K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: A2 **Roll No.: 1911027**

Experiment No. 04

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

TITLE: Implementation of Basic Process management algorithms - Preemptive (SRTN, RR, priority)

AIM: To implement basic Process management algorithms (Round Robin,SRTN, Priority)

Expected Outcome of Experiment:

CO 2. To understand the concept of process, thread and resource management.

Books/ Journals/ Websites referred:

1. **Silberschatz A., Galvin P., Gagne G. "Operating Systems Principles", Willey Eight edition.**
2. **Achyut S. Godbole , Atul Kahate "Operating Systems" McGraw Hill Third Edition.**
3. **William Stallings, "Operating System Internal & Design Principles", Pearson.**
4. **Andrew S. Tanenbaum, "Modern Operating System", Prentice Hall.**

Pre Lab/ Prior Concepts:

Most systems have a large number of processes with short CPU bursts interspersed between I/O requests and a small number of processes with long CPU bursts. To provide good time-sharing performance, we may preempt a running process to let

another one run. The ready list, also known as a run queue, in the operating system keeps a list of all processes that are ready to run and not blocked on some I/O or other system request, such as a semaphore. Then entries in this list are pointers to the process control block, which stores all information and state about a process.

When an I/O request for a process is complete, the process moves from the *waiting* state to the *ready* state and gets placed on the run queue.

The process scheduler is the component of the operating system that is responsible for deciding whether the currently running process should continue running and, if not, which process should run next. There are four events that may occur where the scheduler needs to step in and make this decision:

1. The current process goes from the *running* to the *waiting* state because it issues an I/O request or some operating system request that cannot be satisfied immediately.

2. The current process terminates.

3. A timer interrupt causes the scheduler to run and decide that a process has run for its allotted interval of time and it is time to move it from the *running* to the *ready* state.

4. An I/O operation is complete for a process that requested it and the process now moves from the *waiting* to the*ready* state. The scheduler may then decide to preempt the currently-running process and move this *ready* process into the *running* state.

The decisions that the scheduler makes concerning the sequence and length of time that processes may run is called the scheduling algorithm (or scheduling policy). These decisions are not easy ones, as the scheduler has only a limited amount of information about the processes that are ready to run. A good scheduling algorithm should:

1. Be fair – give each process a fair share of the CPU, allow each process to run in a reasonable amount of time.
2. Be efficient – keep the CPU busy all the time.

3. Maximize throughput – service the largest possible number of jobs in a given amount of time; minimize the amount of time users must wait for their results.
4. Minimize response time – interactive users should see good performance
5. Minimize overhead – don't waste too many resources. Keep scheduling time and context switch time at a minimum.
6. Maximize resource use – favor processes that will use underutilized resources. There are two motives for this. Most devices are slow compared to CPU operations. We'll achieve better system throughput by keeping devices busy as often as possible. The second reason is that a process may be holding a key resource and other, possibly more important, processes cannot use it until it is released. Giving the process more CPU time may free up the resource quicker.
7. Avoid indefinite postponement – every process should get a chance to run eventually.

Description of the application to be implemented:

Round Robin Algorithm

To schedule processes fairly, a round-robin scheduler generally employs time-sharing, giving each job a time slot or quantum (its allowance of CPU time), and interrupting the job if it is not completed by then. The job is resumed next time a time slot is assigned to that process. If the process terminates or changes its state to waiting during its attributed time quantum, the scheduler selects the first process in the ready queue to execute. In the absence of time-sharing, or if the quanta were large relative to the sizes of the jobs, a process that produced large jobs would be favored over other processes. Round-robin algorithm is a pre-emptive algorithm as the scheduler forces the process out of the CPU once the time quota expires. Round Robin is the preemptive process scheduling algorithm. Each process is provided a fix time to execute, it is called a quantum. Once a process is executed for a given time period, it is preempted and other process executes for a given time period. Context switching is used to save states of preempted processes.

Characteristics:

- Round robin is a pre-emptive algorithm

- The CPU is shifted to the next process after fixed interval time, which is called time quantum/time slice.
- The process that is preempted is added to the end of the queue.
- Round robin is a hybrid model which is clock-driven
- Time slice should be minimum, which is assigned for a specific task that needs to be processed. However, it may differ OS to OS.

Advantages:

- There is fairness since every process gets equal share of CPU.
- A round-robin scheduler generally employs time-sharing, giving each job a time slot or quantum.
- Each process get a chance to reschedule after a particular quantum time in this scheduling.

Disadvantages:

- Gantt chart seems to come too big (if quantum time is less for scheduling. For Example:1 ms for big scheduling.)
- Time consuming scheduling for small quantams.
- There is Low throughput.

