# Operating Systems Lab
# CPU Scheduling Algorithms

*By: Muhammad Ahsan*

In this lab we will look at the following CPU Scheduling Techniques

1. FCFS CPU Scheduling Algorithm
2. Round Robin CPU Scheduling Algorithm
3. Priority CPU Scheduling Algorithm

Before we start with scheduling algorithms you first should know about different terminologies used in this lab. Following are the different times which will be considered in scheduling algorithms.

i. **Burst time**
   - Burst time is the total time taken by the process for its execution on the CPU.

ii. **Arrival time**
   - Arrival time is the time when a process enters into the ready state and is ready for its execution.

iii. **Exit time**
   - Exit time is the time when a process completes its execution and exit from the system.

iv. **Response time**
   - Response time is the time spent when the process is in the ready state and gets the CPU for the first time.

v. **Waiting time**
   - Waiting time is the total time spent by the process in the ready state waiting for CPU.
   - i.e Waiting Time = Turnaround Time – Burst Time

vi. **Turnaround time**
   - Turnaround time is the total amount of time spent by the process from coming in the ready state for the first time to its completion.
   - i.e Turnaround Time = Completion Time – Arrival Time

## (1) FCFS CPU Scheduling Algorithm

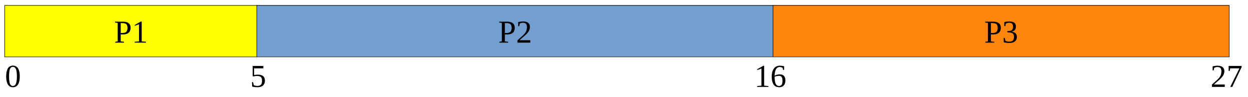
The CPU scheduling algorithm First Come, First Served (**FCFS**), also known as First In, First Out (**FIFO**), allocates the CPU to the processes in the order they are queued in the ready queue. **FCFS** uses non-preemptive scheduling, which means that once a CPU has been assigned to a process, it stays assigned to that process until it is either not terminated or may be interrupted by an I/O interrupt.

**Example:**

Following is the example of FCFS scheduling.

| **Process** | **Arrival Time** | **Burst Time** |
|---|---|---|
| P1 | 0 | 5 |
| P2 | 0 | 11 |
| P3 | 0 | 11 |

**Process Flow:**



Here turn arround time of each process is as follows.

i. P1 turnaround time: 5-0 = 5
ii. P2 turnaround time: 16-0 = 16
iii. P3 turnaround time: 27-0 = 27

**Implementation:**

Here is the source code of the C program for the FCFS Scheduling.

```
#include <stdio.h>
int main()
{
    int pid[15];
    int bt[15];
    int n;
    printf("Enter the number of processes: ");
    scanf("%d",&n);
    printf("Enter process id of all the processes: ");
    for(int i=0;i<n;i++){
        scanf("%d",&pid[i]);
    }
    printf("Enter burst time of all the processes: ");
    for(int i=0;i<n;i++){
        scanf("%d",&bt[i]);
    }
    int i, wt[n];
    wt[0]=0;
    //for calculating waiting time of each process
    for(i=1; i<n; i++){
        wt[i]= bt[i-1]+ wt[i-1];
    }
    printf("Process ID     Burst Time     Waiting Time     TurnAround Time\n");
    float twt=0.0;
    float tat= 0.0;
    for(i=0; i<n; i++){
        printf("%d\t\t", pid[i]);
        printf("%d\t\t", bt[i]);
        printf("%d\t\t", wt[i]);
        //calculating and printing turnaround time of each process
        printf("%d\t\t", bt[i]+wt[i]);
        printf("\n");
        //for calculating total waiting time
        twt += wt[i];
        //for calculating total turnaround time
        tat += (wt[i]+bt[i]);
    }
    float att,awt;
    //for calculating average waiting time
    awt = twt/n;
    //for calcuating average turnaround time
    att = tat/n;
    printf("Avg. waiting time= %f\n",awt);
    printf("Avg. turnaround time= %f\n",att);
    return 0;
}
```

## (2) Round Robin CPU Scheduling Algorithm

**Round Robin Scheduling** is a CPU scheduling algorithm in which each process is executed for a fixed time slot. Since the resources are snatched after the time slot, round robin is preemptive.
**Preemptive:** In this type, resources can be voluntarily snatched.
**Non-Preemptive:** In this type, if a process is once started, it will execute completely i.e resources cannot be snatched.

### Round Robin Scheduling Algorithm:

Step 1: Start the Program.
Step 2: Input the number of processes.
Step 3: Input the burst time and arrival time of each process and the limit of the time slot.
Step 4: Push all processes into the ready queue according to their arrival time. Then execute each process upto time slot and push left over process in queue again for execution.
Step 5: After a process is completely executed, print its turnaround time and waiting time.

**Example:**
Following is the example of round robin scheduling.

| **Process** | **Arrival Time** | **Burst Time** |
|---|---|---|
| P1 | 0 | 10 |
| P2 | 1 | 8 |
| P3 | 2 | 7 |

**Time Slot** is 5 Sec.

First **P1** is executed for 5 seconds, left burst time is 5 sec
Then **P2** is executed for 5 seconds, left burst time is 3 sec
Then **P3** is executed for 5 seconds, left burst time is 2 sec
Then **P1** is executed for 5 seconds, execution of **P1** is completed.
Then **P2** is executed for 3 seconds, execution of **P2** is completed.
Then **P3** is executed for 2 sec, execution of **P3** is completed.

