# Assignment 1

# Operating System Processes

Process Creation
Shared Memory
Bounded Queue

By:

**MHD Maher Azkoul**
**438017578**
**s438017578@st.uqu.edu.sa**
**course: operating systems, class: 3, no: 27**

**Anas Mohammad Nawawi**
**438008655**
**s438008655@st.uqu.edu.sa**
**course: operating systems, class: 3, no: 25**

Professor:

**pr. AbdulBaset Gaddah**

**Umm Al-Qura University**
**College of Computers and Information Systems**
**Operating Systems Course – 14012203 - 1441**

Oct 17, 2019

Github link

Content:

**Umm Al-Qura University**
**Operating Systems – 1441**
**class: 3**
**Assignment 1: processes**
**Date: Thr - 17 Oct 2019**

**Student: MHD Maher Azkuol**
**ID: 438017578**
**Student: Anas Nawawi**
**ID: 438008655**
**Pr. AbdulBaset Gaddah**

---

## Part I - Process Creation:

Write a C program called fork02.c that uses fork() system call to create two child processes within a parent:

The goal of this program is to create two children processes, each child will iterate and print a text on the screen on each iteration. And the parent will track their status.

Sample output:

```
Parent process is born, my pid is: 32380
First child is born, my pid is: 32381
My parent pid is: 32380
First child executes iteration 0
First child executes iteration 1
First child executes iteration 2
First child executes iteration 3
First child executes iteration 4
First child dies quietly
Second child is born, my pid is: 32382
My parent pid is: 32380
Second child executes iteration 0
Second child executes iteration 1
Second child executes iteration 2
Second child executes iteration 3
Second child executes iteration 4
Second child dies quietly
Child completed: pid = 32381   status = 0
Child completed: pid = 32382   status = 0
Parent process dies quietly
```

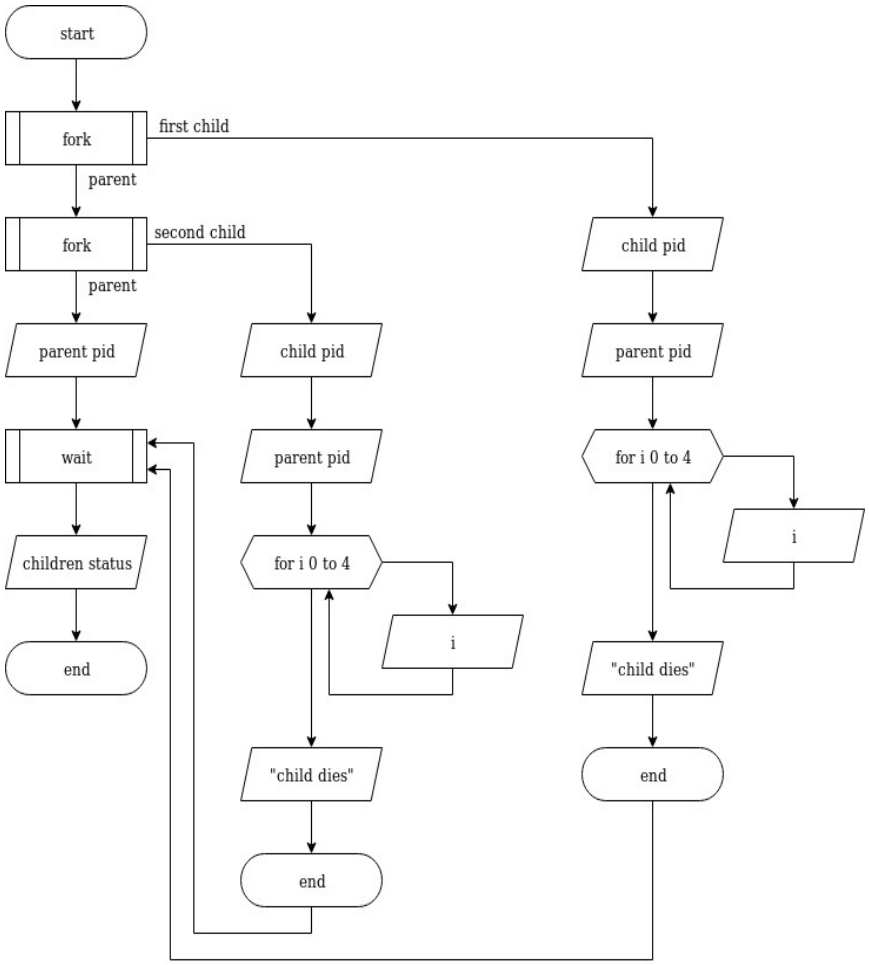
---

## Explanation:

Parent process had created first child and second child before any output had been printed.

