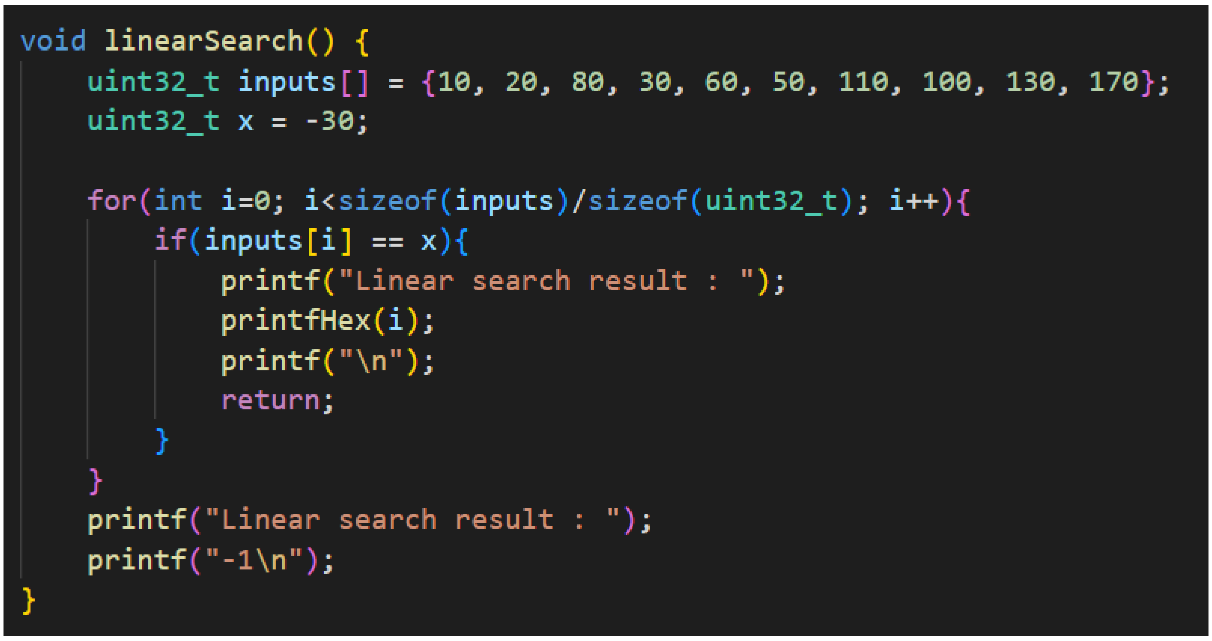
Operating Systems
Homework 1 - Report

Barış Ayyıldız
1901042252

Common Parts

Programs

Linear Search

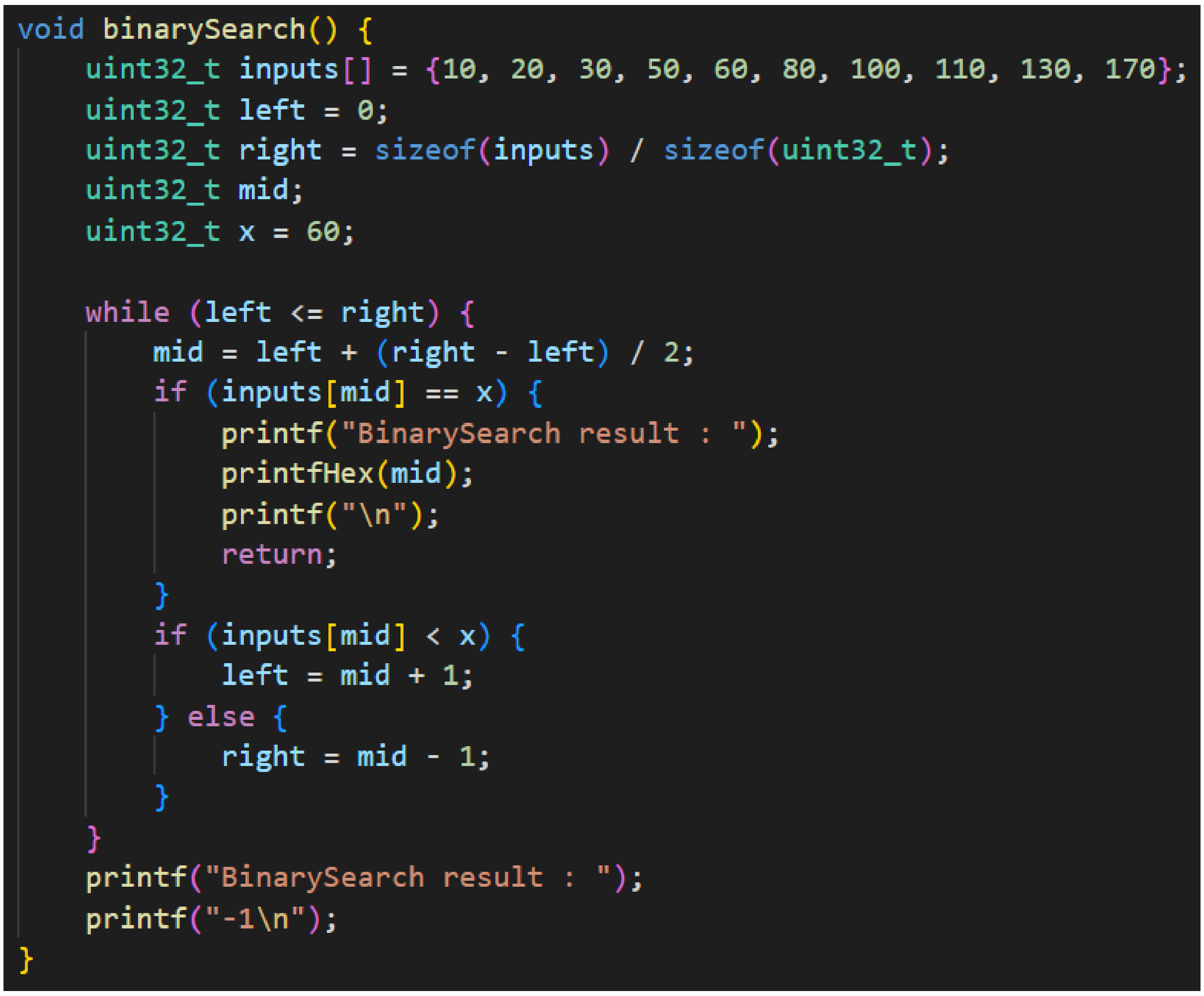


```c
void linearSearch() {
    uint32_t inputs[] = {10, 20, 80, 30, 60, 50, 110, 100, 130, 170};
    uint32_t x = -30;

    for(int i=0; i<sizeof(inputs)/sizeof(uint32_t); i++){
        if(inputs[i] == x){
            printf("Linear search result : ");
            printfHex(i);
            printf("\n");
            return;
        }
    }
    printf("Linear search result : ");
    printf("-1\n");
}
```

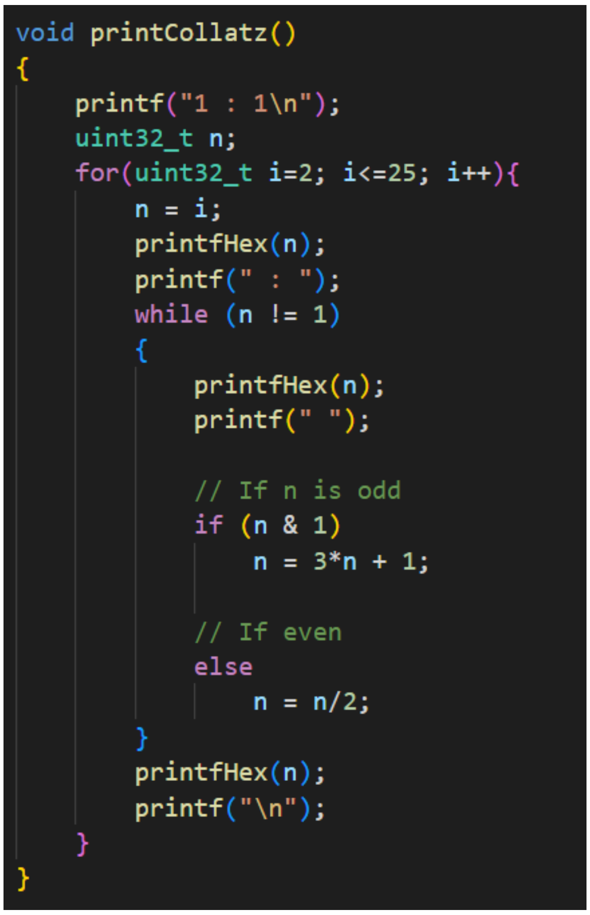
Binary Search

The input array in the documentation was not sorted and I thought that would be a mistake because it was almost a sorted array. That's why I used a sorted array instead of implementing a sorting algorithm.

