



## **CSCE3401: Operating Systems**

### **Project 2: Report**

**Submitted by:** Mohammed Abuelwafa

**Student ID:** 900172603

**Date:** 9<sup>th</sup> May 2020

**Spring 2020**

## **Abstract**

The unisex bathroom problem is one of the classical problems that are used to assess the validity and efficiency of synchronization solutions that uses various synchronization tool such as binary mutex, binary semaphores, counting semaphores and monitors. In this paper, the author presents a solution to this problem using binary and counting semaphores. The problem statement, the detailed solution and the implementation details are provided. Moreover, some scenarios to proof the correctness of the solution and a deadlock free proof are provided as well.

## Table of contents

Abstract .....	2
Table of figures .....	4
1. Introduction .....	5
2. Problem statement and Synchronization solution .....	6
3. Implementation .....	7
4. Proof of correctness scenarios (Test Cases) .....	12
5. A proof that the solution is deadlock free.....	16

## Table of figures

Figure 1: test case 1.....	12
Figure 2: test case 2.....	13
Figure 3: test case 3 (a) .....	13
Figure 4: test case 3 (b).....	14
Figure 5: test case 3 (c) .....	14
Figure 6: test case 3 (d).....	15
Figure 7: test case 3 (e) .....	15
Figure 8: Resource allocation graph .....	16

## 1. Introduction

Computer architectures nowadays have evolved to focus more on multicore systems rather than single core systems in order to improve performance by trying to parallelize code as much as possible in order to enhance concurrency even more. Hence, this introduced to system designers and developers more challenges such as creating, managing threads and handling their issues. Racing conditions and critical section problems are of huge concern in multi-threaded programming. A lot of classical problems are used often to validate a proposed synchronization solution such as the bounded buffer problem, the producer and consumer problem and the dining philosophers. In this paper, we are dealing with a similar classical problem which is concerned with a unisex bathroom. In this report, the author is going to present his solution, scenarios that try to prove its correctness and a proof that it is free of deadlocks.

## 2. Problem statement and Synchronization solution

The Problem has the following conditions to organize accessing the bathroom (a single resource):

- If a woman is accessing the bathroom, other women may enter, but no men and vice versa
- The maximum capacity of the bathroom is 5 people of the same gender.

In order to explain the solution, one must understand the problem and what should be considered in order to reach the best solution. One find that the bathroom is in one of the following states:

1. Empty
2. Men present
3. Women present

In order to solve this problem using counting semaphores and binary semaphores. One finds that we need to use around 6 semaphores having the following descriptions:

1. Empty: it is a binary semaphore that represents whether the bathroom is empty or occupied.
2. Man: it is a binary semaphore that is locked or unlocked depending on whether a male thread is in its critical section or not.
3. Woman: it is a binary semaphore that is locked or unlocked depending on whether a female thread is in its critical section or not.
4. Man\_count : it is a binary semaphore that indicates the maximum number of simultaneous allowable men in the bathroom.
5. Woman\_count : it is a binary semaphore that indicates the maximum number of simultaneous allowable women in the bathroom.
6. Barrier : it is a binary semaphore that is concerned with ensuring fairness between men and women threads in using the bathroom.

Before discussing the implementation, one needs to discuss the problem of starvation that may occur in this problem with some synchronization solutions. First of all, assume that initially the bathroom is empty, then a man comes. He starts using the bathroom and while that is happening, a man and a woman comes in. the man can enter since we did not exceed the maximum capacity, but the woman waits. Assume that another man comes his allowed to enter as well and the woman is still waiting. Assume that a flow of women and men comes. One finds that men are only allowed to enter the bathroom while this is not the case for women, and this may cause starvation. Hence, we need to avoid starvation and here comes the importance of the semaphore called *barrier* in this scenario. This would be clearer while discussing the implementation.

