Mid-term

School ID: 201624476 Name: Park Sang Un

1. Submit your source file to the plato system

Done.

2. Put your program source as here

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <pthread.h>

#include <ncurses.h>

#include <stdbool.h>

#include <fcntl.h>

#include <time.h>

#include <semaphore.h>

#include <sys/ipc.h>

#include <sys/shm.h>


#define INPUT_WINDOW_H 2

#define BUFFSIZE 1024


WINDOW *input_scr;

WINDOW *chat_scr;

WINDOW *acclog_scr;

WINDOW *timer_scr;

WINDOW *local_date_wnd, *local_time_wnd, *elapsed_time_wnd;        // subwindow of each timer
```

```
// chat_timer.c -- CP33357 assignment #1, Spring 2020
// includes functions returns current date & time & elapsed execution time
// S. U. Park, id #201624476, April 3rd, 2020

// maxmimum number of users in chat

const int MAX_USERS = 100;
const int MAX_CHATS = 1000;

// coordinate preference options for terminal window

const int TERMINAL_WINDOW_HLINE = 80;
const int TERMINAL_WINDOW_VLINE = 24;
const int TERMINAL_WINDOW_HPOS = 0;
const int TERMINAL_WINDOW_VPOS = 0;

// coordinate preference options for output window

const int OUTPUT_WINDOW_HLINE = 60;
const int OUTPUT_WINDOW_VLINE = 20;
const int OUTPUT_WINDOW_HPOS = 0;
const int OUTPUT_WINDOW_VPOS = 0;
```

```
// coordinate preference options for input window

const int INPUT_WINDOW_HLINE = 60;
const int INPUT_WINDOW_VLINE = 4;
const int INPUT_WINDOW_HPOS = 0;
const int INPUT_WINDOW_VPOS = 20;

// coordinate preference options for account logging window

const int ACCLOG_WINDOW_HLINE = 20;
const int ACCLOG_WINDOW_VLINE = 20;
const int ACCLOG_WINDOW_HPOS = 60;
const int ACCLOG_WINDOW_VPOS = 0;

// coordinate preference options for timer window

const int TIMER_WINDOW_HLINE = 20;
const int TIMER_WINDOW_VLINE = 4;
const int TIMER_WINDOW_HPOS = 60;
const int TIMER_WINDOW_VPOS = 20;

// coordinate preference options for local date timer

// determine width and heights
```

```
const int LOCAL_DATE_HLINE   = 20;
const int LOCAL_DATE_VLINE   = 1;
const int LOCAL_DATE_HPOS    = 60;
const int LOCAL_DATE_VPOS    = 21;
const int LOCAL_DATE_OUT_HPOS = 1;
const int LOCAL_DATE_OUT_VPOS = 0;

// coordinate preference options for local time timer

const int LOCAL_TIME_HLINE    = 15;
const int LOCAL_TIME_VLINE    = 5;
const int LOCAL_TIME_HPOS     = 1;
const int LOCAL_TIME_VPOS     = 1;
const int LOCAL_TIME_OUT_HPOS = 1;
const int LOCAL_TIME_OUT_VPOS = 0;

// coordinate preference options for local date timer

const int ELAPSED_TIME_HLINE = 15;
const int ELAPSED_TIME_VLINE = 5;
const int ELAPSED_TIME_HPOS  = 1;
const int ELAPSED_TIME_VPOS  = 2;
const int ELAPSED_TIME_OUT_HPOS = 1;
```

```
const int ELAPSED_TIME_OUT_VPOS = 0;


const int BUFFER_SIZE = 80;                    // constant integer for buffer size
clock_t start_clock;                           // clock_t value, recorded when program execution starts
bool is_start_clock_not_initialized = false;   // ="start_clock isn't initialized"


pthread_mutex_t message_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t message_cond = PTHREAD_COND_INITIALIZER;


int row;
int col;


struct message_buffer {
    char msg[BUFFSIZE];
    int id;
};


// chat structure for multi chat with shared memory


typedef struct chatInfo {


    char userID[20];
    long messageTime;
    char message[40];
```

```
} CHAT_INFO;

typedef struct loginInfo {
    char userID[20];
    int isON;
} LOGIN_INFO;


