Project 2 Design

1. Semaphore List

(Sidenote: max\_doc\_threads = 30 and max\_patients\_threads = 3)

* + **max\_capacity**: Initially set to 30. Purpose is to only allow up to 30 patients threads to enter the waiting room.
  + **patient\_at\_desk**: Initially set to 0. Purpose is to alert the receptionist that a patient thread is ready to be registered.
  + **nurseSemaphore[max\_doc\_threads]**: Initially set to 0. Purpose is to call a specific nurse thread using their nurseID.
  + **doctorSemaphore[max\_doc\_threads]**: Initially set to 0. Purpose is to call a specific doctor thread using their doctorID.
  + **patientSemaphore[max\_patients\_threads]**: Initially set to 0. Purpose is to call a specific patient thread using their patientID.
  + **register\_complete[max\_patients\_threads]**: Initially set to 0. Purpose is to let the specific patient thread that their registration is complete.
  + **patient\_in\_waiting\_room[max\_patients\_threads]:** Initially set to 0. Purpose is to let the receptionist know that the specific patient thread is now sitting in waiting room and ready to be called by a nurse.
  + **tell\_doctor\_symptoms[max\_patients\_threads]:** Initially set to 0. Purpose is to let the doctor know that the specific patient thread is telling the doctor their symptoms.
  + **doctor\_advice[max\_patients\_threads]**: Initially set to 0. Purpose is to let the specific patient thread know that their doctor has given them advice about their symptoms.
  + **finishedNurse[max\_patients\_threads]**: Initially set to 0. Purpose is to let the nurse that their specific patient has left the office and can now free the nurse to serve the next patient.
  + **finishedDoc[max\_patients\_threads]**: Initially set to 0. Purpose is to let the doctor that their specific patient has left the office and can now free the doctor to serve the next patient.
  + **mutex\_1**: Initially set to 1. Purpose is to achieve mutual exclusion in patient threads where only one patient thread is allowed to be registered one at a time.
  + **mutex\_2**: Initially set to 1. Purpose is to achieve mutual exclusion in receptionist thread where it is allowed to only register one patient at a time.
  + **mutex\_3[max\_patients\_threads]**: Initially set to 1. Purpose is to achieve mutual exclusion in specific nurse thread where that nurse may only tend to one patient at a time.
  + **mutex\_4[max\_patients\_threads]**: Initially set to 1. Purpose is to achieve mutual exclusion in specific doctor thread where that doctor may only tend to one patient at a time.

1. Pseudocode

/\* program doctorVisit \*/

semaphore max\_capacity = 30;

semaphore patient\_at\_desk = 0;

semaphore nurseSemaphore[3] = {0};

semaphore doctorSemaphore[3] = {0};

semaphore patientSemaphore[30] = {0};

semaphore register\_complete[30] = {0};

semaphore patient\_in\_waiting\_room[30] = {0};

semaphore tell\_doctor\_symptoms[30] = {0};

semaphore doctor\_advice[30] = {0};

semaphore finishedNurse[30] = {0};

semaphore finishedDoc[30] = {0};

semaphore mutex\_1 = 1;

semaphore mutex\_2 = 1;

semaphore mutex\_3[30] = {1};

semaphore mutex\_4[30] = {1};

void patient(int patientArg){

int patientID = patientArg;

//Assign random doctor to patient and save result in hashmap //patientToDoc

patientToDoc[patientID] = randDoc(docNum);

wait(max\_capacity);

wait(mutex\_1);

enqueueP(patientID);

enter\_waiting\_room();

signal(patient\_at\_desk);

wait(register\_complete[patientID]);

sit\_in\_waiting\_room();

signal(patient\_in\_waiting\_room[patientID]);

signal(mutex\_1); //end mutual exclusion 1.

//Interact with Doctor and Nurse Threads.

wait(patient\_semaphore[patientID]);

enter\_doctor\_office();

signal(tell\_doctor\_symptoms[patientID]);

wait(doctor\_advice[patientID]);

receive\_advice();

leave\_office();

signal(finishedDoc[patientID]);

signal(finishedNurse[patientID]);

signal(max\_capacity);

exit(); //end thread

}

void receptionist(int recArg){ //recArg will be null so no need to save it.

int patientID = 0;

int nurseID = 0;

while(true){

wait(patient\_at\_desk);

wait(mutex\_2);

patientID = dequeueP();

nurseID = patientToDoc[patientID]; //get ID from hashmap

if(nurseID == 0){

enqueuePatAtDoc0(patientID); //add to doctor 0 queue

}

else if(nurseID ==1){

enqueuePatAtDoc1(patientID); //add to doctor 1 queue

}

else{

enqueuePatAtDoc2(patientID); //add to doctor 2 queue

}

register\_patient();

signal(register\_complete[patientID]);

wait(patient\_in\_waiting\_room[patientID]);

signal(nurseSemaphore[nurseID]);

signal(mutex\_2); //end mutual exclusion 2.

}

}

void nurse(int nurseArg){

int nurseID = nurseArg;

int patientID = 0;

while(true){

wait(nurseSemaphore[nurseID]);

wait(mutex\_3[nurseID]);

if(nurseID == 0){

patientID = getPatAtTopOfDoc0();

}

else if(nurseID == 1){

patientID = getPatAtTopOfDoc1();

}

else{

patientID = getPatAtTopOfDoc2();

}

take\_to\_doctor\_office();

signal(doctorSemaphore[nurseID]); //nurseID is same as doctorID

signal(patientSemaphore[patientID]);

wait(finishedNurse[patientID]);

signal(mutex\_3[nurseID]);

}

}

void doctor(int docArg){

int doctorID = docArg;

int patientID = 0;

while(true){

wait(doctorSemaphore[doctorID]);

wait(mutex\_4[doctorID]);

if(doctorID == 0){

patientID = dequeuePatAtDoc0();

}

else if(doctorID == 1){

patientID = dequeuePatAtDoc1();

}

else{

patientID = dequeuePatAtDoc2();

}

wait(tell\_doctor\_symptoms[patientID]);

listen\_to\_patient();

signal(doctor\_advice[patientID]);

wait(finishedDoc[patientID]);

signal(mutex\_4[doctorID]);

}

}