System Programming Lecture - Report 8

Student Number: 201520740

Name: LIN WEI

Exercise 1

**circular\_buffer.h**

#include <pthread.h>

#ifndef CIRCULAR\_BUFFER\_H

#define CIRCULAR\_BUFFER\_H

#define QSIZE 100

typedef struct

{

pthread\_mutex\_t buf\_lock;

int start;

int num\_full;

pthread\_cond\_t notfull;

pthread\_cond\_t notempty;

void \*data[QSIZE];

}circ\_buf\_t;

#endif

**circular\_buffer.c**

#include <stdio.h>

#include <pthread.h>

#include <stdint.h>

#include "circular\_buffer.h"

/\* Clean up handler of the circular buffer. \*/

void circular\_buffer\_cleanup(void \*cbt)

{

circ\_buf\_t \*cbp = (circ\_buf\_t \*) cbt;

pthread\_mutex\_unlock(&cbp->buf\_lock);

}

/\* Put data into the circluar buffer. \*/

void put\_cb\_data(circ\_buf\_t \*cbp, void \*data)

{

int s = (intptr\_t) data;

printf("Enqueue connection request from client %d\n", s);

pthread\_mutex\_lock(&cbp->buf\_lock);

pthread\_cleanup\_push(circular\_buffer\_cleanup, (void \*) cbp);

while(cbp->num\_full == QSIZE)

{

pthread\_cond\_wait(&cbp->notfull, &cbp->buf\_lock);

}

cbp->data[(cbp->start + cbp->num\_full) % QSIZE] = data;

cbp->num\_full++;

pthread\_cond\_signal(&cbp->notempty);

pthread\_cleanup\_pop(1);

}

/\* Get data from the circular buffer. \*/

void \*get\_cb\_data(circ\_buf\_t \*cbp)

{

void \*data;

pthread\_mutex\_lock(&cbp->buf\_lock);

pthread\_cleanup\_push(circular\_buffer\_cleanup, (void \*) cbp);

while(cbp->num\_full == 0)

{

pthread\_cond\_wait(&cbp->notempty, &cbp->buf\_lock);

}

data = cbp->data[cbp->start];

cbp->start = (cbp->start + 1) % QSIZE;

cbp->num\_full--;

pthread\_cond\_signal(&cbp->notfull);

pthread\_cleanup\_pop(1);

int s = (intptr\_t) data;

printf("Dequeue connection request from client %d\n", s);

return(data);

}

Exercise 2

/\* The worker thread. \*/

void \*worker(void \*arg)

{

pthread\_t processor;

int data;

while(1)

{

pthread\_create(&processor, NULL, process, (void \*) (intptr\_t) get\_cb\_data(cbt));

pthread\_detach(processor);

}

}

Exercise 3

Program

**main.c**

#include <stdlib.h>

#include <stdio.h>

#include <stdint.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"

#include "circular\_buffer.h"

#define DEFAULT\_SERVER\_PORT 10000

#ifdef SOMAXCONN

#define LISTEN\_BACKLOG SOMAXCONN

#else

#define LISTEN\_BACKLOG 5

#endif

#define WORKER\_THREAD\_NUM 3

char \*program\_name = "sp6-server";

circ\_buf\_t \*cbt;

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);

}

/\* The connection handling thread. \*/

void \*process(void \*sock)

{

int s = (intptr\_t) sock;

char c;

FILE \* in;

char greeting [50];

sprintf(greeting, "Hello, client %d.\n", s);

write(s, greeting, strlen(greeting));

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);

}

/\* The worker thread. \*/

void \*worker(void \*arg)

{

pthread\_t processor;

int data;

while(1)

{

pthread\_create(&processor, NULL, process, (void \*) (intptr\_t) get\_cb\_data(cbt));

pthread\_detach(processor);

}

}

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;

}

/\* Initialize the circular buffer. \*/

void circ\_buf\_init()

{

cbt = (circ\_buf\_t \*)malloc(sizeof(circ\_buf\_t));

pthread\_mutex\_init(&cbt->buf\_lock, NULL);

cbt->start = 0;

cbt->num\_full = 0;

pthread\_cond\_init(&cbt->notfull, NULL);

pthread\_cond\_init(&cbt->notempty, NULL);

}

/\* The thread to enqueue connection requests into the buffer. \*/

void\* enqueue\_request(void \*sock)

{

int sock\_fd = (intptr\_t) sock;

struct sockaddr\_in client\_addr;

int addr\_len = sizeof(client\_addr);

int client\_sock\_fd;

struct pollfd fds;

int ready;

fds.fd = sock\_fd;

fds.events = POLLIN;

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

/\* The program will block here if there is no pending connection \*/

/\* 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);

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));

put\_cb\_data(cbt, (void \*) (intptr\_t) client\_sock\_fd);

}

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;

int i;

pthread\_t t, workers[WORKER\_THREAD\_NUM], enq;

