CS 4348.006 Project 2 Design Zach Tang

List of every semaphore

printMutex = 1

printMutex has an initial value of 1. It exists so threads can have full control over the output buffer when they need to print something. Therefore, output between different functions is not interleaved due to switching between threads.

start = 0

start has an initial value of 0. It exists to block threads from starting prematurely. The threads are only allowed to start after all people in the simulation (threads) have been initialized successfully.

created = 0

created has an initial value of 0. It exists to block the main thread from continuing to execute until all people in the simulation (threads) have been initialized successfully.

roomCountMutex = 1

roomCountMutex has an initial value of 1. It exists to ensure single access to the roomCount global variable for assigning rooms to guests. Therefore, no 2 guests will get the same room assigned by different front desk employees.

peopleCountMutex = 1

peopleCountMutex has an initial value of 1. It exists to ensure single access to the peopleCount global variable for tracking the people successfully initialized in the simulation. Therefore, only one thread can update the peopleCount variable at a time so that the value is accurate at all times to determine when all the people have been successfully initialized.

guestDeskQueueMutex = 1

guestDeskQueueMutex has an initial value of 1. It exists to ensure single access to the guestDeskQueue for front desk employees to serve guests from. Therefore, a guest will only ever get served by a single front desk employee.

guestBellhopQueueMutex = 1

guestBellhopQueueMutex has an initial value of 1. It exists to ensure single access to the guestBellhopQueue for bellhop employees to serve guests from. Therefore, a guest will only ever get served by a single bellhop employee.

guestDeskReady = 0

guestDeskReady has an initial value of 0. It exists to ensure that a desk employee tries to serve a guest only when the guest is ready in the guestDeskQueue. Otherwise, the desk employee would be serving air.

guestBellhopReady = 0

guestBellhopReady has an initial value of 0. It exists to ensure that a bellhop employee tries to serve a guest only when the guest is ready in the guestBellhopQueue. Otherwise, the bellhop employee would be serving air.

guestAssigned[25] = {0}

guestAssigned is an array of 25 semaphores each with an initial value of 0. It's 25 semaphores because that is the maximum number of guests possible in the simulation. These semaphores each correspond to a unique guest.

They exist to ensure the guest waits to be assigned a room and to receive a key from the front desk employee before leaving.

guestTaken[25] = {0}

guestTaken is an array of 25 semaphores each with an initial value of 0. It's 25 semaphores because that is the maximum number of guests possible in the simulation. These semaphores each correspond to a unique guest. They exist to ensure the guest waits for their bags to be taken by the bellhop employee before leaving to their room.

guestDelivered[25] = {0}

guestDelivered is an array of 25 semaphores each with an initial value of 0. It's 25 semaphores because that is the maximum number of guests possible in the simulation. These semaphores each correspond to a unique guest. They exist to ensure the guest waits for their bags to be delivered by the bellhop employee before retiring for the evening.

desk[2] = {0}

desk is an array of 2 semaphores each with an initial value of 0. It's 2 semaphores because that is the number of front desk employees in the simulation. These semaphores each correspond to a unique front desk employee. They exist to ensure the front desk employee waits for the guest to leave the front desk before serving the next available guest.

deskEmployees = 2

deskEmployees has an initial value of 2. It exists to ensure that the guest only queues himself in the guestDeskQueue when a front desk employee is available to serve him.

tip[2] = {0}

tip is an array of 2 semaphores each with an initial value of 0. It's 2 semaphores because that is the number of front desk employees in the simulation. These semaphores each correspond to a unique bellhop employee. They exist to ensure the bellhop employee waits for the guest to tip before leaving the room to serve other guests.

bellhopEmployees = 2

bellhopEmployees has an initial value of 2. It exists to ensure the guest only queues himself in the guestBellhopQueue when a bellhop employee is available to serve him.

