OPERATING SYSTEM

REPORT

ŞEHMUS ACAR
161044085
CSE312

Purpose

The purpose of this documentation is to analyse how to solved this problem and which solution way has been used.

Design

Exit Mechanism

Firstly I want to explain what features are added to the project.Firstly in the initial project , there isn't any exit mechanism from a process that's why we can not finish any task , if we finish any task we get the interrupt handler error. For this feature I added a field to the task class , so each task has its own "isDone" field and this field has been setted as false in the constructors. I set this field as true in just exit() function so the operating system can detect which task has been finished and which task should be deleted during the interrupts via this field.

```
class Task
{
friend class TaskManager;
private:
    common::uint8_t stack[4096]; // 4 KiB
    CPUState* cpustate;
    int secret_id;
    int parent_id;
public:
    Task(GlobalDescriptorTable *gdt, void entrypoint());
    Task(GlobalDescriptorTable *gdt, int entrypoint(), int ret_value);
    Task(Task& task, GlobalDescriptorTable *gdt, uint32_t esp);
    void setCpuState(CPUState* cpustate, GlobalDescriptorTable *gdt);
    ~Task();
    CPUState* getCpuState();
    bool isDone;
    bool settedEntryPoint;
    uint32_t based_eip;
    static int id;
    inline int getId(){return secret_id;}
    inline int getParentId(){return parent_id;}
};
```

```
void exit(){
    taskManager.getCurrentTask()->isDone = true;
    for(int i=0;i<1000000;i++)
        for(int j=0;j<1000000;j++);
}
```

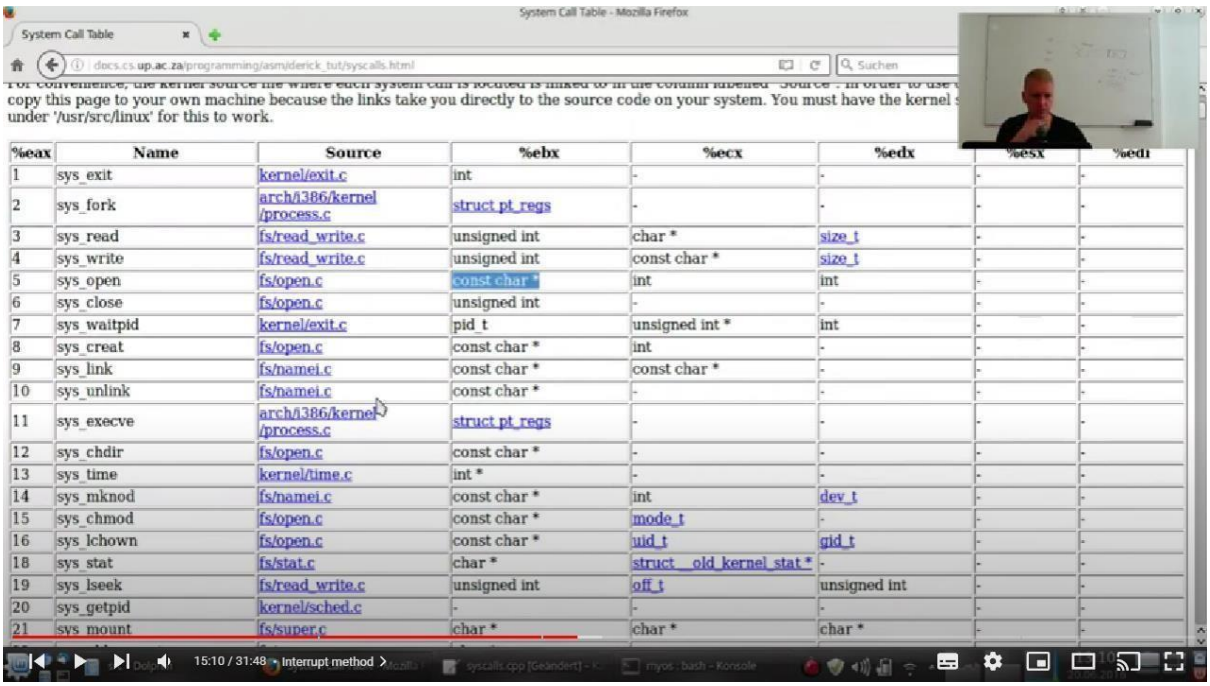
I had to put some loop after the setting isDone line because we are making the deleting section in the interrupt handler so after the setting isDone as true the function needs to wait for the interrupt.

