



**THE AMERICAN  
UNIVERSITY IN CAIRO**  
الجامعة الأمريكية بالقاهرة

**CS3401: Operating systems**

**Dr. Amr El-Kadi**

**Unisex Bathroom Problem**

Mahmoud Ahmed

900172747

## **Abstract**

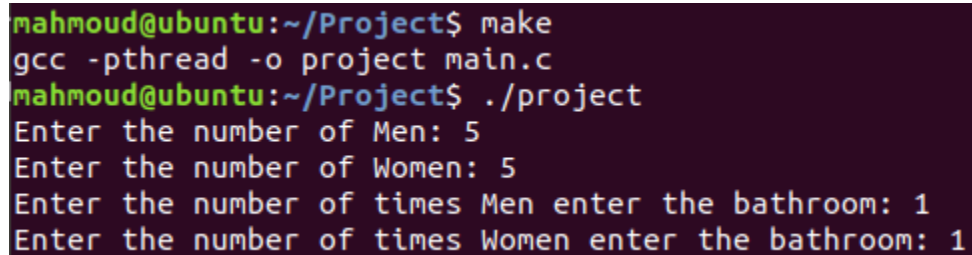
The unisex bathroom problem is one of the problem of synchronization in computer science. It requires to allow only one type of threads to execute a certain section of the code while preventing the other group from executing. In order to solve the problem, this project used 6 semaphores to ensure accurate mutual exclusion and efficient performance in the code. The code launches 6 threads and each of them launches the male and female threads in parallel to ensure concurrency of the execution. For the number of allowed threads of the same gender, after several experiments, it was found that 3 is the most efficient allowable running threads in the critical section. In addition, to ensure fairness, each thread sleep for 500 microsecond after entering the bathroom. The code showed no possible scenario for deadlocks using the 6 semaphores and proved to be working efficiently.

# 1 Running the project

First the makefile is run in terminal using the *make* command

In order to run the project, enter *./project* in the terminal.

The program will then ask the user to enter the number of men, number of women, number of times that a man enters the bathroom, number of times that a woman enters the bathroom.



```
mahmoud@ubuntu:~/Project$ make
gcc -pthread -o project main.c
mahmoud@ubuntu:~/Project$ ./project
Enter the number of Men: 5
Enter the number of Women: 5
Enter the number of times Men enter the bathroom: 1
Enter the number of times Women enter the bathroom: 1
```

Figure 1.1: running the executable file

Note: the total number of generated male and female threads must not exceed 32000 on most devices since this is almost the maximum allowed number of threads on linux systems. The program will return a segmentation fault if this happens.

## 2 Semaphores

There are two types of semaphores used in this project

### 2.1 Binary semaphores

These are the semaphores which only allow one thread to enter the critical section at a time.

The following are the used binary semaphore in the code:

- 1- *toilet\_sem*: behaves as a lock that indicates that a gender is currently using the toilet and prevents the other gender from using it.
- 2- *man\_sem*: behaves as a lock that allows only one man to increment or decrement the shared *m\_counter* variable to prevent racing
- 3- *woman\_sem*: behaves as a lock that allows only one woman to increment or decrement the shared *w\_counter* variable to prevent racing
- 4- *queue*: acts as the waiting queue of the bathroom. Each thread tries to lock it. If it fails, it waits for its turn to enter.

### 2.2 Counting semaphores

- 1- *man\_sem\_counter*: allows only 3 threads of the male gender to execute the critical section
- 2- *woman\_sem\_counter*: allows only 3 threads of the female gender to execute the critical section

### 2.3 Resource sharing:

#### 2.3.1 Among different genders

There are two shared semaphores among the two genders, *toilet\_sem*, and *queue*. *toilet\_sem* is used to insure that only one gender can enter the bathroom while blocking the other gender. *queue* is used to give a first in first out priority to the waiting threads to avoid starvation of one gender.

### 2.3.2 Among same gender

A total of three semaphores are shared between each gender. For example, in case of men, *man\_sem*, *man\_sem\_counter*, *queue* are shared between them. *queue* does the same functionality described previously. *man\_sem* insures that only one male can increment or decrement the *men\_count* variable in addition to printing the statement that a man wants to enter the bathroom. *man\_sem\_counter* is used to allow only three men at a time to enter the critical section in which a print statement and a sleep function is used. The same equivalent semaphores are also used for the women.

## 3 Data structures

### 3.1 Pthreads

A default data structure in linux that creates the required threads need to simulate the problem

### 3.2 Semaphore

A default data structure in linux that create a lock which allows a certain number of threads to execute based on the value passed to it in the initialization.

