Operating System
2024, Spring

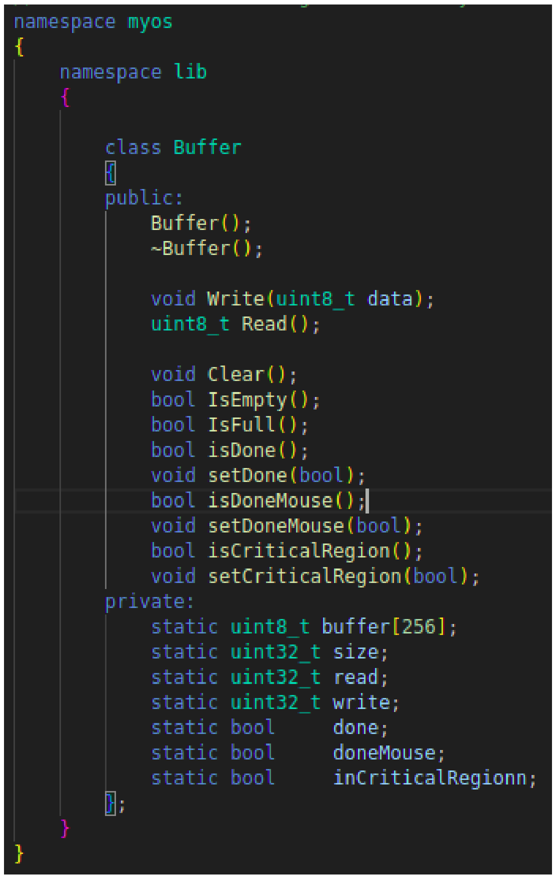
Homework | Project - 01

Ahmet Yigit - 200104004066

Custom Implementations

- Multitasking.h - Multitasking.cpp
    - Task Class
        - pid - ppid - priority - state - parentWait - runningTime
            - added
        - Task default constructor added
    - Task Manager
        - GlobalDescriptorTable - Fork - Exit - Wait - printPrcessesTable -getPid - IncreasePriority
            - added
    - With this implementations, I can control every proces in my os.
    - Processes can do exit-fork etc operations.
    - There are a lot of custom schedulers for each task. Because there is a lot of design differently for each scheduler.
    - Fork's working way is to copy current task information to new task (forked) and copy current state to forked task.
- Syscalls.h - Syscalls.cpp
    - Exit-Fork-Waitpid-Getpid
        - added
    - Adding parameter to current constructor to get taskManager.
    - With taskManager in syscalls i can do whatever i want if there is fork/exit or something else i can decide what i do every interrupt.

- Also with coding with assembly i can move one every interrupt to specialized taskManager function.

- Types.h
  - State, Priority and pid_t
    - added.
- Buffer.h - Buffer.cpp
  - That is my class to handle all of the user inputs.
  - All variables are static.
    - When user input a num it's add to buffer
    - Whenever there is a need to get data from a buffer, first of all the function clears the buffer and takes user input.
    - It gives user input char by char FIFO, so conservation to number/decimal is so simple.
    - Because variables are static, every chapter of the OS can move on the same buffer. That is the reason why I use this technique.
    - 

```
namespace myos
{
    namespace lib
    {

        class Buffer
        {
        public:
            Buffer();
            ~Buffer();

            void Write(uint8_t data);
            uint8_t Read();

            void Clear();
            bool IsEmpty();
            bool IsFull();
            bool isDone();
            void setDone(bool);
            bool isDoneMouse();
            void setDoneMouse(bool);
            bool isCriticalRegion();
            void setCriticalRegion(bool);
        private:
            static uint8_t buffer[256];
            static uint32_t size;
            static uint32_t read;
            static uint32_t write;
            static bool     done;
            static bool     doneMouse;
            static bool     inCriticalRegionn;
        };
    }
}
```

- Stdlib.h - Stdlib.cpp
    - Linear - binary searches, collatz algorithm also long_running_program definition are in here.
    - Also needed functions like
        - printf - getIntegerFromKeyboard - readBuffer - longXXX
    - GetIntegerFromKeyboard
        - It uses a buffer and it reads until the user hits the enter button.
        - It has a **CRITICAL REGION.**
    - xxxLong
        - They are nested in two loops to get other interrupts.
    - Others are well known search algorithms.
- Kernel.cpp
    - There is a definition of a lot of functions to handle given tasks.
    - If you want to run strategy-tasks you have to uncomment the lines like in the test cases. SOME OF THEM NEED UNCOMMENT ALSO SCHEDULER.
    - There is no change to the main point.
- Mouse.cpp - Keyboard.cpp
    - When user input is detected they save in the buffer.

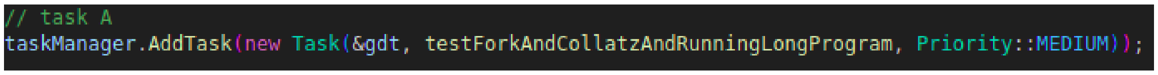
Design Think / Working Mentality Basically

- First of all I add init function
    - When the init function forks there are at least two processes one of them is init and other(s) are runnable processes.
    - When interrupt occurs
    - If a program in a critical region is going on the same esp.
    - Otherwise
        - It goes scheduler
            - Scheduler decides what is the next process to go or start.
            - Robin Round - It get at closest ready one
            - Preemptive - It gets the most priority.
    - When all processes die you can see that only the init process is running.

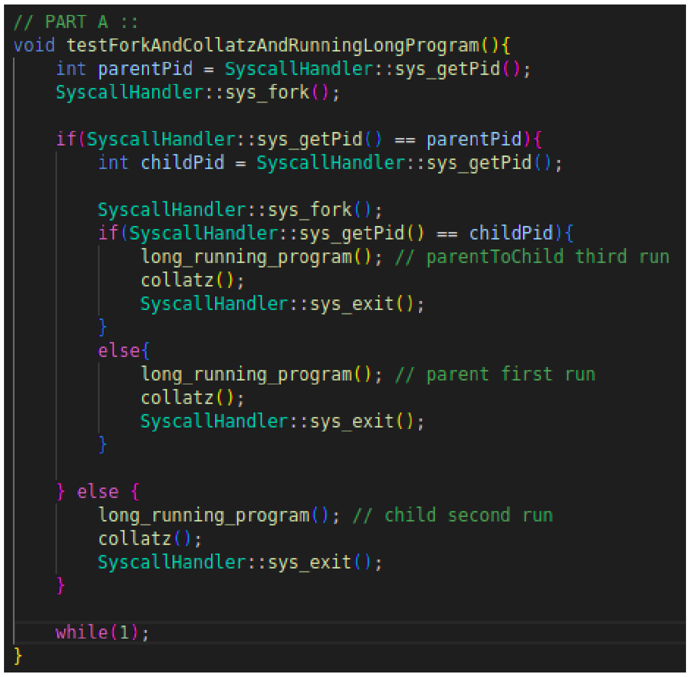
README.TXT | Working Steps

- **Task A**
    - Uncomment addTask for TASK A in kernel.cpp
    - make clean
    - make
- **Task B**
    - **Strategy 1**
        - Uncomment addTask for TASK B in kernel.cpp / firstStrategy
        - make clean
        - make
    - **Strategy 2**
        - Uncomment addTask for TASK B in kernel.cpp second strategy
        - make clean
        - make
    - **Strategy 3**
        - Uncomment addTask for TASK B in kernel.cpp third strategy
        - Make changes in line 78 in multitasking.cpp
        - make clean 'enter' make
    - **Strategy 4**
        - Uncomment addTask for TASK A in kernel.cpp also TASK 4 icin scheduler in multitasking.cpp also comment current scheduler
        - make clean 'enter' make
- **Task C**
    - **Strategy 1**
        - Uncomment addTask for TASK C in kernel.cpp / task c first strategy
        - make clean
        - make
    - **Strategy 2**
        - Uncomment addTask for TASK C in kernel.cpp / task c second strategy
        - Uncomment else part line 196-197 in interrupts.cpp make clean make

Task A

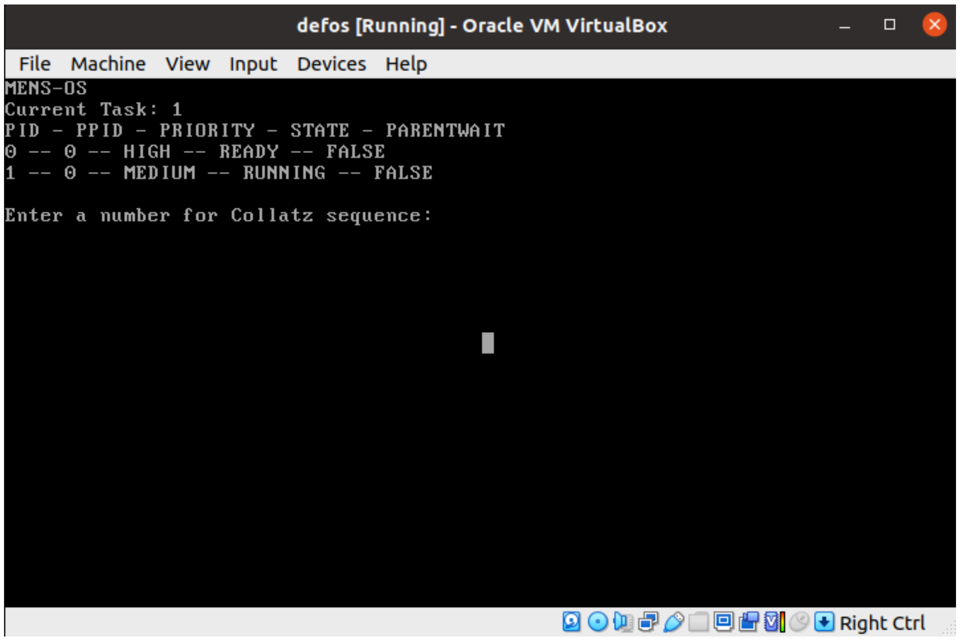
- System calls addTask in mainKernel and goes
  - testForkAndCollatzAndRunningLongProgram
  - In here there are 2 fork system calls for creating 2^2=4 processes but we do not do anything in the parent process because there is a need to create 3 processes in the definition of homework.
- 

```
// task A
taskManager.AddTask(new Task(&gdt, testForkAndCollatzAndRunningLongProgram, Priority::MEDIUM));
```

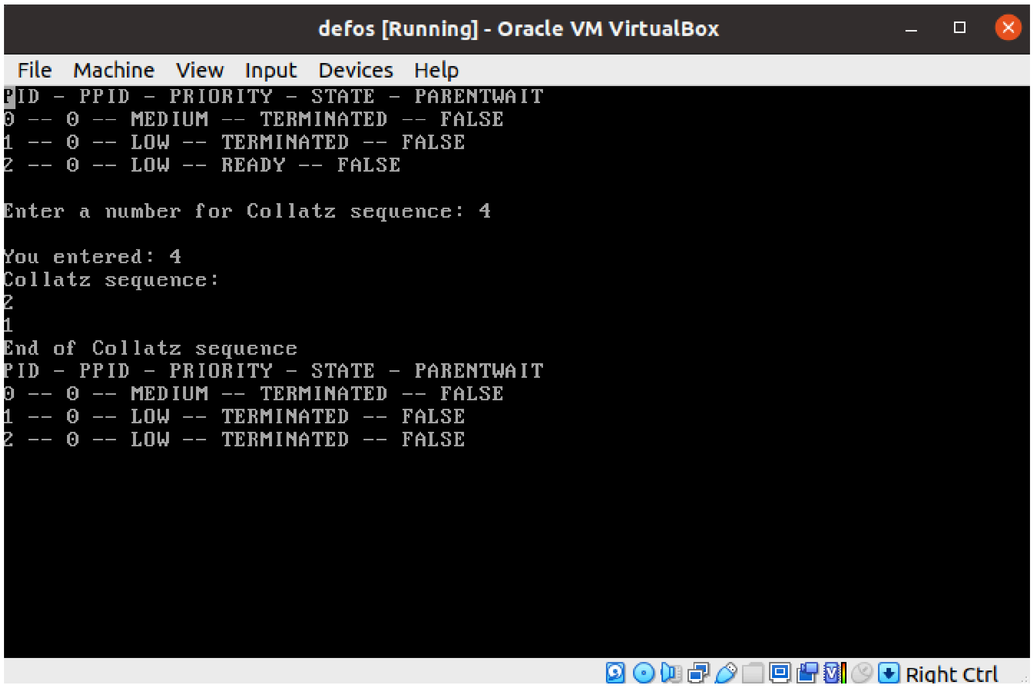


```
// PART A ::
void testForkAndCollatzAndRunningLongProgram(){
    int parentPid = SyscallHandler::sys_getPid();
    SyscallHandler::sys_fork();

    if(SyscallHandler::sys_getPid() == parentPid){
        int childPid = SyscallHandler::sys_getPid();

        SyscallHandler::sys_fork();
        if(SyscallHandler::sys_getPid() == childPid){
            long_running_program(); // parentToChild third run
            collatz();
            SyscallHandler::sys_exit();
        }
        else{
            long_running_program(); // parent first run
            collatz();
            SyscallHandler::sys_exit();
        }

    } else {
        long_running_program(); // child second run
        collatz();
        SyscallHandler::sys_exit();
    }

    while(1);
}
```

- After each long_xx anc collatz functions are done, the system kills these processes.

