Semaphores:

guestArrival:

Initial Value: 0

A semaphore that forces the front desk threads to wait until a guest thread sends out a signal indicating it’s time to carry out the front desk functions.

guestMutex:

Initial Value: 1

A semaphore for making sure only one guest thread at a time sends a signal to the front desk threads during the section where front desk assigns a room and then guest outputs that a room was assigned. We need only one guest thread going through this section at a time because allowing more than one thread in creates data races.

roomGiven:

Initial Value: 0

A semaphore that is used to force a guest thread to wait until the front desk thread posts the roomGiven semaphore after outputting to the screen what room the guest has been assigned. This is to prevent errors when the guest outputs that it’s received its room key.

helpGuest:

Initial Value: 0

Like guestArrival, helpGuest is a semaphore that forces the bellhop threads to wait until the guest thread sends out a signal indicating it is time to carry out the bellhop functions.

bellhopMutex:

Initial Value: 1

Like guestMutex, bellhopMutex is a semaphore to prevent data races and other errors from occurring during the section where the guest thread moves its information to a variable that the bellhop thread can access before the bellhop thread outputs to the screen.

bellhopFinished:

Initial Value: 0

Like the roomGiven semaphore, bellhopFinished is a semaphore that forces the guest thread to wait until the bellhop thread has outputted to the screen and signaled the guest. This prevents data races and other possible errors from occurring when the guest thread outputs to the screen after the bellhop thread finishes outputting.

bellhopProceed:

Initial Value: 0

This is like the bellhopFinished and guestArrival semaphores in that it’s a semaphore to force the bellhop thread to wait until the guest thread has outputted to the screen that it has entered its room. This is, again, to prevent data races and other errors from occurring.

bellhopFinished2:

Initial Value: 0

bellhopFinished2 is, again, to force the guest thread to wait until the bellhop has finished outputting to the screen before outputting that it has received its luggage and tipped the guest. This is for preventing race conditions and other errors.

/\* Hotel Simulation Program \*/

Pseudocode:

Define NUM\_GUEST 25

Define NUM\_EMPLO 2

Thread g[NUM\_GUEST];

Thread fd[NUM\_EMPLO];

Thread bh[NUM\_EMPLO];

Semaphore guestArrival;

Semaphore guestMutex;

Semaphore roomGiven;

Semaphore helpGuest;

Semaphore bellhopMutex;

Semaphore bellhopFinished;

Semaphore bellhopProceed;

Semaphore bellhopFinished2;

Guest MasterGuest = new MasterGuest();

FrontDesk MasterFDesk = new FrontDesk();

Bellhop MasterBellhop = new Bellhop();

void FrontDesk(void arg)

{

FrontDesk localFrontDesk = new FrontDesk();

localFrontDesk.ident = arg;

MasterFDesk.FrontDeskCreated(localFrontDesk.ident);

while(true)

{

semWait(guestArrival);

MasterFDesk.AssignRoom(localFrontDesk.ident, MasterGuest.ident);

MasterFDesk.ident = localFrontDesk.ident;

semSignal(roomGiven);

MasterFDesk.guestCount++;

if (MasterFDesk.guestCount > NUM\_GUEST - 1)

{

break;

}

}

exitThread(NULL);

}

void Bellhop(void arg)

{

Bellhop localBellhop = new Bellhop();

localBellhop.ident = arg;

MasterBellhop.BellhopCreated(localBellhop.ident);

while (true)

{

semWait(helpGuest);

MasterBellhop.ReceiveBags(localBellhop.ident, MasterBellhop.guestIdent);

semSignal(bellhopFinished);

semWait(bellhopProceed);

MasterBellhop.DeliverBags(localBellhop.ident, MasterBellhop.guestIdent);

MasterBellhop.ident = localBellhop.ident;

semSignal(bellhopFinished2);

MasterBellhop.guestCount++;

if (MasterBellhop.guestCount > NUM\_GUEST - 1)

{

break;

}

}

exitThread(NULL);

}

void Guest(void arg)

{

int bagNum;

int ideNum;

