

# Operating Systems Homework #01 Part B

**Berru Lafcı - 1901042681**

*I just did the first and second strategy in this part. So, there is no priority scheduling.*

The rest of the code is the same except kernel.cpp.

I generated random numbers with the clock counter. It uses the rdtsc assembly instruction to read the time-stamp counter, which provides a high-resolution timer value. The counter value is then manipulated with a linear congruential generator algorithm to produce a random number. The resulting value is adjusted to fall within the specified range and returned.

```
common::int32_t generateRandomNumber(int min, int max)
{
    uint64_t counter;
    int32_t num;

    // Get the clock counter
    asm volatile("rdtsc" : "=A"(counter));

    // Generate a random number using the clock counter
    counter = counter * 1103515245 + 12345;
    num = (int32_t)((counter >> 16) & 0xFFFFFFFF) % (max - min);

    if (num < 0)
        num += max;

    return num + min;
}
```

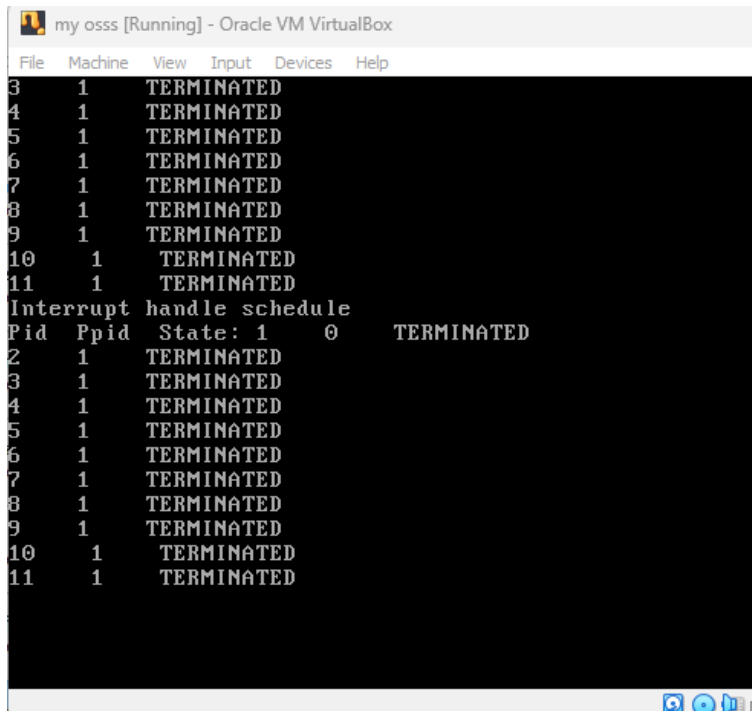
**First Strategy:** Waitpid(i+2) because my parent process starts with pid 1.

```
void FirstStrategy() {  
    // First strategy: Randomly select a task and run it 10 times  
    common::uint32_t pid1 = 0;  
    int parentpid = getpid();  
    int loop_time = 10;  
  
    int randomNumber = generateRandomNumber(1,4);  
  
    printf("Random Number: ");  
    printInt(randomNumber);  
    printf("\n");  
  
    for(int i=0; i < loop_time; i++){  
        // printf("Loop: ");  
        // printInt(i+1);  
        // printf("\n");  
  
        switch (randomNumber)  
        {  
            case 1:  
                pid1 = fork(parentpid);  
                if(pid1 == 0){  
                    taskCollatz();  
                    exit();  
                }  
                break;  
            case 2:  
                pid1 = fork(parentpid);  
                if(pid1 == 0){  
                    taskBinarySearch();  
                    exit();  
                }  
                break;  
            case 3:  
                pid1 = fork(parentpid);  
                if(pid1 == 0){  
                    taskLinearSearch();  
                    exit();  
                }  
                break;  
            case 4:  
                pid1 = fork(parentpid);  
                if(pid1 == 0){
```

```
                case 4:  
                    pid1 = fork(parentpid);  
                    if(pid1 == 0){  
                        common::uint32_t res;  
                        res = long_running_program(1000);  
                        printInt(res);  
                        exit();  
                    }  
                    break;  
                default:  
                    break;  
            }  
        }  
  
        // The parent process waits for all child processes to complete  
        for (int i = 0; i < loop_time; i++) {  
            waitpid(i+2);  
            //printf("\n");  
        }  
  
        exit();  
    }  
}
```

