OPERATING SYSTEMS

Introduction

The operating system is that part that deals and manages the physical components directly, which is the kernel that we do not see and do not deal with directly, but we do not dispense with its services that are the reason for running the rest of the programs and interfaces that we deal with. Some add to the OS the cover and the graphical interface through which we use the system.

In our project, we will explain OS history, services, operations and structures.

History of Operating System

The history of operating systems was closely related to computer engineering. Through the architecture of computers, the history of operating systems that were implemented on them will be studied. The development of operating systems has witnessed several stages in several decades.

2.1 First Generations

In the 1940's, operating systems were not present in computers. The devices were rudimentary and the programs were inserted into rows of connectivity boards each time. At that time, assembly languages and programming languages were unknown and operating systems were non-existent and unknown.

2.2 Second Generation

In the early 1950's, the modus operandi was somewhat improved as punch cards were used. The first operating systems for the IBM 701 were implemented in General Motors laboratories. At that time, the system was only able to run one task at a time. Programs and data are provided in packages, so operating systems were called single-stream batch processing systems.

2.3 Third Generation

In the 1960's, operating systems remained grouped systems, but they evolved from the pursuit of operating several functions in one single function instead of by making better use of computer resources. There are many functions in the main memory at the same time, so the concept of multiple programming has been developed in order to move from processing one job to another as needed, thus achieving two benefits, which are providing many functions at the same time, and the second benefit is keeping the peripheral devices in use.

In other words, in this generation, through multiple programming, the problem of the CPU remaining in a suspended state when the current task is finished due to waiting for the completion of the other I / O process has been resolved. In multi-processing, the memory is divided into several parts, and in each section there is a different function. Thus when the I / O is waiting for a task, the CPU can be used by another task.

Operating systems in this generation feature spooling technology. In this technique, a high-speed device is run in the input and output between the executable program and a low-speed device. Also, operating systems in this generation are characterized by a timesharing technology from multiple programming techniques, in which a system responds to user requests quickly because each user has his own station and interacts with the computer through it. The importance of this technology in increasing the number of users at the same time.

2.4 Fourth Generation

This generation saw the development of large-scale integration circuits, so the operating system and chips were introduced into computers. Desktop computers were manufactured using microprocessor technology.

This generation saw the domination of two operating systems: MSDOS for devices using the Intel 8088 CPU. In addition, UNIX system for devices using the Motorola 6899.

Operating Systems Services

Operating systems services include providing an implementation environment for programs, and services for implementing programs easily for users. These services include:

- **Program execution**

Operating systems offer many activities that are encapsulated as a process, such as name servers, printer spooling, and other user programs and system programs. Executing a process includes code and data that needs to be processed, logs, and system resources.
In terms of managing programs, the operating system performs several activities that include loading the program into memory and then executing it, and providing mechanisms to deal with deadlock, connection process, and synchronization of operations.

- **I/O operations**

Programs may require a file or an I / O device, and devices may require special functions such as clearing the display or DVD drive.

- **File System manipulation**

Programs may read and write files. Also, you need to search for a specific file or create files and delete them. Operating systems provide several types of file systems and include permission management to allow access to them

- **Communication**

Communication and information exchange between operations, whether performed on the same computer or on different devices linked together by a computer network. Shared memory is used to carry out communications and pass messages.

- **Error Detection**

Detecting and correcting errors, such as errors in I / O devices or memory devices, where each operating system determines the appropriate action for each type of error.

- **Resource Allocation**

Allocate resources to multiple users working at the same time. Different types of resources are managed using the operating system such as main memory, and file storage.

- **Protection**

Protection means being able to fully control access to system resources. The process should not be able to interfere with other processes when more than one operation is executed at the same time

Operating System Algorithms

Of all the scheduling algorithms, the simplest is the non-preemptive FCFS algorithm.

Algorithm principle: Processes use central processing units (CPUs) in the order they are requested. Just as you buy things to queue, whoever ranks first will be executed first, and will not be interrupted during their execution. When other people want to enter the memory to be executed, they have to wait in line. If something happens during execution, it doesn't want to line up now, and the next command will consist. If he wanted to line up again at this time, he could only stand at the end of the line.

