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

Batch:	**Roll No.:**
Experiment No. 03	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

TITLE: Implementation of Basic Process management algorithms – Non Pre-emptive (FCFS , SJF, priority)

AIM: To implement basic Non –Pre-emptive Process management algorithms (FCFS , SJF , 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:

First-Come, First-Served Scheduling:

Simplest CPU scheduling algorithm that schedules according to arrival times of processes. First come first serve scheduling algorithm states that the process that requests the CPU first is allocated the CPU first. It is implemented by using the FIFO queue. When a process enters the ready queue, its PCB is linked to the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue. FCFS is a non-preemptive scheduling algorithm.

Characteristics of FCFS:

1) FCFS supports non-preemptive and preemptive CPU scheduling algorithms.
2) Tasks are always executed on a First-come, First-serve concept.
3) FCFS is easy to implement and use.
4) This algorithm is not much efficient in performance, and the wait time is quite high.

Shortest job first :

Shortest Job first has the advantage of having a minimum average waiting time among all scheduling algorithms. It is a Greedy Algorithm. It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of ageing. It is practically infeasible as Operating System may not know burst time and therefore may not sort them. While it is not possible to predict execution time, several methods can be used to estimate the execution time for a job, such as a weighted average of previous execution times. SJF can be used in specialized environments where accurate estimates of running time are available.

Priority scheduling

In the Non Preemptive Priority scheduling, The Processes are scheduled according to the priority number assigned to them. Once the process gets scheduled, it will run till the completion. Generally, the lower the priority number, the higher is the priority of the process.

Implementation details: (printout of code)

FCFS

```
k=int(input("Please enter number of processes "))


arrivalTime=[int(input("Please enter arrival time for
process "+str(i)+" ")) for i in range(k)]


reqTime=[int(input("Please enter estimated time for
process "+str(i)+" ")) for i in range(k)]



timeQuantaDecided =20000


'''

#Test case
```

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

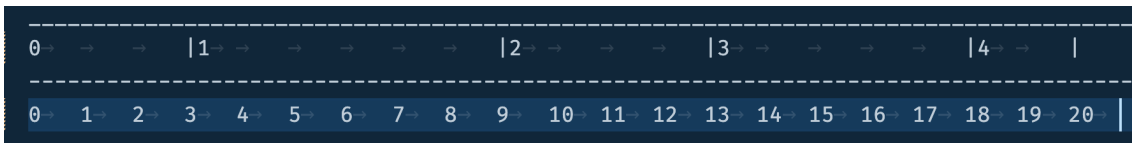
```
k=5

arrivalTime=[0,2,4,6,7]

reqTime=[2,3,4,5,6]

timeQuantaDecided =2000

'''

processes=range(k)

timeQuanta = timeQuantaDecided

reqTimeCopy=[i for i in reqTime]

queue=[]

answer=[]

waiting=[0 for i in range(k)]



for i in range(100):

for process in processes: #check new processes comming

if(arrivalTime[process]==i):

queue.append(process)

if(reqTime[queue[0]]==0):# if process still over next
```

```
timeQuanta= timeQuantaDecided

queue.pop(0)

elif(timeQuanta==0):

timeQuanta= timeQuantaDecided

queue.append(queue[0])

queue.pop(0)

if(len(queue)==0 and i!=0):

break

currentProcess=queue[0]

print(i,"\t", currentProcess,"\t\t\t",queue)

for waitingProcess in processes:

# if process arrived and waiting and present in
queue(not terminated)

if(waitingProcess!=currentProcess and
arrivalTime[waitingProcess]<=i and waitingProcess in
queue):

waiting[waitingProcess]+=1

answer.append(currentProcess)

reqTime[currentProcess]-=1
```

```
timeQuanta-=1

print("Total time spent waiting for processes is given
by", waiting)

print("Average waiting
Time",sum(waiting)/len(waiting))

turnaroundTimeArray=[int(waiting[i])+int(reqTimeCopy[i
]) for i in processes]

print("Total turnaround time for processes is given
by",turnaroundTimeArray )

print("Average turnaround
Time",sum(turnaroundTimeArray)/len(turnaroundTimeArray
))

print("\n")

print("\n_____________________________________________________
_____________________________________________________
_____________________________________________________
________")



temp=-1

for i in range(0,len(answer)):

if(i==0):
```

```
print("|",answer[i],end="\t")

temp=answer[i]

else:

if(temp==answer[i]):

print("", end="\t")

continue

else:

print("|",answer[i],end="\t")

temp=answer[i]

print("|\n-----------------------------------------
-----------------------------------------------------
-----------------------------------------------------
---------")

for i in range(0,len(answer)+1):

print(i,end="\t")
```

OUTPUT-



SJF Shortest job first

```
'''

SJF (non preemptive)

k=int(input("Please enter number of processes "))


arrivalTime=[int(input("Please enter arrival time for
process "+str(i)+" ")) for i in range(k)]


reqTime=[int(input("Please enter estimated time for
process "+str(i)+" ")) for i in range(k)]




priority=[int(input("Please enter priority for process
"+str(i)+" ")) for i in range(k)]







'''

#Test case
```

