ISP Report

Formative Assessment

Group 10

- ❏ Ephraim Awinzoya Adongo
- ❏ Given Edward
- ❏ Joyce Njeri Miiri
- ❏ Zubery Ayyub Msemo
- ❏ Christine Wasike

8th November, 2020

MEETINGS

Project Planning

- ❏ Date: Friday, 27th November, 2020
- ❏ Time: 4:00pm - 5:00pm (EAT)
- ❏ Venue: Zoom
- ❏ Attendees: Everyone

Recording

First meeting

Team Lead

Through the meeting, we worked with equal responsibilities and mutual understanding so we opted out the Team Lead role.

Problem selection

We started with discussing our first impressions of the two questions.

Ephraim felt he'd have an easier time implementing question 2 since he had previously worked on a similar problem in Java.

Joyce found the workflow of a stopwatch easier to implement since it requires a running loop with functions to start, pause, continue, or stop and reset the stopwatch. Seemed pretty straightforward.

Christine led us in going through the instructions.

For question 1, Joyce and Zubeir expressed concerns with working with processes and threads from previous assignments, and felt this was more of a concept question that required firm knowledge in implementing signals, threads and processes. We explored theories on using signals to capture keyboard inputs, or otherwise a c function to capturing keyboard inputs. In the end. Christine advised to choose question 1 if we wanted to challenge our understanding of these topics.

We spent more time in question 2. Everyone was deeply interested in understanding the requirements of the question in relation to the normal functioning of an elevator as we know it. It was agreed that though some concepts were vague, we would accommodate what was excluded from the requirements after we had implemented the basic functioning of an elevator, and include additional functioning such as the elevator accommodating more than 10 people afterwards, and including it in the readMe file.

Edward Given joined the call just in time to join in selecting the question we would work on.

We finally chose to work on the second question because:

- It was easier for us to ask more questions regarding how the simulation would work alongside others
- We felt everyone would have solid inputs to attaining full functionality of the program.
- Question 1 felt restrictive in its implementation, group members would have streamlined line of thought, as opposed to question 2 where its implementation felt more flexible

Scheduling

We agreed to have a live coding session for at most 2 hours in the morning hours which we felt would help us work faster, even if we didn't have a running code at the end, we would have an easy to follow framework.

Live Coding Session 1

- ❏ Date: Saturday, 28th November, 2020
- ❏ Time: 9:00am - 11:00am (EAT)
- ❏ Venue: Zoom
- ❏ Attendees: Zubeir, Christine, Ephraim & Joyce.

Recording

Second meeting(a)

Second meeting(b)

Code Collaboration

Github - ISP-Group-Work

Brainstorming

We first mapped out the requirements and scope of the problem in a text file.

Design

We wanted to implement the basic functionalities of an elevator first before diving into threading by assigning random floor numbers to 10 people, sorting the floor numbers in ascending order, then looping through floor numbers while dropping off people at their destinations.

Roadblocks

We realized there were inconsistencies in terms of the synchronization between the loops for identifying people's destinations and the one for identifying which floor we were currently on. This made us realize that threading would greatly improve the accuracy in when a person arrived at their destination and when it was time for the elevator to proceed to the next stop.

We also somewhat got confused while tracking different variables and their values but managed to fix it in the end, especially through using intuitive naming conventions.

Live Coding Session 2

- ❏ Date: Monday, 30th November, 2020
- ❏ Time: 9:00am - 10:30am (EAT)
- ❏ Venue: Zoom
- ❏ Attendees: Zubeir, Christine, & Joyce.

Implementation

We brainstormed how to implement threading in our program. The session was mostly spent doing research and trying out different applications of multithreading in our program. Though we didn’t have a working code at the end, we had mapped out the process steps we’d take to implement the threads for the next coding session.

Live Coding Session 3

- ❏ Date: Sunday, 6th December, 2020
- ❏ Time: 3:00am - 5:00am (EAT)
- ❏ Venue: Zoom
- ❏ Attendees: Ephraim, & Joyce.

Implementation

We implemented threading of the people and elevator movements through creating ten threads with individual thread ids for the ten people, and 1 thread for the elevator. We also used mutex to synchronize thread activities, and printed the outcomes. In the end, we had a working code that simulated an elevator, though the elevator ran ten times, while we wanted the elevator to start at floor 1, load and offload people, and end simulation at floor 8.

Live Coding Session 4

- ❏ Date: Tuesday, 8th December, 2020
- ❏ Time: 9:00am - 11:00am (EAT)
- ❏ Venue: Zoom
- ❏ Attendees: Christine, & Joyce.

Final Implementation

We used two functions:

- main()
- elevator()
- cmpfunc()

Defined constants:

MAX_PEOPLE which we defined as ten, the maximum capacity an elevator could hold at a time.

MAX_FLOORS defining it as eight to signify the eight floors that the building had.

Important Global variables:

elevator_state - An integer variable: -1 for stop, 0 for moving up, 1 for moving down

In the main():

Declaration of an error integer that would later be assigned to the return value from creating our passenger threads.

Initialization of the mutex and passing in the pthread mutex variable we declared globally

Creating of the elevator and passenger threads.

Joining all the threads, destroying the mutex lock then finally exiting running a pthread_exit() without error.

In the elevator():

Setting the mutex lock and passing the lock global variable

Generating the random floor numbers

Initializing the elevator state to stop and setting the floor number to 1

Assigning the random floor destinations for the passengers and storing these values in an array that hold 10 digits ranging from 2 - 8.

Simulating the elevator movement by running a loop for as long as the floor destination is not greater than the eighth.

Printing the elevators current floor to simulate motion then setting the elevator state to in motion.

Looping through each passenger destination and running a check on whether the passenger should alight or not.

Within this check, we print the passengers corresponding thread id as well as them alighting at their designated destination.

Outside the inner loop, the elevator state is changed to moving as the out loop proceeds to check the next floor after incrementing the current floor number by 1.

Once the out loop(while loop running through 2nd - 8th) the elevator state changes to stop and a print statement indicating the end of the simulation for moving up follows.

Repeat this process for the elevator moving down. We first check if all the 10 people have arrived at their destination then we start moving down with new random destinations for the passengers who have alighted.

The logic we used here is that even if a passenger has gotten off a previous floor and needs to go down, he/she must wait for the elevator to reach the top and load off all its 10 passengers before it can come down to get them and take them to the lower floors. Either way, the elevator is the control and thus passengers' new destination floors are updated when the elevator has reached its destination and is starting to move down or up again.

We realize the elevator can hold a maximum of 10 people, but we did not include taking up extra people from consecutive floors because we are working with 10 threads for 10 people already created before, who should have ideally already been dropped off as the elevator is moving up/down.

User needs to press 'CTRL+C' to end the simulation.

We run the pthread_exit to exit all threads

Individual Contributions

The workload was mostly done by Joyce, Christine and Ephraim.