Implementation details: (printout of code)

CODE:

```
def sort(process):
    for i in process:
        for j in range(0,len(process)-1):
            if(int(process[j][1])>int(process[j+1][1])):
                temp=process[j]
                process[j]=process[j+1]
                process[j+1]=temp

def checkarrival(process,processtaken,time):
    temp=list()
    for i in range(0,len(process)):
```

```
        if(int(process[i][1])==time and process[i][0] not in processtaken
):
            temp.append(process[i])
    return temp

print("------------------------------------------------------------------
------------------------------------------------------------------------
---------------")
print("------------------------------------------------------------------
--------RR--------------------------------------------------------------
---------------")
print("------------------------------------------------------------------
------------------------------------------------------------------------
---------------")
n=int(input("Enter number of processes : "))
print("------------------------------------------------------------------
------------------------------------------------------------------------
---------------")
process=list()
for i in range(0,n):
    inp=input("Enter PID, Arrival Time, Burst Time : ").split()
    process.append(inp)
print("------------------------------------------------------------------
------------------------------------------------------------------------
---------------")
qm=int(input("Enter time quantam : "))
totaltime=0
sort(process)
dic=dict()
dic2=dict()
for i in process:
    dic[i[0]]=i[1]
    dic2[i[0]]=i[2]
print("------------------------------------------------------------------
------------------------------------------------------------------------
---------------")
k=1
totaltime=totaltime+int(process[0][1])+int(process[0][2])
flag=0
while(flag!=1):
    if(totaltime>=int(process[k][1])):
```

```
        temptime=0
        for i in range(k,n):
            if(int(process[i][1])<=totaltime):
                k=k+1
                temptime=temptime+int(process[i][2])
                if(k==n):
                    flag=1
        totaltime=totaltime+temptime
    else:
        totaltime=totaltime+1
processtaken=list()
readyqueue=list()
processing=list()
waiting=0
turnaround=0
waitinglist=list()
turnaroundlist=list()
waitinglist.append(int(process[0][1]))
turnaroundlist.append(int(process[0][2]))
duration=list()
finalprocesssequence=list()
print("Ready queue instances : ")
tempqm=qm
another=0
for i in range(int(process[0][1]),totaltime+1):
    print("Time : ",i,end=" ")
    pro=checkarrival(process,processtaken,i)
    for j in pro:
        if(j not in processtaken):
            readyqueue.append(j)
            processtaken.append(j)
    if(another==1):
        print("Ready queue : ",end=" ")
        for i in range(1,len(readyqueue)):
            print("[",readyqueue[i],"]",end=" ")
    else:
        print("Ready queue : ",readyqueue,end=" ")
    if(len(processing)==0):
        if(len(readyqueue)>0):
            processing.append(readyqueue[0])
            qm=tempqm
```

```
            if(another!=1):
                print("Processing queue : ",processing,end=" ")
            finalprocesssequence.append(readyqueue[0])
            readyqueue.pop(0)
            if(another==1):
                processing[0][2]=int(processing[0][2])-1
                qm=qm-1
                another=0
                print("Processing queue : ",processing,end=" ")
                duration.append(processing[0][0])
                if(int(processing[0][2])==0):
                    processing.pop()
                    if(len(readyqueue)>0):
                        processing.append(readyqueue[0])
                        qm=tempqm
                        print("Processing queue : ",processing,end=" ")
                        finalprocesssequence.append(readyqueue[0])
                        readyqueue.pop(0)
    else:
        processing[0][2]=int(processing[0][2])-1
        qm=qm-1
        print("Processing queue : ",processing,end=" ")
        duration.append(processing[0][0])
        if(int(processing[0][2])==0):
            processing.pop()
            if(len(readyqueue)>0):
                processing.append(readyqueue[0])
                qm=tempqm
                print("Processing queue : ",processing,end=" ")
                finalprocesssequence.append(readyqueue[0])
                readyqueue.pop(0)
    if(len(pro)==0 and len(processtaken)!=len(process) and len(processing
)==0):
        duration.append("X")
    print()
    if(qm==0):
        if(len(processing)>0):
            if(processing[0][2]!=0):
                readyqueue.append(processing[0])
                processing.pop(0)
                another=1
```

```
print("------------------------------------------------------------------------
------------------------------------------------------------------------
---------------")
temp=list()
t=0
temp2=list()
for i in range(0,len(duration)-1):
    t=t+1
    flag=0
    if(duration[i]!=duration[i+1]):
        temp2=[duration[i],t]
        temp.append(temp2)
        flag=1
    if(i==len(duration)-2):
        if(flag==1):
            temp2=[duration[i+1],t+1]
            temp.append(temp2)
        else:
            temp2=[duration[i],t+1]
            temp.append(temp2)
if(int(process[0][1])>0):
    for i in temp:
        i[1]=i[1]+int(process[0][1])
print("Gantt Chart  : ")
start=int(process[0][1])
forcalculation=list()
for i in temp:
    lst=list()
    lst=[None]*3
    if(i[0]!="X"):
        print("(",start,") P",i[0],"(",i[1],")",end=" ")
        lst[0]=start
        lst[1]=i[0]
        lst[2]=i[1]
        forcalculation.append(lst)
        for j in range(start,i[1]):
            print("  ",end=" ")
    else:
        print("(",start,") Idle (",i[1],")",end=" ")
        for j in range(start,i[1]):
            print("  ",end=" ")
```

```
    start=i[1]
print("\n-------------------------------------------------------------------
-------------------------------------------------------------------------
-----------------")
finalwar=list()
tempwar=list()
finalcalculation2=list()
for i in range(len(forcalculation)-1,-1,-1):
    for j in process:
        if(j[0]==forcalculation[i][1] and j[0] not in tempwar):
            finalwar.append(forcalculation[i])
            tempwar.append(j[0])
for i in finalwar:
    finalcalculation2.append(i)
print("Waiting time of processes : ")
for i in range(0,len(finalcalculation2)):
    print("P",finalcalculation2[i][1]," : ",finalcalculation2[i][2]-
int(dic2[finalcalculation2[i][1]])-int(dic[finalcalculation2[i][1]]))
    waiting=waiting+finalcalculation2[i][2]-
int(dic2[finalcalculation2[i][1]])-int(dic[finalcalculation2[i][1]])
print("-------------------------------------------------------------------
-------------------------------------------------------------------------
---------------")
print("Average waiting time  : ",waiting/len(process))
print("-------------------------------------------------------------------
-------------------------------------------------------------------------
---------------")
print("Turnaround time of processes : ")
for i in range(0,len(finalcalculation2)):
    print("P",finalcalculation2[i][1]," : ",finalcalculation2[i][2]-
int(dic[finalcalculation2[i][1]]))
    turnaround=turnaround+finalcalculation2[i][2]-
int(dic[finalcalculation2[i][1]])
print("-------------------------------------------------------------------
-------------------------------------------------------------------------
---------------")
print("Average turnaround time : ",turnaround/len(process))
print("-------------------------------------------------------------------
-------------------------------------------------------------------------
---------------")
```

