EOPSY LAB5

Barber problem

Michał Skarzyński 293054

Table of content

Task 1
Problem analysis 2
Technical issues 3
Algorithm 4
Program input 4
Initial conditions 4
Program output 5

TASK

The analogy is based upon a hypothetical barber shop with many barbers serving women and man. In a barber shop there are N0 barbers serving only man, N1 barbers serving only woman and N2 barbers serving both women and men. Each barber has one barber's chair in a cutting room. In a waiting room there are M chairs. When the barber finishes cutting a customer's hair, he dismisses the customer and goes to the waiting room to see if there are others waiting. If there are, he brings one of them (but only if he is able to cut hair, i.e. the barber brings a man into the cutting room only if he can cut his hair, etc.) back to the chair and cuts hair (it lasts a random time). If there are none, he returns to the chair and sleeps in it. Each customer can be either a male client or a female client. When a client arrives, he/she checks what barbers are doing. If there is any barber sleeping (who is able to serve the client), the customer wakes him up, and sits in the cutting room chair. If all barbers (able to serve the client) are cutting hair, the customer stays in the waiting room. If there is a free chair in the waiting room, the customer sits in it and waits their turn. If there is no free chair, the customer leaves.

Implement the C language the sleeping barber problem.

The program must be parameterized by:

- N0, N1, N2 - number of barbers,
- M - number of chairs in waiting room.

PROBLEM ANALYSIS

In my solution to the problem, I wanted to make the best use of semaphores built-in waiting mechanisms and their ability to notify about a state change.

My whole solution is based on what I will call " Opportunities" in the rest of this text, this is the information transmitted using a semaphore which is incremented in the right places while other parts of the code are waiting for it.

I have two “Opportunities” in my code - the “Opportunity” of serving a man and the “Opportunity” of serving a woman.

In my code there is no separate instance of barber - it is only an abstract part of the code dealing with the change of states of these Opportunities.

At the very beginning, semaphore values describing Opportunities have value:

- For man = N0 + N2
- For woman = N1 + N2

The problem can be divided into two subtasks. The first is the moment when the customer comes to the store and comes out with the initiative and the desire to get to the barber. In this case, a special mutex can be placed which I named ASSIGMENET during which the client checks if there is a free barber - so de facto checks whether the variable describing the number of free barbers suitable for his/her sex is greater than zero and if not whether the number of unisex barbers is greater than zero. The client will prioritize choosing a specialized barber. If there is a possibility of assigning a barber to a given person then the variable describing number of free barbers of given type is decremented. Client holds value of barber type assigned to him and uses the sleep function to simulate hair cutting.

At this point, the customer also takes the Opportunity.

- If barber is a male barber then customer decreases Opportunity for a male.
- If barber is a female barber then customer decreases Opportunity for a female.
- If barber is a unisex barber then customer decreases Opportunity for male and a female.

It is very important to note that Opportunity is not necessarily equal to the number of available barbers.

Now let's look at what happens when there are no available barbers for a given customer.

First, the customer checks if there is a free place in the waiting room. If so, it takes it.

Then using the mechanism of semaphores tries to perform an operation reducing the Opportunity for its sex while trying to check if the mutex responsible for ASSIGEMENT is open. In my code, these are atomic operations performed one after the other as an array of operations.

Importantly, BOTH operations must be able to be performed so that the client's code can go further in flow of execution.

Let's move for a moment to the client who is currently finishing his hair cutting. When he does, he must notify the appropriate sexes about a suitable Opportunity.

- If barber was a male barber then customer increases Opportunity for a male.
- If barber was a female barber then customer increases Opportunity for a female.
- If barber was a unisex barber then customer increases Opportunity for male and a female.

We also remember that the ASSIGNMENT mutex is raised at virtually any time because it is checked quite rarely and immediately raised after checking.

The person in the waiting room gets a notification about their gender using the Opportunity semaphore and can finally reduce it. He also sets up ASSIGEMENT mutex to be able to assign himself a barber without fear that a new customer will come and take him away.

If the barber has been assigned, we must remember that the next step, just before starting the hair cutting, will be taking Opportunity - but it has just been reduced - so after the safe assignment of the barber, the client increases the Opportunity of his sex by one. At this point, it doesn't matter anymore because the ASSIGEMENT mutex has guaranteed us a safe assignment of the barber and, moreover, guaranteed us a safe handling of Opportunity.

We can imagine this situation in the following way: when a man comes out from a barber, he screams to the whole queue about the fact that the barber has the ability to cut someone, at this moment someone from the queue jumps up and determines that he will be assigned the barber. Approaching the barber and welcoming him, he additionally declares that he has just taken someone's place - but by the fact that he does it in two ways (he screams from the queue and then says it while saying hello to the barber) at some point he must increase his Opportunity so that his condition is always well established. Fortunately, this entire period is protected by the ASSIGMNENT mutex.

It is also worth noting at this point that until the customer approaches the N2 type barber and does not determine his status, the barber will be in a superposition and he will emit Opportunity for both sexes.

