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**

#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!