- BinarySearch
    - Ex; Input : {10, 20, 80, 30, 60, 50, 110, 100, 130, 170} x = 110; Output : 6


```c
void binarySearch() {
    uint32_t inputs[] = {10, 20, 30, 50, 60, 80, 100, 110, 130, 170};
    uint32_t left = 0;
    uint32_t right = sizeof(inputs) / sizeof(uint32_t);
    uint32_t mid;
    uint32_t x = 60;

    while (left <= right) {
        mid = left + (right - left) / 2;
        if (inputs[mid] == x) {
            printf("BinarySearch result : ");
            printfHex(mid);
            printf("\n");
            return;
        }
        if (inputs[mid] < x) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    printf("BinarySearch result : ");
    printf("-1\n");
}
```

Collatz

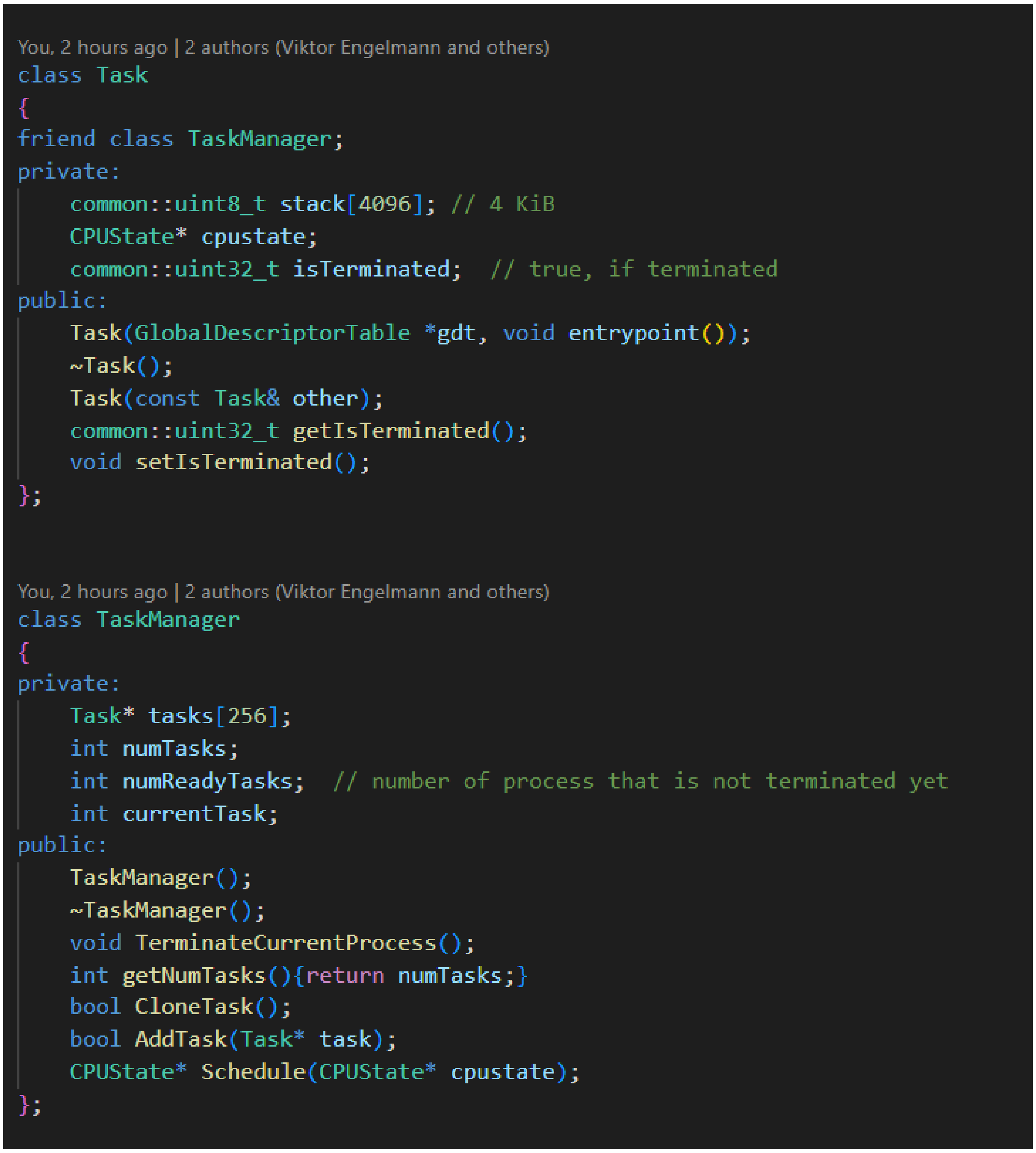

```c
void printCollatz()
{
    printf("1 : 1\n");
    uint32_t n;
    for(uint32_t i=2; i<=25; i++){
        n = i;
        printfHex(n);
        printf(" : ");
        while (n != 1)
        {
            printfHex(n);
            printf(" ");

            // If n is odd
            if (n & 1)
                n = 3*n + 1;

            // If even
            else
                n = n/2;
        }
        printfHex(n);
        printf("\n");
    }
}
```

Multitasking

I made some modifications to the Task class in the given codebase. First I have added the **isTerminated** variable in the Task class. It is set to 1 when a program terminates. I have also defined getters and setters for this variable.

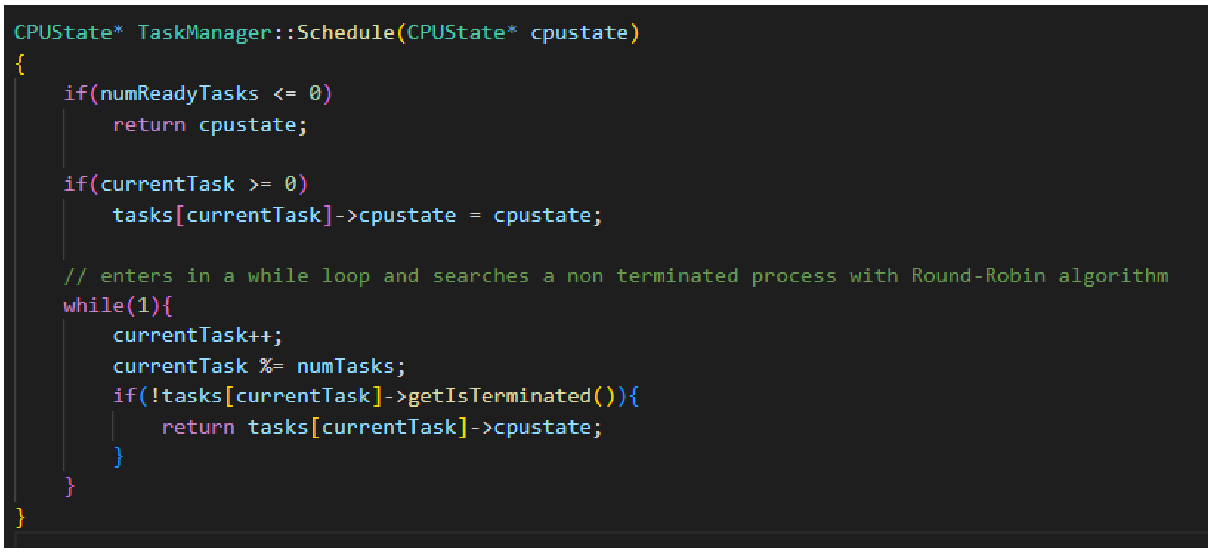
In the TaskManager I have added **numReadyTasks**. It holds the information of how many of the tasks are not terminated.

Then I have the **CloneTask** function, it clones the current running task to handle fork system calls. And the **TerminateCurrentProcess** function terminates the running process when it is called.



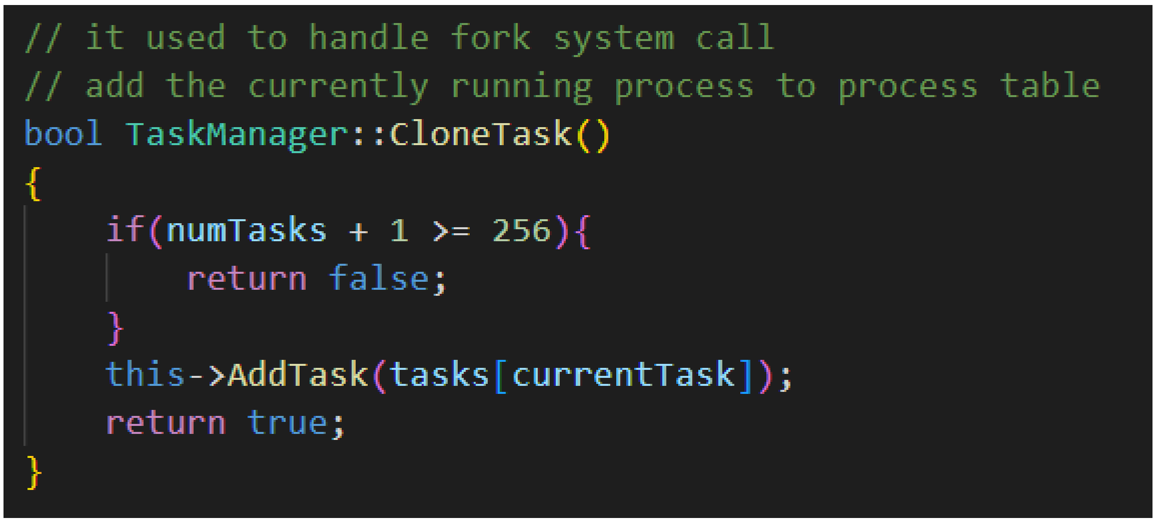
```
You, 2 hours ago | 2 authors (Viktor Engelmann and others)
class Task
{
friend class TaskManager;
private:
    common::uint8_t stack[4096]; // 4 KiB
    CPUState* cpustate;
    common::uint32_t isTerminated;  // true, if terminated
public:
    Task(GlobalDescriptorTable *gdt, void entrypoint());
    ~Task();
    Task(const Task& other);
    common::uint32_t getIsTerminated();
    void setIsTerminated();
};


You, 2 hours ago | 2 authors (Viktor Engelmann and others)
class TaskManager
{
private:
    Task* tasks[256];
    int numTasks;
    int numReadyTasks;  // number of process that is not terminated yet
    int currentTask;
public:
    TaskManager();
    ~TaskManager();
    void TerminateCurrentProcess();
    int getNumTasks(){return numTasks;}
    bool CloneTask();
    bool AddTask(Task* task);
    CPUState* Schedule(CPUState* cpustate);
};
```

This is the schedule function, I made some updates on that as well. If the number of ready tasks is zero it returns the current cpustate. Then it enters a while loop, it searches for the next non terminated task and. It follows the Round-Robin algorithm.