OUTPUT:

```
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------RR----------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
Enter number of processes : 4
----------------------------------------------------------------------------------------------------------------------------------------------------------------
Enter PID, Arrival Time, Burst Time : 1 0 5
Enter PID, Arrival Time, Burst Time : 2 1 4
Enter PID, Arrival Time, Burst Time : 3 2 2
Enter PID, Arrival Time, Burst Time : 4 3 1
----------------------------------------------------------------------------------------------------------------------------------------------------------------
Enter time quantam : 2
----------------------------------------------------------------------------------------------------------------------------------------------------------------
Ready queue instances :
Time :  0 Ready queue :  [['1', '0', '5']] Processing queue :  [['1', '0', '5']]
Time :  1 Ready queue :  [['2', '1', '4']] Processing queue :  [['1', '0', 4]]
Time :  2 Ready queue :  [['2', '1', '4'], ['3', '2', '2']] Processing queue :  [['1', '0', 3]]
Time :  3 Ready queue :  [ ['3', '2', '2'] ] [ ['1', '0', 3] ] [ ['4', '3', '1'] ] Processing queue :  [['2', '1', 3]]
Time :  4 Ready queue :  [['3', '2', '2'], ['1', '0', 3], ['4', '3', '1']] Processing queue :  [['2', '1', 2]]
Time :  5 Ready queue :  [ ['1', '0', 3] ] [ ['4', '3', '1'] ] [ ['2', '1', 2] ] Processing queue :  [['3', '2', 1]]
Time :  6 Ready queue :  [['1', '0', 3], ['4', '3', '1'], ['2', '1', 2]] Processing queue :  [['3', '2', 0]] Processing queue :  [['1', '0', 3]]
Time :  7 Ready queue :  [['4', '3', '1'], ['2', '1', 2]] Processing queue :  [['1', '0', 2]]
Time :  8 Ready queue :  [['4', '3', '1'], ['2', '1', 2]] Processing queue :  [['1', '0', 1]]
Time :  9 Ready queue :  [ ['2', '1', 2] ] [ ['1', '0', 1] ] Processing queue :  [['4', '3', 0]] Processing queue :  [['2', '1', 2]]
Time :  10 Ready queue :  [['1', '0', 1]] Processing queue :  [['2', '1', 1]]
Time :  11 Ready queue :  [['1', '0', 1]] Processing queue :  [['2', '1', 0]] Processing queue :  [['1', '0', 1]]
Time :  12 Ready queue :  [] Processing queue :  [['1', '0', 0]]
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
Gantt Chart  :
( 0 ) P 1 ( 2 )     ( 2 ) P 2 ( 4 )     ( 4 ) P 3 ( 6 )     ( 6 ) P 1 ( 8 )     ( 8 ) P 4 ( 9 )     ( 9 ) P 2 ( 11 )     ( 11 ) P 1 ( 12 )
----------------------------------------------------------------------------------------------------------------------------------------------------------------
Waiting time of processes :
P 1  :  7
P 2  :  6
P 4  :  5
P 3  :  2
----------------------------------------------------------------------------------------------------------------------------------------------------------------
Average waiting time  :  5.0
----------------------------------------------------------------------------------------------------------------------------------------------------------------
Turnaround time of processes :
P 1  :  12
P 2  :  10
P 4  :  6
P 3  :  4
----------------------------------------------------------------------------------------------------------------------------------------------------------------
Average turnaround time :  8.0
----------------------------------------------------------------------------------------------------------------------------------------------------------------
```

Shortest Remaining Time First Algorithm :

Shortest remaining time, also known as shortest remaining time first (SRTF), is a scheduling method that is a preemptive version of shortest job next scheduling. In this scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. Since the currently executing process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses, the process will either run until it completes or get preempted if a new process is added that requires a smaller amount of time. Shortest remaining time is advantageous because short processes are handled very quickly. The system also requires very little overhead since it only makes a decision when a process completes or a new process is added, and when a new process is added the algorithm only needs to compare the currently executing process with the new process, ignoring all other processes currently waiting to execute.