- 
defos [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

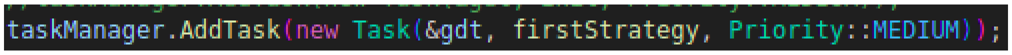
```
MENS-OS
Current Task: 1
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- HIGH -- READY -- FALSE
1 -- 0 -- MEDIUM -- RUNNING -- FALSE

Enter a number for Collatz sequence:
```

- 
defos [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

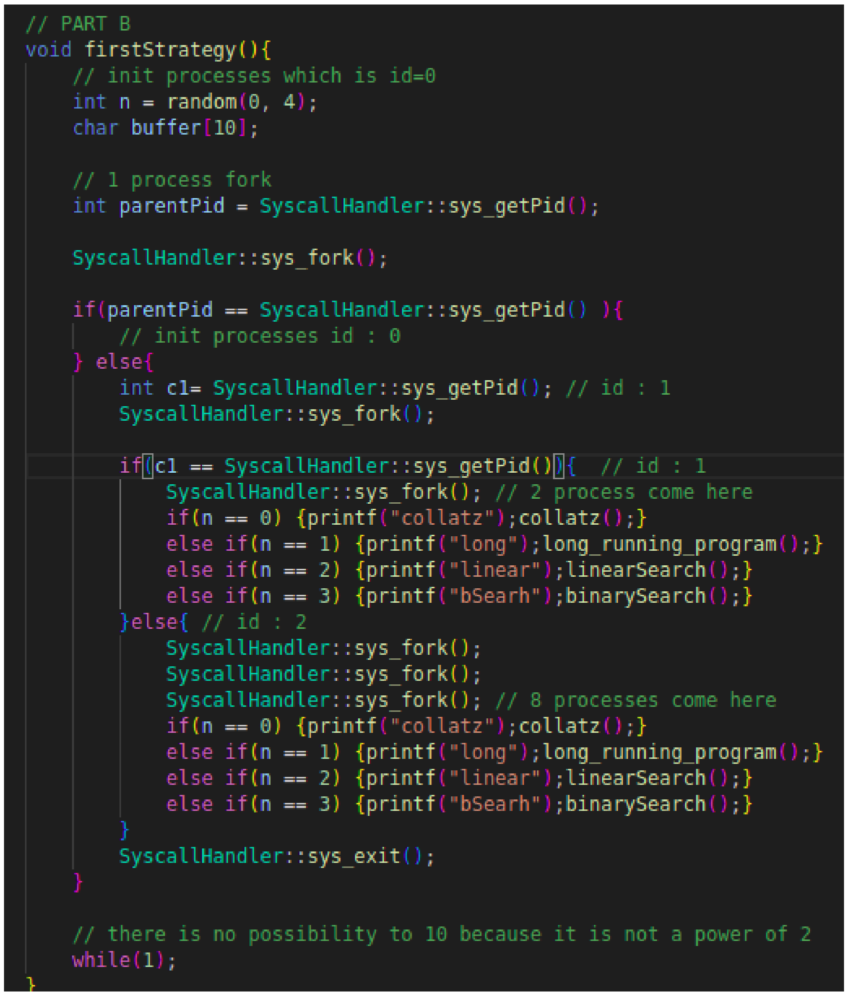
```
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- MEDIUM -- TERMINATED -- FALSE
1 -- 0 -- LOW -- TERMINATED -- FALSE
2 -- 0 -- LOW -- READY -- FALSE

Enter a number for Collatz sequence: 4

You entered: 4
Collatz sequence:
2
1
End of Collatz sequence
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- MEDIUM -- TERMINATED -- FALSE
1 -- 0 -- LOW -- TERMINATED -- FALSE
2 -- 0 -- LOW -- TERMINATED -- FALSE
```

- Medium priority is init processes or parent processes and pid=1 and pid=2 are the child processes which are low priority by default. After all processes are done, all of them are terminated.

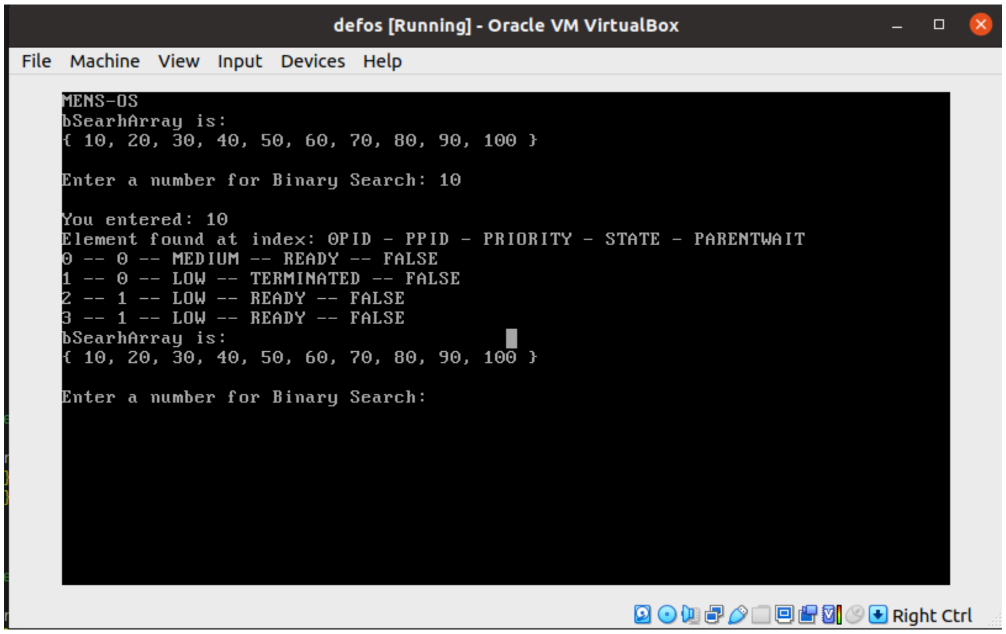
Task B

- ***First Strategy***
  - There is an init process which is always run and its child process runs 10 times at random selected functions.
  - I have to implement 3 forks after each other because I have to get 8 processes in here and the other two processes come from one fork which is scooped like the code.
  - 

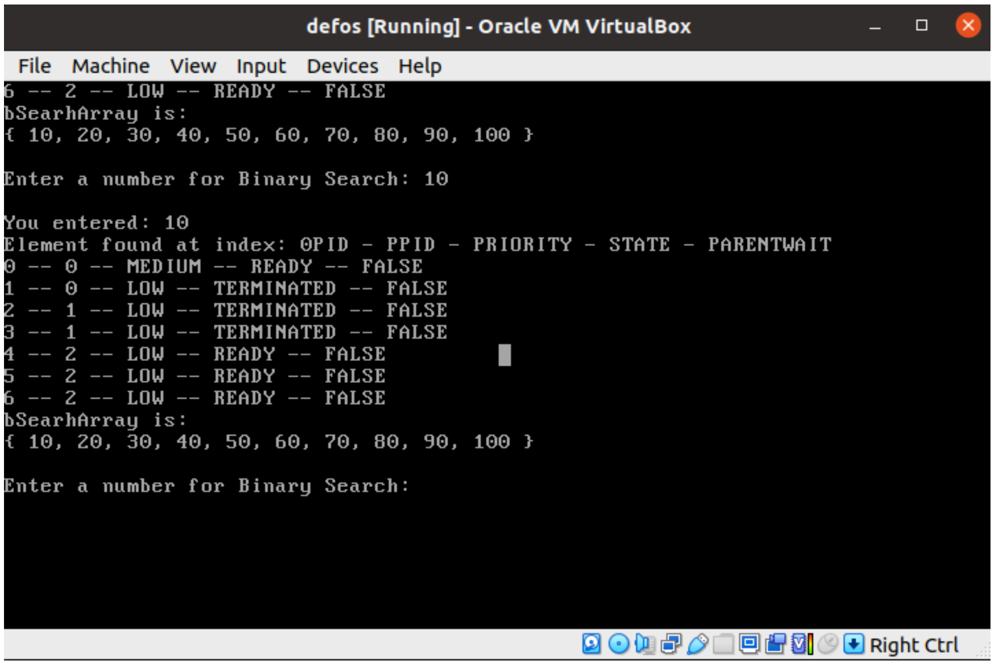
```
taskManager.AddTask(new Task(&gdt, firstStrategy, Priority::MEDIUM));
```

  - 

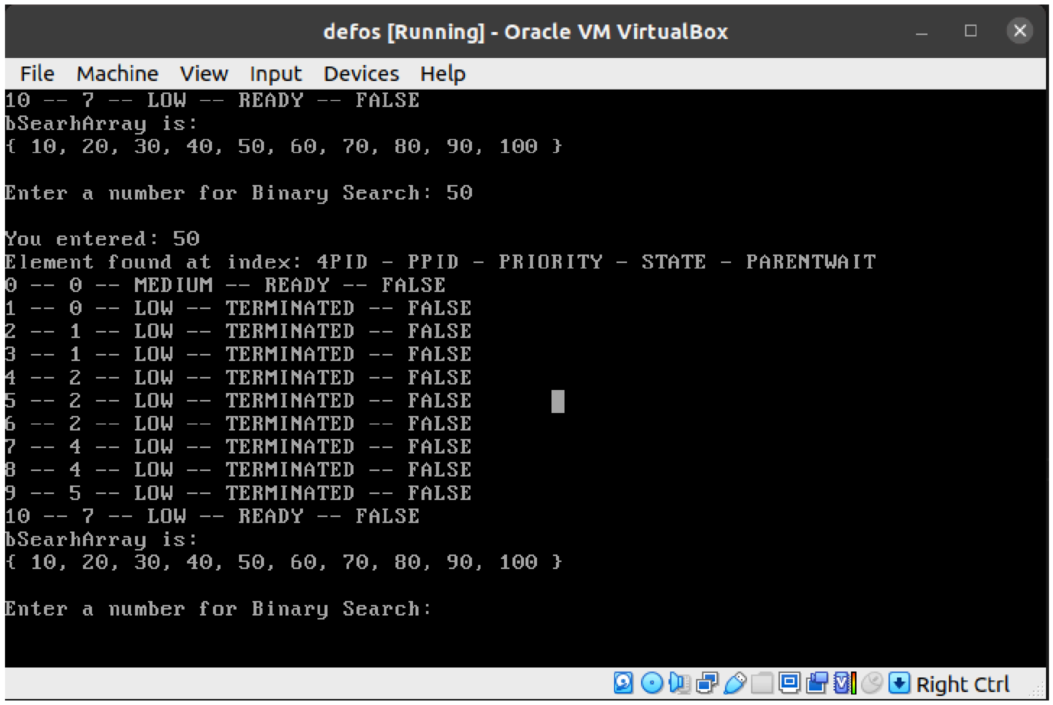
```
// PART B
void firstStrategy(){
    // init processes which is id=0
    int n = random(0, 4);
    char buffer[10];

    // 1 process fork
    int parentPid = SyscallHandler::sys_getPid();

    SyscallHandler::sys_fork();

    if(parentPid == SyscallHandler::sys_getPid() ){
        // init processes id : 0
    } else{
        int c1= SyscallHandler::sys_getPid(); // id : 1
        SyscallHandler::sys_fork();

        if(c1 == SyscallHandler::sys_getPid()){  // id : 1
            SyscallHandler::sys_fork(); // 2 process come here
            if(n == 0) {printf("collatz");collatz();}
            else if(n == 1) {printf("long");long_running_program();}
            else if(n == 2) {printf("linear");linearSearch();}
            else if(n == 3) {printf("bSearh");binarySearch();}
        }else{ // id : 2
            SyscallHandler::sys_fork();
            SyscallHandler::sys_fork();
            SyscallHandler::sys_fork(); // 8 processes come here
            if(n == 0) {printf("collatz");collatz();}
            else if(n == 1) {printf("long");long_running_program();}
            else if(n == 2) {printf("linear");linearSearch();}
            else if(n == 3) {printf("bSearh");binarySearch();}
        }
        SyscallHandler::sys_exit();
    }

    // there is no possibility to 10 because it is not a power of 2
    while(1);
}
```

- 
defos [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

```
MENS-OS
bSearhArray is:
{ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 }

Enter a number for Binary Search: 10

You entered: 10
Element found at index: 0PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- MEDIUM -- READY -- FALSE
1 -- 0 -- LOW -- TERMINATED -- FALSE
2 -- 1 -- LOW -- READY -- FALSE
3 -- 1 -- LOW -- READY -- FALSE
bSearhArray is:
{ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 }

Enter a number for Binary Search:
```

