloc(sizeof(*arg));
                if(arg == NULL){
                        fprintf(stderr, "No space for thread arg.\n");
                        exit(EXIT_FAILURE);
                }
                *arg = client_sock_fd;
                pthread_create(&processor, NULL, process, arg);
                pthread_detach(processor, NULL);
        }

}

int
main(int argc, char **argv)
{
    char *port_number = NULL;
    int ch, sock, server_port = DEFAULT_SERVER_PORT;
    int debug_mode = 0;

    while ((ch = getopt(argc, argv, "dp:")) != -1) {
        switch (ch) {
        case 'd':
            debug_mode = 1;
            break;
		case 'p':
			port_number = optarg;
			break;
		case '?':
		default:
			usage();
		}
	}
	argc -= optind;
	argv += optind;

	if (port_number != NULL)
		server_port = strtol(port_number, NULL, 0);

	/* server_port で listen し，socket descriptor を sock に代入 */
	sock = open_accepting_socket(server_port);

	if (!debug_mode) {
		logutil_syslog_open(program_name, LOG_PID, LOG_LOCAL0);
		daemon(0, 0);
	}

	/*
	 * 無限ループで sock を accept し，accept したらそのクライアント用
	 * のスレッドを作成しプロトコル処理を続ける.
	 */
	main_loop(sock);

	/*NOTREACHED*/
	return (0);
}
```

## Result

### 1. Server Terminal

```
lw@lw-VirtualBox:~/Documents/Report6/Exercise1$ ./a.out -d
Waiting for connection...
Accepted connection from 345196312, port: 40718
hello
Accepted connection from 345196312, port: 40719
hi
Accepted connection from 345196312, port: 40720
my name is
lw
Disconnected from client 4.
Disconnected from client 5.
^C
```

### 2. Client 1 Terminal

```
lw@lw-VirtualBox:~$ telnet localhost 10000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hello
^CConnection closed by foreign host.
```

### 3. Client 2 Terminal

```
lw@lw-VirtualBox:~$ telnet localhost 10000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hi
^CConnection closed by foreign host.
```

### 4. Client 3 Terminal

```
lw@lw-VirtualBox:~$ telnet localhost 10000
Trying 127.0.0.1...
Connected to localhost.
```