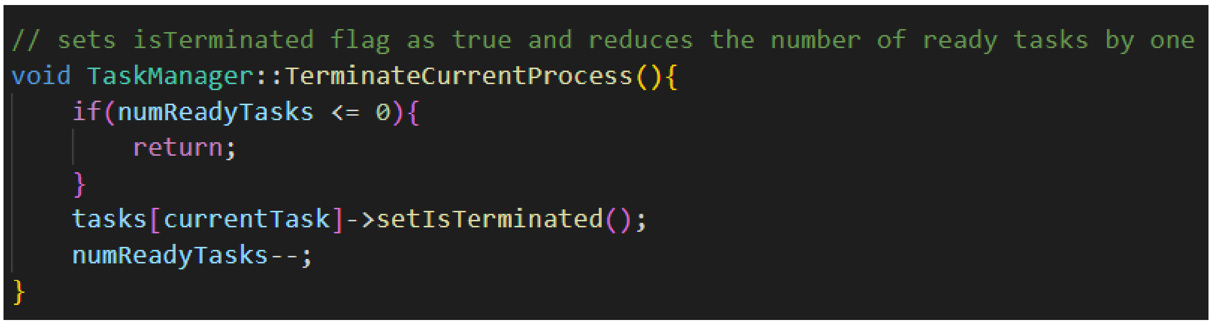
```
CPUState* TaskManager::Schedule(CPUState* cpustate)
{
    if(numReadyTasks <= 0)
        return cpustate;

    if(currentTask >= 0)
        tasks[currentTask]->cpustate = cpustate;

    // enters in a while loop and searches a non terminated process with Round-Robin algorithm
    while(1){
        currentTask++;
        currentTask %= numTasks;
        if(!tasks[currentTask]->getIsTerminated()){
            return tasks[currentTask]->cpustate;
        }
    }
}
```

This is the implementation of the **CloneTask** function. If the number of tasks plus one is not exceeding 256 it adds the current task in the table



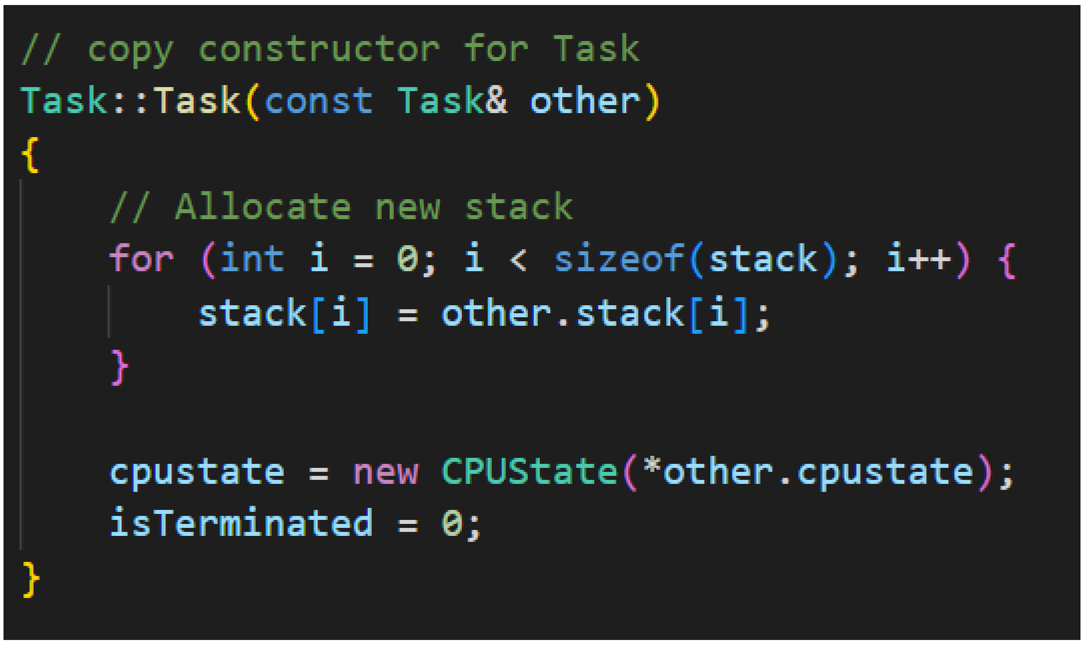
```
// it used to handle fork system call
// add the currently running process to process table
bool TaskManager::CloneTask()
{
    if(numTasks + 1 >= 256){
        return false;
    }
    this->AddTask(tasks[currentTask]);
    return true;
}
```

This is the **TerminateCurrentProcess** function. If the number of ready tasks is greater than zero it sets the current tasks isTerminated flag as true and reduces the number of ready tasks by one.

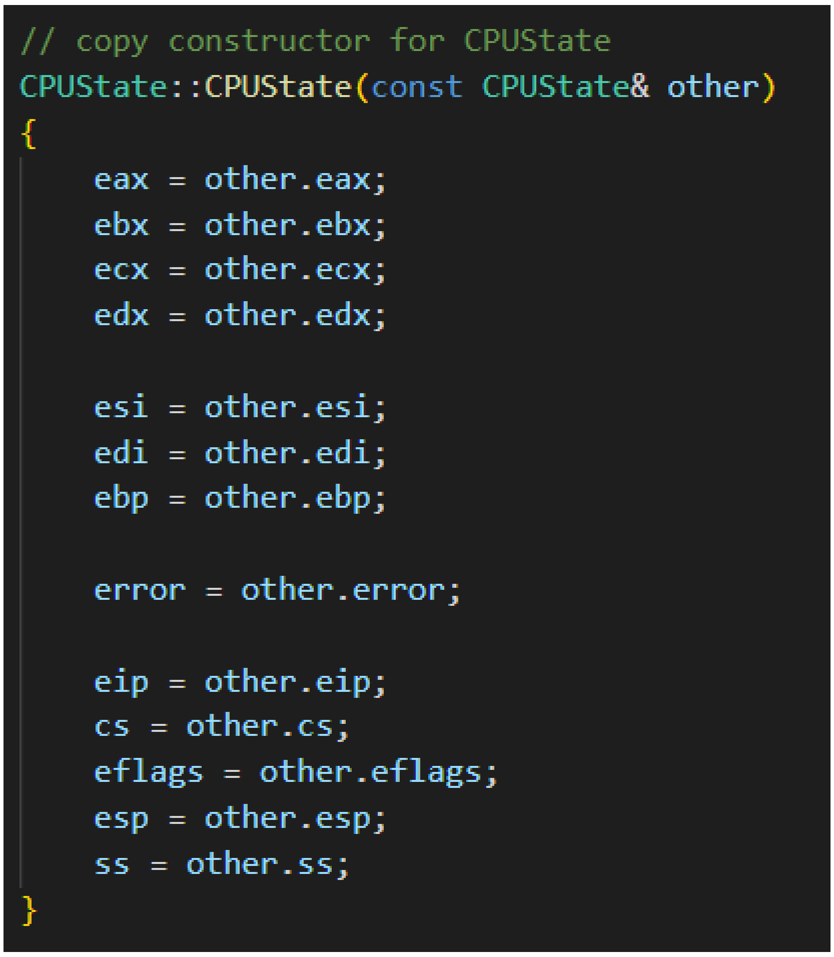


```cpp
// sets isTerminated flag as true and reduces the number of ready tasks by one
void TaskManager::TerminateCurrentProcess(){
    if(numReadyTasks <= 0){
        return;
    }
    tasks[currentTask]->setIsTerminated();
    numReadyTasks--;
}
```

These are the copy constructors that are called inside the CloneTask function



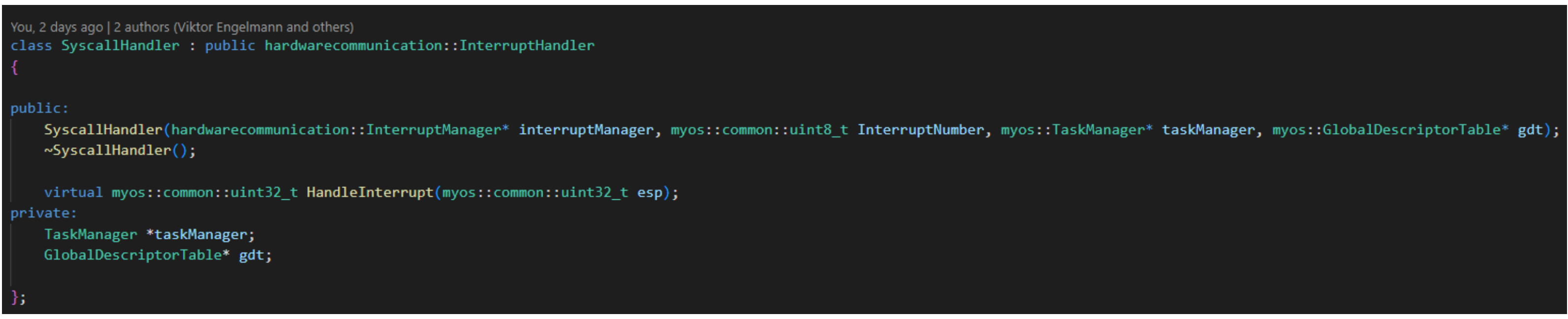
```cpp
// copy constructor for Task
Task::Task(const Task& other)
{
    // Allocate new stack
    for (int i = 0; i < sizeof(stack); i++) {
        stack[i] = other.stack[i];
    }

    cpustate = new CPUState(*other.cpustate);
    isTerminated = 0;
}
```



```cpp
// copy constructor for CPUState
CPUState::CPUState(const CPUState& other)
{
    eax = other.eax;
    ebx = other.ebx;
    ecx = other.ecx;
    edx = other.edx;

    esi = other.esi;
    edi = other.edi;
    ebp = other.ebp;

    error = other.error;

    eip = other.eip;
    cs = other.cs;
    eflags = other.eflags;
    esp = other.esp;
    ss = other.ss;
}
```