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

```
k=5

arrivalTime=[0,2,4,6,7]

reqTime=[2,3,4,5,6]

priority=[3,2,1,4,5]




processes=range(k)



reqTimeCopy=[i for i in reqTime]

queue=[]

answer=[]

waiting=[0 for i in range(k)]



for i in range(100):

for process in processes: #check new processes comming

if(arrivalTime[process]==i):

queue.append(process)

if(len(queue)==1): # for first process
```

```
currentProcess=process

if(reqTime[queue[0]]==0):# if process ended, then only
go to next process

queue.pop(0)

if(len(queue)==0 and i!=0):

break # all processes over

currentProcess = reqTime.index(min([reqTime[process]
for process in range(len(processes)) if process in
queue]))

queue.remove(currentProcess)

queue.insert(0,currentProcess)

if(len(queue)==0 and i!=0):

break # all processes over

currentProcess = currentProcess # non preemptive



print(i,"\t", currentProcess,"\t\t\t",queue)

for waitingProcess in processes:

# if process arrived and waiting and present in
queue(not terminated)
```

```
if(waitingProcess!=currentProcess and
arrivalTime[waitingProcess]<=i and waitingProcess in
queue):

waiting[waitingProcess]+=1

answer.append(currentProcess)

reqTime[currentProcess]-=1



print("Total time spent waiting for processes is given
by", waiting)

print("Average waiting
Time",sum(waiting)/len(waiting))

turnaroundTimeArray=[int(waiting[i])+int(reqTimeCopy[i
]) for i in processes]

print("Total turnaround time for processes is given
by",turnaroundTimeArray )

print("Average turnaround
Time",sum(turnaroundTimeArray)/len(turnaroundTimeArray
))

print("\n")

print("\n_____
_____
```

```
_____________________________________________________________
_________")



temp=-1

for i in range(0,len(answer)):

if(i==0):

print("|",answer[i],end="\t")

temp=answer[i]

else:

if(temp==answer[i]):

print("", end="\t")

continue

else:

print("|",answer[i],end="\t")

temp=answer[i]

print("|\n-----------------------------------------
-----------------------------------------------------
-----------------------------------------------------
---------")

for i in range(0,len(answer)+1):
```

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

```
print(i,end="\t")
```

```
-----------------------------------------------------------------------------------------
| 0       | 1             | 2             | 3             | 4             |
-----------------------------------------------------------------------------------------
0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19  20
```

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

Priority

```
'''

k=int(input("Please enter number of processes "))


arrivalTime=[int(input("Please enter arrival time for
process "+str(i)+" ")) for i in range(k)]


reqTime=[int(input("Please enter estimated time for
process "+str(i)+" ")) for i in range(k)]



priority=[int(input("Please enter priority for process
"+str(i)+" ")) for i in range(k)]




'''

#Test case
```

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

```
k=5

arrivalTime=[0,2,4,6,7]

reqTime=[2,3,4,5,6]

priority=[3,2,1,4,5]




processes=range(k)



reqTimeCopy=[i for i in reqTime]

queue=[]

answer=[]

waiting=[0 for i in range(k)]



for i in range(100):

for process in processes: #check new processes comming

if(arrivalTime[process]==i):

queue.append(process)

if(len(queue)==1): # for first process
```

```
currentProcess=process

if(reqTime[queue[0]]==0):# if process ended, then only
go to next process

queue.pop(0)

if(len(queue)==0 and i!=0):

break # all processes over

currentProcess = priority.index(min([priority[process]
for process in range(len(processes)) if process in
queue]))

queue.remove(currentProcess)

queue.insert(0,currentProcess)

if(len(queue)==0 and i!=0):

break # all processes over

currentProcess = currentProcess # non preemptive



print(i,"\t", currentProcess,"\t\t\t",queue)

for waitingProcess in processes:

# if process arrived and waiting and present in
queue(not terminated)
```

```
if(waitingProcess!=currentProcess and
arrivalTime[waitingProcess]<=i and waitingProcess in
queue):

waiting[waitingProcess]+=1

answer.append(currentProcess)

reqTime[currentProcess]-=1



print("Total time spent waiting for processes is given
by", waiting)

print("Average waiting
Time",sum(waiting)/len(waiting))

turnaroundTimeArray=[int(waiting[i])+int(reqTimeCopy[i
]) for i in processes]

print("Total turnaround time for processes is given
by",turnaroundTimeArray )

print("Average turnaround
Time",sum(turnaroundTimeArray)/len(turnaroundTimeArray
))

print("\n")

print("\n_____
_____
```

```
_____________________________________________________________
_________")



temp=-1

for i in range(0,len(answer)):

if(i==0):

print("|",answer[i],end="\t")

temp=answer[i]

else:

if(temp==answer[i]):

print("", end="\t")

continue

else:

print("|",answer[i],end="\t")

temp=answer[i]

print("|\n-----------------------------------------
-----------------------------------------------------
-----------------------------------------------------
---------")

for i in range(0,len(answer)+1):
```

```
print(i,end="\t")
```

```
-------------------------------------------------------------------------------------
| 0		| 1		|		|		|		| 3		|		|		|		| 4		| 2		|		|		|
-------------------------------------------------------------------------------------
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
```

Conclusion:

Thus we have implemented non-preemptive scheduling algorithms. We have understood how different algorithms work in detail and their implementation. These schemes perform less operations than preemptive operations because once a process is scheduled, it runs till completion. However this can lead to starving of other processes.

Post Lab Objective Questions

1. What is the ready state of a process?
   a) when process is scheduled to run after some execution
   b) when process is unable to run until some task has been completed
   c) when process is using the CPU
   d) none of the mentioned

Ans:a

2. A process stack does not contain
   a) function parameters
   b) local variables
   c) return addresses
   d) PID of child process

Ans:b

3. A process can be terminated due to
   a) normal exit
   b) fatal error
   c) killed by another process
   d) all of the mentioned

Ans:a

Date: 25 sept **Signature of faculty in-charge**