Characteristics:

- This Algorithm is the preemptive version of SJF scheduling.
- In SRTF, the execution of the process can be stopped after certain amount of time.
- Scheduler schedules the process with the least remaining burst time among the list of available processes and the running process.

Advantages:

- In SRTF, process with the smallest established burst time completes first, including new arrivals.
- In SRTF, running process may be preempted with the arrival of new process with shortest estimated time.
- Superior turnaround time than SJF.

Disadvantages:

- SRTF has higher overload than SJF.
- SRTF must keep track of the elapsed time of the running process and must handle preemptions.

Implementation details: (printout of code)

CODE:

```
def sort(process):
    for i in process:
        for j in range(0,len(process)-1):
            if(int(process[j][1])>int(process[j+1][1])):
                temp=process[j]
                process[j]=process[j+1]
                process[j+1]=temp

def checkarrival(process, time):
    temp=list()
    for i in range(0,len(process)):
        if(int(process[i][1])==time and process[i][0]):
            temp.append(process[i])
    return temp

def sjf(readyqueue,processing):
    if(len(processing)!=0):
        if(processing[0][2]!=0):
            readyqueue.insert(0,processing[0])
    for i in readyqueue:
        for j in range(0,len(readyqueue)-1):
            if(int(readyqueue[j][2])>int(readyqueue[j+1][2])):
                temp=readyqueue[j]
                readyqueue[j]=readyqueue[j+1]
                readyqueue[j+1]=temp
    return readyqueue

print("------------------------------------------------------------------------
------------------------------------------------------------------------
---------------")
print("------------------------------------------------------------------------
------SRTF--------------------------------------------------------------------
---------------")
print("------------------------------------------------------------------------
------------------------------------------------------------------------
---------------")
```

```
n=int(input("Enter number of processes : "))
print("-------------------------------------------------------------------
-------------------------------------------------------------------
---------------")
process=list()
for i in range(0,n):
    inp=input("Enter PID, Arrival Time, Burst Time : ").split()
    process.append(inp)
totaltime=0
sort(process)
dic=dict()
dic2=dict()
for i in process:
    dic[i[0]]=i[1]
    dic2[i[0]]=i[2]
print("-------------------------------------------------------------------
-------------------------------------------------------------------
---------------")
k=1
totaltime=totaltime+int(process[0][1])+int(process[0][2])
flag=0
while(flag!=1):
    if(totaltime>=int(process[k][1])):
        temptime=0
        for i in range(k,n):
            if(int(process[i][1])<=totaltime):
                k=k+1
                temptime=temptime+int(process[i][2])
                if(k==n):
                    flag=1
        totaltime=totaltime+temptime
    else:
        totaltime=totaltime+1
processtaken=list()
readyqueue=list()
processing=list()
waiting=0
turnaround=0
waitinglist=list()
turnaroundlist=list()
waitinglist.append(int(process[0][1]))
```

```
turnaroundlist.append(int(process[0][2]))
duration=list()
finalprocesssequence=list()
printing=list()
another=0
print("Ready queue instances : ")
for i in range(int(process[0][1]),totaltime+1):
    flagtemp=0
    another=0
    print("Time : ",i,end=" ")
    pro=checkarrival(process,i)
    for j in pro:
        readyqueue.append(j)
        processtaken.append(j)
    if(len(readyqueue)>1 or (len(readyqueue)>=1 and len(processing)!=0) o
r len(readyqueue)==1):
        readyqueue=sjf(readyqueue,processing)
        if(len(processing)!=0):
            processing.pop()
        processing.append(readyqueue[0])
        finalprocesssequence.append(readyqueue[0])
        readyqueue.pop(0)
        another=1
        print("Ready queue : ",end=" ")
        if(processing[0][2]==dic2[processing[0][0]]):
            print(processing[0],end=" ")
        for i in readyqueue:
            print(i,end=" ")
    if(another==0):
        print("Ready queue : []",end=" ")
        another=0
    print("Processing queue : ",*processing,end=" ")
    if(len(printing)>0):
        print(printing[0],end=" ")
        printing.pop()
    if(len(processing)>0):
        processing[0][2]=int(processing[0][2])-1
        duration.append(processing[0][0])
        if(int(processing[0][2])==0):
            printing.append(processing[0])
            processing.pop()
```

```
            flagtemp=1
            if(len(readyqueue)>0):
                processing.append(readyqueue[0])
                finalprocesssequence.append(readyqueue[0])
                readyqueue.pop(0)
    if(len(readyqueue)==0 and len(processing)==0 and len(processtaken)!=n
 and flagtemp!=1):
        duration.append("X")
    print()
print("----------------------------------------------------------------------
----------------------------------------------------------------------
---------------")
temp=list()
t=0
temp2=list()
for i in range(0,len(duration)-1):
    t=t+1
    flag=0
    if(duration[i]!=duration[i+1]):
        temp2=[duration[i],t]
        temp.append(temp2)
        flag=1
    if(i==len(duration)-2):
        if(flag==1):
            temp2=[duration[i+1],t+1]
            temp.append(temp2)
        else:
            temp2=[duration[i],t+1]
            temp.append(temp2)
if(int(process[0][1])>0):
    for i in temp:
        i[1]=i[1]+int(process[0][1])
print("Gantt Chart  : ")
start=int(process[0][1])
forcalculation=list()
for i in temp:
    lst=list()
    lst=[None]*3
    if(i[0]!="X"):
        print("(",start,") P",i[0],"(",i[1],")",end=" ")
        lst[0]=start
```

```
        lst[1]=i[0]
        lst[2]=i[1]
        forcalculation.append(lst)
        for j in range(start,i[1]):
            print("  ",end=" ")
    else:
        print("(",start,") Idle (",i[1],")",end=" ")
        for j in range(start,i[1]):
            print("  ",end=" ")
    start=i[1]
print("\n----------------------------------------------------------------
----------------------------------------------------------------------
-----------------")
finalwar=list()
tempwar=list()
finalcalculation2=list()
for i in range(len(forcalculation)-1,-1,-1):
    for j in process:
        if(j[0]==forcalculation[i][1] and j[0] not in tempwar):
            finalwar.append(forcalculation[i])
            tempwar.append(j[0])
for i in finalwar:
    finalcalculation2.append(i)
print("Waiting time of processes : ")
for i in range(0,len(finalcalculation2)):
    print("P",finalcalculation2[i][1]," : ",finalcalculation2[i][2]-
int(dic2[finalcalculation2[i][1]])-int(dic[finalcalculation2[i][1]]))
    waiting=waiting+finalcalculation2[i][2]-
int(dic2[finalcalculation2[i][1]])-int(dic[finalcalculation2[i][1]])
print("----------------------------------------------------------------
----------------------------------------------------------------------
---------------")
print("Average waiting time  : ",waiting/len(process))
print("----------------------------------------------------------------
----------------------------------------------------------------------
---------------")
print("Turnaround time of processes : ")
for i in range(0,len(finalcalculation2)):
    print("P",finalcalculation2[i][1]," : ",finalcalculation2[i][2]-
int(dic[finalcalculation2[i][1]]))
```

```
    turnaround=turnaround+finalcalculation2[i][2]-
int(dic[finalcalculation2[i][1]])
print("--------------------------------------------------------------------------
--------------------------------------------------------------------------
---------------")
print("Average turnaround time : ",turnaround/len(process))
print("--------------------------------------------------------------------------
--------------------------------------------------------------------------
---------------")
```