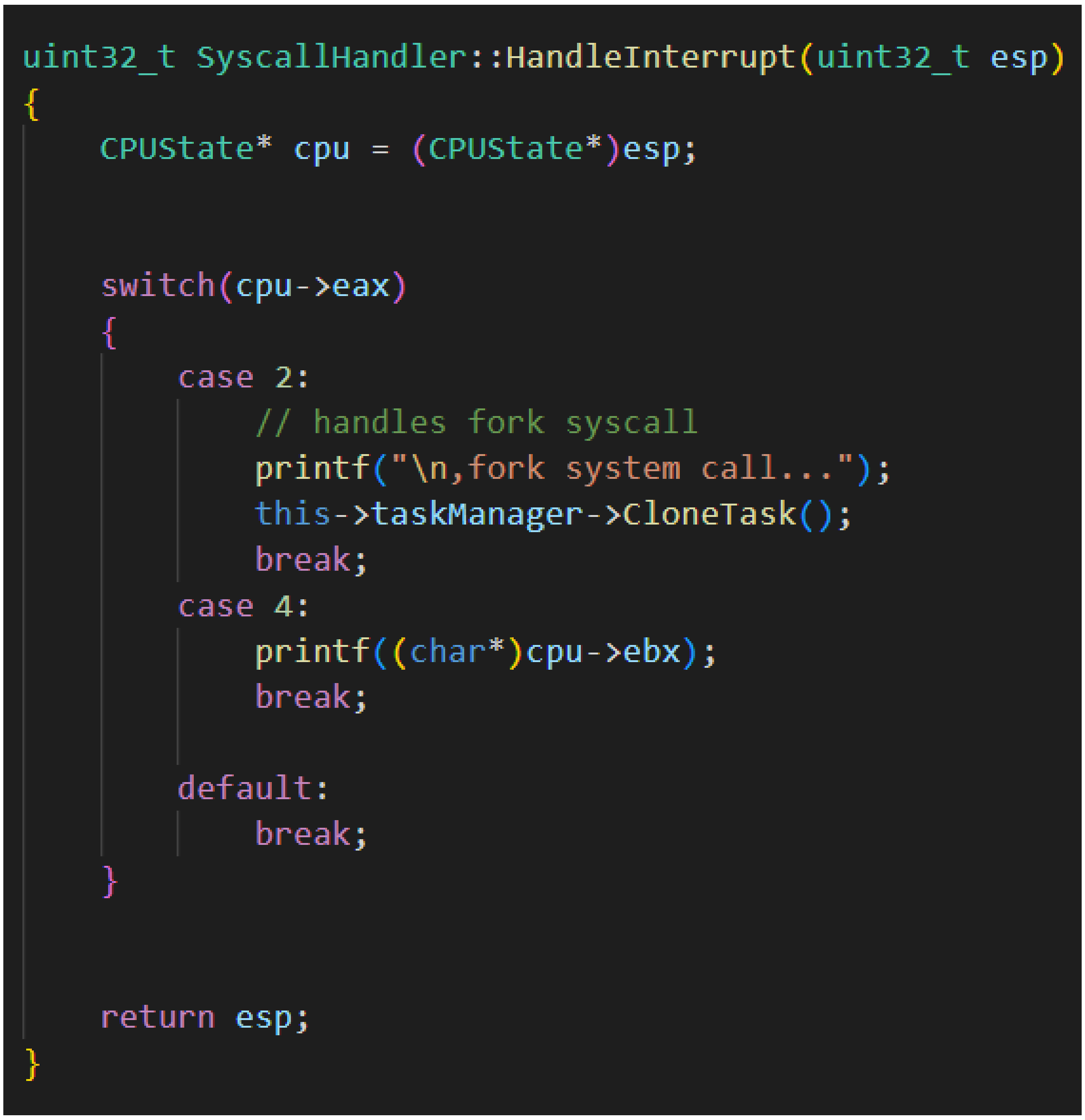
System Call

I changed the constructor of the **SysCallHandler** class, now it also requires the task manager and the global descriptor table.



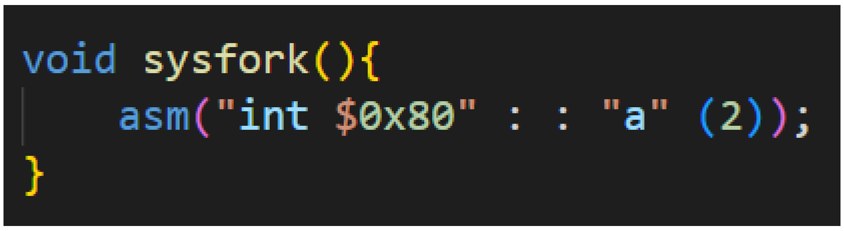
```
You, 2 days ago | 2 authors (Viktor Engelmann and others)
class SyscallHandler : public hardwarecommunication::InterruptHandler
{

public:
    SyscallHandler(hardwarecommunication::InterruptManager* interruptManager, myos::common::uint8_t InterruptNumber, myos::TaskManager* taskManager, myos::GlobalDescriptorTable* gdt);
    ~SyscallHandler();

    virtual myos::common::uint32_t HandleInterrupt(myos::common::uint32_t esp);
private:
    TaskManager *taskManager;
    GlobalDescriptorTable* gdt;

};
```

And in the **HandleInterrupt** function, I handle fork system calls. It is the case 2. Here it calls the CloneTask function I explained above.



```
uint32_t SyscallHandler::HandleInterrupt(uint32_t esp)
{
    CPUState* cpu = (CPUState*)esp;


    switch(cpu->eax)
    {
        case 2:
            // handles fork syscall
            printf("\n,fork system call...");
            this->taskManager->CloneTask();
            break;
        case 4:
            printf((char*)cpu->ebx);
            break;

        default:
            break;
    }


    return esp;
}
```

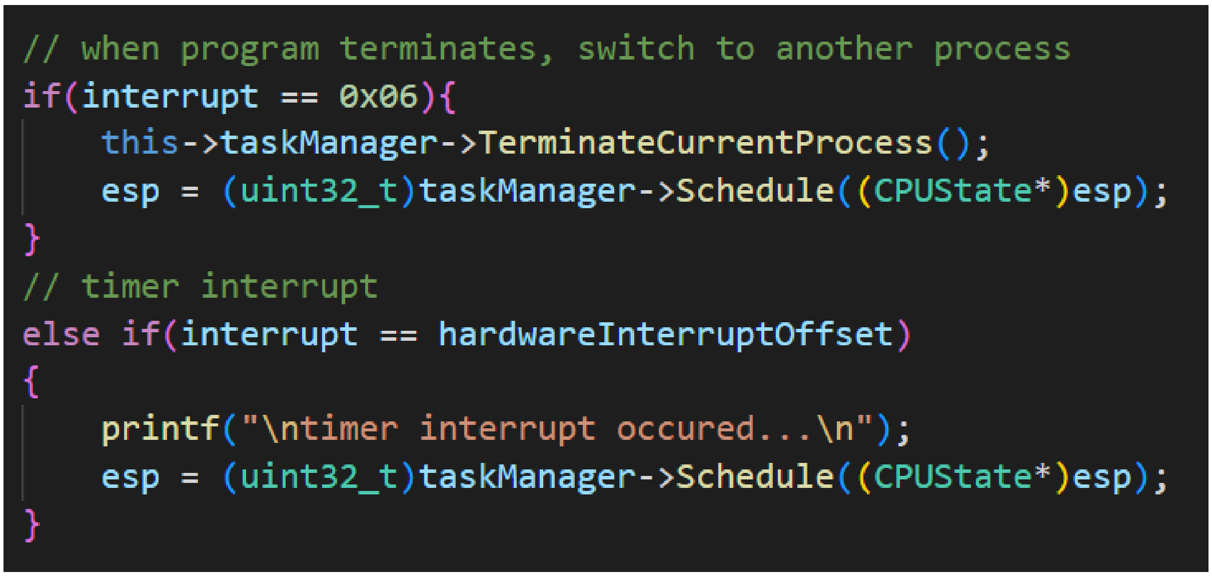
This is my fork function. It runs a certain assembly code, after the execution of this code program jumps to the HandleInterrupt function above.



```
void sysfork(){
    asm("int $0x80" : : "a" (2));
}
```

Interrupts

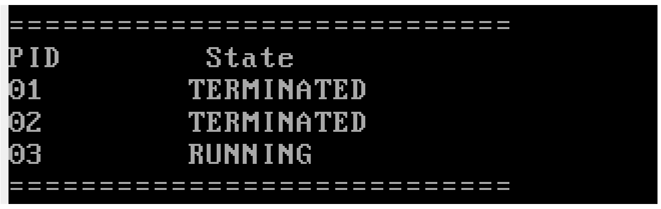
Inside **DoHandleInterrupt** function I added this condition:



```
// when program terminates, switch to another process
if(interrupt == 0x06){
    this->taskManager->TerminateCurrentProcess();
    esp = (uint32_t)taskManager->Schedule((CPUState*)esp);
}
// timer interrupt
else if(interrupt == hardwareInterruptOffset)
{
    printf("\ntimer interrupt occured...\n");
    esp = (uint32_t)taskManager->Schedule((CPUState*)esp);
}
```

It first checks if the interrupt number is **0x06** it terminates the current process by calling TerminateCurrentProcess. It came to my attention that this is the interrupt number used when a program is terminated. If it is not equal to this specific interrupt number it checks if it is equal to hardwareInterruptOffset. If that's the case it means that this was a timer interrupt and it prints '**timer interrupt occurred…**' text to show it.

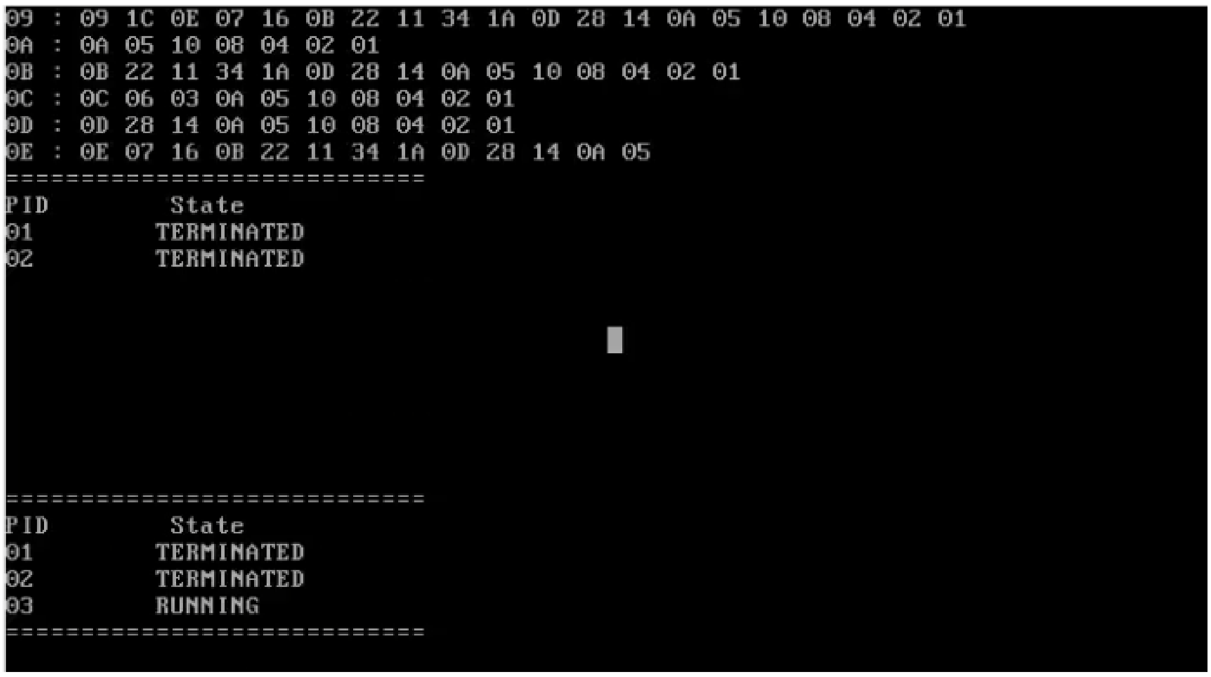
Actually, I was planning to print the information in the process table on the screen in this way, but the programs were running too fast and capturing screenshots would have been difficult.



```
============================
PID          State
01          TERMINATED
02          TERMINATED
03          RUNNING
============================
```

Additionally, the printf function was not working properly, and in cases where the number of characters printed on the screen exceeded the size of the screen, there could be distortions.

