Computer Science 159.335 - Operating Systems & Concurrent Programming
Assignment 2 Due 14th Sept 2015

Lecturer: Dr. Martin Johnson
Telephone: 4140800 ext 43142
Office: INMS 3.24
Email: M.J.Johnson@massey.ac.nz

Introduction

The purpose of this assignment is to use semaphores to finish a lift simulator program.
Available on the stream is an unfinished program called a2-2015.c. It simulates a number of lifts in a large building.
The building is described by the following data structure:

```
typedef struct {
   int waitingtogoup;    /* the number of people waiting to go up */
   int waitingtogodown;  /* the number of people waiting to go down */
   semaphore up_arrow;   /* people going up wait on this */
   semaphore down_arrow; /* people going down wait on this */
} Floor_info;

Floor_info floor[NFLOORS];
```

Each floor has two semaphores that people can wait upon, up_arrow and down_arrow.
A lift is described by the following data structure:

```
typedef struct {
   int no;               /* which lift is it */
   int position;         /* which floor it is on */
   int direction;        /* going UP or DOWN */
   int peopleinlift;     /* number of people in the lift */
   int stops[NFLOORS];   /* for each stop how many people are waiting to get off */
   semaphore stopsem[NFLOORS]; /* people in the lift wait on one of these */
} Lift_info;
```

Each lift contains a semaphore for each floor that people inside can wait upon.
The algorithm for a person is:

```
while(1) {
   wait for a while
   pick a different floor to go to
   if going up
      press up arrow button and wait
   otherwise
      press down arrow button and wait
   get into lift
   press button for floor to go to and wait
   get out of lift
}
```

The algorithm for a lift is:

```
 while(1) {
     drop off all passengers waiting to get out at this floor
     if going up or empty
         if empty
            direction=up
         pick up all passengers on this floor waiting to go up
     if going down or empty
         if empty
            direction=down
         pick up all passengers on this floor waiting to go down
     move lift
     if on top or ground floors
         change direction
  }
```

You need to complete the program by adding semaphore calls to communicate between threads and also for mutual exclusion.

The program creates a thread for each lift and for a fixed number of people. Lifts go up and down, picking up people and taking them where they want to go.

A person starts on a certain floor ('from' in the code) and thinks of a random floor to travel to ('to' in the code) . They then wait for a lift (going in the correct direction) by waiting on a semaphore ('up_arrow' or 'down_arrow') associated with the 'from' floor. When the lift signals this semaphore, the person gets in and waits for the 'to' floor to be reached (by waiting on another semaphore). When the 'to' floor has been reached, they get out, wait a while and start another journey.

There are two semaphores on each floor, one for people going down and one for people going up (up_arrow and down_arrow). In every lift there is a semaphore for each floor (stopsem[]). The people in the lift wait on these before getting out.

The lines marked with --- at the start need completing, most of these just require wait or signal calls. You will need to use a global variable (of type Lift_info *) to pass a pointer to a lift from the lift thread to the person thread.

When you have correctly completed the marked lines, the program should work OK, but every now and then the screen will not be drawn properly (the lifts will write over the building) . This is happening because the threads are all trying to draw to the screen at the same time.

Add a mutual exclusion semaphore to the code to remove this problem.

Change LIFTSPEED, GETINSPEED and GETOUTSPEED to 10.
It should work reasonably well for a while but eventually the lifts will stop picking up people and the program may even crash. This is happening because the threads use shared data.

Change LIFTSPEED, GETINSPEED and GETOUTSPEED to 0 and PEOPLESPEED to 1, this will make things go as fast a possible.
Now the program will be even more unreliable.

Add mutual exclusion semaphores to the code so that the program works reliably at full speed. Think about everywhere in the program that shared data is accessed.

Submit your assignment electronically using Stream. Your submission must be a single source file written in ANSI C. Please do not include any extra header files. Your program should include comments as documentation. Do not open any external data files. Set the SPEEDs to maximum for submission. Your program must compile using the GCC compiler.

See Stream for instructions on how to compile and run the code on a Unix machine.

This assignment will be worth 10% of the complete paper.

Marks will be awarded for correct use of semaphores.

Marks will be subtracted for obvious copying and for late submissions.