### 3.3 Param

A C struct used to pass the 3 parameters of the pthread function since it only allows for one void pointer argument to be passed.

This struct has 3 members: id, Num, itr. Id represents the id of the thread, num represents the number of threads we need to create for this gender. Finally, itr represent the number of time the man or woman is going to enter the bathroom.

## 4 Code Analysis

Since men and women have the same behavior, a generic description of the functions will be shown below

### 4.1 Gender\_enters Function

This function simulates what happens when someone enters a queue to the bathroom. If the person is in the queue, one he is the head of the queue and no one from the other gender is in the bathroom, he can enter.

The function behaves according to the following algorithm:

- 1- Try to lock the queue semaphore to get to enter the bathroom. If it is locked wait until it is unlocked and it's allowed to move to the next step
- 2- Lock the gender semaphore to ensure mutual exclusion and prevent any other thread from the same gender to enter the critical section. If it fails, wait until it's allowed
- 3- Check if this is the first thread of this gender to enter the critical section (the bathroom). If it is, lock the toilet semaphore to prevent threads from the other gender to enter
- 4- Increment the number of threads of this gender by one
- 5- Unlock the gender semaphore
- 6- Unlock the queue semaphore to represent that a thread was removed from the queue and the following threads can try to execute
- 7- Decrease the value of the gender counter semaphore to try to enter the bathroom

Sample code:

```

// this function simulates what happens when a man enters the queue
void man_enters(int i){
    sem_wait(&queue); // lock the queue semaphore in order to give this thread a priority while waiting
    sem_wait(&man_sem); // lock the man semaphore in order to be able to increment and check for equality without any race condition
    if(m_count == 0) sem_wait(&toilet_sem); // lock the bathroom representing that it is used by men. if this fails, wait untill women leave
    m_count++;
    printf("\nMan #%d wants to in the Bathroom\n",i);

    sem_post(&man_sem); // unlock man semaphore to allow others to run the critical section
    sem_post(&queue); // unlock the queue representing that a new thread can be the head of the queue
    sem_wait(&man_sem_counter); // decrease the counting semaphore to allow only 3 threads to in the bathroom
}

```

Figure 4.1: The man\_enters function

## 4.2 gender\_leaves function

the function simulates what happens when a person leaves the bathroom. If this was the last person, the thread would unlock the toilet mutex to allow the other gender to enter if its their turn.

The function behaves according to the following algorithm:

- 1- increment or post the gender counting semaphore to allow for any previously waiting threads to enter the critical section (the bathroom)
- 2- try to lock the gender semaphore. if it fails, wait until it's allowed
- 3- decrement the number of running threads of this gender by one
- 4- if this was the last thread, unlock the toilet semaphore to allow the other gender threads to enter the bathroom if applicable
- 5- unlock the gender semaphore, indicating that the thread left the bathroom

Sample code:

```

// this function simulates what happen when a man leaves the bathroom
void man_leaves(int i){
    sem_post(&man_sem_counter); // increment the counting semaphore to allow any new male thread to enter the bathroom

    sem_wait(&man_sem); // lock the man semaphore to secure the decrement and equality operation from any race conditions
    m_count--;
    printf("\nMan #%d left the Bathroom\n",i);

    if(m_count == 0) sem_post(&toilet_sem); // free the bathroom from men and allow the other gender to enter if applicable

    sem_post(&man_sem);
}

```

Figure 4.2:The man\_leaves function

## 4.3 gender\_thread function

This is the function that each gender thread executes once it is created.

First, the function extracts the function arguments from the void pointer arg.

Then, based on the user's input, it runs the gender\_enters, gender\_leaves for the number of iterations that the user specified. After the thread return successfully from the gender\_enters function, it sleeps for 250 microseconds if it was a male and 500 microseconds if it was a female thread to simulate the real world process.

Sample code:

```

// the thread function for the number of threads representing the men
void* male_thread(void* arg)
{
    parm * thread_arg = arg;    // convert the void pointer to a parm pointer to get the arguments
    int id = thread_arg->id;
    int itr = thread_arg->itr;
    for(int i=0; i<itr; i++){    // loop for the number of times passed by the user
        man_enters(id);        // call the entering function
        printf("\nMan #%d is in the Bathroom for time number #%d\n",id,i+1);
        usleep(250);          // sleep for 250 microsecond
        man_leaves(id);        // call the leaving function
    }
    pthread_exit(0);
}

```

Figure 4.3: The male\_thread function

#### 4.4 gender\_execution function

This is the function that gender thread execute once it is launched in the main. It begins by extracting the passed arguments from the void pointer arg. Afterwards, it creates a dynamic array of pthreads with the size of the gender threads that the user entered. It also creates a dynamic array of parameters struct equal in size to the pthreads array so each thread can have its own parameters.