Algorithm advantages: easy to understand, easy to implement, only requires a queue (FIFO), which is fairly fair

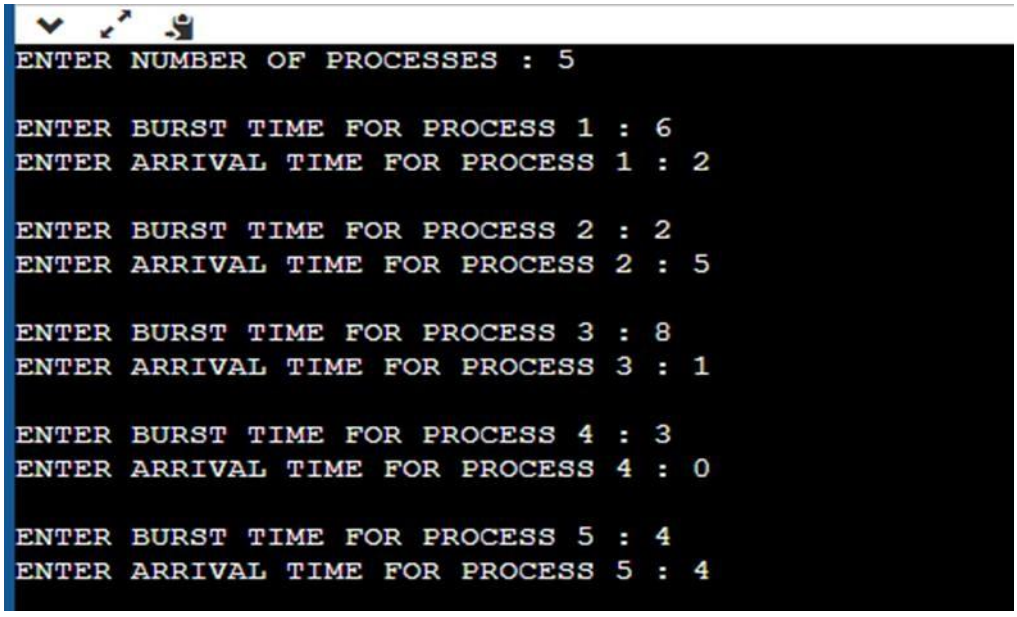
Disadvantages of the algorithm: it is more suitable for long operations, not short ones, for CPU occupied operations, but not for I/O occupied operations