OUTPUT:

```
--------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------SRTF----------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
Enter number of processes : 3
--------------------------------------------------------------------------------------------------------------------------------------------
Enter PID, Arrival Time, Burst Time : 1 0 8
Enter PID, Arrival Time, Burst Time : 2 1 2
Enter PID, Arrival Time, Burst Time : 3 4 3
--------------------------------------------------------------------------------------------------------------------------------------------
Ready queue instances :
Time :  0 Ready queue :  ['1', '0', '8'] Processing queue :  ['1', '0', '8']
Time :  1 Ready queue :  ['2', '1', '2'] ['1', '0', 7] Processing queue :  ['2', '1', '2']
Time :  2 Ready queue :  ['1', '0', 7] Processing queue :  ['2', '1', 1]
Time :  3 Ready queue : [] Processing queue :  ['1', '0', 7] ['2', '1', 0]
Time :  4 Ready queue :  ['3', '4', '3'] ['1', '0', 6] Processing queue :  ['3', '4', '3']
Time :  5 Ready queue :  ['1', '0', 6] Processing queue :  ['3', '4', 2]
Time :  6 Ready queue :  ['1', '0', 6] Processing queue :  ['3', '4', 1]
Time :  7 Ready queue : [] Processing queue :  ['1', '0', 6] ['3', '4', 0]
Time :  8 Ready queue : [] Processing queue :  ['1', '0', 5]
Time :  9 Ready queue : [] Processing queue :  ['1', '0', 4]
Time :  10 Ready queue : [] Processing queue :  ['1', '0', 3]
Time :  11 Ready queue : [] Processing queue :  ['1', '0', 2]
Time :  12 Ready queue : [] Processing queue :  ['1', '0', 1]
Time :  13 Ready queue : [] Processing queue :  ['1', '0', 0]
--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
Gantt Chart  :
(0)P1(1)  (1)P2(3)    (3)P1(4)  (4)P3(7)      (7)P1(13)
--------------------------------------------------------------------------------------------------------------------------------------------
Waiting time of processes :
P1  :  5
P3  :  0
P2  :  0
--------------------------------------------------------------------------------------------------------------------------------------------
Average waiting time  :  1.6666666666666667
--------------------------------------------------------------------------------------------------------------------------------------------
Turnaround time of processes :
P1  :  13
P3  :  3
P2  :  2
--------------------------------------------------------------------------------------------------------------------------------------------
Average turnaround time :  6.0
--------------------------------------------------------------------------------------------------------------------------------------------
```

Priority scheduling:

In Preemptive Priority Scheduling, at the time of arrival of a process in the ready queue, its Priority is compared with the priority of the other processes present in the ready queue as well as with the one which is being executed by the CPU at that point of time. The One with the highest priority among all the available processes will be given the CPU next. The difference between preemptive priority scheduling and non preemptive priority scheduling is that, in the preemptive priority scheduling, the job which is being executed can be stopped at the arrival of a higher priority job. Once all the jobs get available in the ready queue, the algorithm will behave as non-preemptive priority scheduling, which means the job scheduled will run till the completion and no preemption will be done.