Each process among all three has begun executing after creating, and started to do their jobs.

For the last three lines of the output, the parent process has waited its two children to finish their job, then it printed their status.



---

## Part II - Shared Memory:

The Lucky number 5: using POSIX shared memory in Linux or other Unix-like systems, write two different C programs (MyServer.c and MyClient.c) that respectively implement the behavior of server and client:

The goal of the program is to guess values and let the second part to check to the right guess. The two parts will communicate via POSIX-shared memory.

First part (MyServer.c) will scan values and replace each non-true value with -1. If the right value was founded, then that program will display a message and destroy shared memory, then it will be terminated.

Second part (MyClient .c) will generate random numbers and replace -1 with it. If the right value was founded by the server, then that program will display a message and detach from shared memory, then it will be terminated.

---

Sample output:

MyServer:

```
Initial status of shared memory.
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -
1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
now the server is listening to the client.

Numbers generated by the client and read by the server.
4, 7, 18, 16, 14, 16, 7, 13, 10, 2, 3, 8, 11, 20, 4, 7, 1, 7, 13,
17, 12, 9, 8, 10, 3, 11, 3, 4, 8, 16, 10, 3,

after replacing values other than lucky number to -1.
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -
1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,

Numbers generated by the client and read by the server.
3, 19, 10, 8, 14, 17, 12, 3, 10, 14, 2, 20, 5, 18, 19, 5, 16, 11,
14, 7, 12, 1, 17, 14, 3, 11, 17, 2, 6, 6, 5, 8,

after replacing values other than lucky number to -1.
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 5, -1, -1, 5, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 5, -1,

The lucky number [5] is found –the server will be terminated
soon.

the server says goodbye!
```

MyClient:

```
initial start of the client - no search for the lucky number

after replacing values of -1 to new random numbers.
4, 7, 18, 16, 14, 16, 7, 13, 10, 2, 3, 8, 11, 20, 4, 7, 1, 7, 13,
17, 12, 9, 8, 10, 3, 11, 3, 4, 8, 16, 10, 3,

after replacing values of -1 to new random numbers.
3, 19, 10, 8, 14, 17, 12, 3, 10, 14, 2, 20, 5, 18, 19, 5, 16, 11,
14, 7, 12, 1, 17, 14, 3, 11, 17, 2, 6, 6, 5, 8,

after replacing values of -1 to new random numbers.
17, 6, 7, 10, 14, 18, 5, 16, 3, 6, 15, 8, 5, 15, 5, 5, 4, 11, 8,
9, 17, 19, 9, 5, 4, 12, 15, 20, 13, 1, 5, 17,

The lucky number [5] is found -the cleint will be terminated.

the client says goodbye!
```

## Explanation:

First of all, the server has created an array of integers inside a shared memory that it created. Then it initialized it with -1s. Also, the server saved a number 5 as a “lucky number”.