That’s why I decided to keep the outputs short and only printed out ‘timer interrupt occurred…’

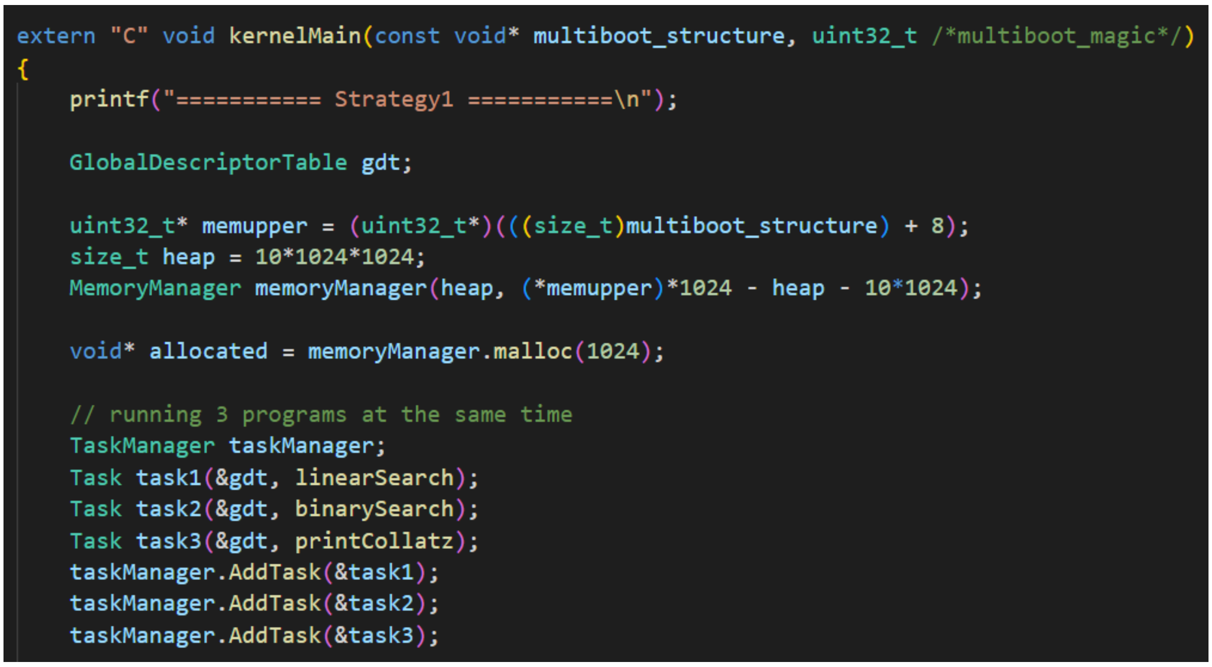


```
09 : 09 1C 0E 07 16 0B 22 11 34 1A 0D 28 14 0A 05 10 08 04 02 01
0A : 0A 05 10 08 04 02 01
0B : 0B 22 11 34 1A 0D 28 14 0A 05 10 08 04 02 01
0C : 0C 06 03 0A 05 10 08 04 02 01
0D : 0D 28 14 0A 05 10 08 04 02 01
0E : 0E 07 16 0B 22 11 34 1A 0D 28 14 0A 05
============================
PID          State
01          TERMINATED
02          TERMINATED



============================
PID          State
01          TERMINATED
02          TERMINATED
03          RUNNING
============================
```

Strategies

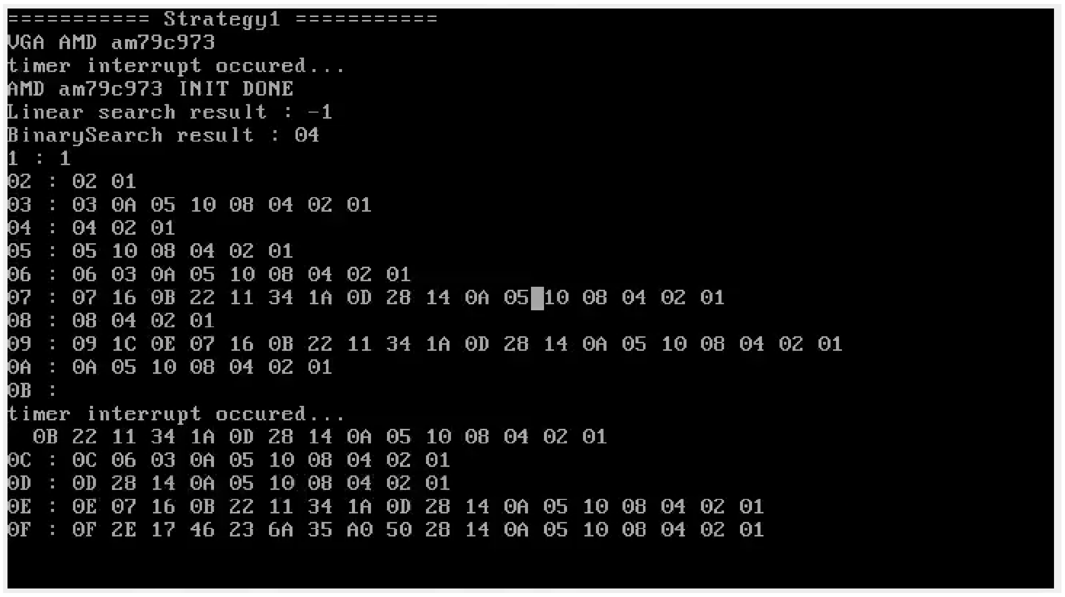
Strategy One

Inside the kernelMain function I load the 3 tasks together in memory.

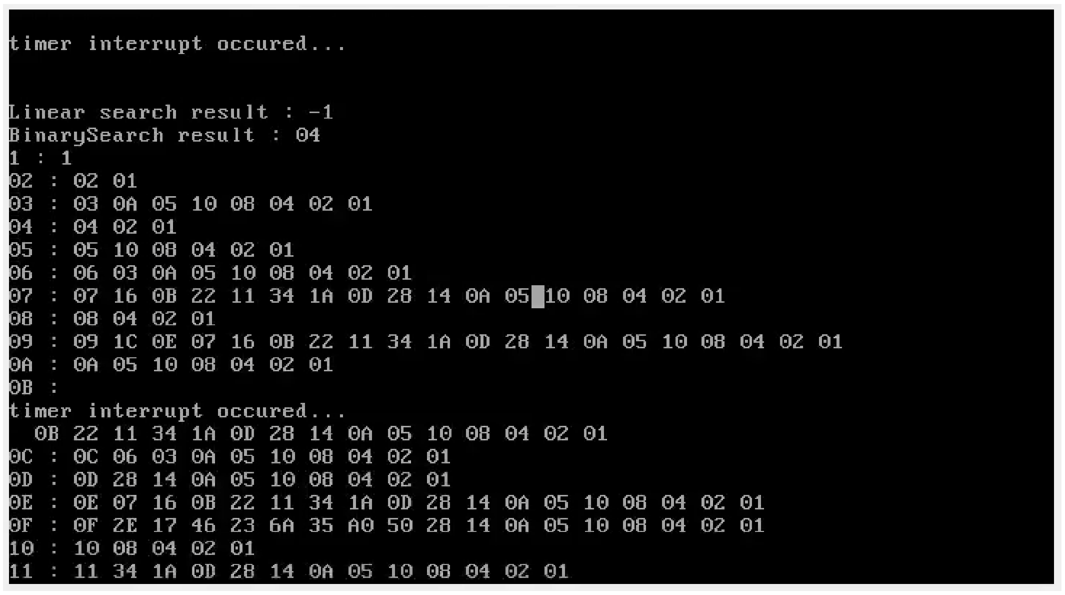


```
extern "C" void kernelMain(const void* multiboot_structure, uint32_t /*multiboot_magic*/)
{
    printf("=========== Strategy1 ===========\n");

    GlobalDescriptorTable gdt;

    uint32_t* memupper = (uint32_t*)(((size_t)multiboot_structure) + 8);
    size_t heap = 10*1024*1024;
    MemoryManager memoryManager(heap, (*memupper)*1024 - heap - 10*1024);

    void* allocated = memoryManager.malloc(1024);

    // running 3 programs at the same time
    TaskManager taskManager;
    Task task1(&gdt, linearSearch);
    Task task2(&gdt, binarySearch);
    Task task3(&gdt, printCollatz);
    taskManager.AddTask(&task1);
    taskManager.AddTask(&task2);
    taskManager.AddTask(&task3);
```