In the interrupt handler I checked the number of task and if it is bigger than 0 it's checking if the current task is done or not. If the current task is done , firstly it's scheduling because the cpu should not be empty. So after the scheduling , it's removing the task from the process table.

```
if(taskManager->getNumTasks() > 0 ){
    if(taskManager->getCurrentTask()->isDone == true){
        printf("Removing the task which id is ");
        printfHex(taskManager->indexOfCurrentTask());
        printf(" total task number is ");
        printfHex(taskManager->getNumTasks());
        printf("\n");
        int curTask = taskManager->indexOfCurrentTask();
        esp = (uint32_t)taskManager->Schedule((CPUState*)esp);
        taskManager->deleteTask(curTask);
    }
    else{
        esp = (uint32_t)taskManager->Schedule((CPUState*)esp);
    }
}
```

FORK

For this feature I use the code which is in the 20.video of implements own operation id youtube playlist. There is a system calls handler and I modified this handler too.I checked the system call table from this video and I get the values of eax and ebx registers of fork system call from this table.



%eax	Name	Source	%ebx	%ecx	%edx	%esx	%edi
1	sys_exit	kernel/exit.c	int	-	-	-	-
2	sys_fork	arch/i386/kernel /process.c	struct pt_regs	-	-	-	-
3	sys_read	fs/read_write.c	unsigned int	char *	size_t	-	-
4	sys_write	fs/read_write.c	unsigned int	const char *	size_t	-	-
5	sys_open	fs/open.c	const char *	int	int	-	-
6	sys_close	fs/open.c	unsigned int	-	-	-	-
7	sys_waitpid	kernel/exit.c	pid_t	unsigned int *	int	-	-
8	sys_creat	fs/open.c	const char *	int	-	-	-
9	sys_link	fs/namei.c	const char *	const char *	-	-	-
10	sys_unlink	fs/namei.c	const char *	-	-	-	-
11	sys_execve	arch/i386/kernel /process.c	struct pt_regs	-	-	-	-
12	sys_chdir	fs/open.c	const char *	-	-	-	-
13	sys_time	kernel/time.c	int *	-	-	-	-
14	sys_mknod	fs/namei.c	const char *	int	dev_t	-	-
15	sys_chmod	fs/open.c	const char *	mode_t	-	-	-
16	sys_lchown	fs/open.c	const char *	uid_t	gid_t	-	-
18	sys_stat	fs/stat.c	char *	struct __old_kernel_stat *	-	-	-
19	sys_lseek	fs/read_write.c	unsigned int	off_t	unsigned int	-	-
20	sys_getpid	kernel/sched.c	-	-	-	-	-
21	sys_mount	fs/super.c	char *	char *	char *	-	-

As can be seen , the eax value of fork system call is 2 so I added a case to switch case in the syscalls.cpp for fork function.

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



    switch(cpu->eax)
    {
        case 2:
            fork(cpu->eip);
            break;
        case 4:
            printf((char*)cpu->esp);
            break;
        default:
            break;
    }


    return esp;
}
```

And I created a fork function which is getting the current task and creating a child task from the current task.