### 3. Implementation

We have seven functions in our solution organized as the following:

1. Man\_wants\_to\_enter ()
2. Woman\_wants\_to\_enter ()
3. Man\_leaves()
4. Woman\_leaves()
5. Men( )
6. Women( )
7. Main( )

```
void man_wants_to_enter ()
{
    sem_wait(&Barrier);
    sem_wait(&man); // main inside
    M_count = M_count + 1; // increment number of men
    if(M_count == 1){
        sem_wait(&empty);} // raise the sign that there is a male inside
    sem_post(&man); // man exits
    sem_post(&Barrier);
    sem_wait(&man_count); // men are inside and did not exceed maximum capacity
}
```

The flow of this function is explained as what follows; the barrier semaphore calls the wait () function and the same applies to the man semaphore. Then the man enters its critical section and increases the count of men inside the bathroom by one. If this is the first man to enter the bathroom, then the empty semaphore calls the wait( ) function as well to indicate that the bathroom is occupied now. After that both the main semaphore and the barrier's semaphore are signaled available by calling the post ( ) function. However, the man\_count semaphore calls the wait function in order to decrease the available men capacity by one.

Similarly, the same logic applies to `woman_want_to_enter ( )` function with the associated semaphores.

```
void woman_wants_to_enter ()
{
    sem_wait(&Barrier);
    sem_wait(&woman);
    W_count = W_count + 1 ;
    if(W_count == 1){
        sem_wait(&empty);}
    sem_post(&woman);
    sem_post(&Barrier);
    sem_wait(&woman_count);
}
```

Regarding the `man_leave ( )` function, the `man_count` semaphore calls the `post ( )` function in order to signal that there is room for an extra man to enter (free place is acquired again).

However, the man semaphore calls the `wait()` function for preparation for the man thread to enter its critical section to decrement the count of men inside the bathroom. If this is the last man to exit, then the empty semaphore signals calling the `post ( )` function to indicate that the bathroom is empty now. Finally, the man semaphore signals calling the `post ( )` function in order to indicate that the bathroom is empty now.

```
void man_leaves ()
{
    //printf("Man id # %d leaves the bathroom\n", *i);
    sem_post(&man_count); // men exit
    sem_wait(&man); // man acquire the lock to decrement
    M_count = M_count - 1; // decerementing (critical section)
    if(M_count==0){
        sem_post(&empty);} // no men are inside
    sem_post(&man); // man releases the lock
}
```



Similarly, the same logic applies to `woman_leaves ( )` function with the associated semaphores.

```
void woman_leaves ()
{
    // printf("Woman # %d leaves the bathroom\n",*i);
    sem_post(&woman_count);
    sem_wait(&woman);
    W_count = W_count - 1 ;
    if(W_count==0){
        sem_post(&empty);}
    sem_post(&woman);
}
```

The function `men ( )` is the function that the thread executed. It contains the code for entering and exiting the bathroom. Moreover, it has the loop that runs a number of times that is equal to the number of times the man is going to use the bathroom which is a user input. Moreover, it has a call to sleep function that represents some sort of delay as an indicator of doing some processing.

```
void men (int * i)
{
    int j;
    for ( j =0; j < Number_of_using_bathroom_males ; j++) // looping multiple
times according to Number_of_using_bathroom_males
    {
        man_wants_to_enter();
        printf("Man # %d is in the bathroom\n", *i);
        sleep(2);
        printf("Man # %d leaves the bathroom\n", *i);
        man_leaves();

    }
    pthread_exit(0);
}
```

Similarly, the same logic applies to woman ( ) function.

```
void women(int * i)
{
    int j;
    for ( j =0; j < Number_of_using_bathroom_females ; j++) // looping multiple
times according to Number_of_using_bathroom_females
    {
        woman_wants_to_enter();
        printf("Woman # %d is in the bathroom\n", *i);
        sleep(2);

        printf("Woman # %d leaves the bathroom\n",*i);
        woman_leaves();

    }
    pthread_exit(0);
}
```