Code

```
#include<iostream>
#define MAX_PROCESS 10
using namespace std;
class process
{
public:
int process_num; int burst_time; int arrival_time;
int response_time; bool processed=false; int waiting_time;
int turnaround_time; void input_process(int); int get_at()
{
return arrival_time;
}
};
void process::input_process(int count)
{
process_num=count+1;
cout<<"\nENTER BURST TIME FOR PROCESS "<<count+1<<" : ";
cin>>burst_time;
cout<<"ENTER ARRIVAL TIME FOR PROCESS "<<count+1<<" : ";
cin>>arrival_time;
}
void calc_wait_tat(process*,int); void average(process*,int);
void display(process*,int); void FCFS(process* , int); void SJF(process*,int);

int main()
{
process p[MAX_PROCESS],temp; int num,i,j;
cout<<"ENTER NUMBER OF PROCESSES : ";
cin>>num; for(i=0;i<num;++i) p[i].input_process(i); for(i=0;i<num;++i)
{
for(j=i+1;j<num;++j)
{

if(p[i].get_at()>p[j].get_at())
{
temp=p[i]; p[i]=p[j]; p[j]=temp;
}
}
}
FCFS(p,num);
SJF(p,num); return 0;
}
void FCFS(process* p , int num)
{
calc_wait_tat(p,num); cout<<endl<<"First Come First Served";
display(p,num);
}
void SJF(process *p,int num)
{ process temp; p[0].waiting_time=0;
```

```
p[0].turnaround_time=p[0].burst_time; p[0].processed=true;
int timed=p[0].burst_time; bool check=true;
for(int i=0;i<num;++i)
{
for(int j=i+1;j<num;++j)
{
if(p[i].burst_time>p[j].burst_time)
{
temp=p[i]; p[i]=p[j]; p[j]=temp;
}
}
}

while(check)
{
check=false;
for(int i=0; i<num;i++)
{
if(timed>p[i].arrival_time && p[i].processed !=true)
{
p[i].waiting_time=timed-p[i].arrival_time;

p[i].turnaround_time=p[i].waiting_time+p[i].burst_time; p[i].processed=true;
timed=timed + p[i].burst_time; check=true;

break;
}
}
}
cout<<endl<<"Shortest Job First (SJF)"<<endl; display(p,num);

}
void calc_wait_tat(process *p,int n)
{
int i; p[0].response_time=0; for(i=1;i<n;++i)
{
p[i].response_time=p[i-1].burst_time+p[i-1].response_time;
if(p[i].response_time<p[i].arrival_time)
p[i].response_time=p[i].arrival_time;
}
p[0].waiting_time=0; for(i=1;i<n;++i)
p[i].waiting_time=p[i].response_time-p[i].arrival_time;
for(i=0;i<n;++i) p[i].turnaround_time=p[i].waiting_time+p[i].burst_time;
}
void average(process *p,int n)
{
float avg_wt=0,avg_tat=0; for(int i=0;i<n;++i)
{
avg_wt+=(float)p[i].waiting_time; avg_tat+=(float)p[i].turnaround_time;
}
avg_wt/=n; avg_tat/=n;
```

```
        cout<<"\n\nAVERAGE WAITING TIME : "<<avg_wt; cout<<"\nAVERAGE TURN
        AROUND TIME : "<<avg_tat;
        }
        void display(process *p,int n)
        {       cout<<endl;
        cout<<"Processes "<<" Burst time "<<" Waiting time "<<" Turn around time\n";
        for (int i=0;i<n;i++)

        {
        cout<<"\n"<<p[i].process_num<<"\t\t"<<p[i].burst_time<<"\t"<<p[i].waiting_time<<"\t\t"<
        <p[i].turnaround_time;
        }
        average(p,n);
}
```



```
ENTER NUMBER OF PROCESSES : 5

ENTER BURST TIME FOR PROCESS 1 : 6
ENTER ARRIVAL TIME FOR PROCESS 1 : 2

ENTER BURST TIME FOR PROCESS 2 : 2
ENTER ARRIVAL TIME FOR PROCESS 2 : 5

ENTER BURST TIME FOR PROCESS 3 : 8
ENTER ARRIVAL TIME FOR PROCESS 3 : 1

ENTER BURST TIME FOR PROCESS 4 : 3
ENTER ARRIVAL TIME FOR PROCESS 4 : 0

ENTER BURST TIME FOR PROCESS 5 : 4
ENTER ARRIVAL TIME FOR PROCESS 5 : 4
```

```
First Come First Served
Processes   Burst time   Waiting time   Turn around time

   4            3            0              3
   3            8            2              10
   1            6            9              15
   5            4            13             17
   2            2            16             18

AVERAGE WAITING TIME : 8
AVERAGE TURN AROUND TIME : 12.6
Shortest Job First (SJF)

Processes   Burst time   Waiting time   Turn around time

   2            2            4              6
   4            3            0              3
   5            4            7              11
   1            6            1              7
   3            8            14             22

AVERAGE WAITING TIME : 5.2
AVERAGE TURN AROUND TIME : 9.8
```

Conclusion:

FCFS algorithm is easy to implement but The process with less execution time suffer i.e. waiting time is often quite long. Here, first process will get the CPU first, other processes can get CPU only after the current process has finished it's execution then the processes will have to wait more unnecessarily, this will result in more average waiting time This effect results in lower CPU and device utilization.

SJF is Shorter process are performed first so the waiting time is less but SJF may cause starvation, if shorter processes keep coming. This problem is solved by aging.

Reference

- A. Silberschatz, P. B. Galvin and, G. Gagne," OPERATING SYSTEM CONCEPTS", 2013