```
void myos::fork(uint32_t esp){
    Task* temp = taskManager.getCurrentTask();
    Task* task3 = new Task(*temp, &gdt, esp);
    taskManager.AddTask(task3);
}
```

And there is where I'm calling the system calls handler with the parameters whose are belong to the fork function and in this function I'm checking if there is any switch it returns 0 otherwise it returns 1;

```
int forkk(){
    int beforeFork = taskManager.getCurrentTask()->getId();
    CPUState* cpustate = taskManager.getCurrentTask()->getCpuState();
    asm("int $0x80" : : "a" (2), "b" (cpustate));
    int afterFork = taskManager.getCurrentTask()->getId();
    if(beforeFork == afterFork){
        return 1; // From parent
    }
    else{
        return 0; // From child
    }
}
```

But at this point there is a trick. I added a field whose name is **settedEntryPoint** and it has been setted as false in the constructor of child processes. This means that the entry points of child processes have been set in the first scheduling and so they can not reach the code block where the before fork functions.

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

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

    if(++currentTask >= numTasks)
        currentTask %= numTasks;
    if(tasks[currentTask]->settedEntryPoint == false){
        tasks[currentTask] = new Task(*tasks[0],&gdt,-1);
        tasks[currentTask]->settedEntryPoint = true;
        tasks[currentTask]->isDone = false;
    }
    return tasks[currentTask]->cpustate;
}
```

Test Cases Of Fork Function

I wrote some functions for testing the fork. You can see the screenshots of functions and outputs below.

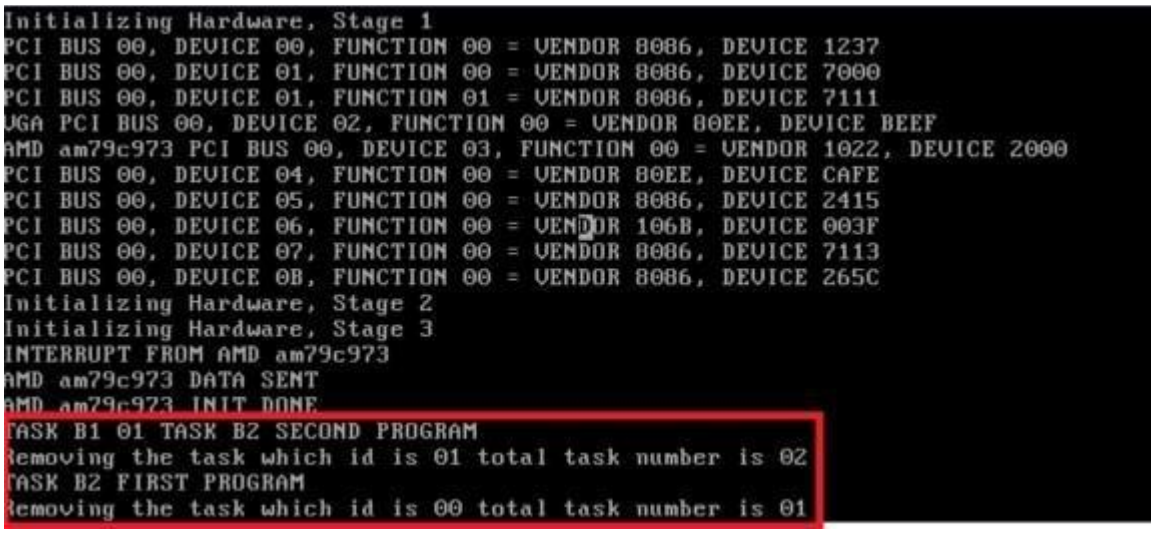
Function

```
void TestFork()
{
    printf("TASK B1 ");
    printfHex(taskManager.getNumTasks());
    printf(" ");
    for(int i=0;i<100000;i++)
        for(int j=0;j<100000;j++);
    forkk();
    for(int i=0;i<100000;i++)
        for(int j=0;j<100000;j++);
    if(taskManager.indexOfCurrentTask() == 0)
        printf("TASK B2 FIRST PROGRAM\n");
    else
        printf("TASK B2 SECOND PROGRAM\n");
    exit();
}
```

Expectation

The expectation is that the parent process will print the TASK B1 and TASK B2 FIRST PROGRAM and the child process will print **the just TASK B2 SECOND PROGRAM.**

Screenshot of output



As you can see , it prints TASK B1 just 1 time so that means that the child process is working after the fork function.

