**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();**

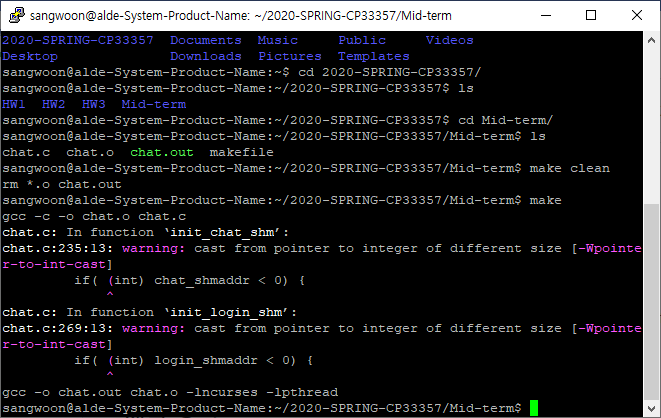
**endwin();**

**return 0;**

**}**

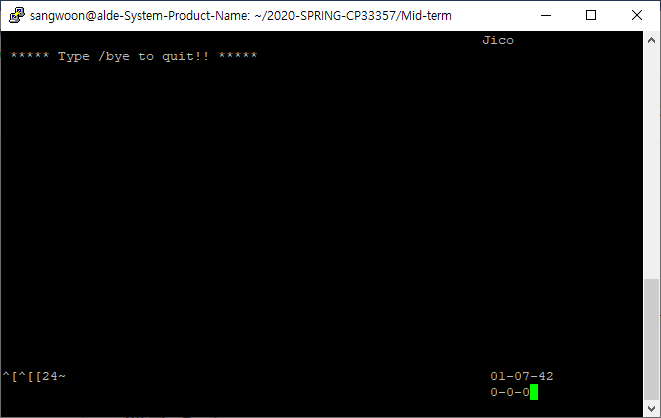
**3. You must show the building result after compiling and linking your source codes. You must show no warnings and errors (Use gcc -Wall option).**

**(Put a screen shot of your C debugging output)**



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.**



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.