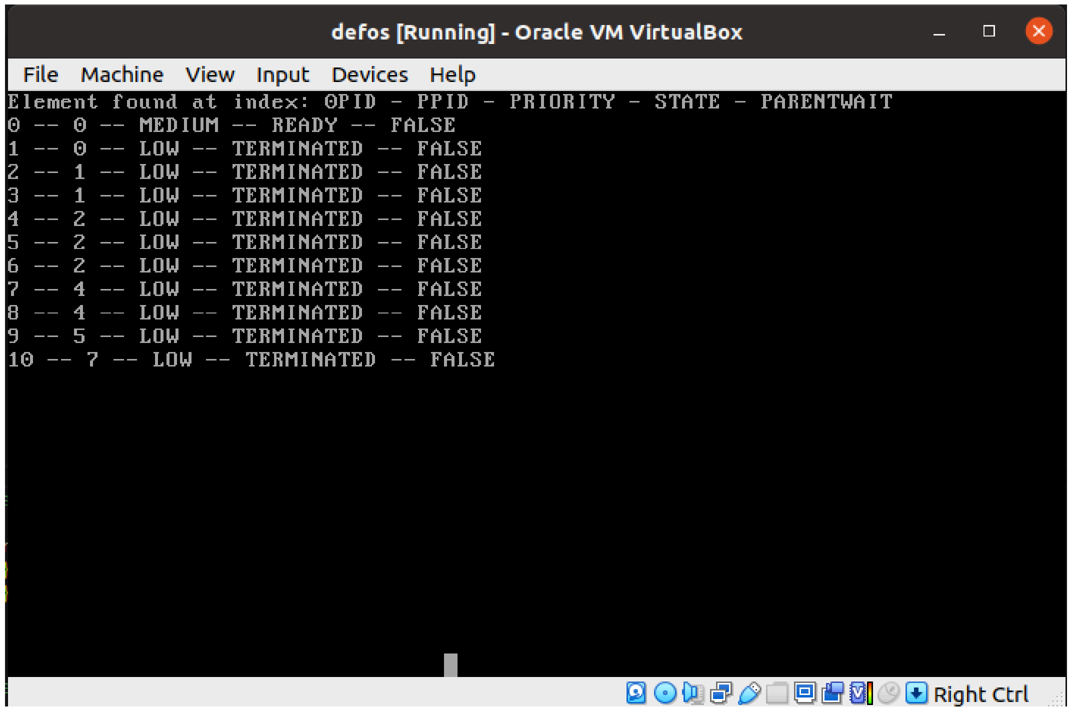
- 
defos [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

```
6 -- 2 -- LOW -- READY -- FALSE
bSearhArray is:
{ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 }

Enter a number for Binary Search: 10

You entered: 10
Element found at index: 0PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- MEDIUM -- READY -- FALSE
1 -- 0 -- LOW -- TERMINATED -- FALSE
2 -- 1 -- LOW -- TERMINATED -- FALSE
3 -- 1 -- LOW -- TERMINATED -- FALSE
4 -- 2 -- LOW -- READY -- FALSE
5 -- 2 -- LOW -- READY -- FALSE
6 -- 2 -- LOW -- READY -- FALSE
bSearhArray is:
{ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 }

Enter a number for Binary Search:
```

- 

defos [Running] - Oracle VM VirtualBox

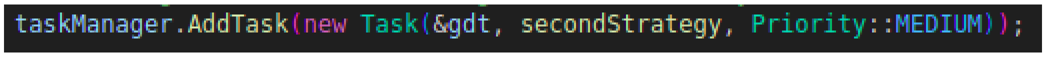
```
10 -- 7 -- LOW -- READY -- FALSE
bSearhArray is:
{ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 }

Enter a number for Binary Search: 50

You entered: 50
Element found at index: 4PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- MEDIUM -- READY -- FALSE
1 -- 0 -- LOW -- TERMINATED -- FALSE
2 -- 1 -- LOW -- TERMINATED -- FALSE
3 -- 1 -- LOW -- TERMINATED -- FALSE
4 -- 2 -- LOW -- TERMINATED -- FALSE
5 -- 2 -- LOW -- TERMINATED -- FALSE
6 -- 2 -- LOW -- TERMINATED -- FALSE
7 -- 4 -- LOW -- TERMINATED -- FALSE
8 -- 4 -- LOW -- TERMINATED -- FALSE
9 -- 5 -- LOW -- TERMINATED -- FALSE
10 -- 7 -- LOW -- READY -- FALSE
bSearhArray is:
{ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 }

Enter a number for Binary Search:
```

- 

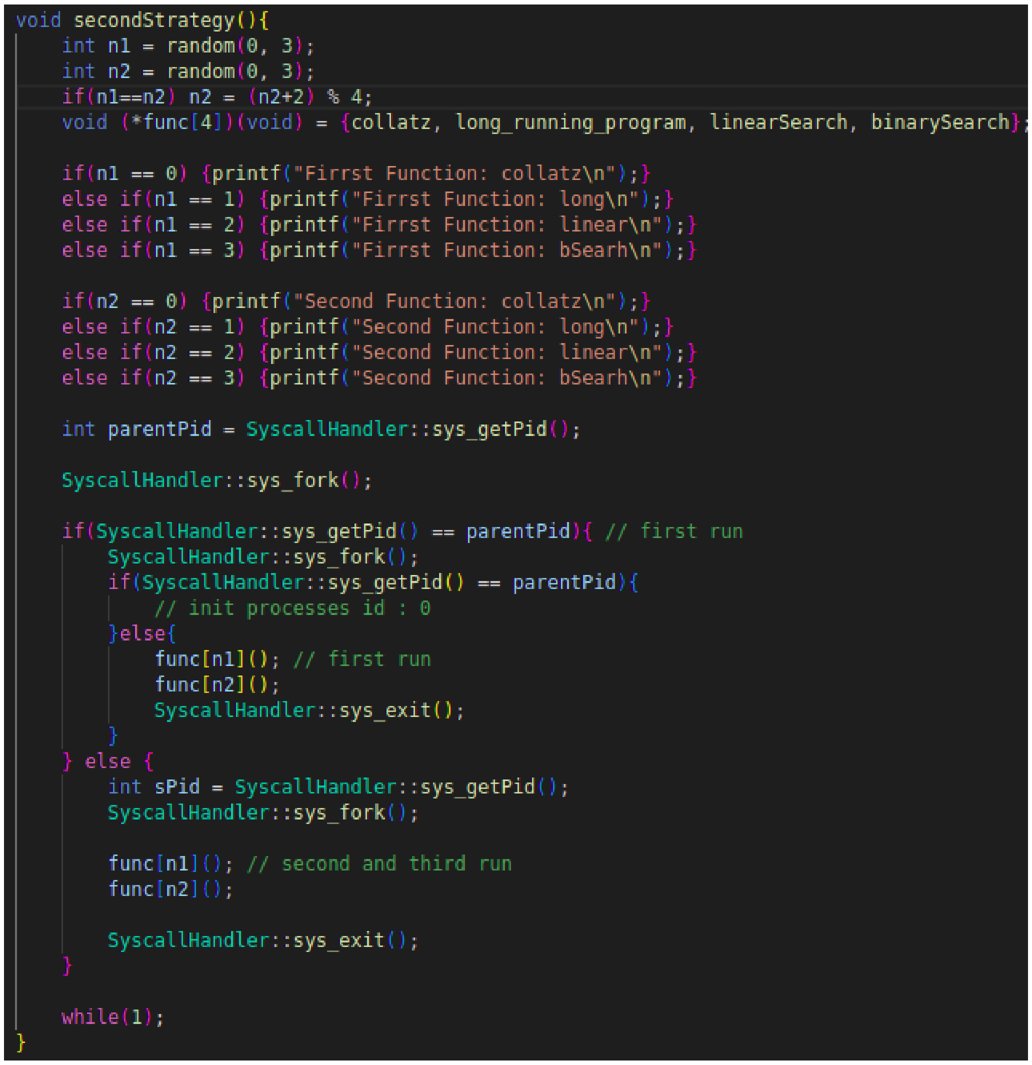
defos [Running] - Oracle VM VirtualBox

```
Element found at index: 0PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- MEDIUM -- READY -- FALSE
1 -- 0 -- LOW -- TERMINATED -- FALSE
2 -- 1 -- LOW -- TERMINATED -- FALSE
3 -- 1 -- LOW -- TERMINATED -- FALSE
4 -- 2 -- LOW -- TERMINATED -- FALSE
5 -- 2 -- LOW -- TERMINATED -- FALSE
6 -- 2 -- LOW -- TERMINATED -- FALSE
7 -- 4 -- LOW -- TERMINATED -- FALSE
8 -- 4 -- LOW -- TERMINATED -- FALSE
9 -- 5 -- LOW -- TERMINATED -- FALSE
10 -- 7 -- LOW -- TERMINATED -- FALSE
```

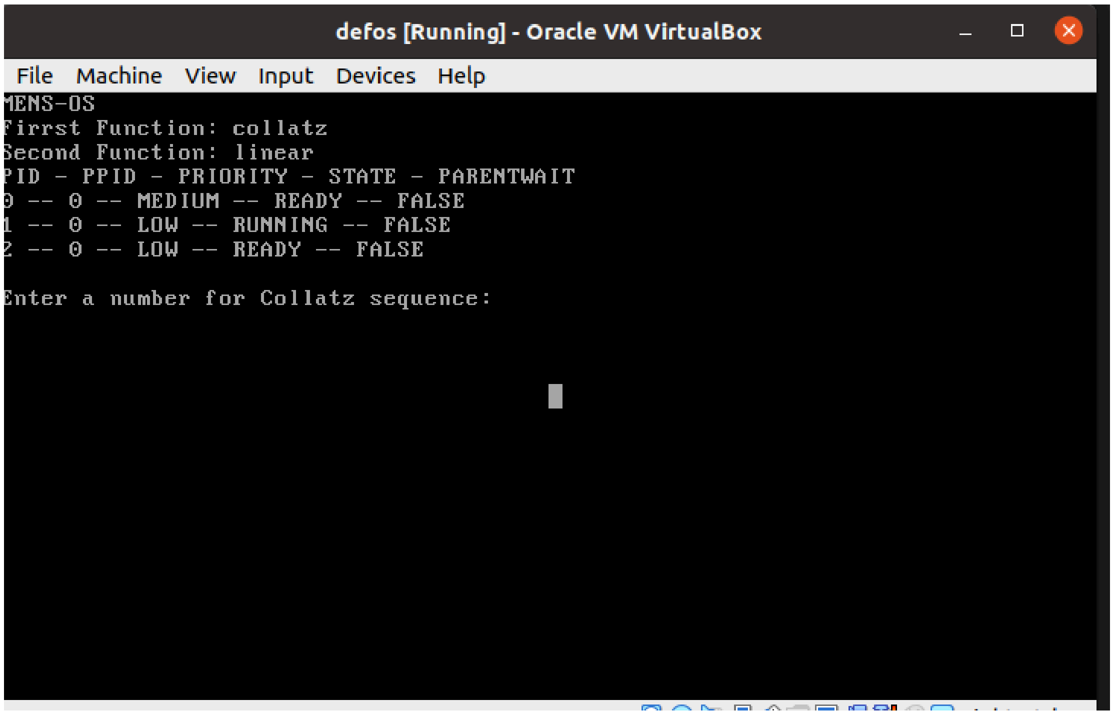
- As you can see, all 10 processes which are created by forks here and their parent processes are defined as expected. Also init processes are still ready to run so it can take process to handle.
- **I PRINTED TABLE AFTER TERMINATINATION OF PROCESSES. SO WE DO NOT SEE ANY RUNNING. AFTER NOW I PRINT ALL OF THEM.**

- **Second Strategy**
    - There is an init process and its 3 child process runs a random selected function. Working mentality is the same as the first strategy.
    - 