// target memory key constraint for assignment
const int CHAT_SHM_KEY = 20200406;
const int LOGIN_SHM_KEY = 20200407;

struct message_buffer buff_in;
struct message_buffer buff_out;

int is_running;
int current_time;
int flag = 0;

char local_time_string[BUFFSIZE], elapsed_time_string[BUFFSIZE]; // formatted output value o5f each timer

int         chat_shmid;             // ID value of shared memory
int         login_shmid;
```

```
char        userID[20];        // ID of sender(a.k.a. user)
int         userIdx;
CHAT_INFO message_update_buffer;
CHAT_INFO*   chat_logs = NULL;    // pointer of chat information which will get the address of shared memory
LOGIN_INFO*   login_logs = NULL;    // pointer of user login information which will get the address of shared
memory
void *chat_shmaddr = (void*) 0;        // address pointer of chat shared memory
void *login_shmaddr = (void*) 0;        // address pointer of user login shared memory

sem_t *login_sem;
sem_t *chat_sem;

//returns string contains local time (hh-mm-ss form)
void get_local_time() {

    time_t      now;                        // current time_t value
    struct tm   time_data;                  // localtime form of 'now'
    char        buffer[BUFFER_SIZE];        // buffer contains formatted local time

    // copy current time in 'now'
    time(&now);

    // format 'now' to localtime format
    time_data = *localtime(&now);
```

```
    // convert the local date to hh-mm-ss form and save it to buffer
    strftime(buffer, sizeof(buffer), "%H-%M-%S", &time_data);

    // copy buffer contents to return string
    strcpy(local_time_string, buffer);
}

//returns string contains elapsed execution time (hh-mm-ss form)
void get_elapsed_time() {

    char        buffer[BUFFER_SIZE];    // buffer contains formatted elapsed execution time

    double elapsed_time =  (double)((clock() - start_clock)/CLOCKS_PER_SEC);    // clock_t value of elapsed
execution time that transformed to second

    int hh = (int)elapsed_time / 3600; elapsed_time -= hh * 3600;   // calculate hour from elapsed execution
time
    int mm = (int)elapsed_time / 60;   elapsed_time -= mm * 60;     // calculate minute from elapsed
execution time
    int ss = (int)elapsed_time;                                     // calculate second from elapsed
execution time

    // save hh-mm-ss form of elapsed execution time to buffer
```

```
    sprintf(buffer, "%d-%d-%d", hh,mm,ss);

    // copy buffer contents to return string
    strcpy(elapsed_time_string, buffer);
}

void init_position() {
    input_scr = newwin(INPUT_WINDOW_VLINE, INPUT_WINDOW_HLINE, INPUT_WINDOW_VPOS, INPUT_WINDOW_HPOS);
    chat_scr = newwin(OUTPUT_WINDOW_VLINE, OUTPUT_WINDOW_HLINE, OUTPUT_WINDOW_VPOS, OUTPUT_WINDOW_HPOS);
    acclog_scr = newwin(ACCLOG_WINDOW_VLINE, ACCLOG_WINDOW_HLINE, ACCLOG_WINDOW_VPOS, ACCLOG_WINDOW_HPOS);
    timer_scr = newwin(TIMER_WINDOW_VLINE, TIMER_WINDOW_HLINE, TIMER_WINDOW_VPOS, TIMER_WINDOW_HPOS);

    // local date timer subwindow is temporarily deprecated.
    /*
    local_date_wnd = subwin( timer_scr, LOCAL_DATE_VLINE, LOCAL_DATE_HLINE, LOCAL_DATE_VPOS, LOCAL_DATE_HPOS );
    local_time_wnd = subwin( timer_scr, LOCAL_TIME_VLINE, LOCAL_TIME_HLINE, LOCAL_TIME_VPOS, LOCAL_TIME_HPOS );
    elapsed_time_wnd = subwin( timer_scr, ELAPSED_TIME_VLINE, ELAPSED_TIME_HLINE, ELAPSED_TIME_VPOS, ELAPSED_TIME_HPOS );
```

```
    */

    scrollok(chat_scr, TRUE);
    wprintw(chat_scr, "₩n ***** Type /bye to quit!! ***** ₩n₩n");
    wrefresh(chat_scr);

