EX No. 3 CPU Scheduling

Dr S.Rajarajan
SASTRA

Objective

Develop programs to demonstrate the various CPU scheduling algorithms and compare their performances through sample sets.

CPU Scheduling

- **Two categories**
  - Preemptive
    - FCFS, Prioriry, SJF
  - Non-preemptive
    - Priority, SRT, Multi-level feedback

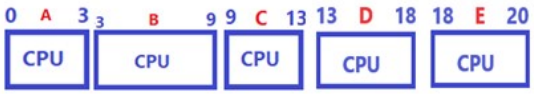
First in first out

Also known as ***F**irst **C**ome, **F**irst **S**erved* (FCFS), is the simplest scheduling algorithm, FIFO simply queues processes in the order that they arrive in the ready queue.

- Take the number of processes as input.
- For each process get the arrival time and burst time as input.
- Run the following steps until all the processes are completed
- Choose the process that arrived first and has not yet completed
- Update its completion time, mark it as completed, and reduce the number of processes to be completed, update current time

FCFS

Process	A	B	C	D	E	
Arrival Time(AT)	0	2	4	6	8	
Service Time(BT)	3	6	4	5	2	
FCFS						
Finish Time	3	9	13	18	20	Mean
Turnaround Time (FT-AT)	3	7	9	12	12	8.60
Normalized TT (TT / BT)	1.00	1.17	2.25	2.4	6	2.56
Waiting Time (TT-BT)	0	1	5	7	10	4.6



Round-robin scheduling

The scheduler assigns a fixed time unit per process, and cycles through them.

- Take the number processes as input.
- For each process get the arrival time and burst time as input.
- Take the RR time quantum as input.
- Run the following steps until all the processes are completed
- Choose the first process arrived which is not yet completed, give it the fixed RR time quantum
- Reduce the burst time of the process by the RR quantum. If the burst time has hit zero then mark it as completed and update its completion time.

Shortest Job First

This is a non-preemptive approach. The process with the lowest burst time among the arrived processes is chosen

- Take the number of processes as input.
- For each process get the arrival time and burst time as input.
- Run the following steps until all the processes are completed
- Choose the process with the smallest burst time among the processes that are not yet completed and have already arrived
- Update its completion time, mark it as completed, reduce the number of processes to be completed and update current time

Process	A	B	C	D	E	
Arrival Time(AT)	0	2	4	6	8	
Service Time(BT)	3	6	4	5	2	
SJF						
Finish Time	3	9	15	20	11	Mean
Turnaround Time (FT-AT)	3	7	11	14	2	7.4
Normalized TT (TT / BT)	1.00	1.17	2.75	2.8	1	1.744
Waiting Time (TT-BT)	0	1	7	9	0	3.4

Shortest Remaining Time

This is a non-preemptive version of the shortest job first algorithm. Initially the process with the lowest burst time is taken for execution. But while processing that process if a better process arrives (process with lesser burst time than the remaining burst time of the currently running process) then immediately preempt or stop the process and start the newly arrived process.

- Take the number processes as input.
- For each process get the arrival time and burst time as input.
- In each iteration, choose the process with the smallest burst time among the already arrived and not yet completed processes and complete it.
- Keep looking for the arrival of smaller processes.
- While the current process is running, if a smaller process arrives then halt the execution of the current process and start the execution of new process. Note that for this process you must compare only the remaining burst time of the executing process with the new process's burst time not the total burst time.
- Reduce the burst time of process as burst time – executed time.
- Repeat the steps until all the processes are completed.

Preemptive SJF or SRT

	P1	P2	P3	P4
AT	0	1	2	3
BT	8	4	9	5
FT	17	5	26	10

0-1	P1(7)
1-5	P2(0)
5-10	P4(0)
10-17	P1(0)
17-25	P3 (0)

Priority scheduling

- Take the number processes as input.
- For each process get the arrival time, burst time as input.
- In each iteration, choose the highest priority process among the already arrived not yet completed processes and complete it.
- Repeat the steps until all the processes are completed

	P1	P2	P3	P4	P5
AT	0	0	0	0	0
BT	10	1	2	1	5
Priority	3	1	4	5	2
FT	**16**	**1**	**18**	**19**	**6**
TT	**16**	**1**	**18**	**19**	**6**

0-1	P2
1-6	P5
6-16	P1
16-18	P3
18-19	P4

The average waiting time is 8.2((6+ 0 + 16 + 18 + 1) /5)

Sample Code for Priority Scheduling

```c
#include <stdio.h>
struct process
{
    int at;  // arrival time
    int st;  // service time
    int pr;  // priority
    int status; // 1- completed, 0 - not completed
    int ft; // finish time
}ready_list[10];
int n;
int main()
{
    int i,cur_time,pid;
    printf("Enter number of processes:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Process %d\n",i+1);
        printf("***********\n");
        printf("Enter Arrival Time:");
        scanf("%d",&ready_list[i].at);
        printf("Enter Burst Time:");
        scanf("%d",&ready_list[i].bt);
        printf("Priority (1-10):");
        scanf("%d",&ready_list[i].pr);
        ready_list[i].status=0;
    }
    i=0; cur_time=0;
    while(i < n)   // until all the processes are completed
    {
        pid=dispatcher(cur_time); // choose the next process for execut
        ready_list[pid].ft=cur_time + ready_list[pid].bt; // update the
        ready_list[pid].status=1;   // Process completed
        cur_time+= ready_list[pid].bt;  // Update clock time
        i++;
    }
    printf("Process\t Arrival Time\t Service Time\t Finish Time\n");
    printf("*******\t ************\t ************\t ***********\n");
    for(i=0;i<n;i++)
```

```
    {
        printf("%d\t\t%d\t\t%d\t\t\t%d\n",i,ready_list[i].at,ready_list
    }
}
int dispatcher(int time)
{
    int i,high_pr=0,index=-1;
    for(i=0;i<n;i++)
    {
        if(ready_list[i].status != 1)   // Process already completed or
        if(ready_list[i].at <= time) // Process's AT is within the curr
        if(ready_list[i].pr > high_pr)  // Choosing high priority proce
        {
            high_pr = ready_list[i].pr;
            index=i;
        }
    }
    return index;
}
```