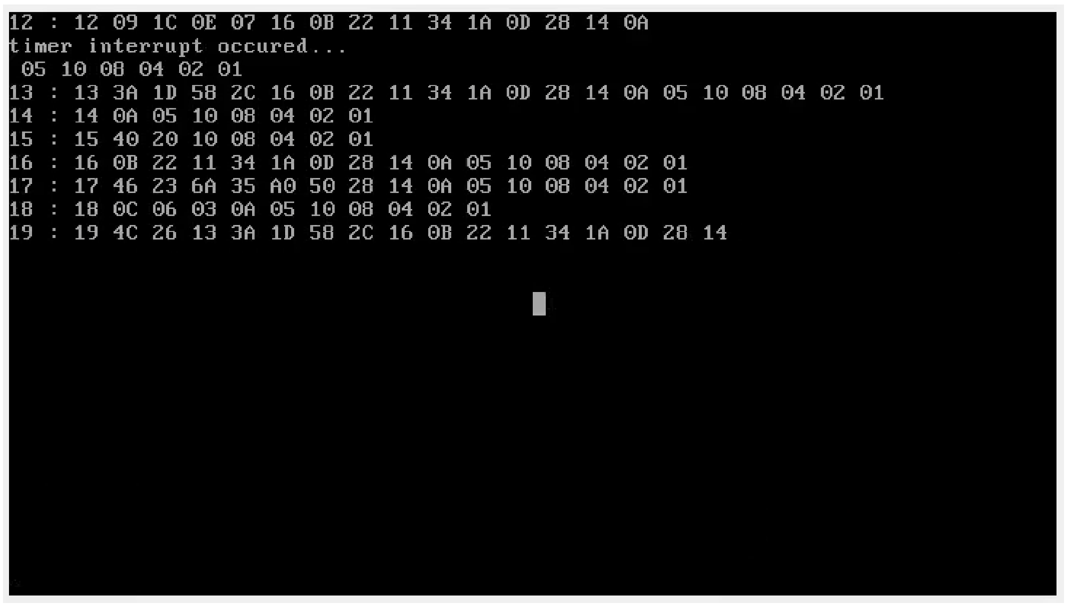
And this is the output I get:



```
=========== Strategy1 ===========
VGA AMD am79c973
timer interrupt occured...
AMD am79c973 INIT DONE
Linear search result : -1
BinarySearch result : 04
1 : 1
02 : 02 01
03 : 03 0A 05 10 08 04 02 01
04 : 04 02 01
05 : 05 10 08 04 02 01
06 : 06 03 0A 05 10 08 04 02 01
07 : 07 16 0B 22 11 34 1A 0D 28 14 0A 05 10 08 04 02 01
08 : 08 04 02 01
09 : 09 1C 0E 07 16 0B 22 11 34 1A 0D 28 14 0A 05 10 08 04 02 01
0A : 0A 05 10 08 04 02 01
0B :
timer interrupt occured...
  0B 22 11 34 1A 0D 28 14 0A 05 10 08 04 02 01
0C : 0C 06 03 0A 05 10 08 04 02 01
0D : 0D 28 14 0A 05 10 08 04 02 01
0E : 0E 07 16 0B 22 11 34 1A 0D 28 14 0A 05 10 08 04 02 01
0F : 0F 2E 17 46 23 6A 35 A0 50 28 14 0A 05 10 08 04 02 01
```



```
timer interrupt occured...


Linear search result : -1
BinarySearch result : 04
1 : 1
02 : 02 01
03 : 03 0A 05 10 08 04 02 01
04 : 04 02 01
05 : 05 10 08 04 02 01
06 : 06 03 0A 05 10 08 04 02 01
07 : 07 16 0B 22 11 34 1A 0D 28 14 0A 05 10 08 04 02 01
08 : 08 04 02 01
09 : 09 1C 0E 07 16 0B 22 11 34 1A 0D 28 14 0A 05 10 08 04 02 01
0A : 0A 05 10 08 04 02 01
0B :
timer interrupt occured...
  0B 22 11 34 1A 0D 28 14 0A 05 10 08 04 02 01
0C : 0C 06 03 0A 05 10 08 04 02 01
0D : 0D 28 14 0A 05 10 08 04 02 01
0E : 0E 07 16 0B 22 11 34 1A 0D 28 14 0A 05 10 08 04 02 01
0F : 0F 2E 17 46 23 6A 35 A0 50 28 14 0A 05 10 08 04 02 01
10 : 10 08 04 02 01
11 : 11 34 1A 0D 28 14 0A 05 10 08 04 02 01
```



```
12 : 12 09 1C 0E 07 16 0B 22 11 34 1A 0D 28 14 0A
timer interrupt occured...
 05 10 08 04 02 01
13 : 13 3A 1D 58 2C 16 0B 22 11 34 1A 0D 28 14 0A 05 10 08 04 02 01
14 : 14 0A 05 10 08 04 02 01
15 : 15 40 20 10 08 04 02 01
16 : 16 0B 22 11 34 1A 0D 28 14 0A 05 10 08 04 02 01
17 : 17 46 23 6A 35 A0 50 28 14 0A 05 10 08 04 02 01
18 : 18 0C 06 03 0A 05 10 08 04 02 01
19 : 19 4C 26 13 3A 1D 58 2C 16 0B 22 11 34 1A 0D 28 14
```

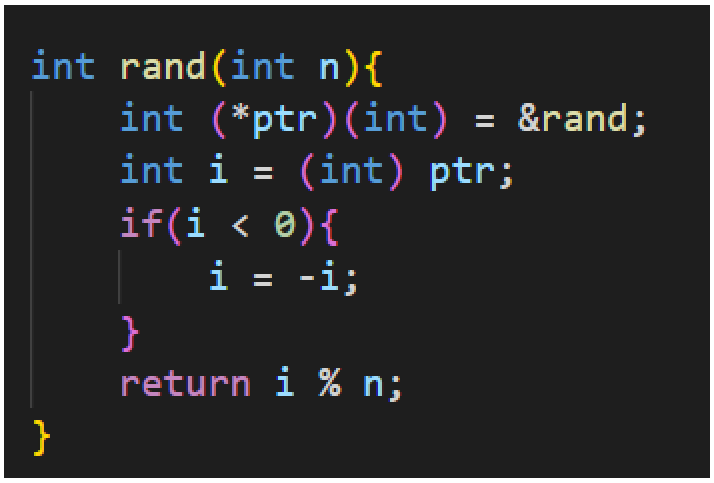
X value was -30 in linear search and it was 60 in binary search so they produced the correct answer.

As you can see, the first linear search and binary search programs were executed and at that time we didn't get any timer interrupts because they were executed very quickly. After that the collatz function started to run. It printed the results for the first 10 integers. While it was printing the collatz numbers for the number 11, it had a timer interrupt. Then it continued to run the same program because at that time this was the only program that was not terminated.

Then it had another timer interrupt when it was printing the values for the integer 18. And again it continued from where it was interrupted.

Strategy Two

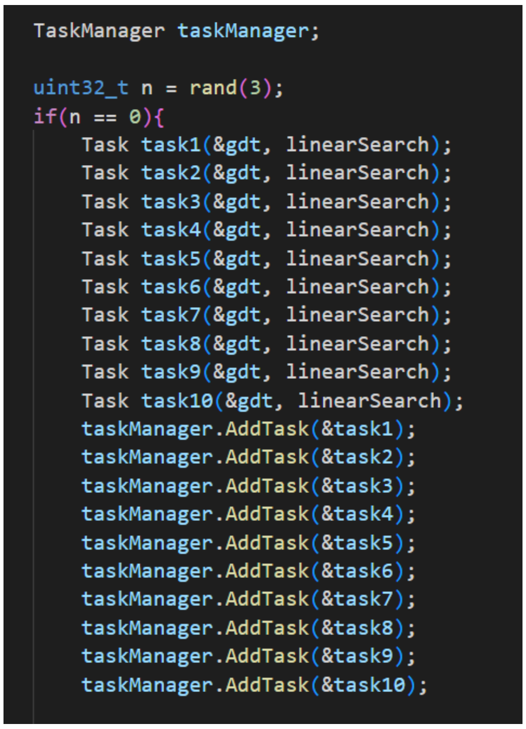
For the 2nd and the 3rd strategies I wrote this simple random number generator function since we don't have access to any C libraries.



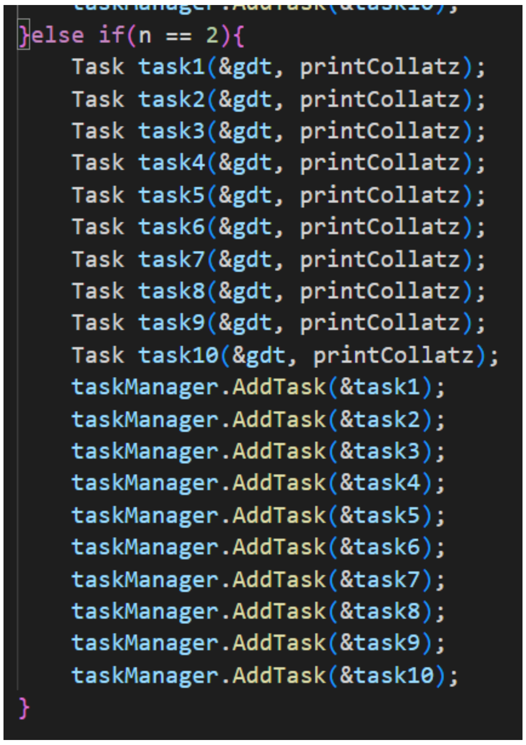
```c
int rand(int n){
    int (*ptr)(int) = &rand;
    int i = (int) ptr;
    if(i < 0){
        i = -i;
    }
    return i % n;
}
```

It takes the address of the function and takes its mode with the parameter n. It’s actually totally random, when it’s called twice it will give the same number because the address of the function stays the same. But it gets the job done for our case.

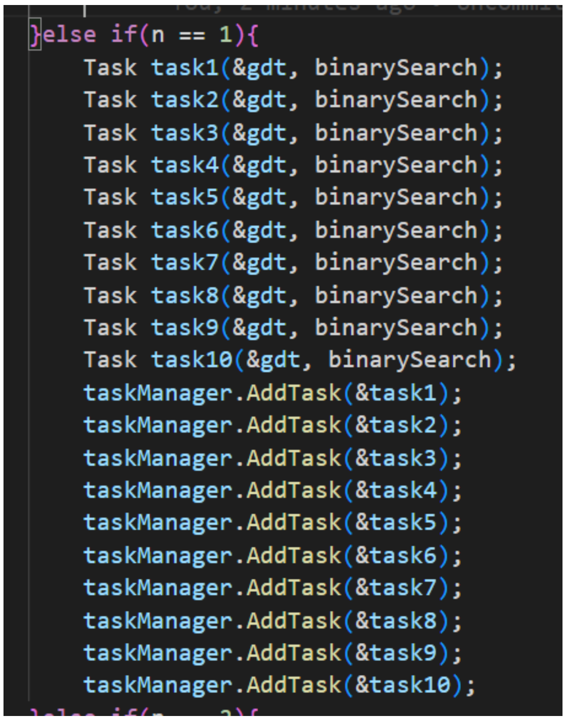
In the main function I call this rand function and based on the resulting number i create tasks and load them to memory.



```cpp
TaskManager taskManager;

uint32_t n = rand(3);
if(n == 0){
    Task task1(&gdt, linearSearch);
    Task task2(&gdt, linearSearch);
    Task task3(&gdt, linearSearch);
    Task task4(&gdt, linearSearch);
    Task task5(&gdt, linearSearch);
    Task task6(&gdt, linearSearch);
    Task task7(&gdt, linearSearch);
    Task task8(&gdt, linearSearch);
    Task task9(&gdt, linearSearch);
    Task task10(&gdt, linearSearch);
    taskManager.AddTask(&task1);
    taskManager.AddTask(&task2);
    taskManager.AddTask(&task3);
    taskManager.AddTask(&task4);
    taskManager.AddTask(&task5);
    taskManager.AddTask(&task6);
    taskManager.AddTask(&task7);
    taskManager.AddTask(&task8);
    taskManager.AddTask(&task9);
    taskManager.AddTask(&task10);
```