Pseudocode

Semaphores and global variables

```
1
2  int MAX_NUM_GUESTS = 25;
3  int NUM_DESK_EMPLOYEES = 2;
4  int NUM_BELLHOP_EMPLOYEES = 2;
5
6  int NUM_GUESTS;
7
8  int peopleCount = 0;
9  int roomCount = 1;
10
11 int guestDeskEmployeeNumber[MAX_NUM_GUESTS];
12 int guestBellhopEmployeeNumber[MAX_NUM_GUESTS];
13 int guestRoom[MAX_NUM_GUESTS];
14
15 queue guestDeskQueue;
16 queue guestBellhopQueue;
17
18 semaphore guestDeskQueueMutex = 1;
19 semaphore guestBellhopQueueMutex = 1;
20 semaphore guestDeskReady = 0;
21 semaphore guestBellhopReady = 0;
22 semaphore guestAssigned[MAX_NUM_GUESTS] = {0};
23 semaphore guestTaken[MAX_NUM_GUESTS] = {0};
24 semaphore guestDelivered[MAX_NUM_GUESTS] = {0};
25
26 semaphore desk[NUM_DESK_EMPLOYEES] = {0};
27 semaphore deskEmployees = 2;
28
29 semaphore tip[NUM_BELLHOP_EMPLOYEES] = {0};
30 semaphore bellhopEmployees = 2;
31
32 semaphore printMutex = 1;
33 semaphore start = 0;
34 semaphore created = 0;
35 semaphore roomCountMutex = 1;
36 semaphore peopleCountMutex = 1;
```

Main functions

```
void guest(int guestNumber) {
    int bags = rand(0, 5) // random number between 0 and 5

    createPerson("Guest", guestNumber);

    enterHotel(guestNumber, bags);

    semWait(deskEmployees);

    enqueue(guestDeskQueue, guestNumber, guestDeskQueueMutex, guestDeskReady);

    semWait(guestAssigned[guestNumber]);

    receiveKey(guestNumber);

    semSignal(desk[guestDeskEmployeeNumber[guestNumber]]);

    if (bags > 2) {
        requestHelp(guestNumber);

        semWait(bellhopEmployees);

        enqueue(guestBellhopQueue, guestNumber, guestBellhopQueueMutex, guestBellhopReady);

        semWait(guestTaken[guestNumber]);
    }

    enterRoom(guestNumber);

    if (bags > 2) {
        semWait(guestDelivered[guestNumber]);

        receiveBags(guestNumber);

        semSignal(tip[guestBellhopEmployeeNumber[guestNumber]]);
    }

    retire(guestNumber);
}
```

```
101 void bellhopEmployee(int bellhopEmployeeNumber) {
102     int guestNumber;
103 
104     while (true) {
105         semWait(guestBellhopReady);
106 
107         dequeue(guestBellhopQueue, guestNumber, guestBellhopQueueMutex);
108 
109         guestBellhopEmployeeNumber[guestNumber] = bellhopEmployeeNumber;
110 
111         takeBags(bellhopEmployeeNumber, guestNumber);
112 
113         semSignal(guestTaken[guestNumber]);
114 
115         semSignal(guestDelivered[guestNumber]);
116 
117         semWait(tip[bellhopEmployeeNumber]);
118 
119         semSignal(bellhopEmployees);
120     }
121 }
```

```
79 void deskEmployee(int deskEmployeeNumber) {
80     int guestNumber;
81 
82     createPerson("Front desk employee", deskEmployeeNumber);
83 
84     while (true) {
85         sem_wait(guestDeskReady);
86 
87         dequeue(guestDeskQueue, guestNumber, guestDeskQueueMutex);
88 
89         guestDeskEmployeeNumber[guestNumber] = deskEmployeeNumber;
90 
91         assignRoom(deskEmployeeNumber, guestNumber);
92 
93         semSignal(guestAssigned[guestNumber]);
94 
95         semWait(desk[deskEmployeeNumber]);
96 
97         semSignal(deskEmployees);
98     }
99 }
```

```
123 void main(int numGuests) {
124     print("Simulation starts");
125     NUM_GUESTS = numGuests;
126 
127     thread deskEmployeeThreads[NUM_DESK_EMPLOYEES];
128     thread bellhopEmployeeThreads[NUM_BELLHOP_EMPLOYEES];
129     thread guestThreads[NUM_GUESTS];
130 
131     for (int i = 0; i < NUM_DESK_EMPLOYEES; i++) {
132         create_thread(deskEmployeeThreads[i], deskEmployee, i);
133     }
134     for (int i = 0; i < NUM_BELLHOP_EMPLOYEES; i++) {
135         create_thread(bellhopEmployeeThreads[i], bellhopEmployee, i);
136     }
137     for (int i = 0; i < NUM_GUESTS; i++) {
138         create_thread(guestThreads[i], guest, i);
139     }
140 
141     semWait(created);
142 
143     for (int i = 0; i < (NUM_GUESTS + NUM_DESK_EMPLOYEES + NUM_BELLHOP_EMPLOYEES); i++) {
144         semSignal(start);
145     }
146 
147     for (int i = 0; i < NUM_GUESTS; i++) {
148         join_thread(guestThreads[i]);
149     }
150 
151     print("Simulation ends");
152 }
```