```
Escape character is '^]'.
my name is
lw
Connection closed by foreign host.
```

## Consideration on Exercise 1

The server can communicate with multiple clients simultaneously, which means each thread created by main_loop runs correctly. The server would say “disconnected” when the corresponding client sends EOF. And the client will be disconnected when the server shuts down.

## Exercise 2

### Program

**main.c**

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <syslog.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <pthread.h>
#include <signal.h>
#include <poll.h>

#include "logutil.h"

#define DEFAULT_SERVER_PORT	10000
#ifdef SOMAXCONN
#define LISTEN_BACKLOG SOMAXCONN
#else
#define LISTEN_BACKLOG 5
```

```
#endif

char *program_name = "sp6-server";

pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;

int bye = 0;

sigset_t byeset;

int
open_accepting_socket(int port)
{
	struct sockaddr_in self_addr;
	socklen_t self_addr_size;
	int sock, sockopt;

	memset(&self_addr, 0, sizeof(self_addr));
	self_addr.sin_family = AF_INET;
	self_addr.sin_addr.s_addr = INADDR_ANY;
	self_addr.sin_port = htons(port);
	self_addr_size = sizeof(self_addr);
	sock = socket(PF_INET, SOCK_STREAM, 0);
	if (sock < 0)
		logutil_fatal("accepting socket: %d", errno);
	sockopt = 1;
	if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR,
	    &sockopt, sizeof(sockopt)) == -1)
		logutil_warning("SO_REUSEADDR: %d", errno);
	if (bind(sock, (struct sockaddr *)&self_addr, self_addr_size) < 0)
		logutil_fatal("bind accepting socket: %d", errno);
	if (listen(sock, LISTEN_BACKLOG) < 0)
```

```
            logutil_fatal("listen: %d", errno);
        return (sock);
}

void
usage(void)
{
        fprintf(stderr, "Usage: %s [option]\n", program_name);
        fprintf(stderr, "option:\n");
        fprintf(stderr, "\t-d\t\t\t\t... debug mode\n");
        fprintf(stderr, "\t-p <port>\n");
        exit(1);
}

void *
process(void* sock)
{
                int s = *((int *) sock);
                char c;
                FILE * in;

                if((in = fdopen(s, "r")) == NULL){
                                fprintf(stderr, "Failed in fdopen(read).\n");
                                exit(EXIT_FAILURE);
                }

                while ((c = getc(in)) != EOF)
                {
                                printf("%c",c);
                }

                fprintf(stderr, "Disconnected from client %d.\n", s);
        fclose(in);
        free(sock);
}

void *
handle_bye(void *arg)
{
        int sig, err;

        while(1){
                err = sigwait(&byeset, &sig);
                if(err){
                        fprintf(stderr, "Failed in sigwait().\n");
                }
                else if(sig != SIGINT && sig != SIGTERM){
                        fprintf(stderr, "Failed in SIGINT or SIGTERM.\n");
                }
                else{
                break;
                }
        }

        pthread_mutex_lock(&m);
        bye = 1;
        pthread_mutex_unlock(&m);

        return NULL;
}

void
main_loop(int sock_fd)
{
  struct sockaddr_in client_addr;
  int addr_len = sizeof(client_addr);
  int client_sock_fd;
  pthread_t processor;
        struct pollfd fds;
  int ready;

        fds.fd = sock_fd;
        fds.events = POLLIN;

  printf("Waiting for connection...\n");

        /* The program will be blocked here if there is no pending connections. */
        /* So we use poll function to only execute the accept function
         when there are data ready for reading.*/
        while(1){
                ready = poll(&fds, 1, 1000);
                //printf("ready: %d\n", ready);
                if(ready){
                        if((client_sock_fd = accept(sock_fd, (struct sockaddr *)&client_addr,
&addr_len)) < 0){
                                fprintf(stderr,"Failed in accept.\n");
                                exit(EXIT_FAILURE);
                        }
                        printf("Accepted connection from %d, port: %d\n",
inet_ntoa(client_addr.sin_addr),
                                        ntohs(client_addr.sin_port));

                        /*Malloc space to pass arguments to thread.*/
                        int *arg = malloc(sizeof(*arg));
                        if(arg == NULL){
                                fprintf(stderr, "No space for thread arg.\n");
                                exit(EXIT_FAILURE);
                        }
                        *arg = client_sock_fd;
                        pthread_create(&processor, NULL, process, arg);
                        pthread_detach(processor);
                }

                pthread_mutex_lock(&m);
                if(bye){
                        printf("bye\n");
                        pthread_mutex_unlock(&m);
                        exit(0);
                }
                pthread_mutex_unlock(&m);
        }

}

int
main(int argc, char **argv)
{
    char *port_number = NULL;
    int ch, sock, server_port = DEFAULT_SERVER_PORT;
    int debug_mode = 0;
                                pthread_t t;

    while ((ch = getopt(argc, argv, "dp:")) != -1) {
        switch (ch) {
        case 'd':
            debug_mode = 1;
            break;
        case 'p':
            port_number = optarg;
            break;
        case '?':
        default:
            usage();
        }
}
argc -= optind;
argv += optind;

if (port_number != NULL)
        server_port = strtol(port_number, NULL, 0);

/* server_port で listen し，socket descriptor を sock に代入 */
sock = open_accepting_socket(server_port);

if (!debug_mode) {
        logutil_syslog_open(program_name, LOG_PID, LOG_LOCAL0);
        daemon(0, 0);
}

/*
 * 無限ループで sock を accept し，accept したらそのクライアント用
 * のスレッドを作成しプロトコル処理を続ける.
 */

      sigemptyset(&byeset);
      sigaddset(&byeset, SIGINT);
      sigaddset(&byeset, SIGTERM);
      pthread_sigmask(SIG_BLOCK, &byeset, NULL);
      pthread_create(&t, NULL, handle_bye, NULL);

main_loop(sock);
```

```
	/*NOTREACHED*/
	return (0);
}
```

## Result

### 1. Server Terminal

```
lw@lw-VirtualBox:~/Documents/Report6/Exercise2$ ./a.out -d
Waiting for connection...
Accepted connection from 1840895768, port: 40728
Accepted connection from 1840895768, port: 40729
hello
hi
^Cbye
lw@lw-VirtualBox:~/Documents/Report6/Exercise2$
```

### 2. Client 1 Terminal

```
lw@lw-VirtualBox:~$ telnet localhost 10000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hello
Connection closed by foreign host.
```

### 3. Client 2 Terminal

```
lw@lw-VirtualBox:~$ telnet localhost 10000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hi
Connection closed by foreign host.
```

## Consideration on Exercise 2

The server would say “bye” when we press Ctrl-C, that is, send SIGINT signal to it, and the clients are disconnected. I found the program will stop and wait before the while loop in main_loop when there is no pending connections. For this, we can utilize the poll function to only execute the accept function while there are data ready for reading, which in other words means, there are clients trying to connect.

## Review of this lecture

The exercises help me understand the basic of signal and signal mask in Linux programming. And I also learned that the signal hander thread with condition variables make it easier than signal handler to process async signals. Thank you for this lecture!