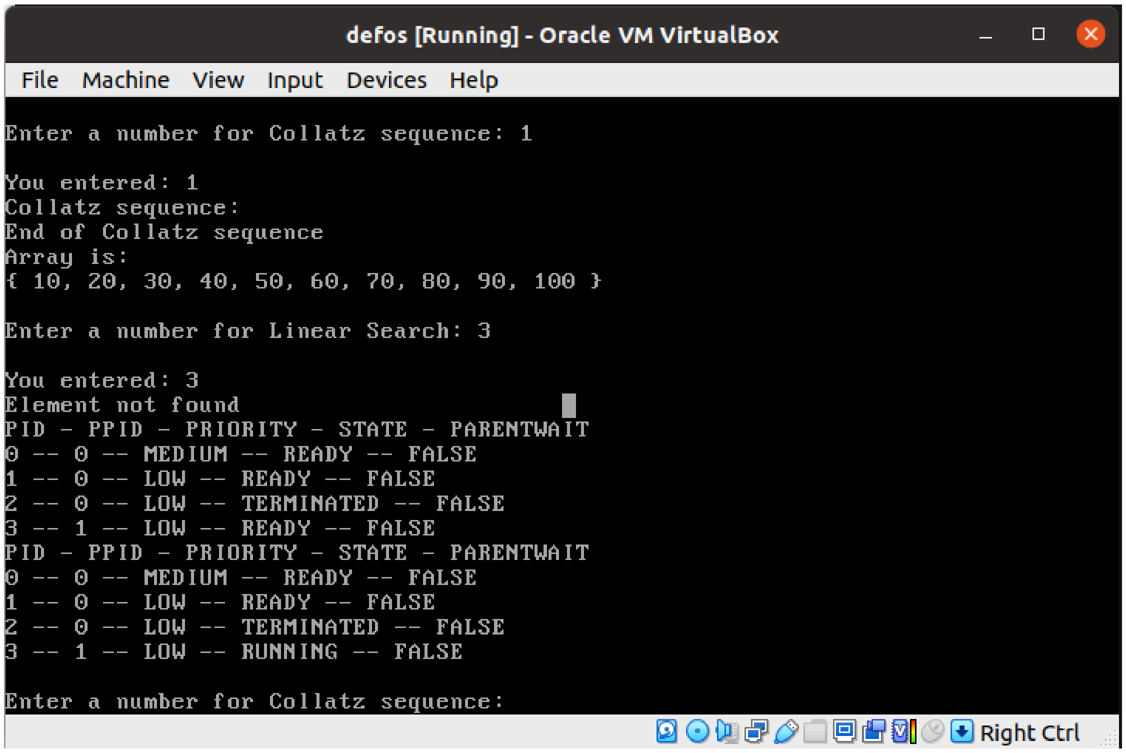
```
taskManager.AddTask(new Task(&gdt, secondStrategy, Priority::MEDIUM));
```

    - 

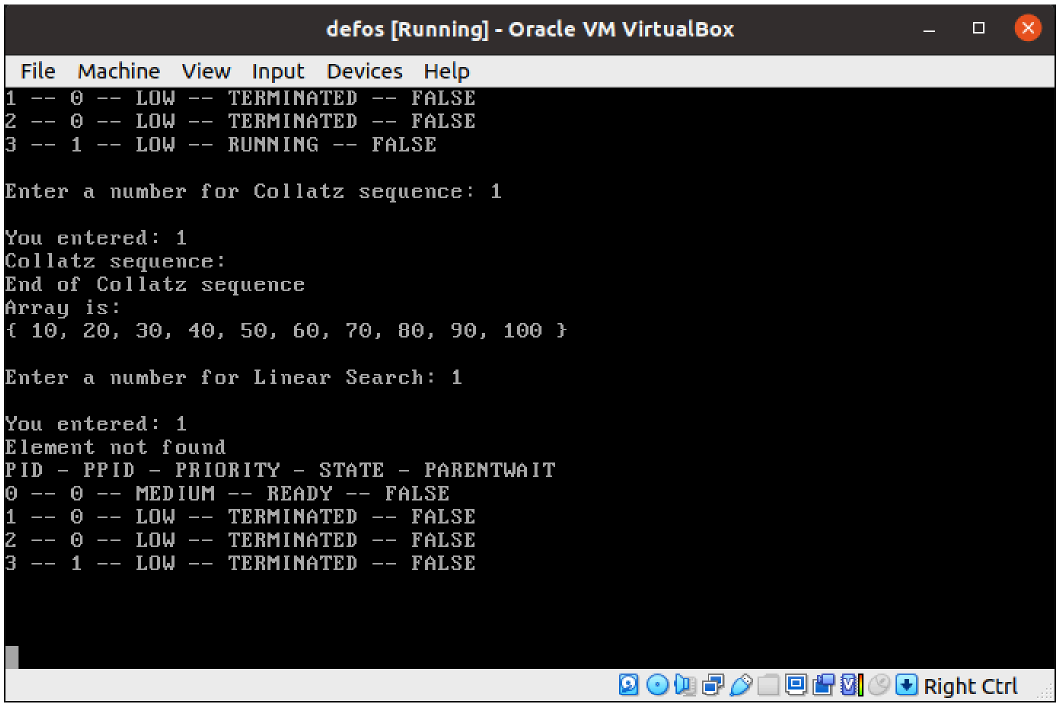
```
void secondStrategy(){
    int n1 = random(0, 3);
    int n2 = random(0, 3);
    if(n1==n2) n2 = (n2+2) % 4;
    void (*func[4])(void) = {collatz, long_running_program, linearSearch, binarySearch};

    if(n1 == 0) {printf("Firrst Function: collatz\n");}
    else if(n1 == 1) {printf("Firrst Function: long\n");}
    else if(n1 == 2) {printf("Firrst Function: linear\n");}
    else if(n1 == 3) {printf("Firrst Function: bSearh\n");}

    if(n2 == 0) {printf("Second Function: collatz\n");}
    else if(n2 == 1) {printf("Second Function: long\n");}
    else if(n2 == 2) {printf("Second Function: linear\n");}
    else if(n2 == 3) {printf("Second Function: bSearh\n");}

    int parentPid = SyscallHandler::sys_getPid();

    SyscallHandler::sys_fork();

    if(SyscallHandler::sys_getPid() == parentPid){ // first run
        SyscallHandler::sys_fork();
        if(SyscallHandler::sys_getPid() == parentPid){
            // init processes id : 0
        }else{
            func[n1](); // first run
            func[n2]();
            SyscallHandler::sys_exit();
        }
    } else {
        int sPid = SyscallHandler::sys_getPid();
        SyscallHandler::sys_fork();

        func[n1](); // second and third run
        func[n2]();

        SyscallHandler::sys_exit();
    }

    while(1);
}
```

○ 
defos [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

```
MENS-OS
Firrst Function: collatz
Second Function: linear
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- MEDIUM -- READY -- FALSE
1 -- 0 -- LOW -- RUNNING -- FALSE
2 -- 0 -- LOW -- READY -- FALSE

Enter a number for Collatz sequence:
```

○ 
defos [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

```
Enter a number for Collatz sequence: 1

You entered: 1
Collatz sequence:
End of Collatz sequence
Array is:
{ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 }

Enter a number for Linear Search: 3

You entered: 3
Element not found
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- MEDIUM -- READY -- FALSE
1 -- 0 -- LOW -- READY -- FALSE
2 -- 0 -- LOW -- TERMINATED -- FALSE
3 -- 1 -- LOW -- READY -- FALSE
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- MEDIUM -- READY -- FALSE
1 -- 0 -- LOW -- READY -- FALSE
2 -- 0 -- LOW -- TERMINATED -- FALSE
3 -- 1 -- LOW -- RUNNING -- FALSE

Enter a number for Collatz sequence:
```

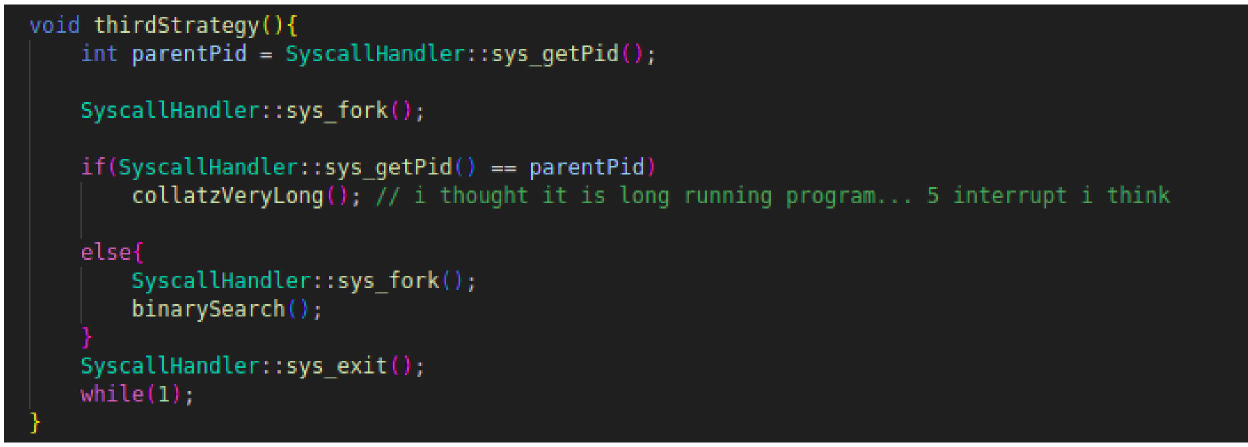
- 
defos [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

```
1 -- 0 -- LOW -- TERMINATED -- FALSE
2 -- 0 -- LOW -- TERMINATED -- FALSE
3 -- 1 -- LOW -- RUNNING -- FALSE

Enter a number for Collatz sequence: 1

You entered: 1
Collatz sequence:
End of Collatz sequence
Array is:
{ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 }

Enter a number for Linear Search: 1

You entered: 1
Element not found
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- MEDIUM -- READY -- FALSE
1 -- 0 -- LOW -- TERMINATED -- FALSE
2 -- 0 -- LOW -- TERMINATED -- FALSE
3 -- 1 -- LOW -- TERMINATED -- FALSE
```

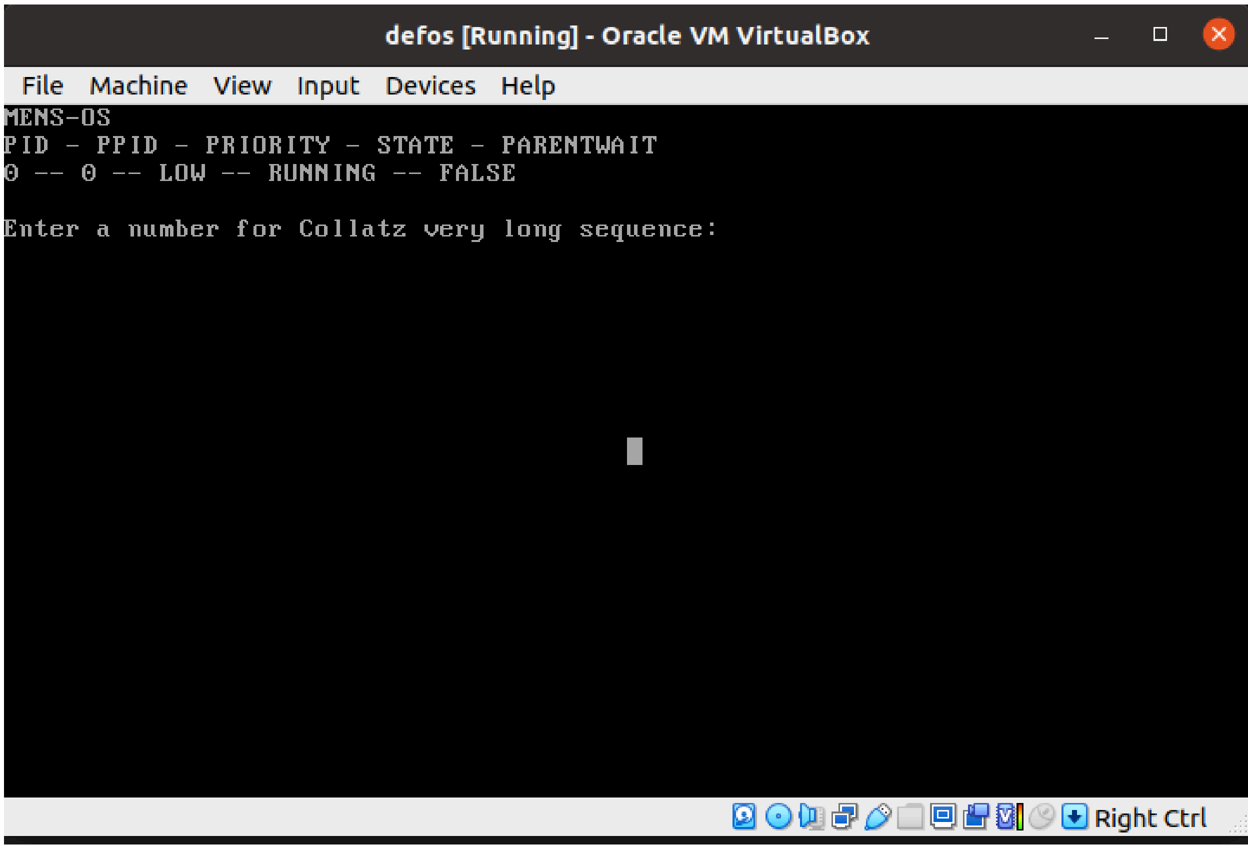
- As you can see, the first and second output contains running operations.
- In the second image after terminating pid=2, pid=3 is ready and starts running after the context switch which is shown in the second printed process table which is in the scheduler. First process table comes from the exit() function.

- **Third Strategy**
  - In here we started to collatzVeryLong and get a lot of time to complete more than 5 interrupts but other binary search is ready to run. It won't run because third strategy is aiming to first in first out if their priority is the same.