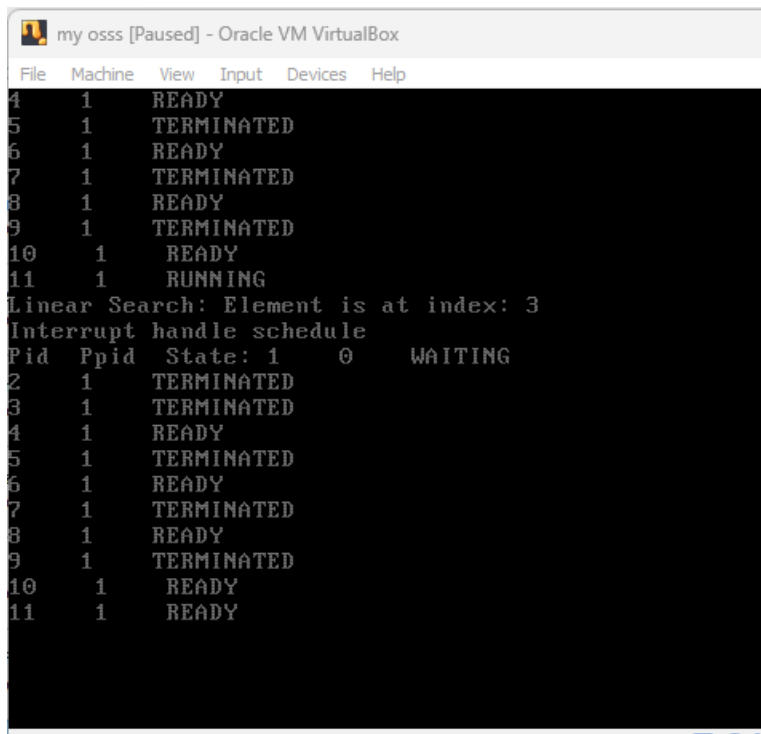
## Outputs:

### With process table:



```
my osss [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
3 1 TERMINATED
4 1 TERMINATED
5 1 TERMINATED
6 1 TERMINATED
7 1 TERMINATED
8 1 TERMINATED
9 1 TERMINATED
10 1 TERMINATED
11 1 TERMINATED
Interrupt handle schedule
Pid Ppid State: 1 0 TERMINATED
2 1 TERMINATED
3 1 TERMINATED
4 1 TERMINATED
5 1 TERMINATED
6 1 TERMINATED
7 1 TERMINATED
8 1 TERMINATED
9 1 TERMINATED
10 1 TERMINATED
11 1 TERMINATED
```

### During the execution:



```
my osss [Paused] - Oracle VM VirtualBox
File Machine View Input Devices Help
4 1 READY
5 1 TERMINATED
6 1 READY
7 1 TERMINATED
8 1 READY
9 1 TERMINATED
10 1 READY
11 1 RUNNING
Linear Search: Element is at index: 3
Interrupt handle schedule
Pid Ppid State: 1 0 WAITING
2 1 TERMINATED
3 1 TERMINATED
4 1 READY
5 1 TERMINATED
6 1 READY
7 1 TERMINATED
8 1 READY
9 1 TERMINATED
10 1 READY
11 1 READY
```

### Without process table:

The screenshot shows a Windows XP desktop environment running as a virtual machine. A command prompt window titled "C:\Program Files\Oracle VM\notebook" contains the following text:  

```
Hello World! --- http://www.AlgorithMan.de  
Random Number: 1  
Interrupt handle schedule  
Collatz sequence for 15 : 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1  
Interrupt handle schedule  
Collatz sequence for 15 : 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1  
Interrupt handle schedule  
Collatz sequence for 15 : 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1  
Interrupt handle schedule  
Collatz sequence for 15 : 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1  
Interrupt handle schedule  
Collatz sequence for 15 : 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1  
Interrupt handle schedule  
Collatz sequence for 15 : 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1  
Interrupt handle schedule  
Collatz sequence for 15 : 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1  
Interrupt handle schedule  
Collatz sequence for 15 : 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1  
Interrupt handle schedule  
Collatz sequence for 15 : 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1  
Interrupt handle schedule  
Collatz sequence for 15 : 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1  
Interrupt handle schedule
```