Scenarios

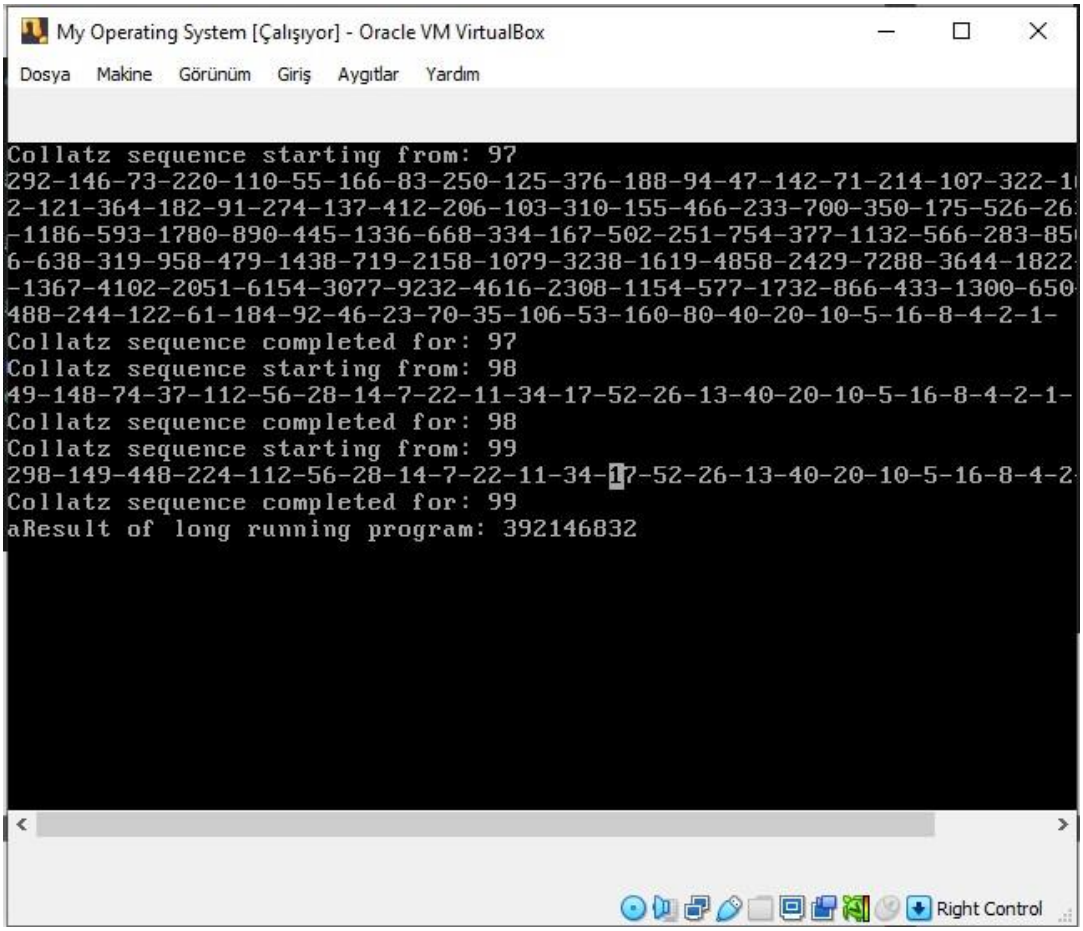
First Scenario for PartA

In this scenario , I'm creating 2 tasks which are collatz and long_running_program and run them 3 times in sequence. These tasks will run until they are finished. You can see the screenshots of functions and outputs below.

```
void partA_firstStrategy() {

    _fork(collatz);
    _fork(long_running_program);
    _fork(collatz);
    _fork(long_running_program);
    _fork(collatz);
    _fork(long_running_program);

    while (taskManager.getNumTasks() > 2) {
        for (int i = 0; i < 100000; i++)
            for (int j = 0; j < 100000; j++);
    }
    exit();
}
```

```
void collatz() {
    for (int i = 1; i < 100; ++i) {
        int n = i;
        printf("Collatz sequence starting from: ");
        printfInt(n);
        printf("\n");