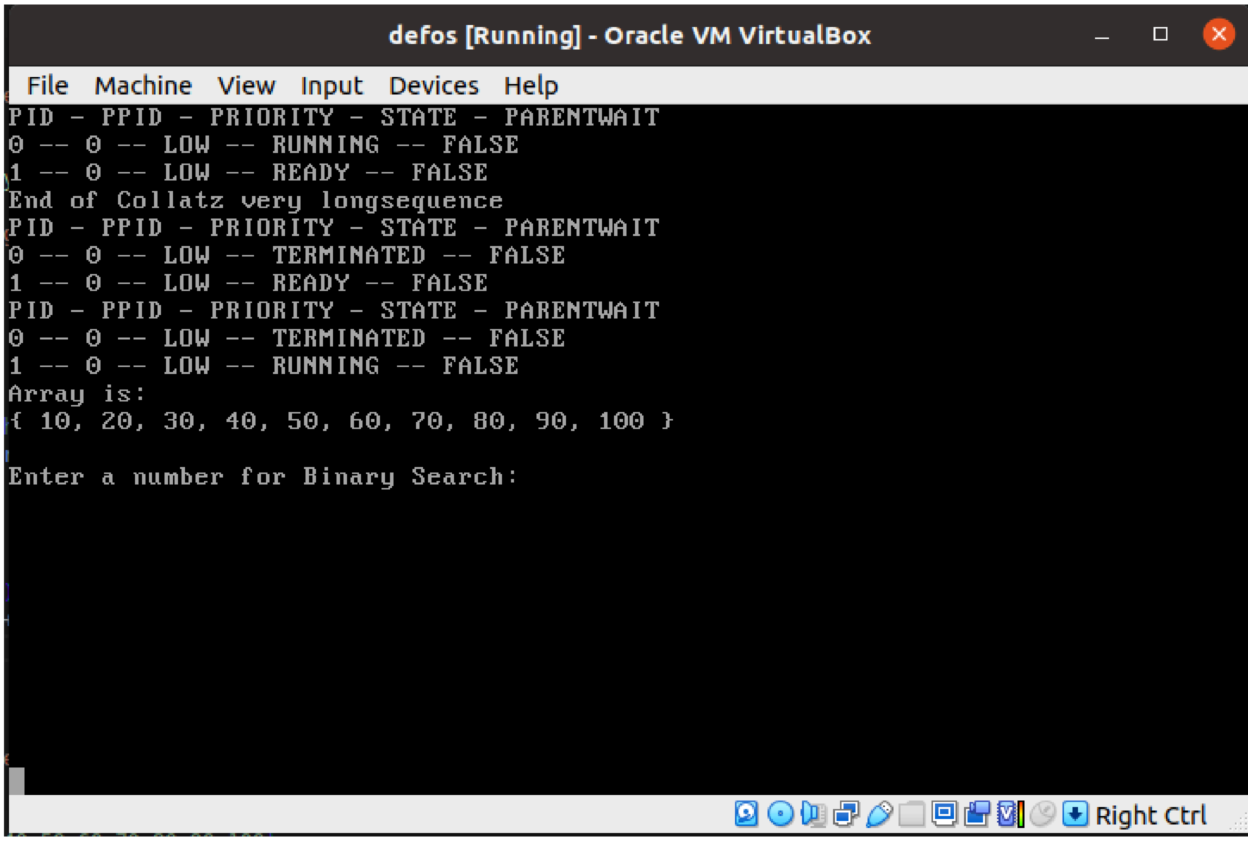
  -
    ```
    taskManager.AddTask(new Task(&gdt, thirdStrategy, Priority::LOW));
    ```



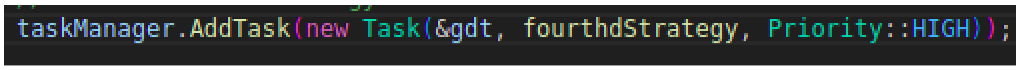
```
void thirdStrategy(){
    int parentPid = SyscallHandler::sys_getPid();

    SyscallHandler::sys_fork();

    if(SyscallHandler::sys_getPid() == parentPid)
        collatzVeryLong(); // i thought it is long running program... 5 interrupt i think

    else{
        SyscallHandler::sys_fork();
        binarySearch();
    }
    SyscallHandler::sys_exit();
    while(1);
}
```


defos [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

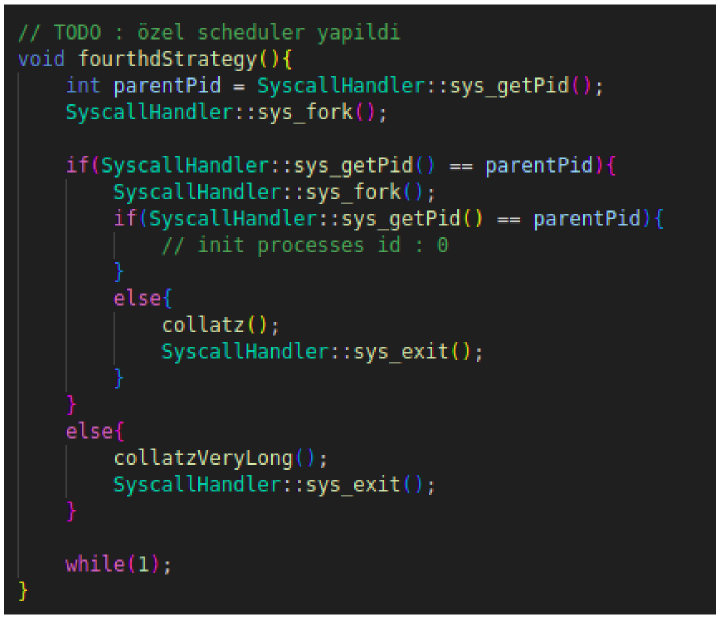
```
MENS-OS
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- LOW -- RUNNING -- FALSE

Enter a number for Collatz very long sequence:
```

- 
defos [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

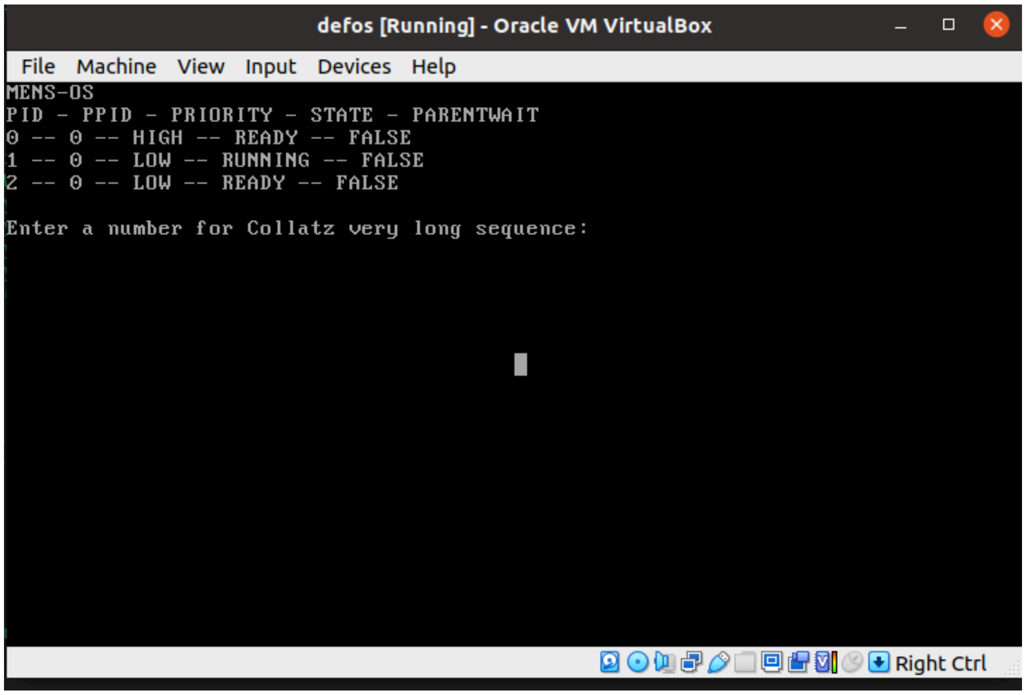
```
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- LOW -- RUNNING -- FALSE
1 -- 0 -- LOW -- READY -- FALSE
End of Collatz very longsequence
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- LOW -- TERMINATED -- FALSE
1 -- 0 -- LOW -- READY -- FALSE
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- LOW -- TERMINATED -- FALSE
1 -- 0 -- LOW -- RUNNING -- FALSE
Array is:
{ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 }

Enter a number for Binary Search:
```

  - You can see that while the first process is running and the second is ready after being terminated the second will run. That is the expected result. If you can get other interrupts in the first process running time there is no context switch to the second process.
  - YOU HAVE TO USE A PREEMPTIVE SCHEDULER FOR THAT TEST CASE! YOU HAVE TO UNCOMMENT THAT LINES AND COMMENT CURRENT SCHEDULER.
- **Fourth Strategy**
  - I use Collactz and very long Collaczt to compare. Because normal Collactz has done one interrupt so it won't change but LongCollactz has done more than 10 interrupt. So it's priority will change. There is an init process as it was.
  - 

```
taskManager.AddTask(new Task(&gdt, fourthdStrategy, Priority::HIGH));
```

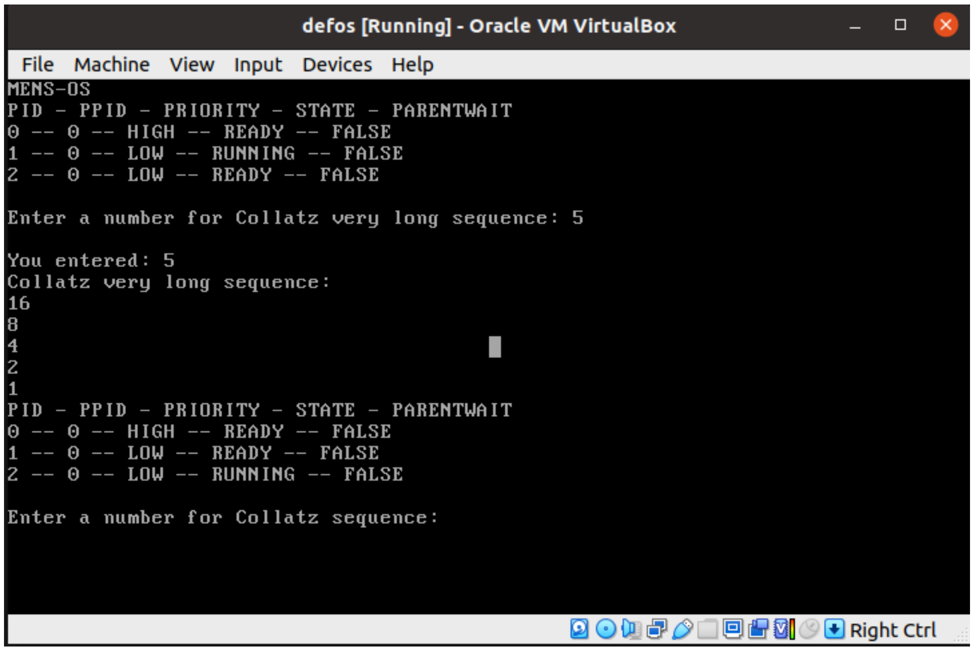


```
// TODO : özel scheduler yapildi
void fourthdStrategy(){
    int parentPid = SyscallHandler::sys_getPid();
    SyscallHandler::sys_fork();

    if(SyscallHandler::sys_getPid() == parentPid){
        SyscallHandler::sys_fork();
        if(SyscallHandler::sys_getPid() == parentPid){
            // init processes id : 0
        }
        else{
            collatz();
            SyscallHandler::sys_exit();
        }
    }
    else{
        collatzVeryLong();
        SyscallHandler::sys_exit();
    }

    while(1);
}
```

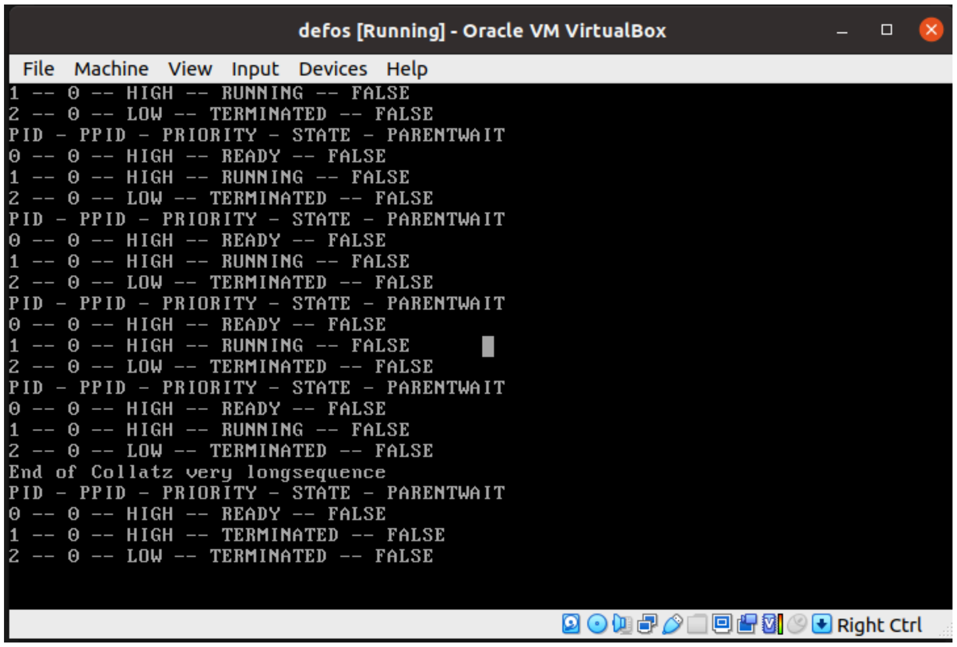


defos [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

