Operating system

Assignment 8

NAME:SOURABH PATEL

ADMISSION NO: U19CS082

Q. The synchronization problem called sleeping barber is described as follows:

A barber shop has a single barber, a single barber's chair in a small room, and a large waiting room with n seats. The barber and the barber's chair are visible from the waiting rooms. After servicing one customer, the barber checks whether any customers are waiting in the waiting room. If so, he admits one of them and starts serving him; otherwise, he goes to sleep in the barber's chair. A customer enters the waiting room only if there is at least one vacant seat and either waits for the barber to call him if the barber is busy, or wakes the barber if he is asleep. Identify the synchronization requirements between the barber and customer processes.

a. Code the barber and customer processes such that deadlocks do not arise.

We use 3 semaphores. Semaphore customers counts waiting customers; semaphore barbers is the number of idle barbers (0 or 1); and mutex is used for mutual exclusion. A shared data variable customers1 also counts waiting customers. It is a copy of customers. But we need it here because we can't access the value of semaphores directly. We also need a semaphore cutting which ensures that the barber won't cut another customer's hair before the previous customer leaves.

```
// shared data
semaphore customers = 0;
 semaphore barbers = 0;
semaphore cutting = 0;
semaphore mutex = 1;
 int customer1 = 0;
void barber() {
   while(true) {
   wait(customers); //sleep when there are no waiting customers
   wait(mutex); //mutex for accessing customers1
```

```
customers1 = customers1 - 1;
signal(barbers);
signal(mutex);
cut_hair();
          }
}

void customer() {
wait(mutex); //mutex for accessing customers1
if (customers1 < n) {
  customers1 = customers1 + 1;
  signal(customers);
  signal(mutex);
   wait(barbers); //wait for available barbers
            get_haircut();
          }
           else {
             //do nothing (leave) when all chairs are used.
signal(mutex);
          }
}
cut_hair(){
             waiting(cutting);
}
get_haircut(){
//get hair cut for some time;
  signal(cutting);
}
```

b. Consider the Sleeping-Barber Problem with the modification that there are k barbers and k barber chairs in the barber room, instead of just one. Write a program to coordinate the barbers and the customers.

```
//shared data
 semaphore waiting_room_mutex=1;
semaphore barber_room_mutex = 1;
semaphore barber_chair_free = k;
 semaphore sleepy_barbers = 0;

semaphore barber_chairs[k] = {0, 0, 0, …};
 int barber_chair_states[k] = {0, 0, 0, …};
 int num_waiting_chairs_free = N;
 boolean customer_entry( ) {
  // try to make it into waiting room
  wait(waiting_room_mutex);
if (num_waiting_chairs_free == 0) {
            signal(waiting_room_mutex);
return false;
          }
  num_waiting_chairs_free--; // grabbed a chair
signal(waiting_room_mutex);
  // now, wait until there is a barber chair free
  wait(barber_chair_free);
                // a barber chair is free, so release waiting room chair
             wait(waiting_room_mutex);
         wait(barber_room_mutex);
  num_waiting_chairs_free++;
  signal(waiting_room_mutex);
// now grab a barber chair
  int mychair;
```

```
  for (int I=0; I<k; I++) {
  if (barber_chair_states[I] == 0) { // 0 = empty chair
  mychair = I;
  break;
                    }
          }
  barber_chair_states[mychair] = 1; // 1 = haircut needed
 signal(barber_room_mutex);
  // now wake up barber, and sleep until haircut done
  signal(sleepy_barbers);
  wait(barber_chairs[mychair]);
  // great! haircut is done, let's leave.
  // barber has taken care of the barber_chair_states array.
  signal(barber_chair_free);
  return true;
}
void barber_enters() {
while(1) {
   // wait for a customer
  wait(sleepy_barbers);
  // find the customer
  wait(barber_room_mutex);
  int mychair;
            for (int I=0; I<k; I++) {
            if (barber_chair_states[I] == 1) {
  mychair = I;
break; }
                    }
            barber_chair_states[mychair] = 2;// 2 = cutting hair
```

```
signal(barber_room_mutex);
// CUT HAIR HERE
cut_hair(mychair);
// now wake up customer
  wait(barber_room_mutex);
            barber_chair_states[mychair] = 0;
// 0 = empty chair
signal(barber_chair[mychair]);
  signal(barber_room_mutex);
 // all done, we'll loop and sleep again
          }
}
```