Regarding the main function, it mainly prompts the user to enter all the inputs. Moreover, it creates the threads, runs them and finally joins them.

```
int main ()
{

    sem_init(&empty, 0, 1);
    sem_init(&man, 0, 1);
    sem_init(&woman, 0, 1);
    sem_init(&man_count, 0, 5); // Maximum number of males in the bathroom = 5
    sem_init(&woman_count, 0, 5); // Maximum number of females in the bathroom =
5
    sem_init(&Barrier, 0, 1);

    int w , m, max ; // male and females threads counts and the max between both
    printf( "Please enter m:");
    scanf("%d", &m);
    printf( "Please enter w:");
    scanf("%d",&w);
    printf("please enter the number of times each man enters the bathroom:");
    scanf("%d",&Number_of_using_bathroom_males);
    printf("please enter the number of times each woman enters the bathroom:");
    scanf("%d",&Number_of_using_bathroom_females);
```

```

pthread_t  *m_tid , *w_tid;
pthread_attr_t attr;
pthread_attr_init(&attr);
m_tid = ((pthread_t*)malloc(m*sizeof(pthread_t)));
w_tid = ((pthread_t*)malloc(w*sizeof(pthread_t)));
// m_tid = ((pthread_t*)malloc((m+w)*sizeof(pthread_t)));
if (m >= w)
    max=m;
else
    max=w;

int i ;
int success_m , success_w;

int * tmids , *tfids ;
tmids = ((int*)malloc(m*sizeof(int)));
tfids = ((int*)malloc(w*sizeof(int)));
for (i=0;i<max;i++)
{

    if (i < m)
    {
        //then create man thread
        tmids[i]=i;
        success_m = pthread_create(&m_tid[i],&attr, (void *) (int *) men ,
&tmids[i]); //create threads and tmids[i] is used for indexing the threads
        if(success_m!=0)
        { //create returns 0 if thread creation succeeded
            printf("Man thread creation failed\n");
            exit(EXIT_FAILURE);
        }
    }

    if (i < w)
    {
        //then create female thread
        tfids[i]=i;
        success_w = pthread_create(&w_tid[i],&attr, (void*)(int*) women ,
&tfids[i]); //create female thread and tfids[i] is used for indexing the threads
        if(success_w!=0)
        { //create returns 0 if thread creation succeeded
            printf("Woman thread creation failed\n");
        }
    }
}

```

```

        exit(EXIT_FAILURE);
    }
}

for (i = 0 ; i < m ; i++)
{
    pthread_join(m_tid[i],NULL);
}
int j;
for (j = 0 ; j < w ; j++)
{
    pthread_join(w_tid[j],NULL);
}

return 0;
}

```

#### 4. Proof of correctness scenarios (Test Cases)

Mainly these test cases would try to show that the solution produces the correct simulation. Moreover, it would try to focus on the fairness between the two groups of threads.