```
MENS-OS
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- HIGH -- READY -- FALSE
1 -- 0 -- LOW -- RUNNING -- FALSE
2 -- 0 -- LOW -- READY -- FALSE

Enter a number for Collatz very long sequence:
```

- 
defos [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

```
MENS-OS
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- HIGH -- READY -- FALSE
1 -- 0 -- LOW -- RUNNING -- FALSE
2 -- 0 -- LOW -- READY -- FALSE

Enter a number for Collatz very long sequence: 5

You entered: 5
Collatz very long sequence:
16
8
4
2
1
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- HIGH -- READY -- FALSE
1 -- 0 -- LOW -- READY -- FALSE
2 -- 0 -- LOW -- RUNNING -- FALSE

Enter a number for Collatz sequence:
```

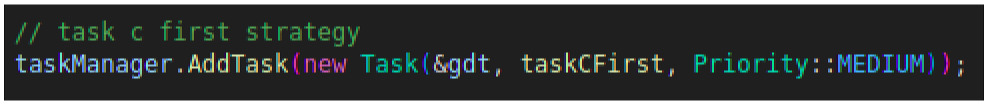
- 
defos [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

```
1 -- 0 -- HIGH -- RUNNING -- FALSE
2 -- 0 -- LOW -- TERMINATED -- FALSE
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- HIGH -- READY -- FALSE
1 -- 0 -- HIGH -- RUNNING -- FALSE
2 -- 0 -- LOW -- TERMINATED -- FALSE
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- HIGH -- READY -- FALSE
1 -- 0 -- HIGH -- RUNNING -- FALSE
2 -- 0 -- LOW -- TERMINATED -- FALSE
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- HIGH -- READY -- FALSE
1 -- 0 -- HIGH -- RUNNING -- FALSE
2 -- 0 -- LOW -- TERMINATED -- FALSE
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- HIGH -- READY -- FALSE
1 -- 0 -- HIGH -- RUNNING -- FALSE
2 -- 0 -- LOW -- TERMINATED -- FALSE
End of Collatz very longsequence
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- HIGH -- READY -- FALSE
1 -- 0 -- HIGH -- TERMINATED -- FALSE
2 -- 0 -- LOW -- TERMINATED -- FALSE
```

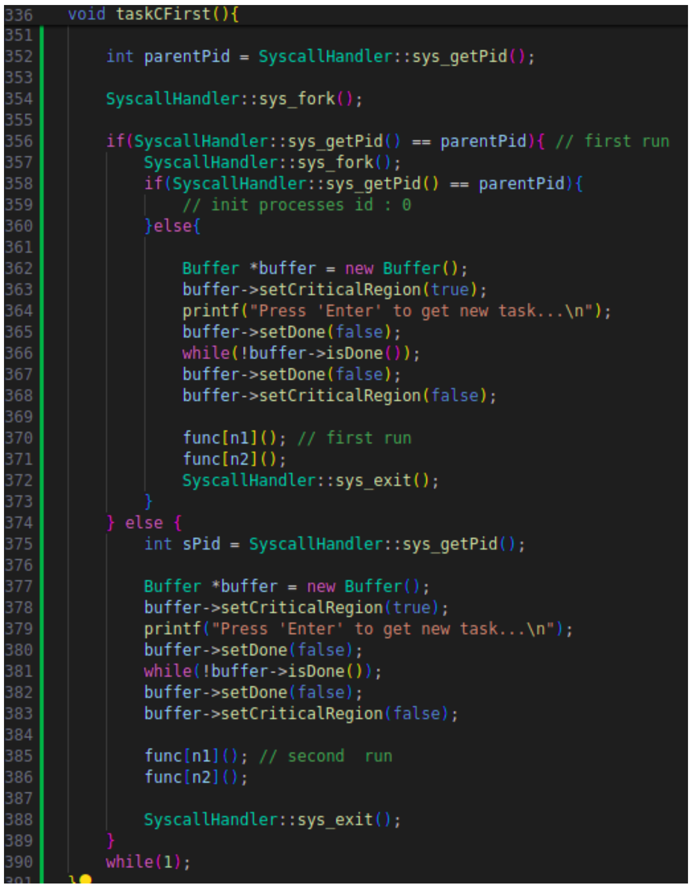
- Now, First picture says that pid=1 and pid=2 is low, after all processes terminated except init our pid=1 is HIGH because it is done in more than 10 interrupts that Project PDF said.
- Also you can see there is no change in priority in one interrupt in the second picture.

- YOU HAVE TO USE A FOURTH TASK ICIN SCHEDULER FOR THAT TEST CASE! YOU HAVE TO UNCOMMENT THAT LINES AND COMMENT CURRENT SCHEDULER

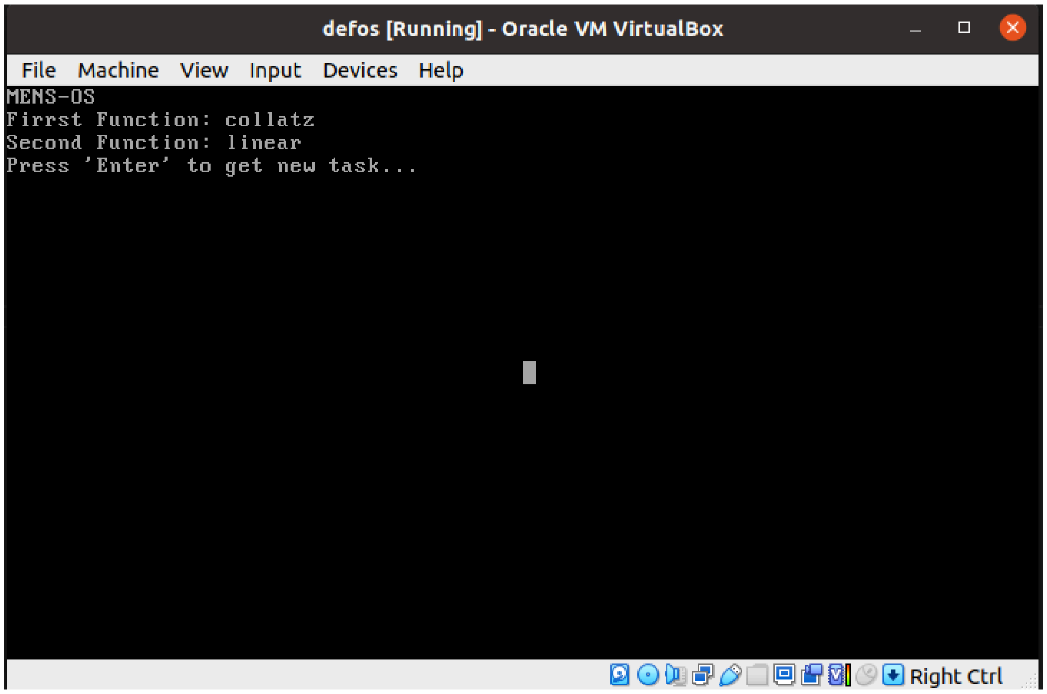
Task C

- **Random Process Spawning with Interactive Input Handling Strategy**
  - Here the system waits for the 'ENTER' keyboard event to start to process.
  - I can add a mouse event to but here there is no need it is the same as the Enter keyboard. I will implement it to Second task of C.
  - All functions are selected randomly like Task B - strategy two.
  - Init processes are always running while others are waiting to enter events.
  - 

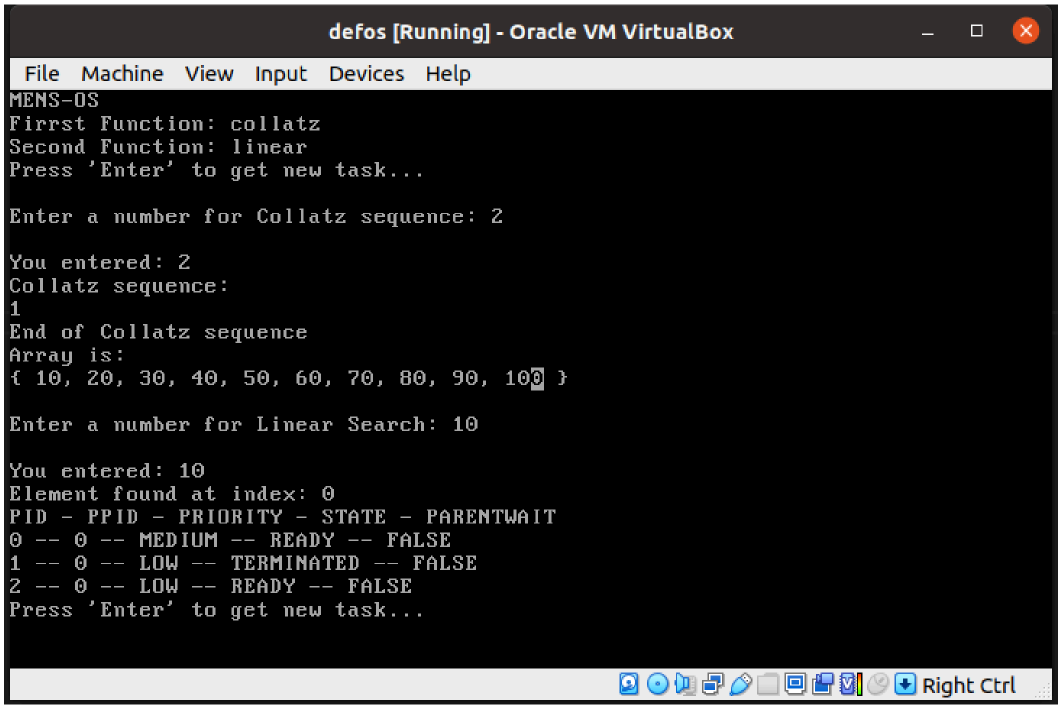
```
// task c first strategy
taskManager.AddTask(new Task(&gdt, taskCFirst, Priority::MEDIUM));
```

  - 

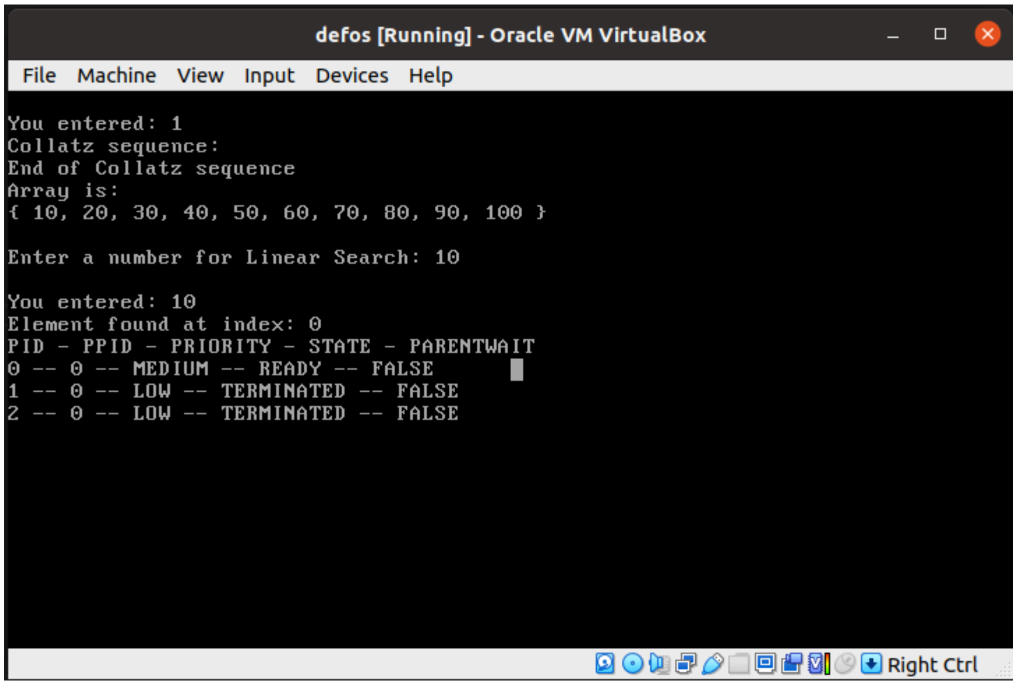
```
void taskCFirst(){

    int parentPid = SyscallHandler::sys_getPid();