Helper functions

```
172 void enqueue(queue q, int guestNumber, semaphore queueMutex, semaphore readyMutex) {
173     semWait(queueMutex);
174     q.push(guestNumber);
175     semSignal(readyMutex);
176     semSignal(queueMutex);
177 }
```

```
179 void dequeue(queue q, int guestNumber, semaphore queueMutex) {
180     semWait(queueMutex);
181     guestNumber = q.front();
182     q.pop();
183     semSignal(queueMutex);
184 }
```

```
154 void createPerson(string p, int id) {
155     semWait(printMutex);
156     print(p, id, "created");
157     semSignal(printMutex);
158 
159     semWait(peopleCountMutex);
160     peopleCount += 1;
161 
162     if (peopleCount == (NUM_GUESTS + NUM_DESK_EMPLOYEES + NUM_BELLHOP_EMPLOYEES)) {
163         // All the necessary people in the simulation have been created
164         //  let the main thread know the simulation can start
165         semSignal(created);
166     }
167     semSignal(peopleCountMutex);
168 
169     semWait(start);
170 }
```

```
221 void enterRoom(int guestNumber) {
222     semWait(printMutex);
223     print("Guest", guestNumber, "enter rooms", guestRoom[guestNumber]);
224     semSignal(printMutex);
225 }
226
227 void receiveBags(int guestNumber) {
228     semWait(printMutex);
229     print("Guest", guestNumber, "receives bags from bellhop", guestBellhopEmployeeNumber[guestNumber], "and gives tip");
230     semSignal(printMutex);
231 }
232
233 void retire(int guestNumber) {
234     semWait(printMutex);
235     print("Guest", guestNumber, "retires for the evening");
236     semSignal(printMutex);
237 }
```

```
186 void assignRoom(int deskEmployeeNumber, int guestNumber) {
187     semWait(roomCountMutex);
188     guestRoom[guestNumber] = roomCount;
189     roomCount += 1;
190     semSignal(roomCountMutex);
191
192     semWait(printMutex);
193     print("Front desk employee", deskEmployeeNumber, "registers guest", guestNumber, "and assigns from", guestRoom[guestNumber]);
194     semSignal(printMutex);
195 }
196
197 void takeBags(int bellhopEmplployeeNumber, int guestNumber) {
198     semWait(printMutex);
199     print("Bellhop", bellhopEmployeeNumber, "receives bags from guest", guestNumber);
200     semSignal(printMutex);
201 }
202
203 void enterHotel(int guestNumber, int bags) {
204     semWait(printMutex);
205     print("Guest", guestNumber, "enters hotel with", bags, "bags");
206     semSignal(printMutex);
207 }
208
209 void receiveKey(int guestNumber) {
210     semWait(printMutex);
211     print("Guest", guestNumber, "receives room key for room", guestRoom[guestNumber], "from front desk employee", guestDeskEmployeeNumber[guestNumber]);
212     semSignal(printMutex);
213 }
214
215 void requestHelp(int guestNumber) {
216     semWait(printMutex);
217     print("Guest", guestNumber, "requests help with bags");
218     semSignal(printMutex);
219 }
```