        while (n != 1) {
            if (n % 2 == 0) {
                n = n / 2;
            } else {
                n = 3 * n + 1;
            }
            printfInt(n);
            printf("-");
        }

        printf("\nCollatz sequence completed for: ");
        printfInt(i);
        printf("\n");
    }

    exit();
}
```

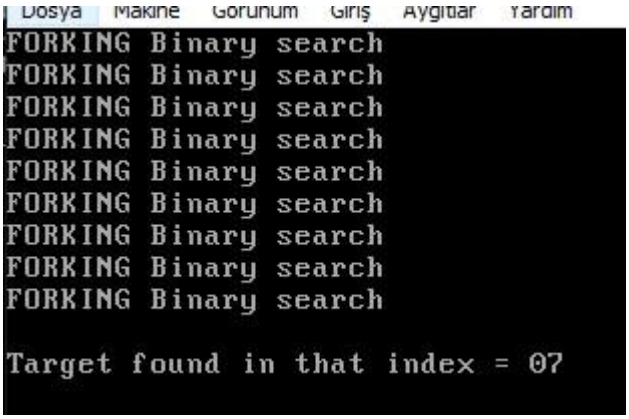
```
void long_running_program() {
    int n = 1000;
    long long result = 0;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            result += i * j;
        }
    }

    printf("Result of long running program: ");
    char buffer[20];
    itoa(result, buffer, 10);
    printf(buffer);
    printf("\n");

    while (true);
}

void _fork(void (*entrypoint)()) {
    Task* temp = new Task(&gdt, entrypoint, 0);
    taskManager.AddTask(temp);
}
```



