```
// GROUP D
// By Corey Green, Robby Hallock, and Kyle McCullough
// decoreyon.green@okstate.edu
// robert.hallock@okstate.edu
// kymccul@okstate.edu
// CS 4323
// finalGroupProject
// 4-26-22

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <semaphore.h>
#include <pthread.h>
#include <sys/types.h>
#include <time.h>
#include "Robby.h"
#include "Corey.h"
#include "Kyle.h"

/*
INPUTS:
# of medical pros.
# of total patients
# patient capaity
# sofa space
# max enter time interval
# patient checkup time
*/
```

```
// allow command line args
int main(int argc, char *argv[])
{
   // Initialize variables
   remainingTasks = 0;
   left = 0;
   buffer = -1;

   srand(time(NULL)); // seed random number generator
   int medicalStaff, totalPatients, roomCapacity, sofaSpace, maxTimeInterval;

   // assigns the arguments to ints
   medicalStaff = atoi(argv[1]);
   totalPatients = atoi(argv[2]);
   roomCapacity = atoi(argv[3]);
   sofaSpace = atoi(argv[4]);
   maxTimeInterval = atoi(argv[5]);
   checkupTime = atoi(argv[6]);

   // sets global variables
   totalRoomCapacity = roomCapacity;
   totalSofaCapacity = sofaSpace;
   maxSofaCapacity = sofaSpace;
   maxPatients = totalPatients;

   // Initialize mutexes
```

```
for (int i = 0; i < 12; i++)
    pthread_mutex_init(&mutex[i], NULL);

// Lock some mutexes for later
pthread_mutex_lock(&mutex[2]);
pthread_mutex_lock(&mutex[4]);
pthread_mutex_lock(&mutex[5]);
pthread_mutex_lock(&mutex[6]);

// Initialize
sem_init(&count[0], 0, 0);
sem_init(&count[1], 0, 0);

// Threads
pthread_t threads[medicalStaff+totalPatients];

// struct arrays for each person type
struct threadStruct contents[totalPatients];
struct threadStruct contentsM[medicalStaff];

// create threads for thread pool
for (int i = 0; i < medicalStaff+totalPatients; i++){
    if (pthread_create(&threads[i], NULL, &mainThreadLoop, NULL)){
        printf("Failed to create thread\n");
    }
}

// create medical staff
for (int i = 0; i < medicalStaff; i++){
```

```
    contentsM[i].id = i;
    contentsM[i].occupation = "Staff";
    struct task t = {
        .selector = 0,
        .args = &contentsM[i],
    };
    queueTask(t);
}

// create patients
for (int i = 0; i < totalPatients; i++){
    int ms = (rand() % maxTimeInterval) + 1;
    contents[i].id = i;
    contents[i].occupation = "Patient";
    struct task t = {
        .selector = 1,
        .args = &contents[i],
    };
    queueTask(t);
    usleep(ms * 1000);
}

// waits for all patients to finish
while(1){
    pthread_mutex_lock(&mutex[9]);
    if (left >= totalPatients){
        pthread_mutex_unlock(&mutex[9]);
        break;
    }
```

```
    pthread_mutex_unlock(&mutex[9]);
}

// signal medical staff to finish
for (int i = 0; i < medicalStaff; i++){
    sem_post(&count[0]);
}

// stop all threads
for (int i = 0; i < totalPatients+medicalStaff; i++){
    struct task t = {
        .selector = 2
    };
    queueTask(t);
}

// join threads
for (int i = 0; i < medicalStaff+totalPatients; i++){
    if (pthread_join(threads[i], NULL)){
        printf("Failed to join thread\n");
    }
}



// print summary
summary.patientsAvgWaitTime = summary.patientsAvgWaitTime/summary.successfulCheckups;
summary.medicalProAvgWaitTime = summary.medicalProAvgWaitTime/medicalStaff;
printf("Statistical Summary:\n");
printf("--------------------------------------------------------------------------------\n");
```

```
    printf("Number of successful checkups: %d \n", summary.successfulCheckups);
    printf("Average waiting time for Medical Professionals: %ldms \n",
summary.medicalProAvgWaitTime);
    printf("Number of Patients that left: %d \n", summary.patientsThatLeft);
    printf("Average wait time for patients: %ldms \n", summary.patientsAvgWaitTime);

    // destroy all mutexes
    for (int i = 0; i < 12; i++){
        pthread_mutex_destroy(&mutex[i]);
    }

    // destroy all semaphores
    for (int i = 0; i < 2; i++){
        sem_destroy(&count[i]);
    }

    return 0;
}

/*
Reference:

Title: Thread Pools with function pointers in C
Author: codeVault
Source Code: https://code-vault.net/lesson/w1h356t5vg:1610029047572

*/
```