    SyscallHandler::sys_fork();

    if(SyscallHandler::sys_getPid() == parentPid){ // first run
        SyscallHandler::sys_fork();
        if(SyscallHandler::sys_getPid() == parentPid){
            // init processes id : 0
        }else{

            Buffer *buffer = new Buffer();
            buffer->setCriticalRegion(true);
            printf("Press 'Enter' to get new task...\n");
            buffer->setDone(false);
            while(!buffer->isDone());
            buffer->setDone(false);
            buffer->setCriticalRegion(false);

            func[n1](); // first run
            func[n2]();
            SyscallHandler::sys_exit();
        }
    } else {
        int sPid = SyscallHandler::sys_getPid();

        Buffer *buffer = new Buffer();
        buffer->setCriticalRegion(true);
        printf("Press 'Enter' to get new task...\n");
        buffer->setDone(false);
        while(!buffer->isDone());
        buffer->setDone(false);
        buffer->setCriticalRegion(false);

        func[n1](); // second  run
        func[n2]();

        SyscallHandler::sys_exit();
    }
    while(1);
```

- 
defos [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

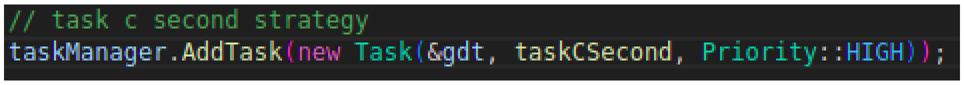
```
MENS-OS
Firrst Function: collatz
Second Function: linear
Press 'Enter' to get new task...
```

- 
defos [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

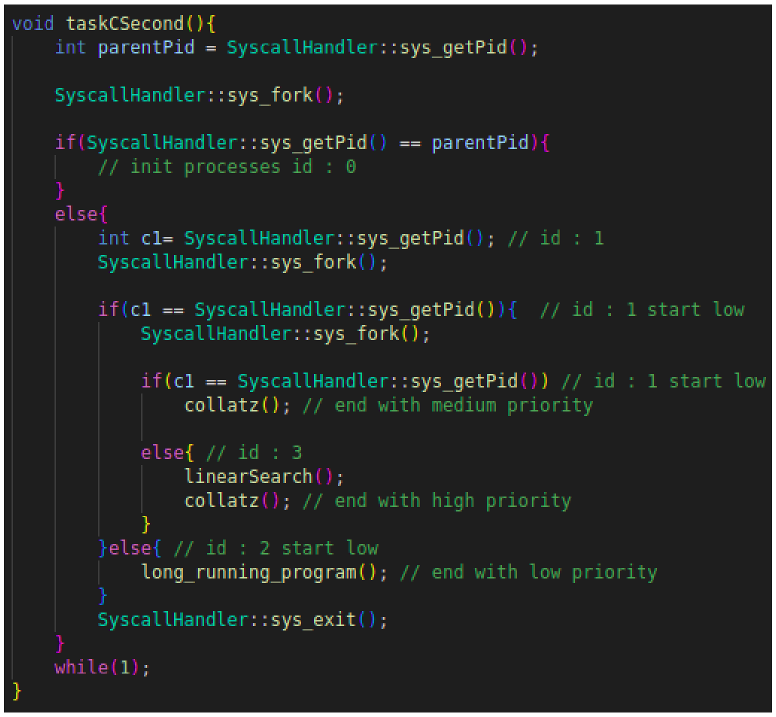
```
MENS-OS
Firrst Function: collatz
Second Function: linear
Press 'Enter' to get new task...

Enter a number for Collatz sequence: 2

You entered: 2
Collatz sequence:
1
End of Collatz sequence
Array is:
{ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 }

Enter a number for Linear Search: 10

You entered: 10
Element found at index: 0
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- MEDIUM -- READY -- FALSE
1 -- 0 -- LOW -- TERMINATED -- FALSE
2 -- 0 -- LOW -- READY -- FALSE
Press 'Enter' to get new task...
```

- 
defos [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

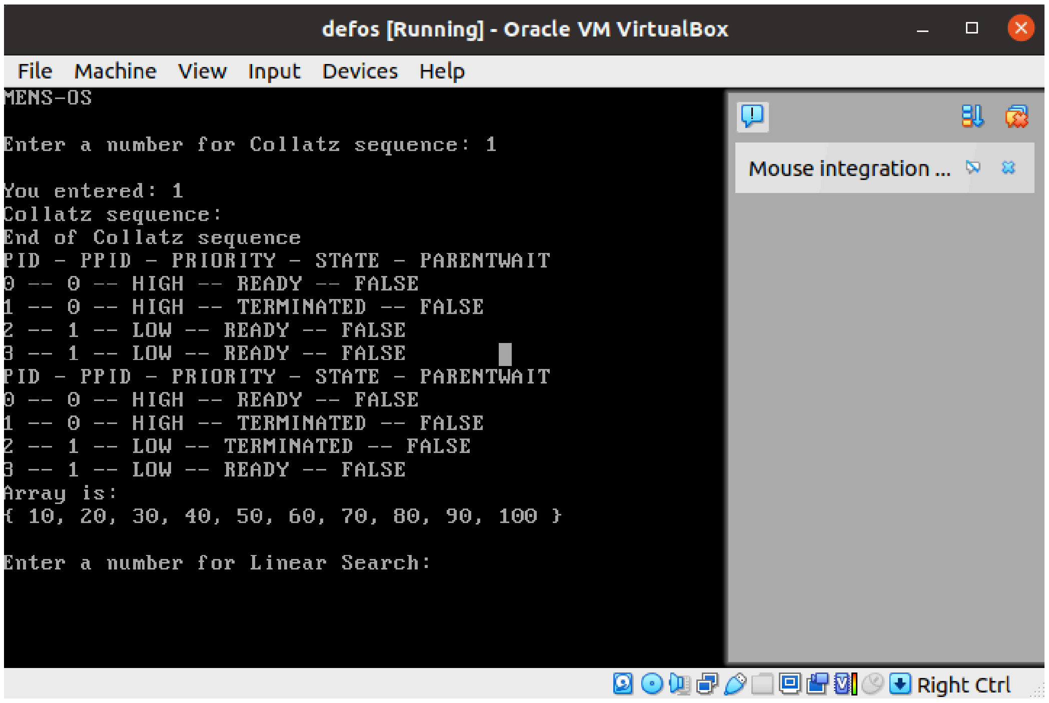
```
You entered: 1
Collatz sequence:
End of Collatz sequence
Array is:
{ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 }

Enter a number for Linear Search: 10

You entered: 10
Element found at index: 0
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- MEDIUM -- READY -- FALSE
1 -- 0 -- LOW -- TERMINATED -- FALSE
2 -- 0 -- LOW -- TERMINATED -- FALSE
```

- **Interactive Input Priority Strategy**
    - It is so simple to do that, because I use the critical region for input from users, so when processing in the critical region its priority must be increased.
    - I do that in interrupts.cpp control of critical region areas.
    - It works as expected.
    - 

```
// task c second strategy
taskManager.AddTask(new Task(&gdt, taskCSecond, Priority::HIGH));
```

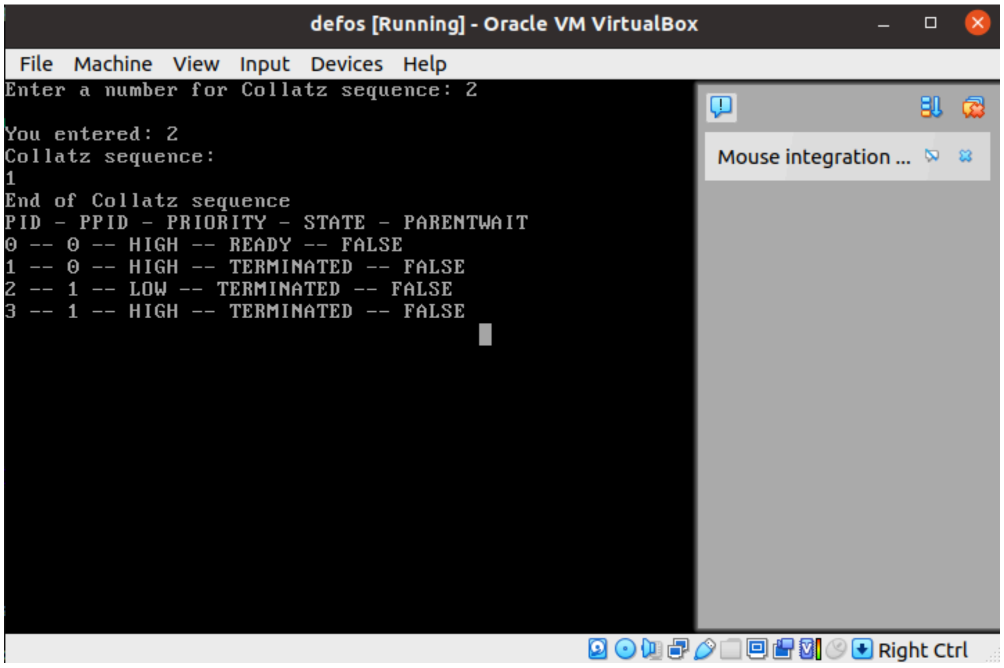
- 

```
void taskCSecond(){
    int parentPid = SyscallHandler::sys_getPid();

    SyscallHandler::sys_fork();

    if(SyscallHandler::sys_getPid() == parentPid){
        // init processes id : 0
    }
    else{
        int c1= SyscallHandler::sys_getPid(); // id : 1
        SyscallHandler::sys_fork();

        if(c1 == SyscallHandler::sys_getPid()){  // id : 1 start low
            SyscallHandler::sys_fork();

            if(c1 == SyscallHandler::sys_getPid()) // id : 1 start low
                collatz(); // end with medium priority

            else{ // id : 3
                linearSearch();
                collatz(); // end with high priority
            }
        }else{ // id : 2 start low
            long_running_program(); // end with low priority
        }
        SyscallHandler::sys_exit();
    }
    while(1);
}
```

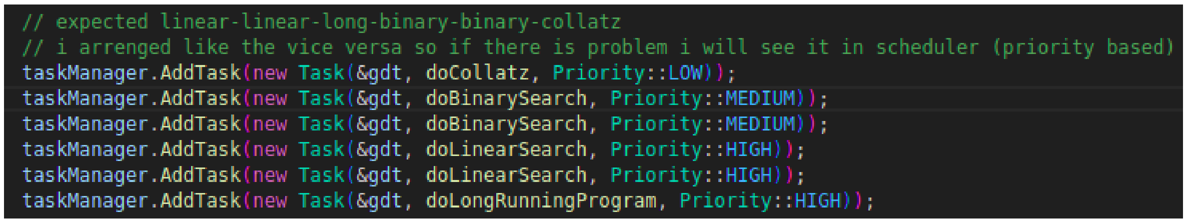
- 

```
MENS-OS

Enter a number for Collatz sequence: 1

You entered: 1
Collatz sequence:
End of Collatz sequence
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- HIGH -- READY -- FALSE
1 -- 0 -- HIGH -- TERMINATED -- FALSE
2 -- 1 -- LOW -- READY -- FALSE
3 -- 1 -- LOW -- READY -- FALSE
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- HIGH -- READY -- FALSE
1 -- 0 -- HIGH -- TERMINATED -- FALSE
2 -- 1 -- LOW -- TERMINATED -- FALSE
3 -- 1 -- LOW -- READY -- FALSE
Array is:
{ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 }

Enter a number for Linear Search:
```

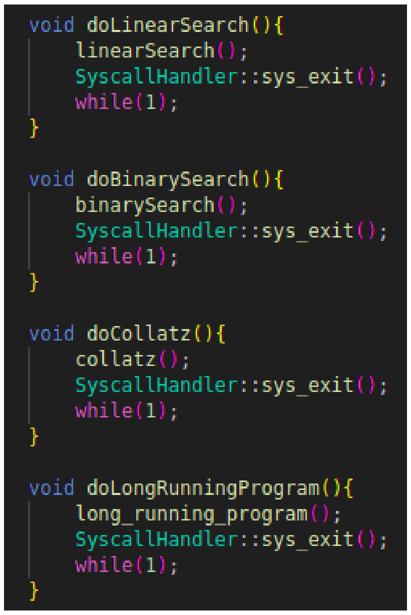
defos [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Mouse integration ...
Right Ctrl

- 
defos [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Mouse integration ...
Right Ctrl

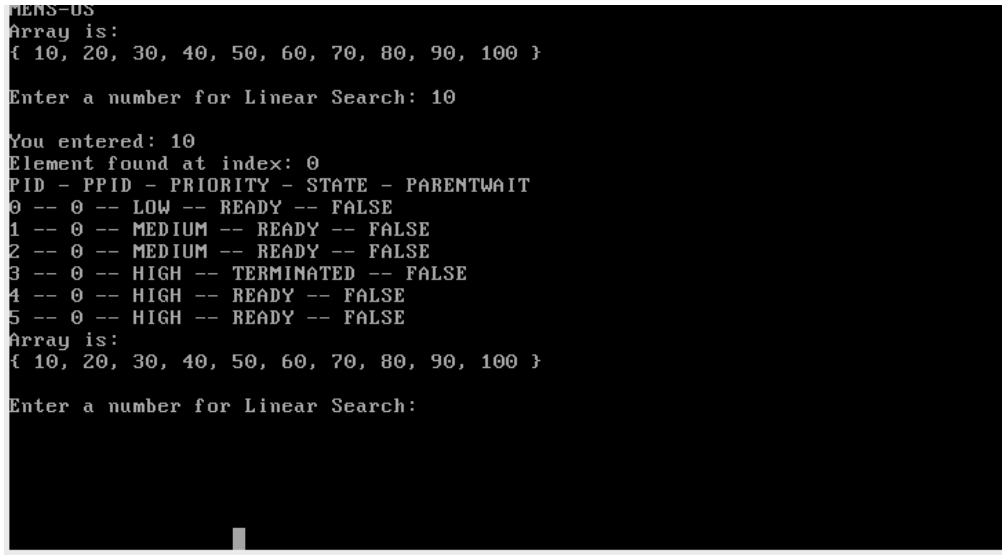
```
Enter a number for Collatz sequence: 2

