PRIORITY SCHEDULING

Name : V.Angu HariHara Prasad

Reg : 11609764

Sec : K1620

Roll :B33

Email:angu123angu@gmail.com

Link :

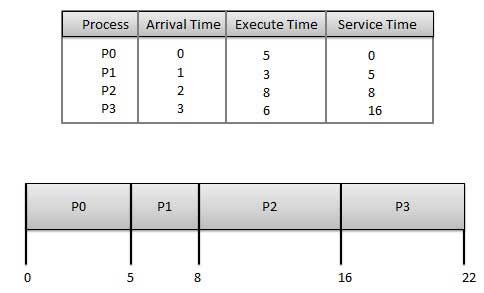
A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithms which we are going to discuss in this chapter −

* First-Come, First-Served (FCFS) Scheduling
* Shortest-Job-Next (SJN) Scheduling
* Priority Scheduling
* Shortest Remaining Time
* Round Robin(RR) Scheduling
* Multiple-Level Queues Scheduling

These algorithms are either **non-preemptive or preemptive**. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

First Come First Serve (FCFS)

* Jobs are executed on first come, first serve basis.
* It is a non-preemptive, pre-emptive scheduling algorithm.
* Easy to understand and implement.
* Its implementation is based on FIFO queue.
* Poor in performance as average wait time is high.



**Wait time** of each process is as follows −

|  |  |
| --- | --- |
| **Process** | **Wait Time : Service Time - Arrival Time** |
| P0 | 0 - 0 = 0 |
| P1 | 5 - 1 = 4 |
| P2 | 8 - 2 = 6 |
| P3 | 16 - 3 = 13 |

Average Wait Time: (0+4+6+13) / 4 = 5.75

**Test cases :**

Order in which processes gets executed

1 3 2

Processes Burst time Waiting time Turn around time

1 10 0 10

3 8 10 18

2 5 18 23

**Algorithm:**

**while(j<=max)**

**{i=1;**

**while(i<=n)**

**{**

**if(P[i]==j)**

**{**

**Wt[i]=w;**

**w=w+B[i];**

**}i++;}**

**j++;**

**Code:**

**#include<iostream>**

**#include<conio.h>**

**#include<stdio.h>**

**using namespace std;**

**class Cpu**

**{**

**int n,Bu[20];**

**float Total\_wt,Average\_wt,A[10],Wt[10],w;**

**public:**

**void Getdata();**

**void Priority();**

**};**

**void Cpu::Getdata()**

**{**

**int i;**

**cout<<"Enter no of processes to run:"<<endl;**

**cin>>n;**

**for(i=1;i<=n;i++)**

**{**

**cout<<"Enter burstTime for each process "<<i<<"= "<<endl;**

**cin>>Bu[i];**

**}**

**}**

**void Cpu::Priority()**

**{**

**int i,B[10],P[10],j;**

**w=0.0;**

**int maximum;**

**Total\_wt=0.0;**

**maximum=1;**

**for(i=1;i<=n;i++)**

**{**

**B[i]=Bu[i];**

**cout<<"Burst time for each process p"<<i<<"= "<<endl;**

**cout<<B[i];**

**cout<<"Give the priority for each process P"<<i<<"= "<<endl;**

**cin>>P[i];**

**if(maximum<P[i])**

**maximum=P[i];**

**}**

**j=1;**

**while(j<=maximum)**

**{**

**i=1;**

**while(i<=n)**

**{**

**if(P[i]==j)**

**{**

**Wt[i]=w;**

**w=w+B[i];**

**}**

**i++;**

**}**

**j++;**

**}**

**for(i=1;i<=n;i++)**

**Total\_wt=Total\_wt+Wt[i];**

**Average\_wt=Total\_wt/n;**

**cout<<"Total Waiting Time="<<Total\_wt<<""<<endl;**

**cout<<"Average Wiating Time="<<Average\_wt<<""<<endl;**

**}**

**int main()**

**{**

**int choice,cho;**

**Cpu cpu;**

**do**

**cout<<" --------------MENU---------------"<<endl;**

**cout<<"1.For process & BurstTime"<<endl;**

**cout<<"2.For Priority"<<endl;**

**cout<<"3.EXIT"<<endl;**

**cout<<"Enter your choice"<<endl;**

**cin>>choice;**

**switch(choice)**

**{**

**case 1:**

**cpu.Getdata();**

**break;**

**case 2:**

**cout<<"\*\*\*\*\*\*\*\*\*\*\*PRIORITY SCHEDULING\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;**

**cpu.Priority();**

**break;**

**case 3:**

**cout<<"";**

**break;**

**}**

**}while(choice<=3);**

**}**

**Safe and Unsafe conditions:**

1. P1 acquires 2 A, 1 B and 1 D more resources, achieving its maximum
   * [available resource: <3 1 1 2> - <2 1 0 1> = <1 0 1 1>]
   * The system now still has 1 A, no B, 1 C and 1 D resource available
2. P1 terminates, returning 3 A, 3 B, 2 C and 2 D resources to the system
   * [available resource: <1 0 1 1> + <3 3 2 2> = <4 3 3 3>]
   * The system now has 4 A, 3 B, 3 C and 3 D resources available
3. P2 acquires 2 B and 1 D extra resources, then terminates, returning all its resources
   * [available resource: <4 3 3 3> - <0 2 0 1> + <1 2 3 4> = <5 3 6 6>]
   * The system now has 5 A, 3 B, 6 C and 6 D resources
4. P3 acquires 1 B and 4 C resources and terminates.
   * [available resource: <5 3 6 6> - <0 1 4 0> + <1 3 5 0> = <6 5 7 6>]
   * The system now has all resources: 6 A, 5 B, 7 C and 6 D
5. Because all processes were able to terminate, this state is safe

**Limitations:**

Like the other algorithms, the Banker's algorithm has some limitations when implemented. Specifically, it needs to know how much of each resource a process could possibly request. In most systems, this information is unavailable, making it impossible to implement the Banker's algorithm. Also, it is unrealistic to assume that the number of processes is static since in most systems the number of processes varies dynamically. Moreover, the requirement that a process will eventually release all its resources (when the process terminates) is sufficient for the correctness of the algorithm, however it is not sufficient for a practical system. Waiting for hours (or even days) for resources to be released is usually not acceptable.

Average waiting time = 9.33333

Average turn around time = 17

**Disadvantages of Priority Scheduling**:

* Indefinite blocking or starvation.
* A **priority scheduling** can leave some low **priority** waiting processes indefinitely for CPU.
* If the system eventually crashes then all unfinished low **priority** processes gets lost.