1. Men = 3 , women = 2 , number of male times = 1 , number of female times = 1.

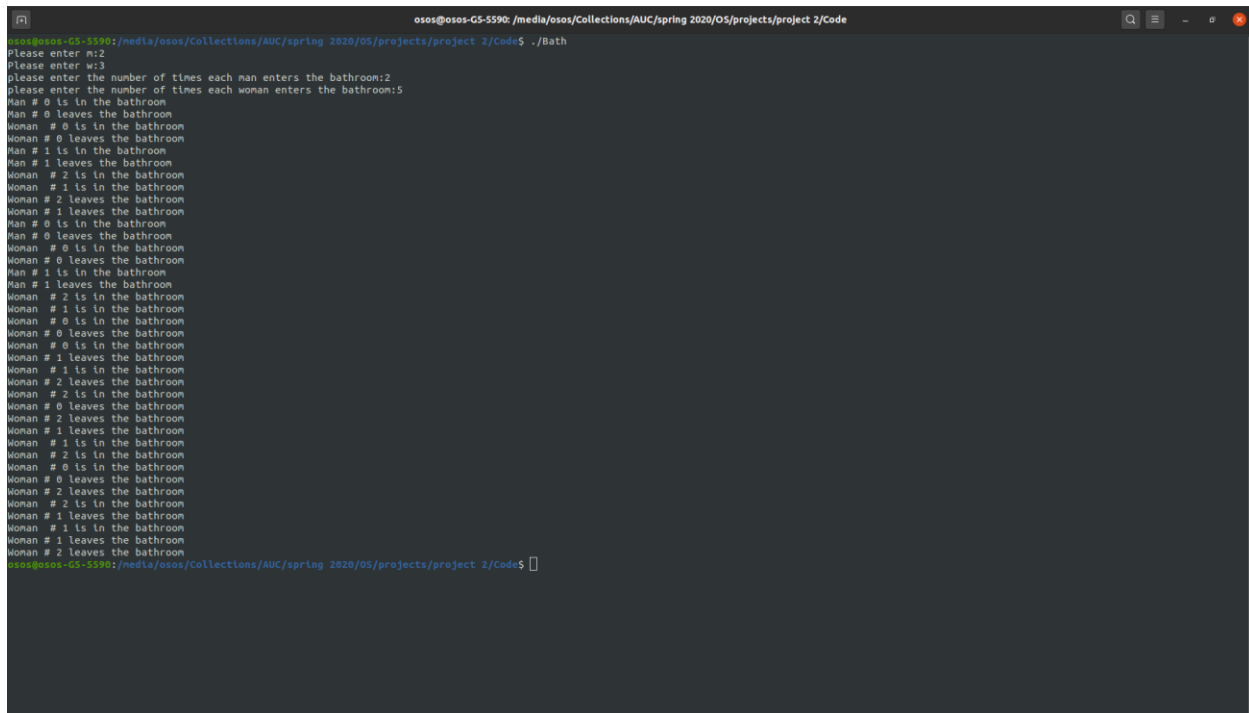
```

osos@osos-G5-5590: /media/osos/Collections/AUC/spring 2020/OS/projects/project 2/Code
Please enter m:3
Please enter w:2
please enter the number of times each man enters the bathroom:1
please enter the number of times each woman enters the bathroom:1
Man # 0 is in the bathroom
Man # 0 leaves the bathroom
Woman # 0 is in the bathroom
Woman # 0 leaves the bathroom
Man # 1 is in the bathroom
Man # 1 leaves the bathroom
Woman # 1 is in the bathroom
Woman # 1 leaves the bathroom
Man # 2 is in the bathroom
Man # 2 leaves the bathroom
osos@osos-G5-5590: /media/osos/Collections/AUC/spring 2020/OS/projects/project 2/Code$

