Department of "Computer Engineering"

"Operating Systems"

Project Final Report

- Event Simulator -

Lecturer: Prof. Dr. Ercan SOLAK

Leyla Abdullayeva - 1904010038
Darına Karimova - 2104011002

Project Description

Our program simulates some aspects of operating systems. The entire simulation is predicated on the text inputs that the program receives from the user.

The program should simulate the demand paging memory management.
When a replacement process appears within the system, it's not loaded into memory.

At the beginning , the program asks the user three questions:

- How much RAM memory is there on the simulated computer?
  The program receives the amount in bytes (not kilobytes). I can enter any number up to 4000000000 (4 billion).

- What is the size of a page/frame?
  The enumeration automatically begins with 0.

- How many hard disks should be in the simulated computer?
  The enumeration starts with 0 as well.

```
int main() {

    unsigned int RAM = 0;
    int numberHardDisks = 0;
    unsigned int pageSize = 0;

    cout << "How much RAM is there?" << endl;
    std::cin >> RAM;

    cout << "What is the size of a page/frame?" << endl;
    cin >> pageSize;

    cout << "How many hard disks does the simulated computer have?" << endl;
    cin >> numberHardDisks;
```

Then the program creates an initial process, which is a system process that cannot be terminated. All other processes in the system are the descendants of this initial one. In our simulation only, this process never uses the CPU or memory.

After that, the simulation begins. The program implements according to the user's inputs. The system does not ask for the confirmation to enter another input. The user inputs signal some system events.

The possible inputs are:

‘A’ input means a replacement process has been created. When the new process has been created no memory is allocated for it. The program places the newly created process at the end of the ready-queue.

The PID for the new process starts from 2 because the first one belongs to the first system process.

'S r' displays what process is currently using the CPU and what processes are waiting within the ready-queue.

'fork' should be a child process which is placed at the end of the ready-queue.

'toQueue' moves the process currently running in the CPU to the end of the ready-queue. In case if there is one process in the ready queue, it places a new process into the CPU.

```
    System os(RAM, pageSize, numberHardDisks);

    string input;
    cin>>input;

    //while(1) {
        //to display which process is using the CPU and which is waiting in the ready queue
        if (input == "S r") {
            os.showCurrentProcess();
        }

        //the process using the CPU forks a child.
        //the child is places in the end of the ready queue
        else if (input == "fork") {
            //os.fork();
        }

        //to move the current process in CPU to the end of the ready queue
        //plus, places a new process into the CPU
        else if  (input == "toQueue"){
            os.fromCPUtoReadyQ();
        }

        //to create a process and places at the end of ready queue
        // if the ready queue is empty it places the process in the CPU
        else if (input == "A"){
            os.createProcess();
        }
    //}

}
```

Scheduling Algorithm

CPU Scheduling is a process of determining which process will own the CPU for execution while another process is on hold. The main task of CPU scheduling is to make sure that whenever the CPU remains idle, the OS at least selects one of the processes available in the ready queue for execution. A Scheduling Algorithm is an algorithm that tells us how much CPU time we can allocate to the processes.

1)First Come First Serve (FCFS)

First Come First Serve is the full form of FCFS. It is the easiest and most simple CPU scheduling algorithm. In this type of algorithm, the process which requests the CPU gets the CPU allocation first. This scheduling method can be managed with a FIFO queue.
As the process enters the ready queue, its PCB (Process Control Block) is linked with the tail of the queue. So, when the CPU becomes free, it should be assigned to the process at the beginning of the queue.

2) Shortest Remaining Time

The full form of SRT is Shortest remaining time. It is also known as SJF preemptive scheduling. In this method, the process will be allocated to the task, which is closest to its completion. This method prevents a newer ready state process from holding the completion of an older process.

3) Round Robin Scheduling

Round robin is the oldest, simplest scheduling algorithm. The name of this algorithm comes from the round-robin principle, where each person gets an equal share of something in turn. It is mostly used for scheduling algorithms in multitasking. This algorithm method helps for starvation-free execution of processes.

Context Switching

Context Switching involves storing the context or state of a process so that it can be reloaded when required and execution can be resumed from the same point as earlier. This is a feature of a multitasking operating system and allows a single CPU to be shared by multiple processes.

```
void print_table(vector<process> p, int n){
    sort(p.begin(),p.end(),cmp1);
    int i;
    cout<<"+-----+--------------+------------+--------------+-----------------+\n";
    cout<<"| PID | Arrival Time | Burst Time | Waiting Time | Turnaround Time |\n";
    cout<<"+-----+--------------+------------+--------------+-----------------+\n";

    for(i=0; i<n; i++) {
        printf("| %2d  |      %2d       |     %2d     |      %2d      |        %2d
        cout<<"+-----+--------------+------------+--------------+-----------------+\n";
    }
    cout<<"\n\n";
}
```

Virtual and Physical memory and address translation using a page table

Virtual addresses are used by the program executed by the accessing process, while physical addresses are used by the hardware, or more specifically, by the random-access memory (RAM) subsystem. The page table is a key component of virtual address translation that is necessary to access data in memory. A virtual memory address comes in and needs to be translated to the physical address

```
int toInt(string bin)
{
    int spot = 0, num = 0;
    for(int i = (bin.size()-1); i >= 0; i--){
            if(bin[i]== '1'){
                num += pow(2,spot);
            }
            spot++;
        }
    return num;
}

string toBinary(int n)
{
    string r;
    while(n!=0) {r=(n%2==0 ?"0":"1")+r; n/=2;}
    while(r.size() != 32) {r = '0' + r;}

    return r;
}

int main()
{
    ifstream address;
    int page, offset, data;
    string binary;
    address.open("address.txt");
    while(!address.eof())
    {

        address >> data;

        binary = toBinary(data);

        page = toInt(binary.substr(16,8));
        offset = toInt(binary.substr(24,8));
```