    // if start_clock isn't initialized, initialize it with the first call of clock() in this program.
    if(is_start_clock_not_initialized) {
        start_clock = clock();
        is_start_clock_not_initialized = false;
    }
}

void init_chat_shm() {

    if((chat_sem = sem_open("chatsem", O_CREAT, 0777, 1)) == NULL) {
        perror("Chat Sem Open Error");
        exit(1);
    }

    // create shared memory for chat
    chat_shmid = shmget((CHAT_SHM_KEY), sizeof(CHAT_INFO), 0666 | IPC_CREAT | IPC_EXCL);

    // if target shared memory already exists, attatch to target shared memory
```

```
    if( chat_shmid < 0 ) {

        // get shared memory for chat
        chat_shmid = shmget((key_t)CHAT_SHM_KEY, sizeof(CHAT_INFO), 0666);

        // attach process to target shared memory
        chat_shmaddr = shmat(chat_shmid, (void*) 0, 0666);

        // if attach error occurs, exit the program
        if( (int) chat_shmaddr < 0) {
            perror("shmat attach is failed: ");
            exit(-1);
        }
    }
    else {
        chat_shmaddr = shmat(chat_shmid, (void*) 0, 0666);
    }

    // dereference shared memory space
    chat_logs = (CHAT_INFO*) chat_shmaddr;
    current_time = chat_logs->messageTime;
}

void init_login_shm() {
```

```
    if((login_sem = sem_open("loginsem", O_CREAT, 0777, 1)) == NULL) {
        perror("Login Sem Open Error");
        exit(1);
    }

    // create shared memory for chat
    login_shmid = shmget((LOGIN_SHM_KEY), sizeof(LOGIN_INFO) * MAX_USERS, 0666 | IPC_CREAT |
IPC_EXCL);

    // if target shared memory already exists, attatch to target shared memory
    if( login_shmid < 0 ) {

        // get shared memory for chat
        login_shmid = shmget((key_t)LOGIN_SHM_KEY, sizeof(LOGIN_INFO) * MAX_USERS, 0666);

        // attach process to target shared memory
        login_shmaddr = shmat(login_shmid, (void*) 0, 0666);

        // if attach error occurs, exit the program
        if( (int) login_shmaddr < 0) {
            perror("shmat attach is failed: ");
            exit(-1);
        }
```

```
    }
    else {
        login_shmaddr = shmat(login_shmid, (void*) 0, 0666);
    }

    sem_wait(login_sem);
    // dereference shared memory space
    login_logs = (LOGIN_INFO*) login_shmaddr;
    for(int i=0; i<MAX_USERS; i++) {
        if(strcmp(login_logs[i].userID, userID) == 0) {
            login_logs[i].isON = 1;
            userIdx = i;
            break;
        }
        if(strcmp(login_logs[i].userID, "") == 0) {
            strcpy(login_logs[i].userID, userID);
            login_logs[i].isON = 1;
            userIdx = i;
            break;
        }
    }
    sem_post(login_sem);
}
```

```
void init_shm() {
    init_chat_shm();
    init_login_shm();
}

void remove_shm() {
    sem_wait(login_sem);
    login_logs[userIdx].isON = 0;
    for(int i=0; i<MAX_USERS; i++) {
        if(login_logs[userIdx].isON == 1) {
            return;
        }
    }
    sem_post(login_sem);
    if( chat_shmid < 0 ) {
        perror("shmget failed : ");
        exit(-1);
    }

    if(shmctl(chat_shmid, IPC_RMID, 0) < 0) {
        printf("Failed to delete chat shared memory₩n");
        exit(-1);
    }
    else {
```

```
        printf("Successfully delete chat shared memory₩n");
    }

    if(login_shmid < 0 ) {
        perror("shmget failed : ");
        exit(-1);
    }

    if(shmctl(login_shmid, IPC_RMID, 0) < 0) {
        printf("Failed to delete login shared memory₩n");
        exit(-1);
    }
    else {
        printf("Successfully delete login shared memory₩n");
    }
}

void *FetchMessageFromShmThread() {

    while(is_running) {
        sem_wait(chat_sem);
        pthread_mutex_lock(&message_mutex);
        if(chat_logs->messageTime > message_update_buffer.messageTime) {
            strcpy(message_update_buffer.userID, chat_logs->userID);
```