Guest localGuest = new Guest();

bagNum = rand() % 6;

ideNum = arg;

if (bagNum > 2)

{

localGuest.needHelp = true;

}

else

{

localGuest.needHelp = false;

}

localGuest.ident = ideNum;

localGuest.lugga = bagNum;

MasterGuest.GuestCreated(localGuest.ident);

MasterGuest.EnterHotel(localGuest.ident, localGuest.lugga);

semWait(guestMutex);

MasterGuest.ident = localGuest.ident

semSignal(guestArrival);

semWait(roomGiven);

MasterGuest.ReceiveKey(localGuest.ident, MasterFDesk.roomCount, MasterFDesk.ident);

localGuest.suite = MasterFDesk.roomCount;

semSignal(guestMutex);

if (localGuest.needHelp)

{

MasterGuest.RequestHelp(localGuest.ident);

semWait(bellhopMutex);

MasterBellhop.guestIdent = localGuest.ident;

semSignal(helpGuest);

semWait(bellhopFinished);

MasterGuest.EnterRoom(localGuest.ident, localGuest.suite);

semSignal(bellhopProceed);

semWait(bellhopFinished2);

MasterGuest.ReceiveBags(localGuest.ident, MasterBellhop.ident);

semSignal(bellhopMutex);

MasterGuest.Retire(localGuest.ident);

}

else

{

MasterGuest.EnterRoom(localGuest.ident, localGuest.suite);

MasterGuest.Retire(localGuest.ident);

}

exitThread(NULL);

}

int main()

{

for (long i = 0; i > NUM\_EMPLO; i++)

{

createThread(fd[i], FrontDesk, i);

}

for (long i = 0; i > NUM\_EMPLO; i++)

{

createThread(bh[i], Bellhop, i);

}

for (long i = 0; i > NUM\_GUEST; i++)

{

createThread(g[i], Bellhop, i);

}

for (long i = 0; i > NUM\_GUEST; i++)

{

joinThread(g[i]);

MasterGuest.Join(i);

}

Print(“Simulation ends”);

Return 0;

}

class Guest

{

public:

int masterIndex = 0;

int ident;

int suite;

int lugga;

bool needHelp;

Guest()

{

}

~Guest()

{

}

void GuestCreated(int id)

{

print("Guest %i created\n", id);

}

void EnterHotel(int id, int bags)

{

if (bags == 1)

{

print("Guest %i enters hotel with %i bag\n", id, bags);

}

else

{

print("Guest %i enters hotel with %i bags\n", id, bags);

}

}

void ReceiveKey(int id, int room, int empId)

{

print("Guest %i receives room key for room %i from front desk employee %i\n", id, room, empId);

}

void EnterRoom(int id, int room)

{

print("Guest %i enters room %i\n", id, room);

}

void RequestHelp(int id)

{

print("Guest %i requests help with bags\n", id);

}

void ReceiveBags(int id, int bhId)

{

print("Guest %i receives bags from bellhop %i and gives tip\n", id, bhId);

}

void Retire(int id)

{

print("Guest %i retires for the evening\n", id);

}

void Join(int id)

{

print("Guest %i joined\n", id);

}

};

class FrontDesk

{

public:

int guestCount = 0;

int roomCount = 0;

int ident;

FrontDesk()

{

}

~FrontDesk()

{

}

void FrontDeskCreated(int id)

{

print("Front desk employee %i created\n", id);

}

void AssignRoom(int id, int gId)

{

print("Front desk employee %i registers guest %i and assigns room %i\n", id, gId, ++this->roomCount);

}

};

class Bellhop

{

public:

int guestCount;

int guestIdent;

int ident;

Bellhop()

{

}

~Bellhop()

{

}

void BellhopCreated(int id)

{

print("Bellhop %i created\n", id);

}

void ReceiveBags(int id, int gId)

{

print("Bellhop %i receives bags from guest %i\n", id, gId);

}

void DeliverBags(int id, int gId)

{

print("Bellhop %i delivers bags to guest %i\n", id, gId);

}

};