You entered: 2
Collatz sequence:
1
End of Collatz sequence
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- HIGH -- READY -- FALSE
1 -- 0 -- HIGH -- TERMINATED -- FALSE
2 -- 1 -- LOW -- TERMINATED -- FALSE
3 -- 1 -- HIGH -- TERMINATED -- FALSE
```

- **Preemptive Scheduler**
  - In the preemptive scheduler I do not use a fork because I want to show you preemptive scheduling is working as expected. With a fork I can do that also but without using it is still parallel programming and working.
  - First of the program loads Collatz in a robin round. We expect first in first out so Collatz works but here the last 3 of them are higher priority so they run in order of linear-linear-long. After high is over, medium processes will work. End of the medium processes low priority which is Collatz it will work and terminate.
  - **I also add a forking way with the init process to get full credit.**
  - 

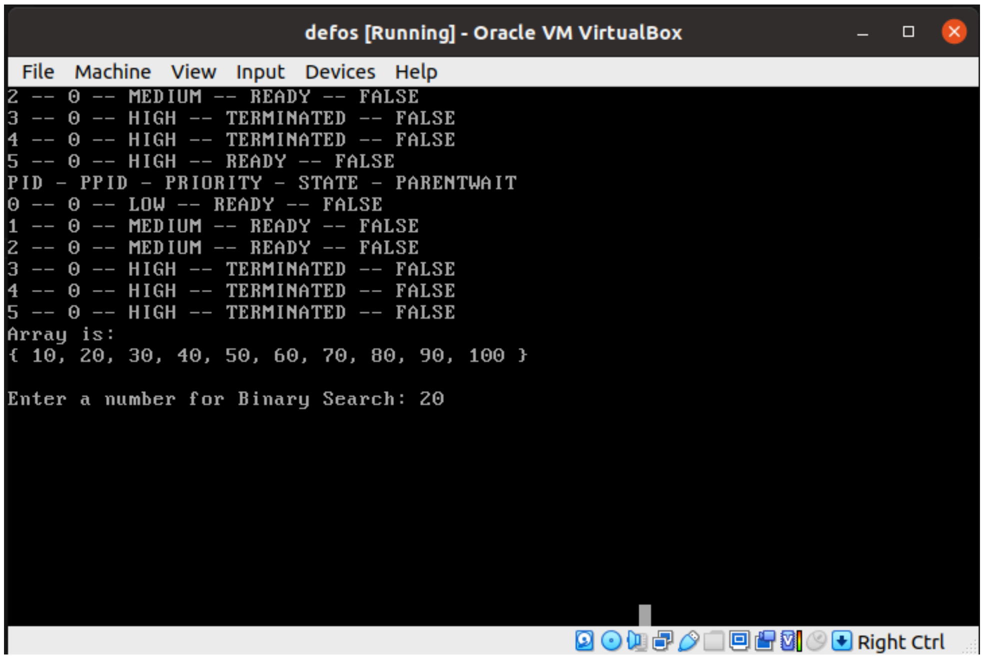
```
// expected linear-linear-long-binary-binary-collatz
// i arrenged like the vice versa so if there is problem i will see it in scheduler (priority based)
taskManager.AddTask(new Task(&gdt, doCollatz, Priority::LOW));
taskManager.AddTask(new Task(&gdt, doBinarySearch, Priority::MEDIUM));
taskManager.AddTask(new Task(&gdt, doBinarySearch, Priority::MEDIUM));
taskManager.AddTask(new Task(&gdt, doLinearSearch, Priority::HIGH));
taskManager.AddTask(new Task(&gdt, doLinearSearch, Priority::HIGH));
taskManager.AddTask(new Task(&gdt, doLongRunningProgram, Priority::HIGH));
```

- 

```
void doLinearSearch(){
    linearSearch();
    SyscallHandler::sys_exit();
    while(1);
}

void doBinarySearch(){
    binarySearch();
    SyscallHandler::sys_exit();
    while(1);
}

void doCollatz(){
    collatz();
    SyscallHandler::sys_exit();
    while(1);
}

void doLongRunningProgram(){
    long_running_program();
    SyscallHandler::sys_exit();
    while(1);
}
```

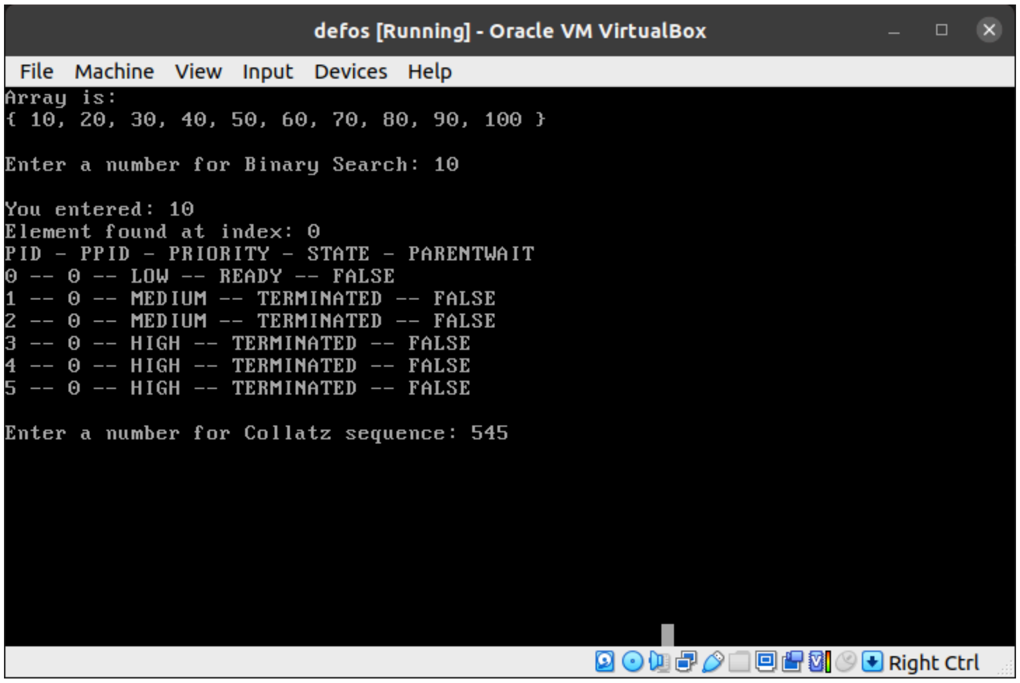
- 

```
MENS-OS
Array is:
{ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 }

Enter a number for Linear Search: 10

You entered: 10
Element found at index: 0
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- LOW -- READY -- FALSE
1 -- 0 -- MEDIUM -- READY -- FALSE
2 -- 0 -- MEDIUM -- READY -- FALSE
3 -- 0 -- HIGH -- TERMINATED -- FALSE
4 -- 0 -- HIGH -- READY -- FALSE
5 -- 0 -- HIGH -- READY -- FALSE
Array is:
{ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 }

Enter a number for Linear Search:
```

- 

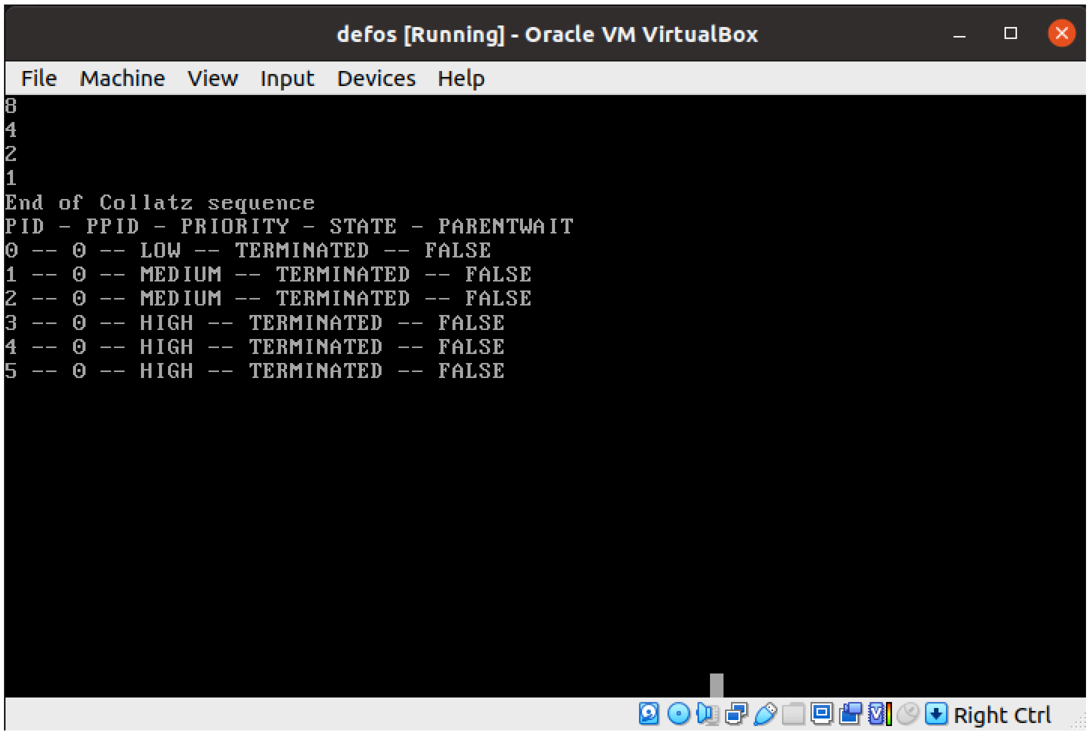
defos [Running] - Oracle VM VirtualBox

```
2 -- 0 -- MEDIUM -- READY -- FALSE
3 -- 0 -- HIGH -- TERMINATED -- FALSE
4 -- 0 -- HIGH -- TERMINATED -- FALSE
5 -- 0 -- HIGH -- READY -- FALSE
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- LOW -- READY -- FALSE
1 -- 0 -- MEDIUM -- READY -- FALSE
2 -- 0 -- MEDIUM -- READY -- FALSE
3 -- 0 -- HIGH -- TERMINATED -- FALSE
4 -- 0 -- HIGH -- TERMINATED -- FALSE
5 -- 0 -- HIGH -- TERMINATED -- FALSE
Array is:
{ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 }

Enter a number for Binary Search: 20
```

- 

defos [Running] - Oracle VM VirtualBox

```
Array is:
{ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 }

Enter a number for Binary Search: 10

You entered: 10
Element found at index: 0
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- LOW -- READY -- FALSE
1 -- 0 -- MEDIUM -- TERMINATED -- FALSE
2 -- 0 -- MEDIUM -- TERMINATED -- FALSE
3 -- 0 -- HIGH -- TERMINATED -- FALSE
4 -- 0 -- HIGH -- TERMINATED -- FALSE
5 -- 0 -- HIGH -- TERMINATED -- FALSE

Enter a number for Collatz sequence: 545
```


defos [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

```
8
4
2
1
End of Collatz sequence
PID - PPID - PRIORITY - STATE - PARENTWAIT
0 -- 0 -- LOW -- TERMINATED -- FALSE
1 -- 0 -- MEDIUM -- TERMINATED -- FALSE
2 -- 0 -- MEDIUM -- TERMINATED -- FALSE
3 -- 0 -- HIGH -- TERMINATED -- FALSE
4 -- 0 -- HIGH -- TERMINATED -- FALSE
5 -- 0 -- HIGH -- TERMINATED -- FALSE
```

- You can see that First off all high processes terminate after medium, at the end of the day low processes die. That is the way that we expected.
- YOU HAVE TO USE A PREEMPTIVE SCHEDULER FOR THAT TEST CASE! YOU HAVE TO UNCOMMENT THAT LINES AND COMMENT CURRENT SCHEDULER.