Afterwards, each thread is launched and passed its corresponding parameter struct, and the calling gender thread waits for all its created threads to finish execution before it can return to the main thread.

The following graph shows a representation of the threads when there are 3 men and 3 women

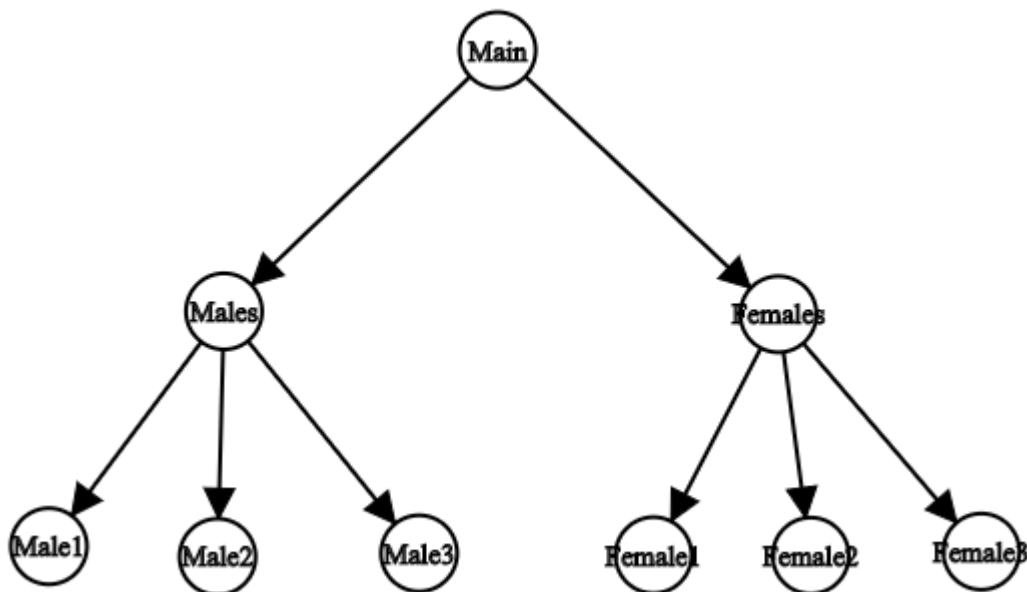


Figure 4.4: Generated threads tree

Sample code:

```

// this function launches the male threads
void* male_execution(void *arg){
    pthread_t *men;    // array of threads for each man
    parm * thread_arg = arg;    // convert the void pointer to a struct pointer
    int num_men = thread_arg->num;
    parm * men_thread_paramaters; // array of structs for each thread

    // allocate memory for the arrays
    men_thread_paramaters = (parm *) malloc(sizeof(parm)*num_men );
    men = (pthread_t*)malloc(sizeof(pthread_t)*num_men);
    int ret;
    for(int i = 0; i < num_men; i++){
        men_thread_paramaters[i].id = i+1;
        men_thread_paramaters[i].itr = thread_arg->itr;
        // launch the ith thread
        ret = pthread_create(&men[i], NULL, male_thread, (void *) &men_thread_paramaters[i]);
        if(ret!=0){
            printf("Creating Male Thread failed");
            pthread_exit(0);
        }
    }
    for(int i =0; i < num_men; i++){
        pthread_join(men[i], NULL);
    }
    //morgan
    free(men); // :V
    free(men_thread_paramaters);
    pthread_exit(0);
}

```

Figure 4.5: The male\_execution function

#### 4.5 Main function

The main function is responsible for getting input from the user to specify the required arguments of the threads. Afterwards, it launches a male execution thread and a female execution thread. Each of these threads launches its gender set of threads in parallel to ensure concurrency between the two genders.

## 5 Handling starvation

Using the queue semaphore gives the priority of execution to the currently waiting thread regardless of the gender. This prevents starvation by giving the lock to each thread to try to execute the entering function. If the thread waits after acquiring the lock, then this means that a thread of the other gender is using the bathroom and they leave, the thread that acquired the queue lock will execute. This represents what happens in real life when for example, two women are in the bathroom and only two people are allowed in the bathroom. The case of starvation happens when a man tries to enter after the women but waits instead for them to exit. If another woman comes and waits with the man, using the queue semaphore would ensure that the thread that came first would execute first.

## 6 Deadlocks

Due to the nature of the solution of the problem. The only shared semaphores between the two genders are toilet\_sem and queue. Both of which are handled efficiently to prevent any deadlocks. This can be shown using resource allocation graphs.