```

Figure 1: test case 1

2. Men = 2, women = 3 , number of male times = 2 , number of female times = 5.



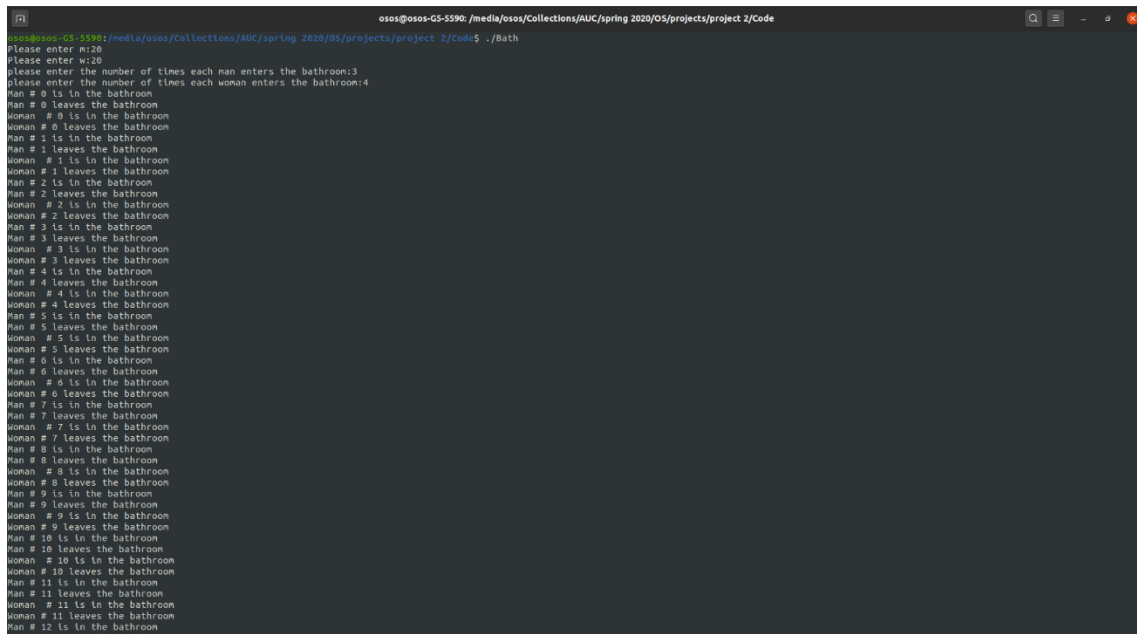
```

osos@osos-G5-5590: /media/osos/Collections/AUC/spring 2020/OS/projects/project 2/Code$ ./Bath
Please enter m:2
Please enter w:3
please enter the number of times each man enters the bathroom:2
please enter the number of times each woman enters the bathroom:5
Man # 0 is in the bathroom
Man # 0 leaves the bathroom
Woman # 0 is in the bathroom
Woman # 0 leaves the bathroom
Man # 1 is in the bathroom
Man # 1 leaves the bathroom
Woman # 2 is in the bathroom
Woman # 2 leaves the bathroom
Woman # 1 leaves the bathroom
Man # 0 is in the bathroom
Man # 0 leaves the bathroom
Woman # 0 is in the bathroom
Woman # 0 leaves the bathroom
Man # 1 is in the bathroom
Man # 1 leaves the bathroom
Woman # 2 is in the bathroom
Woman # 2 leaves the bathroom
Woman # 1 is in the bathroom
Woman # 1 leaves the bathroom
Man # 0 is in the bathroom
Man # 0 leaves the bathroom
Woman # 2 is in the bathroom
Woman # 2 leaves the bathroom
Woman # 1 is in the bathroom
Woman # 1 leaves the bathroom
Man # 1 is in the bathroom
Man # 1 leaves the bathroom
Woman # 2 is in the bathroom
Woman # 2 leaves the bathroom
Woman # 0 is in the bathroom
Woman # 0 leaves the bathroom
Man # 1 is in the bathroom
Man # 1 leaves the bathroom
Woman # 2 is in the bathroom
Woman # 2 leaves the bathroom
Woman # 1 is in the bathroom
Woman # 1 leaves the bathroom
Man # 0 is in the bathroom
Man # 0 leaves the bathroom
Woman # 2 is in the bathroom
Woman # 2 leaves the bathroom
Woman # 1 is in the bathroom
Woman # 1 leaves the bathroom
Man # 1 is in the bathroom
Man # 1 leaves the bathroom
Woman # 2 leaves the bathroom
osos@osos-G5-5590: /media/osos/Collections/AUC/spring 2020/OS/projects/project 2/Code$

```

Figure 2: test case 2

3. Men = 20, women = 20 , number of male times =3, number of female times = 4.



```

osos@osos-G5-5590: /media/osos/Collections/AUC/spring 2020/OS/projects/project 2/Code$ ./Bath
Please enter m:20
Please enter w:20
please enter the number of times each man enters the bathroom:3
please enter the number of times each woman enters the bathroom:4
Man # 0 is in the bathroom
Man # 0 leaves the bathroom
Woman # 0 is in the bathroom
Woman # 0 leaves the bathroom
Man # 1 is in the bathroom
Man # 1 leaves the bathroom
Woman # 1 is in the bathroom
Woman # 1 leaves the bathroom
Man # 2 is in the bathroom
Man # 2 leaves the bathroom
Woman # 2 is in the bathroom
Woman # 2 leaves the bathroom
Man # 3 is in the bathroom
Man # 3 leaves the bathroom
Woman # 3 is in the bathroom
Woman # 3 leaves the bathroom
Man # 4 is in the bathroom
Man # 4 leaves the bathroom
Woman # 4 is in the bathroom
Woman # 4 leaves the bathroom
Man # 5 is in the bathroom
Man # 5 leaves the bathroom
Woman # 5 is in the bathroom
Woman # 5 leaves the bathroom
Man # 6 is in the bathroom
Man # 6 leaves the bathroom
Woman # 6 is in the bathroom
Woman # 6 leaves the bathroom
Man # 7 is in the bathroom
Man # 7 leaves the bathroom
Woman # 7 is in the bathroom
Woman # 7 leaves the bathroom
Man # 8 is in the bathroom
Man # 8 leaves the bathroom
Woman # 8 is in the bathroom
Woman # 8 leaves the bathroom
Man # 9 is in the bathroom
Man # 9 leaves the bathroom
Woman # 9 is in the bathroom
Woman # 9 leaves the bathroom
Man # 10 is in the bathroom
Man # 10 leaves the bathroom
Woman # 10 is in the bathroom
Woman # 10 leaves the bathroom
Man # 11 is in the bathroom
Man # 11 leaves the bathroom
Woman # 11 is in the bathroom
Woman # 11 leaves the bathroom
Man # 12 is in the bathroom
Man # 12 leaves the bathroom