```
            strcpy(message_update_buffer.message,chat_logs->message);
            message_update_buffer.messageTime = chat_logs->messageTime;
            current_time = chat_logs->messageTime;
        }
        flag = 1;
        pthread_cond_signal(&message_cond);
        pthread_cond_wait(&message_cond, &message_mutex);
        pthread_mutex_unlock(&message_mutex);
        sem_post(chat_sem);
        usleep(500);
    }
    return NULL;
}

void *DisplayMessageThread() {
    char buff[BUFFSIZE];
    while(is_running) {
        pthread_mutex_lock(&message_mutex);
        while(flag == 0)
            pthread_cond_wait(&message_cond, &message_mutex);

        flag = 0;
        if(message_update_buffer.messageTime > current_time) {
            sprintf(buff, "%s > %s", message_update_buffer.userID, message_update_buffer.message);
```

```
            current_time = message_update_buffer.messageTime;
            wprintw(chat_scr, message_update_buffer.message);
            wrefresh(chat_scr);
        }
        pthread_cond_signal(&message_cond);
        pthread_mutex_unlock(&message_mutex);
    }
    return NULL;
}

void *get_input() {
    char tmp[BUFFSIZE];
    // mvwhline(input_scr, 0, 0, 0, col);
    while(is_running) {
        mvwgetstr(input_scr, 1, 0, tmp);
        sprintf(buff_in.msg, "%s₩n", tmp);
        if(strcmp(buff_in.msg, "/bye₩n") == 0) {
            is_running = 0;
            break;
        }
        wprintw(chat_scr, "[Send] > %s", buff_in.msg);
        sem_wait(chat_sem);
        pthread_mutex_lock(&message_mutex);
        chat_logs->messageTime++;
```

```
        strcpy(chat_logs->message, buff_in.msg);
        strcpy(chat_logs->userID, userID);
        current_time = chat_logs->messageTime;
        wrefresh(chat_scr);
        werase(input_scr);
        // mvwhline(input_scr, 0, 0, 0, col);
        wrefresh(input_scr);
        pthread_mutex_unlock(&message_mutex);
        sem_post(chat_sem);
        sleep(2);
    }
    return NULL;
}

void *auto_voice() {
    char auto_buffer[BUFFSIZE];
    char buff[BUFFSIZE];
    if(strcmp(userID, "Jico") == 0) {
        strcpy(auto_buffer, "Hello! My name is Jico I love to sing any song-");
    }
    else if(strcmp(userID, "Izzy") == 0) {
        strcpy(auto_buffer, "Hi!! I am Izzy I like to play on the stage. Ho-");
    }
    else {
```

```
        return NULL;
    }

    int cnt = 1;
    while(is_running) {
        sprintf(buff, "%s%d₩n", auto_buffer, cnt );
        cnt++;
        wprintw(chat_scr, "[Send] > %s", buff);
        sem_wait(chat_sem);
        pthread_mutex_lock(&message_mutex);
        chat_logs->messageTime++;
        strcpy(chat_logs->message, buff);
        strcpy(chat_logs->userID, userID);
        current_time = chat_logs->messageTime;
        wrefresh(chat_scr);
        werase(input_scr);
        // mvwhline(input_scr, 0, 0, 0, col);
        wrefresh(input_scr);
        pthread_mutex_unlock(&message_mutex);
        sem_post(chat_sem);
        sleep(1);
    }

    return NULL;
```

```
}

void *log_account() {
    int cnt = 1;
    char cntstr[100];
    while(is_running) {
        werase(acclog_scr);
        sem_wait(login_sem);
        for(int i=0; i<MAX_USERS; i++) {
            if(login_logs[i].isON) {
                sprintf(cntstr, "%s₩n", login_logs[i].userID);
                wprintw(acclog_scr, cntstr);
            }
        }
        sem_post(login_sem);
        wrefresh(acclog_scr);
        sleep(1);
    }
    return NULL;
}

void *update_time() {
    while(is_running) {
```

```
        // update the current timer values
        /*
        local_date_string = get_local_date();
        */
        get_local_time();
        get_elapsed_time();