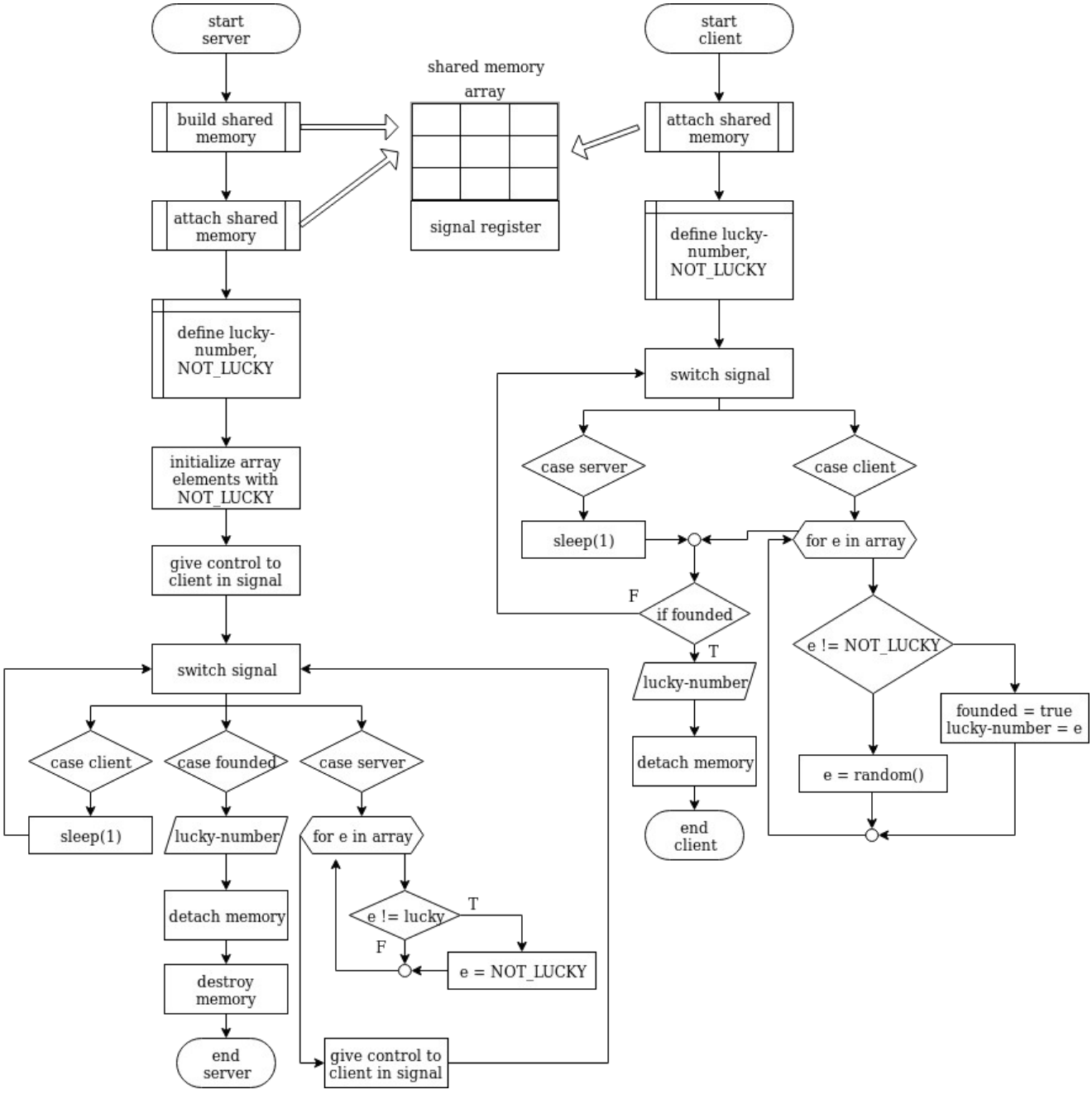
After that, the client had attached to the shared memory and accessed the array, then it replaced all -1s in the array with random numbers.

The server scaned the array with new random numbers and replaced all values with -1 except the lucky number.

The two programs will continue like that until the lucky number is founded in the random numbers generated by the client.

**Umm Al-Qura University**
**Operating Systems – 1441**
**class: 3**
**Assignment 1: processes**
**Date: Thr - 17 Oct 2019**

**Student: MHD Maher Azkuol**
**ID: 438017578**
**Student: Anas Nawawi**
**ID: 438008655**
**Pr. AbdulBaset Gaddah**



---

## Part III - BoundedQueue:

create a bounded queue of size 20, which is shared by both the producer and the consumer. The producer generates data and put it into a queue, at the same time, the consumer takes the data from the same queue:

The goal of the program is to generate random numbers and consume it by another program via shared memory (queue). This program is divided into three subprograms: bounded buffer, producer, consumer.

Sample output:

```
next consumed = 188
next produced = 188
next produced = 57
next consumed = 57
next produced = 108
next consumed = 108
…
next produced = 155
next consumed = 55
Producer died quietly
next consumed = 155
Consumer died quietly
```

---

## Explanation:

First, the bounded queue class will start and run the thread of consumer and the thread of the producer.

The producer will generate 50 random numbers between 1 and 200 and put it in the queue.

The consumer will start consuming the data (random numbers generated by the producer) from the queue.

When producer reaches the 50th number, it will put a “poison bill” to terminate the consumer. The poison bill is nothing but a negative number.