Consider the following scenario: only one person is allowed in the bathroom, and there is a man in the bathroom, a woman came and started waiting for the man to exit, and another man came after the woman. This will result in the following graph

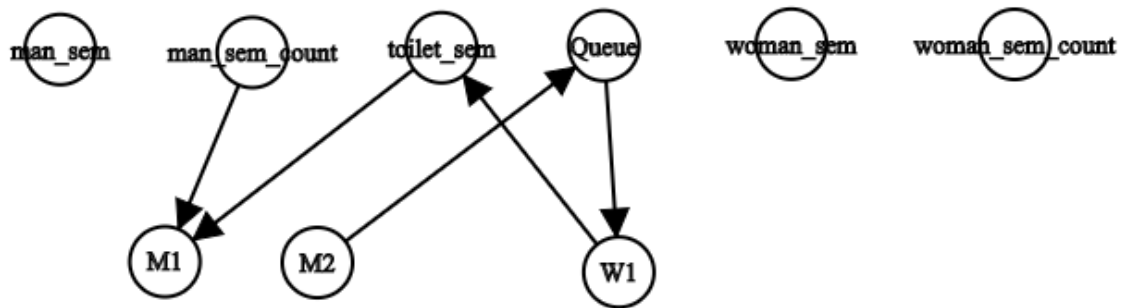


Figure 6.1: First resources allocation graph

As shown, toilet\_sem is allocated to M1 and queue is allocated to W1. M2 comes after W1 and requests access to queue but is blocked since W1 hasn't released it yet as it is waiting to acquire toilet\_sem which is allocated to M1.

Now, once M1 exits, toilet\_sem is released and acquired by W1 which in return releases queue semaphore. M2 can now acquire the queue semaphore and request toilet\_sem semaphore.

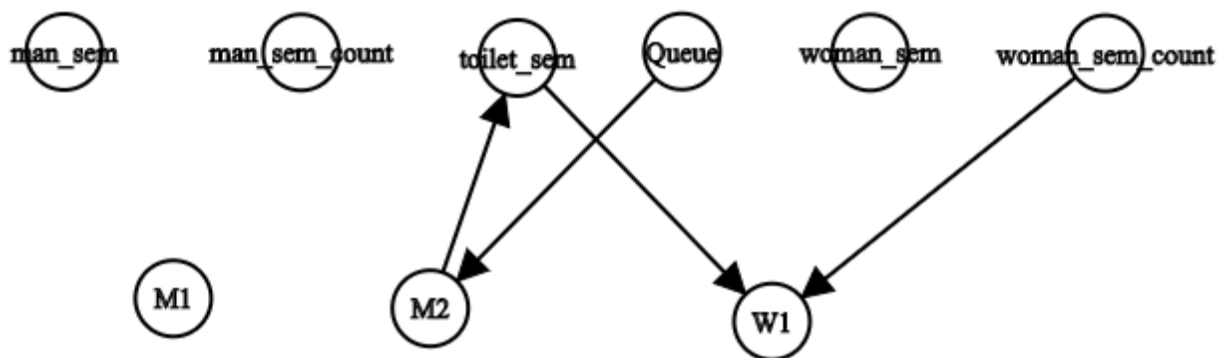


Figure 6.2: Second resources allocation graph

This pattern keeps on alternating between men and woman for any number of threads and any number of allowed people in the bathroom.



## 7 Sample output

```
Man #1 wants to in the Bathroom
Man #1 is in the Bathroom for time number #1
Man #1 left the Bathroom
Woman #1 wants to enter the Bathroom
Woman #1 is in the Bathroom for time number #1
Woman #1 left the Bathroom
Man #5 wants to in the Bathroom
Man #5 is in the Bathroom for time number #1
Man #5 left the Bathroom
Woman #3 wants to enter the Bathroom
Woman #3 is in the Bathroom for time number #1
Woman #2 wants to enter the Bathroom
Woman #2 is in the Bathroom for time number #1
Woman #2 left the Bathroom
Woman #3 left the Bathroom
Man #2 wants to in the Bathroom
Man #2 is in the Bathroom for time number #1
Man #3 wants to in the Bathroom
Man #3 is in the Bathroom for time number #1
Man #3 left the Bathroom
Man #2 left the Bathroom
Woman #5 wants to enter the Bathroom
Woman #5 is in the Bathroom for time number #1
Woman #5 left the Bathroom
Man #4 wants to in the Bathroom
Man #4 is in the Bathroom for time number #1
Man #4 left the Bathroom
Woman #4 wants to enter the Bathroom
Woman #4 is in the Bathroom for time number #1
Woman #4 left the Bathroom
```

*Figure 7.1: Sample output*