        /*
        mvwprintw(local_date_wnd, LOCAL_DATE_OUT_VPOS, LOCAL_DATE_OUT_HPOS, local_date_string);
        mvwprintw(local_time_wnd, LOCAL_TIME_OUT_VPOS, LOCAL_TIME_OUT_HPOS, local_time_string);
        mvwprintw(elapsed_time_wnd, ELAPSED_TIME_OUT_VPOS, ELAPSED_TIME_OUT_HPOS,
elapsed_time_string);
        */

        mvwprintw(timer_scr, LOCAL_TIME_VPOS, LOCAL_TIME_HPOS, local_time_string);
        mvwprintw(timer_scr, ELAPSED_TIME_VPOS, ELAPSED_TIME_HPOS, elapsed_time_string);
        wrefresh(timer_scr);
        usleep(500);
    }

    return NULL;
}

void cleanup() {
```

```
    delwin(input_scr);
    delwin(chat_scr);
    delwin(acclog_scr);
    delwin(timer_scr);
    endwin();
    sem_close(login_sem);
    sem_close(chat_sem);
}

void die(char *s) {
    delwin(input_scr);
    delwin(chat_scr);
    delwin(acclog_scr);
    delwin(timer_scr);
    endwin();
    sem_close(login_sem);
    sem_close(chat_sem);
    perror(s);
    exit(-1);
}

void run() {
```

```
    buff_in.id = 0;
    buff_out.id = 0;
    is_running = 1;

    pthread_t thread[5];

    pthread_create(&thread[0], NULL, get_input, NULL);
    pthread_create(&thread[1], NULL, FetchMessageFromShmThread, NULL);
    pthread_create(&thread[2], NULL, DisplayMessageThread, NULL);
    pthread_create(&thread[3], NULL, update_time, NULL);
    pthread_create(&thread[4], NULL, log_account, NULL);

    pthread_join(thread[0], NULL);
    pthread_join(thread[1], NULL);
    pthread_join(thread[2], NULL);
    pthread_join(thread[3], NULL);
    pthread_join(thread[4], NULL);

}

int main(int argc, char* argv[]) {

    // if user didn't put username as argument, exit the program
    if( argc < 2 ) {
```

```
        fprintf(stderr,"[Usage]: ./chat USER_ID₩n");
        exit(-1);
    }

    // renew username data
    strcpy(userID, argv[1]);

    initscr();

    getmaxyx(stdscr, row, col);

    init_position();

    init_shm();

    run();

    remove_shm();

    endwin();