Characteristics:

- A CPU algorithm that schedules processes based on priority.
- It used in Operating systems for performing batch processes.
- If two jobs having the same priority are READY, it works on a FIRST COME, FIRST SERVED basis.
- In priority scheduling, a number is assigned to each process that indicates its priority level.

Advantages:

- Static priorities work well for real time systems.
- Dynamic priorities work well for general workload.

Disadvantages:

- Low priority jobs can starve.
- It will be a complex issue to find priority of each process.

Implementation details: (printout of code)

CODE:

```
def sort(process):
    for i in process:
        for j in range(0,len(process)-1):
```

```
            if(int(process[j][1])>int(process[j+1][1])):
                temp=process[j]
                process[j]=process[j+1]
                process[j+1]=temp

def checkarrival(process, time):
    temp=list()
    for i in range(0,len(process)):
        if(int(process[i][1])==time and process[i][0]):
            temp.append(process[i])
    return temp

def sjf(readyqueue,processing):
    if(len(processing)!=0):
        if(processing[0][2]!=0):
            readyqueue.insert(0,processing[0])
    for i in readyqueue:
        for j in range(0,len(readyqueue)-1):
            if(int(readyqueue[j][3])>int(readyqueue[j+1][3])):
                temp=readyqueue[j]
                readyqueue[j]=readyqueue[j+1]
                readyqueue[j+1]=temp
    return readyqueue

print("------------------------------------------------------------------------
------------------------------------------------------------------------
--------------")
print("------------------------------------------------------------------------
----PRIORITY-----------------------------------------------------------------
--------------")
print("------------------------------------------------------------------------
------------------------------------------------------------------------
--------------")
n=int(input("Enter number of processes : "))
print("------------------------------------------------------------------------
------------------------------------------------------------------------
--------------")
process=list()
for i in range(0,n):
    inp=input("Enter PID, Arrival Time, Burst Time, Priority : ").split()
    process.append(inp)
```

```
totaltime=0
sort(process)
dic=dict()
dic2=dict()
for i in process:
    dic[i[0]]=i[1]
    dic2[i[0]]=i[2]
print("--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
---------------")
k=1
totaltime=totaltime+int(process[0][1])+int(process[0][2])
flag=0
while(flag!=1):
    if(totaltime>=int(process[k][1])):
        temptime=0
        for i in range(k,n):
            if(int(process[i][1])<=totaltime):
                k=k+1
                temptime=temptime+int(process[i][2])
                if(k==n):
                    flag=1
        totaltime=totaltime+temptime
    else:
        totaltime=totaltime+1
processtaken=list()
readyqueue=list()
processing=list()
waiting=0
turnaround=0
waitinglist=list()
turnaroundlist=list()
waitinglist.append(int(process[0][1]))
turnaroundlist.append(int(process[0][2]))
duration=list()
finalprocesssequence=list()
printing=list()
another=0
print("Ready queue instances : ")
for i in range(int(process[0][1]),totaltime+1):
    flagtemp=0
```

```
    another=0
    print("Time : ",i,end=" ")
    pro=checkarrival(process,i)
    for j in pro:
        readyqueue.append(j)
        processtaken.append(j)
    if(len(readyqueue)>1 or (len(readyqueue)>=1 and len(processing)!=0) o
r len(readyqueue)==1):
        readyqueue=sjf(readyqueue,processing)
        if(len(processing)!=0):
            processing.pop()
        processing.append(readyqueue[0])
        finalprocesssequence.append(readyqueue[0])
        readyqueue.pop(0)
        another=1
        print("Ready queue : ",end=" ")
        if(processing[0][2]==dic2[processing[0][0]]):
            print(processing[0],end=" ")
        for i in readyqueue:
            print(i,end=" ")
    if(another==0):
        print("Ready queue : []",end=" ")
        another=0
    print("Processing queue : ",*processing,end=" ")
    if(len(printing)>0):
        print(printing[0],end=" ")
        printing.pop()
    if(len(processing)>0):
        processing[0][2]=int(processing[0][2])-1
        duration.append(processing[0][0])
        if(int(processing[0][2])==0):
            printing.append(processing[0])
            processing.pop()
            flagtemp=1
            if(len(readyqueue)>0):
                processing.append(readyqueue[0])
                finalprocesssequence.append(readyqueue[0])
                readyqueue.pop(0)
    if(len(readyqueue)==0 and len(processing)==0 and len(processtaken)!=n
 and flagtemp!=1):
        duration.append("X")
```

```
    print()
print("-------------------------------------------------------------------------
-------------------------------------------------------------------------
---------------")
temp=list()
t=0
temp2=list()
for i in range(0,len(duration)-1):
    t=t+1
    flag=0
    if(duration[i]!=duration[i+1]):
        temp2=[duration[i],t]
        temp.append(temp2)
        flag=1
    if(i==len(duration)-2):
        if(flag==1):
            temp2=[duration[i+1],t+1]
            temp.append(temp2)
        else:
            temp2=[duration[i],t+1]
            temp.append(temp2)
if(int(process[0][1])>0):
    for i in temp:
        i[1]=i[1]+int(process[0][1])
print("Gantt Chart  : ")
start=int(process[0][1])
forcalculation=list()
for i in temp:
    lst=list()
    lst=[None]*3
    if(i[0]!="X"):
        print("(",start,") P",i[0],"(",i[1],")",end=" ")
        lst[0]=start
        lst[1]=i[0]
        lst[2]=i[1]
        forcalculation.append(lst)
        for j in range(start,i[1]):
            print("  ",end=" ")
    else:
        print("(",start,") Idle (",i[1],")",end=" ")
        for j in range(start,i[1]):
```

```
            print("   ",end=" ")
    start=i[1]
print("\n--------------------------------------------------------------------
------------------------------------------------------------------------
-----------------")
finalwar=list()
tempwar=list()
finalcalculation2=list()
for i in range(len(forcalculation)-1,-1,-1):
    for j in process:
        if(j[0]==forcalculation[i][1] and j[0] not in tempwar):
            finalwar.append(forcalculation[i])
            tempwar.append(j[0])
for i in finalwar:
    finalcalculation2.append(i)
print("Waiting time of processes : ")
for i in range(0,len(finalcalculation2)):
    print("P",finalcalculation2[i][1]," : ",finalcalculation2[i][2]-
int(dic2[finalcalculation2[i][1]])-int(dic[finalcalculation2[i][1]]))
    waiting=waiting+finalcalculation2[i][2]-
int(dic2[finalcalculation2[i][1]])-int(dic[finalcalculation2[i][1]])
print("--------------------------------------------------------------------
------------------------------------------------------------------------
---------------")
print("Average waiting time  : ",waiting/len(process))
print("--------------------------------------------------------------------
------------------------------------------------------------------------
---------------")
print("Turnaround time of processes : ")
for i in range(0,len(finalcalculation2)):
    print("P",finalcalculation2[i][1]," : ",finalcalculation2[i][2]-
int(dic[finalcalculation2[i][1]]))
    turnaround=turnaround+finalcalculation2[i][2]-
int(dic[finalcalculation2[i][1]])
print("--------------------------------------------------------------------
------------------------------------------------------------------------
---------------")
print("Average turnaround time : ",turnaround/len(process))
print("--------------------------------------------------------------------
------------------------------------------------------------------------
---------------")
```