  
At the bottom of the screen, there are icons for various applications like Internet Explorer, Firefox, and Skype, along with system tray icons for volume and network status.[illegible]

**Second Strategy:** There are 2 for loops to run 2 different task.

```
void SecondStrategy() {  
  
    // Second strategy: Choosing 2 out 4 programs randomly and loading each program 3 times  
    int randomNumber1 = generateRandomNumber(1,4);  
    int randomNumber2 = generateRandomNumber(1,4);  
  
    while(randomNumber1==randomNumber2){  
        randomNumber2 = generateRandomNumber(1,4); // Make sure that the two random numbers are different  
    }  
  
    common::uint32_t pid2=0;  
    common::uint32_t pid3=0;  
  
    int parentpid = getpid();  
  
    printf("Random Number: ");  
    printInt(randomNumber1);  
    printf(" ");  
    printInt(randomNumber2);  
    printf("\n");  
  
    for(int i = 0; i < 3; i++){  
        switch (randomNumber1)  
        {  
            case 1:  
                pid2 = fork(parentpid);  
                if(pid2 == 0){  
                    taskCollatz();  
                    exit();  
                }  
                break;  
            case 2:  
                pid2 = fork(parentpid);  
                if(pid2 == 0){  
                    taskBinarySearch();  
                    exit();  
                }  
                break;  
            case 3:  
                pid2 = fork(parentpid);  
                if(pid2 == 0){  
                    taskLinearSearch();  
                    exit();  
                }  
                break;  
        }  
    }  
}
```

```

        break;
    case 3:
        pid2 = fork(parentpid);
        if(pid2 == 0){
            taskLinearSearch();
            _exit(0);
        }
        break;
    case 4:
        pid2 = fork(parentpid);
        if(pid2 == 0){
            common::uint32_t res;
            res = long_running_program(1000);
            printInt(res);
            _exit(0);
        }
        break;
    default:
        break;
}

}

for(int i = 0; i < 3; i++){
    switch (randomNumber2)
    {
        case 1:
            pid3 = fork(parentpid);
            if(pid3 == 0){
                taskCollatz();
                _exit(0);
            }
            break;
        case 2:
            pid3 = fork(parentpid);
            if(pid3 == 0){
                taskBinarySearch();
                _exit(0);
            }
            break;
        case 3:
            pid3 = fork(parentpid);
            if(pid3 == 0){
                taskLinearSearch();
                _exit(0);
            }
    }
}

```

```

        }
        break;
    case 3:
        pid3 = fork(parentpid);
        if(pid3 == 0){
            taskLinearSearch();
            exitt();
        }
        break;
    case 4:
        pid3 = fork(parentpid);
        if(pid3 == 0){
            common::uint32_t res;
            res = long_running_program(1000);
            printInt(res);
            exitt();
        }
        break;
    default:
        break;
}

}

// The parent process waits for all child processes to complete
for (int i = 0; i < 6; i++) {
    waitpidd(i+2);
    //printf("\n");
}

exitt();
}

```

**Outputs:** You can observe the round robin scheduling with timer interrupt from the different orders of processes.

```
my osss [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Hello World! --- http://www.AlgorithMan.de
Random Number: 1 2
Interrupt handle schedule
Binary Search: Element is at index 4
Interrupt handle schedule
Collatz sequence for 15 : 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
Interrupt handle schedule
Collatz sequence for 15 : 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
Interrupt handle schedule
Binary Search: Element is at index 4
Interrupt handle schedule
Collatz sequence for 15 : 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
Interrupt handle schedule
Binary Search: Element is at index 4
Interrupt handle schedule
Interrupt handle schedule
```

```
my osss [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Hello World! --- http://www.AlgorithMan.de
Random Number: 1 3
Interrupt handle schedule
Collatz sequence for 15 : 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
Interrupt handle schedule
Collatz sequence for 15 : 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
Interrupt handle schedule
Linear Search: Element is at index: 3
Interrupt handle schedule
Interrupt handle schedule
Linear Search: Element is at index: 3
Interrupt handle schedule
Collatz sequence for 15 : 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
Interrupt handle schedule
Interrupt handle schedule
Linear Search: Element is at index: 3
Interrupt handle schedule
Interrupt handle schedule
```