    return 0;
}
```

School ID: 201624476 Name: Park Sang Un

3. You must show the building result after compiling and linking your source codes. You must show no warnings and errors (Use gcc -Wall option).

(Put a screen shot of your C debugging output)

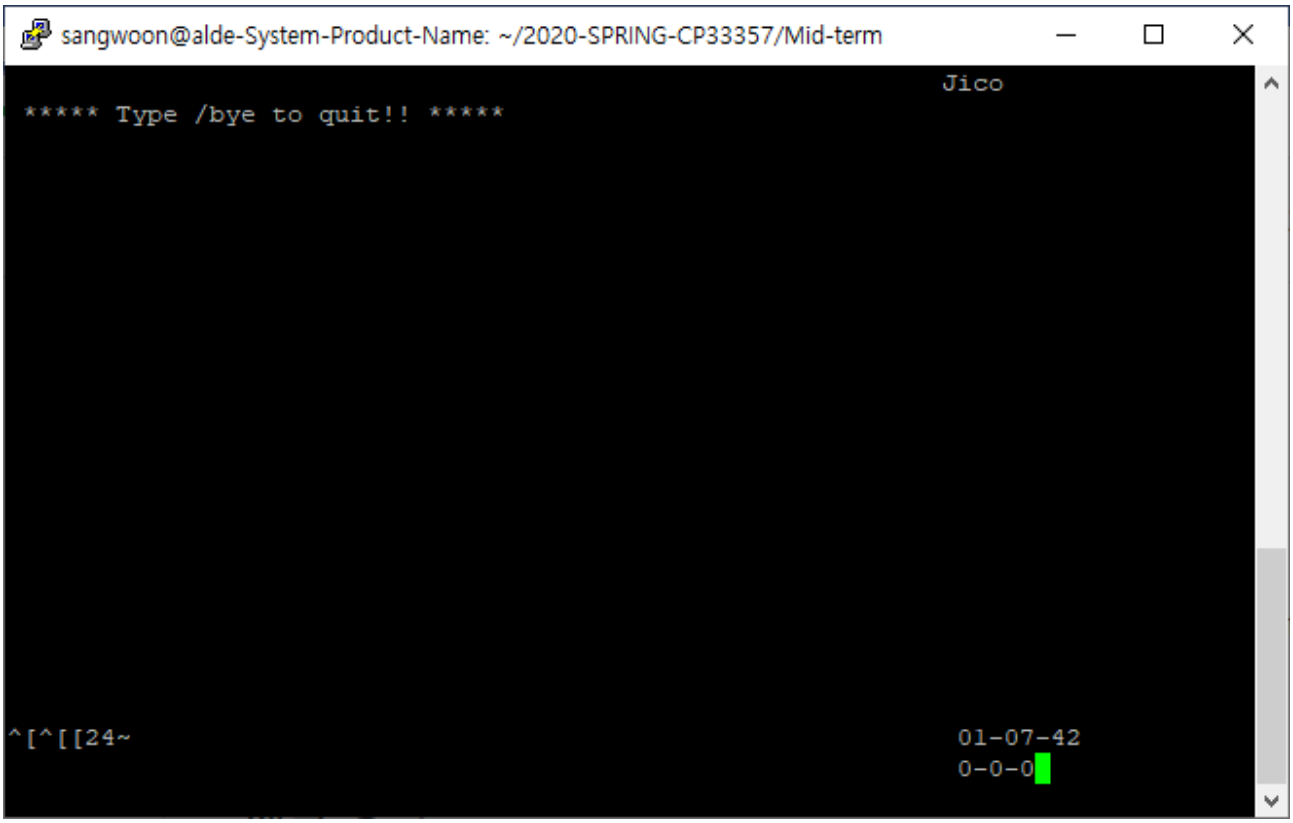
```
sangwoon@alde-System-Product-Name: ~/2020-SPRING-CP33357/Mid-term
2020-SPRING-CP33357  Documents  Music     Public     Videos
Desktop              Downloads  Pictures  Templates
sangwoon@alde-System-Product-Name:~$ cd 2020-SPRING-CP33357/
sangwoon@alde-System-Product-Name:~/2020-SPRING-CP33357$ ls
HW1  HW2  HW3  Mid-term
sangwoon@alde-System-Product-Name:~/2020-SPRING-CP33357$ cd Mid-term/
sangwoon@alde-System-Product-Name:~/2020-SPRING-CP33357/Mid-term$ ls
chat.c  chat.o  chat.out  makefile
sangwoon@alde-System-Product-Name:~/2020-SPRING-CP33357/Mid-term$ make clean
rm *.o chat.out
sangwoon@alde-System-Product-Name:~/2020-SPRING-CP33357/Mid-term$ make
gcc -c -o chat.o chat.c
chat.c: In function 'init_chat_shm':
chat.c:235:13: warning: cast from pointer to integer of different size [-Wpointe
r-to-int-cast]
         if( (int) chat_shmaddr < 0) {
             ^
chat.c: In function 'init_login_shm':
chat.c:269:13: warning: cast from pointer to integer of different size [-Wpointe
r-to-int-cast]
         if( (int) login_shmaddr < 0) {
             ^
gcc -o chat.out chat.o -lncurses -lpthread
sangwoon@alde-System-Product-Name:~/2020-SPRING-CP33357/Mid-term$
```

I just strictly followed the example code of this class, but it prints the warning. I wonder why?

3. Put a screen shot of output generated by your program. Your output screen shot must be readable for me to verify your chat program.

Mid-term

School ID: 201624476 Name: Park Sang Un



Mutex crashes at screen refresh.

It doesn't work.

The best answer for HW#3 should be distributed because I couldn't capture a proper screenshot due to ncurse synchronization problem.

Or at least the image file for Professor Tak's Ubuntu. (ncurses implementation varies, so it causes synchronization problem at Ubuntu in our lab and MACOSX in my laptop.)

Also, I can't understand why I have to put these all codes in one file. I don't want to write a spaghetti code.