OUTPUT:

```
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------PRIORITY----------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
Enter number of processes : 5
----------------------------------------------------------------------------------------------------------------------------------------------------------------
Enter PID, Arrival Time, Burst Time, Priority : 1 0 8 3
Enter PID, Arrival Time, Burst Time, Priority : 2 1 1 1
Enter PID, Arrival Time, Burst Time, Priority : 3 2 3 2
Enter PID, Arrival Time, Burst Time, Priority : 4 3 2 3
Enter PID, Arrival Time, Burst Time, Priority : 5 4 6 4
----------------------------------------------------------------------------------------------------------------------------------------------------------------
Ready queue instances :
Time :  0 Ready queue :  ['1', '0', '8', '3'] Processing queue :  ['1', '0', '8', '3']
Time :  1 Ready queue :  ['2', '1', '1', '1'] ['1', '0', 7, '3'] Processing queue :  ['2', '1', '1', '1']
Time :  2 Ready queue :  ['3', '2', '3', '2'] ['1', '0', 7, '3'] Processing queue :  ['3', '2', '3', '2'] ['2', '1', 0, '1']
Time :  3 Ready queue :  ['1', '0', 7, '3'] ['4', '3', '2', '3'] Processing queue :  ['3', '2', 2, '2']
Time :  4 Ready queue :  ['1', '0', 7, '3'] ['4', '3', '2', '3'] ['5', '4', '6', '4'] Processing queue :  ['3', '2', 1, '2']
Time :  5 Ready queue :  ['4', '3', '2', '3'] ['5', '4', '6', '4'] Processing queue :  ['1', '0', 7, '3'] ['3', '2', 0, '2']
Time :  6 Ready queue :  ['4', '3', '2', '3'] ['5', '4', '6', '4'] Processing queue :  ['1', '0', 6, '3']
Time :  7 Ready queue :  ['4', '3', '2', '3'] ['5', '4', '6', '4'] Processing queue :  ['1', '0', 5, '3']
Time :  8 Ready queue :  ['4', '3', '2', '3'] ['5', '4', '6', '4'] Processing queue :  ['1', '0', 4, '3']
Time :  9 Ready queue :  ['4', '3', '2', '3'] ['5', '4', '6', '4'] Processing queue :  ['1', '0', 3, '3']
Time :  10 Ready queue :  ['4', '3', '2', '3'] ['5', '4', '6', '4'] Processing queue :  ['1', '0', 2, '3']
Time :  11 Ready queue :  ['4', '3', '2', '3'] ['5', '4', '6', '4'] Processing queue :  ['1', '0', 1, '3']
Time :  12 Ready queue :  ['4', '3', '2', '3'] ['5', '4', '6', '4'] Processing queue :  ['4', '3', '2', '3'] ['1', '0', 0, '3']
Time :  13 Ready queue :  ['5', '4', '6', '4'] Processing queue :  ['4', '3', 1, '3']
Time :  14 Ready queue :  [] Processing queue :  ['5', '4', '6', '4'] ['4', '3', 0, '3']
Time :  15 Ready queue :  [] Processing queue :  ['5', '4', 5, '4']
Time :  16 Ready queue :  [] Processing queue :  ['5', '4', 4, '4']
Time :  17 Ready queue :  [] Processing queue :  ['5', '4', 3, '4']
Time :  18 Ready queue :  [] Processing queue :  ['5', '4', 2, '4']
Time :  19 Ready queue :  [] Processing queue :  ['5', '4', 1, '4']
Time :  20 Ready queue :  [] Processing queue :  ['5', '4', 0, '4']
----------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
---------------------------------------------------------------------------------------------------------------------------
Gantt Chart :
( 0 ) P 1 ( 1 )   ( 1 ) P 2 ( 2 )   ( 2 ) P 3 ( 5 )       ( 5 ) P 1 ( 12 )              ( 12 ) P 4 ( 14 )     ( 14 ) P 5 ( 20 )
---------------------------------------------------------------------------------------------------------------------------
Waiting time of processes :
P 5  :  10
P 4  :  9
P 1  :  4
P 3  :  0
P 2  :  0
---------------------------------------------------------------------------------------------------------------------------
Average waiting time  :  4.6
---------------------------------------------------------------------------------------------------------------------------
Turnaround time of processes :
P 5  :  16
P 4  :  11
P 1  :  12
P 3  :  3
P 2  :  1
---------------------------------------------------------------------------------------------------------------------------
Average turnaround time :  8.6
---------------------------------------------------------------------------------------------------------------------------
```