First Scenario for PartB

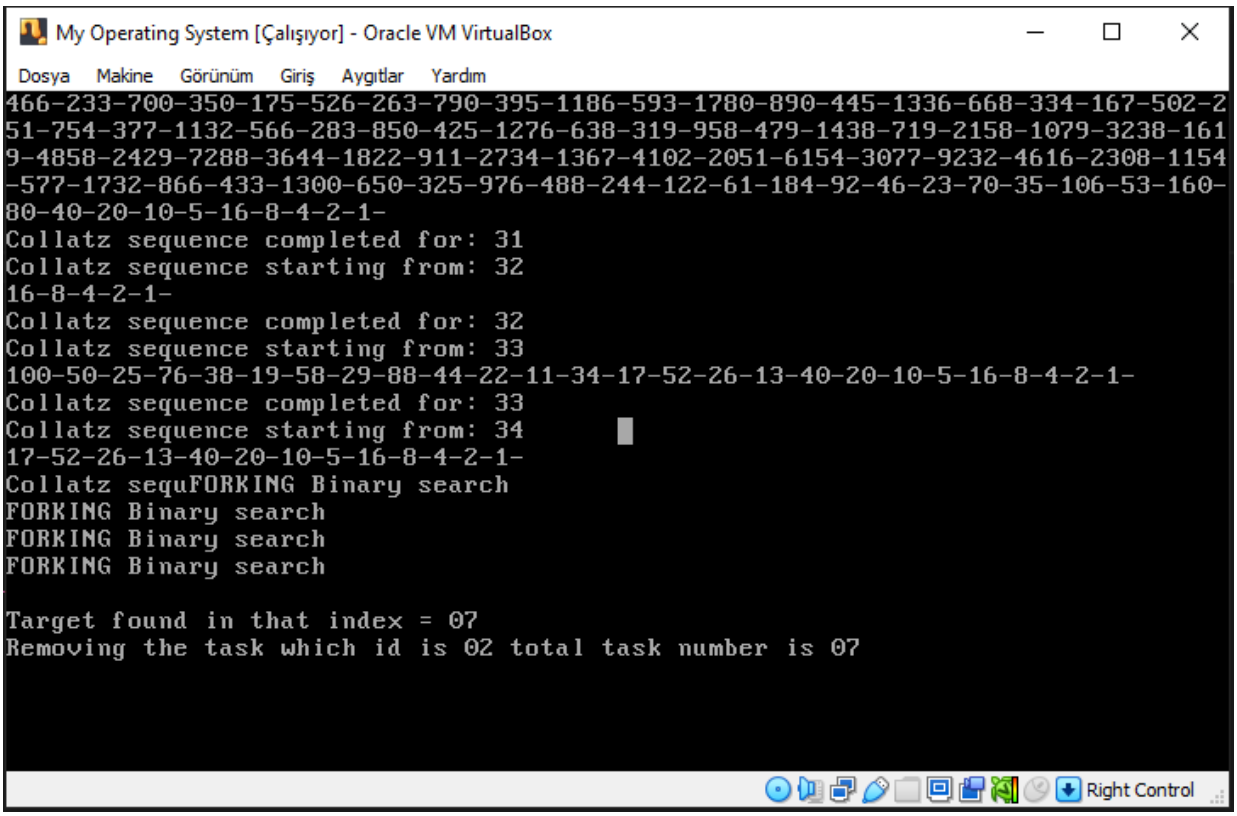
In this scenario I create randomly 4 tasks, collatz , long_running_program ,binarysearch and linearsearch and run them 10 times in sequence. These tasks will run until they are finished. I selected the binary search function for this scenario and I created a task of binarysearch and in this function it will execute the fork function 10 times.

```
void partB_firstStrategy() {
    srand(12345);  // Rastgele sayı üretici fonksiyonu başlatmak için bir seed kullan

    // Rastgele bir program seç
    int programIndex = rand() % 4;
    void (*programs[4])() = {collatz, binarySearch, linearSearch, long_running_program};

    // Seçilen programı 10 kez yükle
    for (int i = 0; i < 10; ++i) {
        _fork(programs[programIndex]);
    }

    // Tüm süreçler bitene kadar sonsuz döngüye gir
    while (taskManager.getNumTasks() > 1) {
        for (int i = 0; i < 100000; i++)
            for (int j = 0; j < 100000; j++);
    }
    exit();
}
```



Second Scenario for PartB

In this scenario I create randomly 2 tasks, collatz , long_running_program ,binarysearch and linearsearch and I'm adding to the process table these tasks. These tasks will run until they are finished. You can see the screenshots of functions and outputs below.



Scenario for PartC

When we disable calling the schedule function, we can run the tasks by pressing the 'a' key on the keyboard.

```
if(interrupt == hardwareInterruptOffset)
{
    /*
    if(taskManager->getNumTasks() > 0){
        if(taskManager->getCurrentTask()->isDone == true){

            printf("Removing the task which id is ");
            printfHex(taskManager->indexOfCurrentTask());
            printf(" total task number is ");
            printfHex(taskManager->getNumTasks());
            printf("\n");
            int curTask = taskManager->indexOfCurrentTask();
            esp = (uint32_t)taskManager->Schedule((CPUState*)esp);
            taskManager->deleteTask(curTask);
        }
        else{
            esp = (uint32_t)taskManager->Schedule((CPUState*)esp);
        }
    }
    */
}
```

I assigned calling the schedule function to the 'a' key.

```
case 0x1E: handler->OnKeyDown('a');
esp = (uint32_t)taskManager->Schedule((CPUState *) esp);
break;
```