```

Figure 3: test case 3 (a)

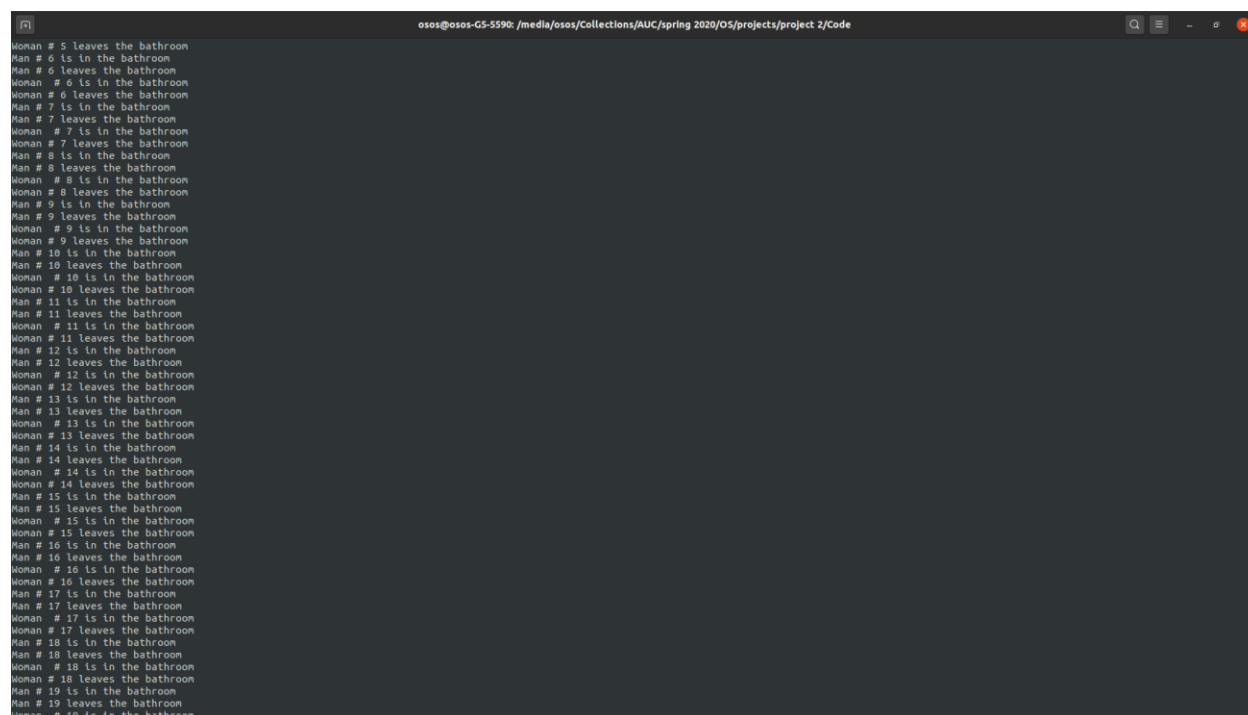


Figure 4: test case 3 (b)