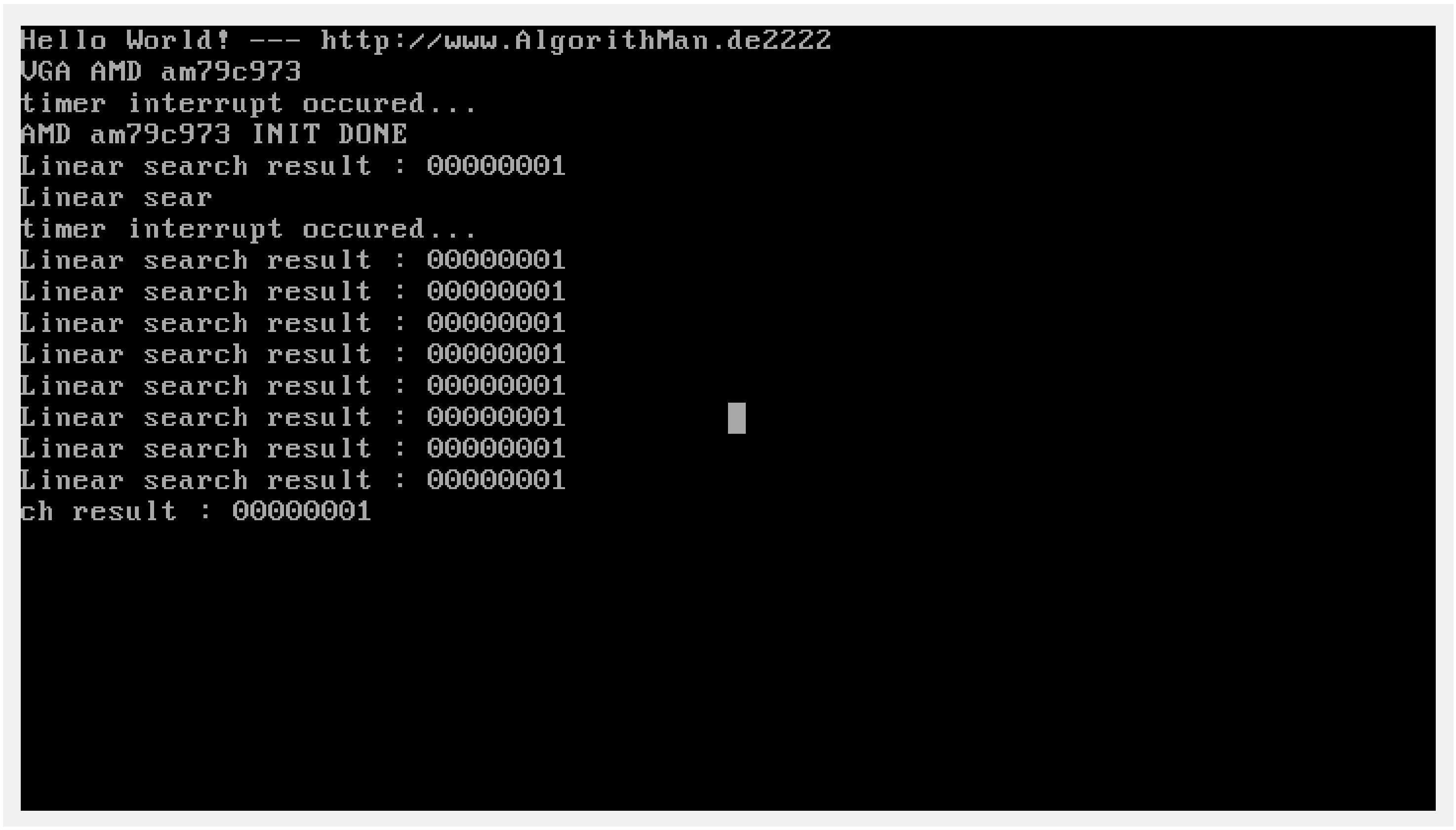
```cpp
}else if(n == 2){
    Task task1(&gdt, printCollatz);
    Task task2(&gdt, printCollatz);
    Task task3(&gdt, printCollatz);
    Task task4(&gdt, printCollatz);
    Task task5(&gdt, printCollatz);
    Task task6(&gdt, printCollatz);
    Task task7(&gdt, printCollatz);
    Task task8(&gdt, printCollatz);
    Task task9(&gdt, printCollatz);
    Task task10(&gdt, printCollatz);
    taskManager.AddTask(&task1);
    taskManager.AddTask(&task2);
    taskManager.AddTask(&task3);
    taskManager.AddTask(&task4);
    taskManager.AddTask(&task5);
    taskManager.AddTask(&task6);
    taskManager.AddTask(&task7);
    taskManager.AddTask(&task8);
    taskManager.AddTask(&task9);
    taskManager.AddTask(&task10);
}
```



```cpp
}else if(n == 1){
    Task task1(&gdt, binarySearch);
    Task task2(&gdt, binarySearch);
    Task task3(&gdt, binarySearch);
    Task task4(&gdt, binarySearch);
    Task task5(&gdt, binarySearch);
    Task task6(&gdt, binarySearch);
    Task task7(&gdt, binarySearch);
    Task task8(&gdt, binarySearch);
    Task task9(&gdt, binarySearch);
    Task task10(&gdt, binarySearch);
    taskManager.AddTask(&task1);
    taskManager.AddTask(&task2);
    taskManager.AddTask(&task3);
    taskManager.AddTask(&task4);
    taskManager.AddTask(&task5);
    taskManager.AddTask(&task6);
    taskManager.AddTask(&task7);
    taskManager.AddTask(&task8);
    taskManager.AddTask(&task9);
    taskManager.AddTask(&task10);
```

In this case where n is 0, it is called linearSearch 10 times. While it was printing the result an interrupt occurred and it continued with a different process. After running other processes it jumped back to the process that was interrupted and finished that as well.

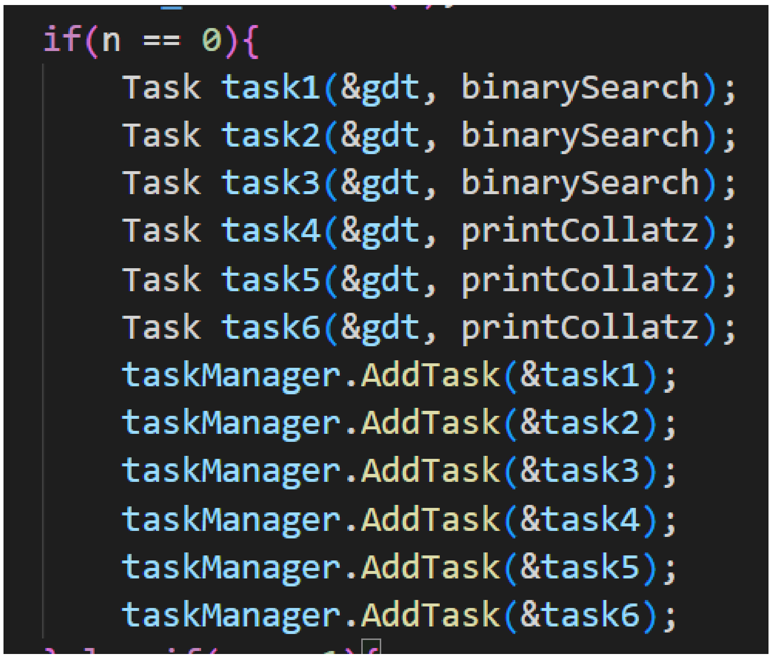
X was 20 in this case



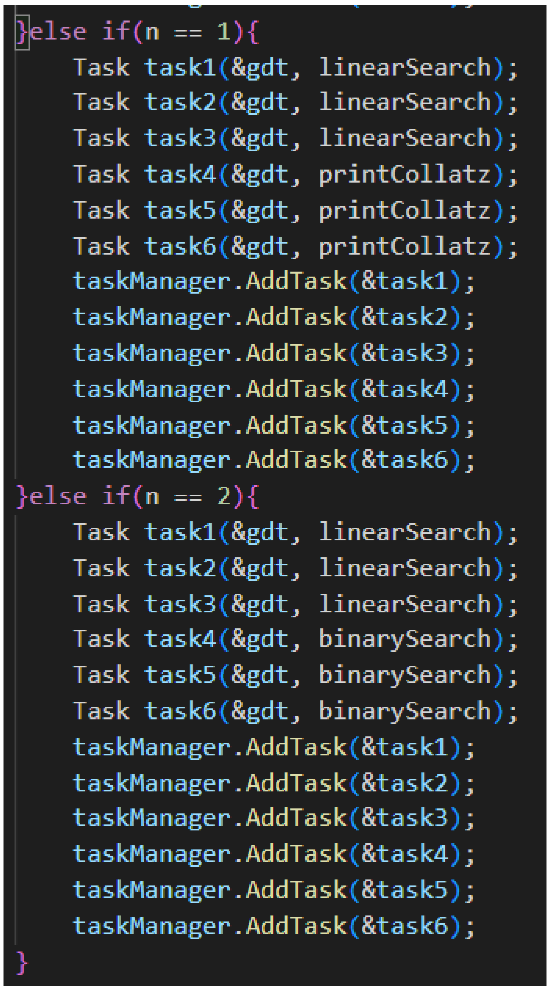
```
Hello World! --- http://www.AlgorithMan.de2222
VGA AMD am79c973
timer interrupt occured...
AMD am79c973 INIT DONE
Linear search result : 00000001
Linear sear
timer interrupt occured...
Linear search result : 00000001
Linear search result : 00000001
Linear search result : 00000001
Linear search result : 00000001
Linear search result : 00000001
Linear search result : 00000001
Linear search result : 00000001
Linear search result : 00000001
ch result : 00000001
```

Strategy Three

This is very similar to the last one, it gets a random number and if it's 0 it doesn't load linearSearch, if it’s 1 doesn’t load binarySearch and finally if it’s 2 it doesn't load collatz.