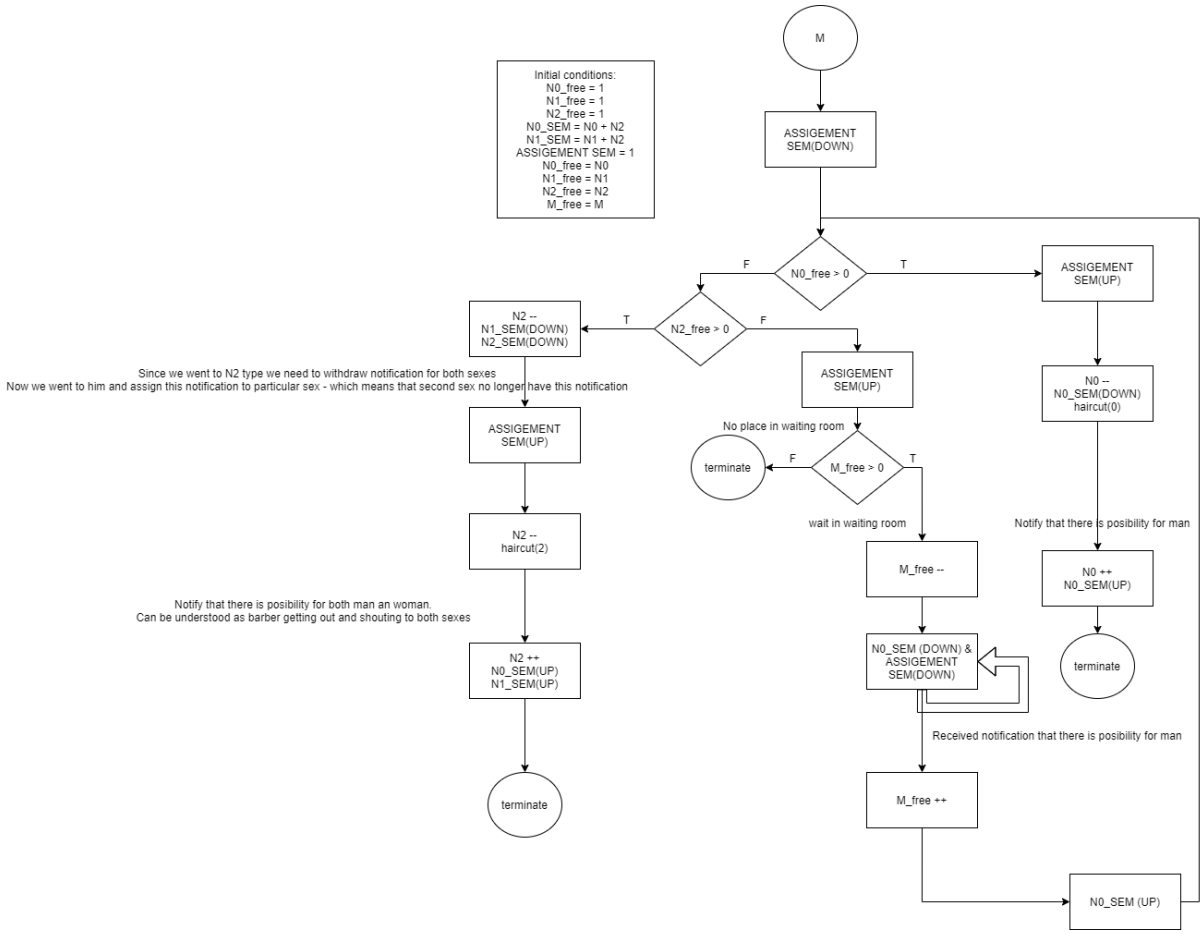
TECHNICAL ISSUES

To share variables such as the number of available barbers and the amount of free space I use shared memory which I create using the shmget command in main process, on child processes I fill it using the shmat command.

In addition to the semaphores described above, which work more like information transfer channels, each variable in shared memory has its own mutex responsible for secure changes of this variable without race condition

To create a semaphore set I use the semget command

ALGORITHM



We can see a flowchart for a male client. An analogous flowchart for female will only be N0 will be converted to N1.

PROGRAM INPUT

The program accepts 4 variables. The number of barbers N0, N1, N2 and the number of available seats in the waiting room M. The program can also accept them as a list of arguments from the command line.

INITIAL CONDITIONS

At the very beginning, the variables in the program are set to the following values:

- N0_free = N0
- N1_free = N1
- N2_free = N2
- M_free = M

Semaphores have values:

- SEM_BARBER_0_FREE = 1

- SEM_BARBER_1_FREE = 1
- SEM_BARBER_2_FREE = 1
- SEM_BARBER_0_NOTIFY = N0 + N2 (this is our Opportunity for male)
- SEM_BARBER_1_NOTIFY = N1 + N2 (this is our Opportunity for female)
- SEM_SEATS = 1
- ASSIGNMENT = 1;

PROGRAM OUTPUT

The program prints the output values as follows:

```
Client [client_id]{sex}: [N0_free, N1_free, N2_free | M_free] : <Action>
```