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);

sock = open\_accepting\_socket(server\_port);

if (!debug\_mode)

{

logutil\_syslog\_open(program\_name, LOG\_PID, LOG\_LOCAL0);

daemon(0, 0);

}

sigemptyset(&byeset);

sigaddset(&byeset, SIGINT);

sigaddset(&byeset, SIGTERM);

pthread\_sigmask(SIG\_BLOCK, &byeset, NULL);

pthread\_create(&t, NULL, handle\_bye, NULL);

circ\_buf\_init();

pthread\_create(&enq, NULL, enqueue\_request, (void \*) (intptr\_t) sock);

for(i = 0; i < WORKER\_THREAD\_NUM; i++)

pthread\_create(&workers[i], NULL, worker, NULL);

pthread\_join(enq, NULL);

return (0);

}

Result

1. Server Terminal

lw@lw-VirtualBox:~/Documents/Report8$ ./a.out -d

Waiting for connection...

Accepted connection from -1458043176, port: 58185

Enqueue connection request from client 4

Dequeue connection request from client 4

hello 4

Accepted connection from -1458043176, port: 58186

Enqueue connection request from client 5

Dequeue connection request from client 5

hello 5

Accepted connection from -1458043176, port: 58187

Enqueue connection request from client 6

Dequeue connection request from client 6

hello 6

Accepted connection from -1458043176, port: 58188

Enqueue connection request from client 7

Dequeue connection request from client 7

hello 7

Accepted connection from -1458043176, port: 58189

Enqueue connection request from client 8

Dequeue connection request from client 8

hello 8

Disconnected from client 4.

1. Client 4 Terminal

lw@lw-VirtualBox:~$ telnet localhost 10000

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

Hello, client 4.

hello 4

^CConnection closed by foreign host.

1. Client 5 Terminal

lw@lw-VirtualBox:~$ telnet localhost 10000

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

Hello, client 5.

hello 5

1. Client 6 Terminal

lw@lw-VirtualBox:~$ telnet localhost 10000

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

Hello, client 6.

hello 6

1. Client 7 Terminal

lw@lw-VirtualBox:~$ telnet localhost 10000

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

Hello, client 7.

hello 7

1. Client 8 Terminal

lw@lw-VirtualBox:~$ telnet localhost 10000

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

Hello, client 8.

hello 8

Consideration on Exercise 3

All these 5 clients connected correctly with the server. When a client tried to connect in, the connection request was added into the buffer. And the worker threads get the connection request from the buffer, then create a thread to communicate with the client. If the server read an EOF char from a client, it will close the connection with that client (like client 4 in our program).

Exercise 4

The program of exercise 4 is the same as exercise 3 except adding some thread cancellation code at the end of main function.

sleep(60);

for(i = 0; i < WORKER\_THREAD\_NUM; i++)

pthread\_cancel(workers[i]);

Result

1. Server Terminal

lw@lw-VirtualBox:~/Documents/Report8$ ./a.out -d

Waiting for connection...

Accepted connection from -111012136, port: 58194

Enqueue connection request from client 4

Dequeue connection request from client 4

Accepted connection from -111012136, port: 58195

Enqueue connection request from client 5

Dequeue connection request from client 5

Accepted connection from -111012136, port: 58196

Enqueue connection request from client 6

Dequeue connection request from client 6

Accepted connection from -111012136, port: 58197

Enqueue connection request from client 7

Dequeue connection request from client 7

Accepted connection from -111012136, port: 58198

Enqueue connection request from client 8

Dequeue connection request from client 8

hi 4

lw@lw-VirtualBox:~/Documents/Report8$

1. Client 4 Terminal

lw@lw-VirtualBox:~$ telnet localhost 10000

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

Hello, client 4.

hi 4

Connection closed by foreign host.

1. Client 5 Terminal

lw@lw-VirtualBox:~$ telnet localhost 10000

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

Hello, client 5.

Connection closed by foreign host.

1. Client 6 Terminal

lw@lw-VirtualBox:~$ telnet localhost 10000

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

Hello, client 6.

Connection closed by foreign host.

1. Client 7 Terminal

lw@lw-VirtualBox:~$ telnet localhost 10000

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

Hello, client 7.

Connection closed by foreign host.

1. Client 8 Terminal

lw@lw-VirtualBox:~$ telnet localhost 10000

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

Hello, client 8.

Connection closed by foreign host.

Consideration on Exercise 4

All these 5 clients connected correctly with the server. The worker threads were cancelled after 60 seconds, after which, all clients were disconnected from the server.

Review of this lecture

In this lecture, I have learned the basic of cancellation mechanism of thread and the concept of thread pool. Since this is the last class, I would like write some comments here. I think the report after each lesson is good because they can improve our programming skills, but sometimes, your PPT slides were quiet difficult to understand. Maybe it will be much better if you can attach some diagrams or list more examples to explain some abstract concepts. Anyway, thank you for this lecture.