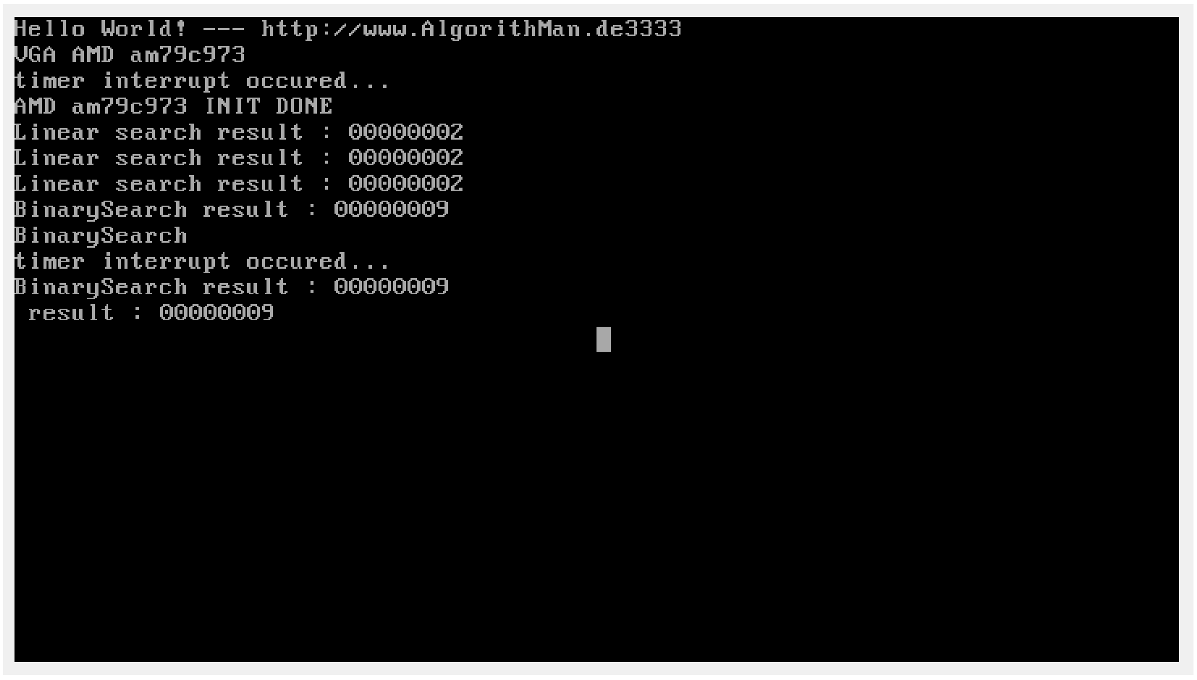


```
if(n == 0){
    Task task1(&gdt, binarySearch);
    Task task2(&gdt, binarySearch);
    Task task3(&gdt, binarySearch);
    Task task4(&gdt, printCollatz);
    Task task5(&gdt, printCollatz);
    Task task6(&gdt, printCollatz);
    taskManager.AddTask(&task1);
    taskManager.AddTask(&task2);
    taskManager.AddTask(&task3);
    taskManager.AddTask(&task4);
    taskManager.AddTask(&task5);
    taskManager.AddTask(&task6);
```



```
}else if(n == 1){
    Task task1(&gdt, linearSearch);
    Task task2(&gdt, linearSearch);
    Task task3(&gdt, linearSearch);
    Task task4(&gdt, printCollatz);
    Task task5(&gdt, printCollatz);
    Task task6(&gdt, printCollatz);
    taskManager.AddTask(&task1);
    taskManager.AddTask(&task2);
    taskManager.AddTask(&task3);
    taskManager.AddTask(&task4);
    taskManager.AddTask(&task5);
    taskManager.AddTask(&task6);
}else if(n == 2){
    Task task1(&gdt, linearSearch);
    Task task2(&gdt, linearSearch);
    Task task3(&gdt, linearSearch);
    Task task4(&gdt, binarySearch);
    Task task5(&gdt, binarySearch);
    Task task6(&gdt, binarySearch);
    taskManager.AddTask(&task1);
    taskManager.AddTask(&task2);
    taskManager.AddTask(&task3);
    taskManager.AddTask(&task4);
    taskManager.AddTask(&task5);
    taskManager.AddTask(&task6);
}
```

And this is the output for the case where n is 2

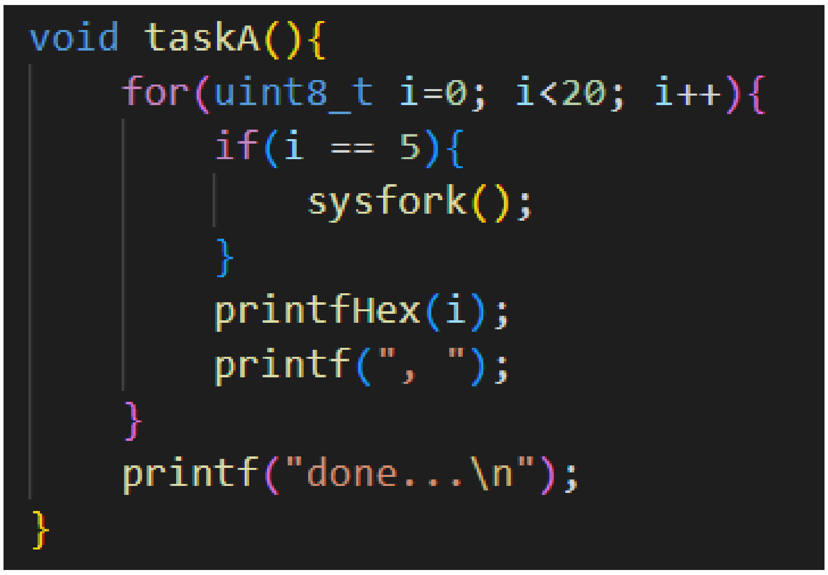


```
Hello World! --- http://www.AlgorithMan.de3333
VGA AMD am79c973
timer interrupt occured...
AMD am79c973 INIT DONE
Linear search result : 00000002
Linear search result : 00000002
Linear search result : 00000002
BinarySearch result : 00000009
BinarySearch
timer interrupt occured...
BinarySearch result : 00000009
 result : 00000009
```

System Calls

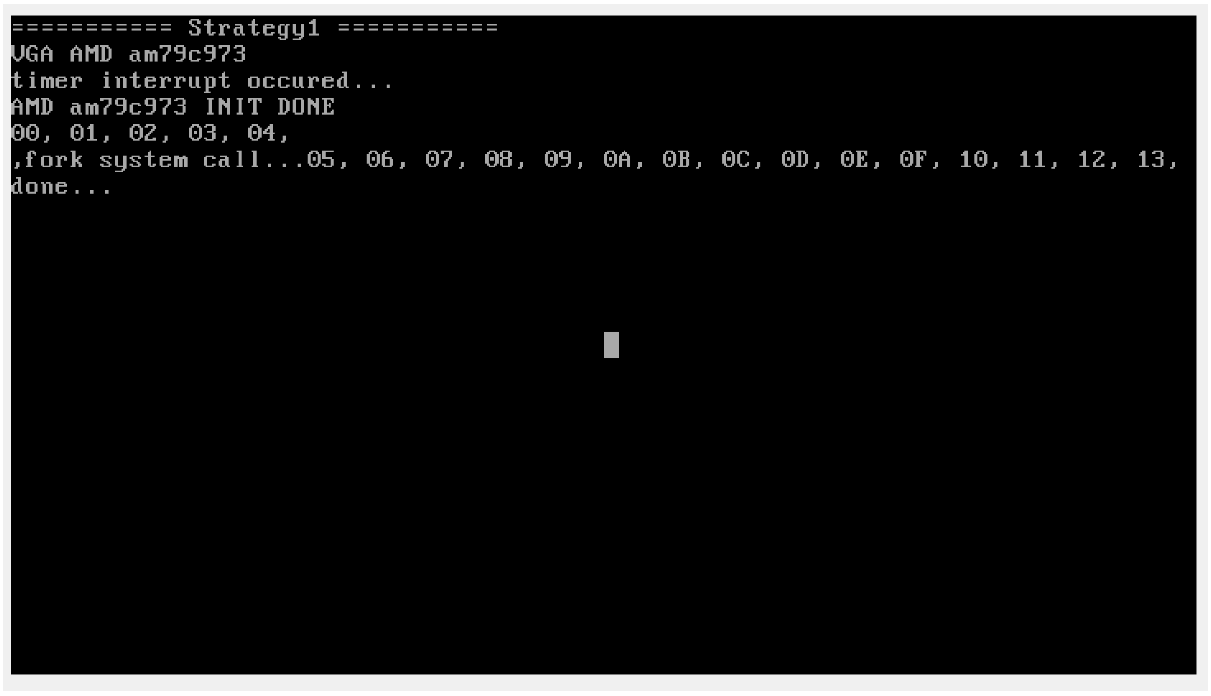
Fork

Inside the **syscalltest.cpp** file I created this task, it basically prints numbers from 0 to 19 and calls fork once. What we normally expect from fork to create a child process based on the parent process. And when the child process is scheduled it should start executing from the point where it's first created.



```
void taskA(){
    for(uint8_t i=0; i<20; i++){
        if(i == 5){
            sysfork();
        }
        printfHex(i);
        printf(", ");
    }
    printf("done...\n");
}
```

But in our case we get an output like this:



```
=========== Strategy1 ===========
VGA AMD am79c973
timer interrupt occured...
AMD am79c973 INIT DONE
00, 01, 02, 03, 04,
,fork system call...05, 06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F, 10, 11, 12, 13,
done...
```

As you can see it printed out 'fork system call', so we can understand that this system call is handled. The reason it fails, I think copy constructors don't copy the variables properly and the child process gets the same stack address and the same cpustate. That's why when a timer interrupt occurs, the scheduled process just continues where the previous is interrupted.

Since the fork system call doesn't work properly I couldn't implement the rest of the system calls.

How to Run

make mykernel1.iso
This will create mykernel1.iso which has the 1st strategy
Strategy1 is in **strategy1.cpp**

make mykernel2.iso
This will create mykernel1.iso which has the 2nd strategy
Strategy2 is in **strategy2.cpp**

make mykernel3.iso
This will create mykernel1.iso which has the 3rd strategy
Strategy3 is in **strategy3.cpp**

make mykernel4.iso
This will create mykernel1.iso which has the system call test
Strategy4 is in **strategy4.cpp**

make clean
This will delete all the object, bin and iso files