Execution of all processes completed

| **Process** | **Burst Time** | **Wait Time** | **Turn Around Time** |
|---|---|---|---|
| P1 | 10 | 20 | 10 |
| P2 | 8 | 22 | 14 |
| P3 | 7 | 23 | 16 |

**Advantages:**

- There is no starvation of resources as each process gets equal share.
- Every Process gets equal allocation of CPU.
- Increases Performance time in terms of response time.

**Disadvantages:**

- More time is wasted on context switching.
- Frequent context switching increases CPU overhead.
- Average Waiting time increases for each process.

**Implementation:**

Here is the source code of the C program for the Round Robin Scheduling.

```
#include<stdio.h>
int main()
{
   //Input no of processed
   int  n;
   printf("Enter Total Number of Processes:");
   scanf("%d", &n);
   int wait_time = 0, ta_time = 0, arr_time[n], burst_time[n], temp_burst_time[n];
   int x = n;
   //Input details of processes
   for(int i = 0; i < n; i++){
      printf("Enter Details of Process %d \n", i + 1);
      printf("Arrival Time:  ");
      scanf("%d", &arr_time[i]);
      printf("Burst Time:   ");
      scanf("%d", &burst_time[i]);
      temp_burst_time[i] = burst_time[i];
   }
```

```
    //Input time slot
    int time_slot;
    printf("Enter Time Slot:");
    scanf("%d", &time_slot);
    //Total indicates total time
    //counter indicates which process is executed
    int total = 0,  counter = 0,i;
    printf("Process ID       Burst Time       Turnaround Time      Waiting Time\n");
    for(total=0, i = 0; x!=0; )  {
        // define the conditions
        if(temp_burst_time[i] <= time_slot && temp_burst_time[i] > 0){
            total = total + temp_burst_time[i];
            temp_burst_time[i] = 0;
            counter=1;
        }
        else if(temp_burst_time[i] > 0){
            temp_burst_time[i] = temp_burst_time[i] - time_slot;
            total  += time_slot;
        }
        if(temp_burst_time[i]==0 && counter==1){
            x--; //decrement the process no.
            printf("\nProcess No %d  \t\t %d\t\t\t\t %d\t\t\t %d", i+1, burst_time[i],
            total-arr_time[i], total-arr_time[i]-burst_time[i]);
            wait_time = wait_time+total-arr_time[i]-burst_time[i];
            ta_time += total -arr_time[i];
            counter =0;
        }
        if(i==n-1){
            i=0;
        }
        else if(arr_time[i+1]<=total){
            i++;
        }
        else{
            i=0;
        }
    }
    float average_wait_time = wait_time * 1.0 / n;
    float average_turnaround_time = ta_time * 1.0 / n;
    printf("\nAverage Waiting Time:%f", average_wait_time);
    printf("\nAvg Turnaround Time:%f", average_turnaround_time);
    return 0;
}
```

## (3) Priority CPU Scheduling Algorithm

**Priority Scheduling** is a CPU scheduling algorithm in which the CPU performs the task having higher priority at first. If two processes have the same priority then scheduling is done on **FCFS** basis (first come first serve). Priority Scheduling is of two types : **Preemptive** and **Non-Preemptive**.

**Example:**
Following is the example of non preemptive scheduling with arrival time zero.

| **Process** | **Burst Time** | **Priority** |
|---|---|---|
| P1 | 5 | 1 |
| P2 | 7 | 6 |
| P3 | 2 | 4 |
| P4 | 3 | 5 |

Since scheduling is non preemptive, which means that the process will be fully executed once its execution is started. So processes will be executed in the same order of priority.

**Order:** P2, P4, P3, P1

P2 will be executed from 0 to 7.
P4 will be executed from 7 to 10.
P3 will be executed from 10 to 12.
P1 will be executed from 12 to 17.

| **Process** | **Burst Time** | **Wait Time** | **Turn Around Time** |
|---|---|---|---|
| P2 | 7 | 0 | 7 |
| P4 | 3 | 7 | 10 |
| P3 | 2 | 10 | 12 |
| P1 | 5 | 12 | 17 |

**Implementation:**
Here is the source code of the C program for the Round Robin Scheduling.

```
#include <stdio.h>
//Function to swap two variables
void swap(int *a,int *b){
   int temp=*a;
```

```
    *a=*b;
    *b=temp;
}
int main(){
    int n;
    printf("Enter Number of Processes: ");
    scanf("%d",&n);
    // b is array for burst time, p for priority and index for process id
    int b[n],p[n],index[n];
    for(int i=0;i<n;i++){
        printf("Enter Burst Time and Priority Value for Process %d: ",i+1);
        scanf("%d %d",&b[i],&p[i]);
        index[i]=i+1;
    }
    for(int i=0;i<n;i++){
        int a=p[i],m=i;
        //Finding out highest priority element and placing it at its desired position
        for(int j=i;j<n;j++){
            if(p[j] > a){
                a=p[j];
                m=j;
            }
        }
        //Swapping processes
        swap(&p[i], &p[m]);
        swap(&b[i], &b[m]);
        swap(&index[i],&index[m]);
    }
    // T stores the starting time of process
    int t=0;
    //Printing scheduled process
    printf("Order of process Execution is\n");
    for(int i=0;i<n;i++){
        printf("P%d is executed from %d to %d\n",index[i],t,t+b[i]);
        t+=b[i];
    }
    printf("\n");
    printf("Process Id     Burst Time   Wait Time    TurnAround Time\n");
    int wait_time=0;
    for(int i=0;i<n;i++){
        printf("P%d          %d          %d          %d\n",index[i],b[i],wait_time,wait_time + b[i]);
        wait_time += b[i];
    }
    return 0;
}
```