Figure 5: test case 3 (c)

```

man # 16 leaves the bathroom
woman # 16 is in the bathroom
man # 16 leaves the bathroom
man # 17 is in the bathroom
man # 17 leaves the bathroom
woman # 17 is in the bathroom
woman # 17 leaves the bathroom
man # 18 is in the bathroom
man # 18 leaves the bathroom
woman # 18 is in the bathroom
woman # 18 leaves the bathroom
man # 19 is in the bathroom
man # 19 leaves the bathroom
woman # 19 is in the bathroom
woman # 0 is in the bathroom
woman # 1 is in the bathroom
woman # 2 is in the bathroom
woman # 3 is in the bathroom
woman # 19 leaves the bathroom
woman # 19 is in the bathroom
woman # 1 leaves the bathroom
woman # 4 is in the bathroom
woman # 2 leaves the bathroom
woman # 6 is in the bathroom
woman # 3 leaves the bathroom
woman # 0 leaves the bathroom
woman # 7 is in the bathroom
woman # 0 is in the bathroom
woman # 4 leaves the bathroom
woman # 6 leaves the bathroom
woman # 9 is in the bathroom
woman # 7 leaves the bathroom
woman # 8 leaves the bathroom
woman # 10 is in the bathroom
woman # 11 is in the bathroom
woman # 12 is in the bathroom
woman # 19 leaves the bathroom
woman # 13 is in the bathroom
woman # 10 leaves the bathroom
woman # 9 leaves the bathroom
woman # 11 leaves the bathroom
woman # 15 is in the bathroom
woman # 12 leaves the bathroom
woman # 16 is in the bathroom
woman # 17 is in the bathroom
woman # 13 leaves the bathroom
woman # 14 is in the bathroom
woman # 18 is in the bathroom
woman # 14 leaves the bathroom
woman # 18 leaves the bathroom
woman # 16 leaves the bathroom
woman # 5 is in the bathroom
woman # 15 leaves the bathroom
woman # 17 leaves the bathroom
woman # 5 leaves the bathroom

```

Figure 7: test case 3 (e)

## 5. A proof that the solution is deadlock free

In order to have a deadlock in a concurrent parallel code, four conditions must hold at the same time which are:

1. Mutual exclusion
2. Hold and wait
3. No preemption
4. Circular wait

We find that in our solution both Mutual exclusion and hold and wait are applicable. However, due to the nature that we are running on Linux operating system that uses the CFS scheduler that is preemptive, the no preemption is violated. Hence, there is no deadlocks. Moreover, there is no circular wait in this solution since all the threads are waiting to use the bathroom (single resource) in a queue manner where the threads of the same gender are allowed to enter the bathroom until it reaches its maximum capacity. Since the time of using the bathroom is fixed, it is guaranteed that the threads will finish using the resource and let others use it at the end. The problem is following the following resource allocation graph where the solution chooses between the different threads.

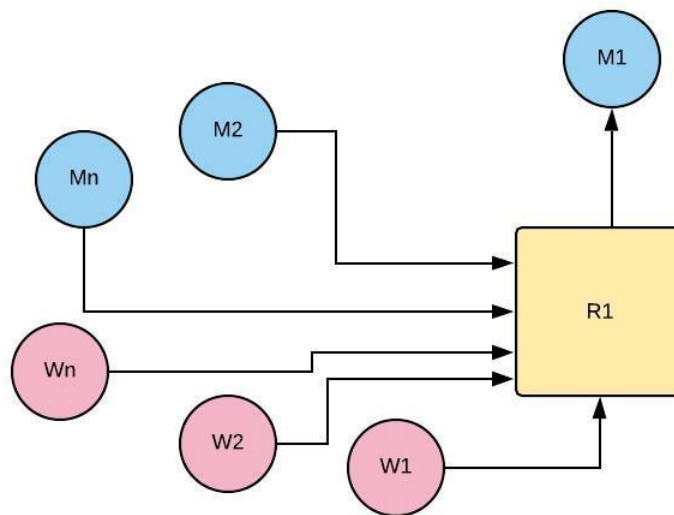


Figure 8: Resource allocation graph