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>
```

```c
#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.
```

2. 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.
```

3. 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
```

4. 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
```

5. 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
```

6. 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$
```

2. 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.
```

3. 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.
```

4. 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.
```

5. 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.
```

6. 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.