Conclusion: Understood the concept of basic process management algorithms in operating system. Understood the theory behind round robin algorithm, shortest remaining time first algorithm and priority scheduling algorithm. Also implemented the same using python programming language.

Post Lab Descriptive Questions

1. Consider three processes, all arriving at time zero, with total execution time of 10, 20 and 30 units, respectively. Each process spends the first 20% of execution time doing I/O, the next 70% of time doing computation, and the last 10% of time doing I/O again. The operating system uses a shortest remaining compute time first scheduling algorithm and schedules a new process either when the running process gets blocked on I/O or when the running process finishes its compute burst. Assume that all I/O operations can be overlapped as much as possible. For what percentage of time does the CPU remain idle?

ANS) Shortest remaining time (SRT) scheduling algorithm selects the process for execution which has the smallest amount of time remaining until completion. Let three processes be p0, p1 and p2. Their execution time is 10, 20 and 30 respectively.
p0 spends first 2 time units in I/O, 7 units of CPU time and finally 1 unit in I/O.
p1 spends first 4 units in I/O, 14 units of CPU time and finally 2 units in I/O.
p2 spends first 6 units in I/O, 21 units of CPU time and finally 3 units in I/O.

PID	AT	IO	BT	IO
P0	0	2	7	1
P1	0	4	14	2
P2	0	6	21	3

AT- Arrival Time, IO-input/output, BT-Burst Time, first process p0 will spend 2 units in IO, next 7 units in BT, then process p1 will spend 14 units in BT (as its 4 units of IO has been spent already when previous process was running) and ten process p2 will spend 21 units in BT (as its 6 units of IO has been spent already when previous processes were running) and atlast 3 units in IO (process p0,p1,p2's last IO included.)

```
idle   p0    p1      p2     idle
0    2     9      23      44      47

Total time spent = 47
Idle time = 2 + 3 = 5
Percentage of idle time = (5/47)*100 = 10.6 %
```

2. What is Starvation?

ANS) Starvation is the name given to the indefinite postponement of a process because it requires some resource before it can run, but the resource, though available for allocation, is never allocated to this process. It is sometimes called livelock, though sometimes that name might be reserved for cases where the waiting process is doing something, but nothing useful, as in a spin lock. However it happens, starvation is self-evidently a bad thing; more formally, it's bad because we don't want a non-functional system. If starvation is possible in a system, then a process which enters the system can be held up for an arbitrary length of time. To avoid starvation, it is often said that we want the system's resources to be shared "fairly". This is an appealing suggestion, but in practice it isn't much use because it doesn't define what we mean by "fairly", so it's really more of a description of the problem than a contribution to its solution. It's more constructive to investigate the paths by which a system can reach a livelocked state, and try to control them.

CAUSES OF STARVATION:

- Processes hand on resources to other processes without control. If decisions about resource allocation are taken locally without considering the overall resource requirements of the system, anomalies can occur. If processes queue for a resource, and the resource is always handed on to the next process in the queue, it is essential that every process awaiting the resource must be placed in the queue.
- Processes priorities are strictly enforced. If a process of worse priority requires a resource in competition with a constant stream of processes of better priority, it might wait for ever.

Solution: Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time.For example, if priority range from 127(low) to 0(high), we could increase the priority of a waiting process by 1 Every 15 minutes. Eventually even a process with an initial priority of 127 would take no more than 32 hours for priority 127 process to age to a priority-0 process.

Date: 30 / 9 / 2021 **Signature of faculty in-charge**