Shape, arrow

Description automatically generated

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

**A LAB REPORT**

**ON:**

**PROCESS SCHEDULING**

**ALGORITHMS**

**Submitted by:**

MILAN SHRESTHA, 075BCT050

Group: BCT C

**Submitted to**:

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

LALITPUR, NEPAL

**THEORY**

**Process Scheduling**

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process based on a particular strategy.

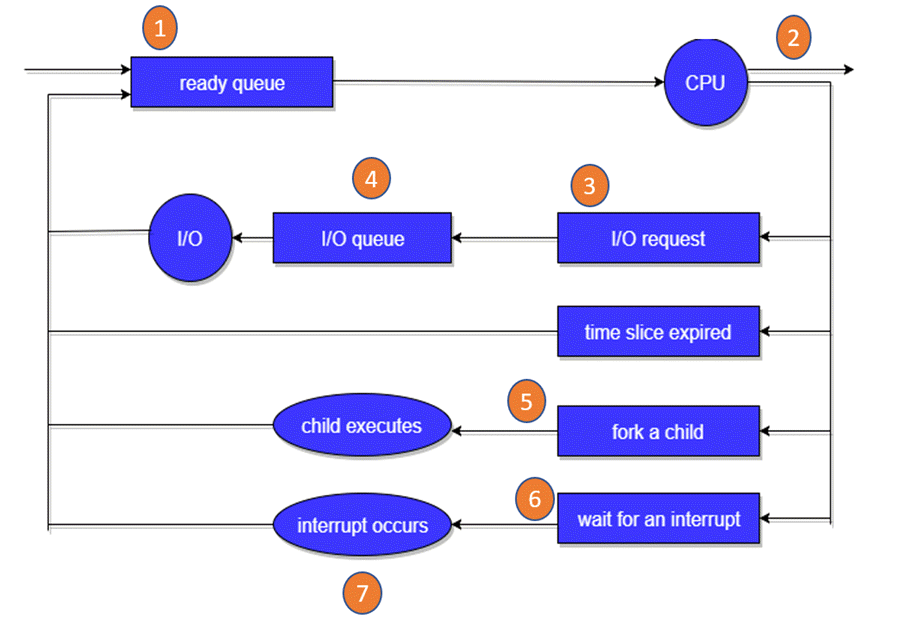
Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

**Process Scheduling Queues**

The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues −

* **Job queue** − This queue keeps all the processes in the system.
* **Ready queue** − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
* **Device queues** − The processes which are blocked due to unavailability of an I/O device constitute this queue.



Process Scheduling Queues

In the above-given Diagram,

* Rectangle represents a queue.
* Circle denotes the resource
* Arrow indicates the flow of the process.

1. Every new process first put in the Ready queue .It waits in the ready queue until it is finally processed for execution. Here, the new process is put in the ready queue and wait until it is selected for execution or it is dispatched.
2. One of the processes is allocated the CPU and it is executing
3. The process should issue an I/O request
4. Then, it should be placed in the I/O queue.
5. The process should create a new subprocess
6. The process should be waiting for its termination.
7. It should remove forcefully from the CPU, as a result interrupt. Once interrupt is completed, it should be sent back to ready queue.

We often use two parameters to determine efficiency of scheduling algorithm: waiting time and turnaround time. Turnaround time is the time between successful completion of a process and its arrival. Waiting time is the time that a process is logically runnable but is suspended by the CPU for executing other process. Waiting time is also alternatively given as difference of turnaround time and the actual CPU execution time.

Some of the process scheduling algorithms are:

**Round Robin Scheduling**

Round robin scheduling is the pre-emptive scheduling in which every process gets executed in a cyclic way, i.e., in this a particular time slice is allotted to each process which is known as time quantum. Every process, which is present in the queue for processing, [CPU](https://ecomputernotes.com/fundamental/introduction-to-computer/what-is-cpu) is assigned to that process for that time quantum. Now, if the execution of the process gets completed in that time quantum, then the process will get terminate otherwise the process will again go to the ready queue, and the previous process will wait for the turn to complete its execution. The process that is pre-empted 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. It is a real time algorithm which responds to the event within a specific time limit.

**Shortest Job First Scheduling**

**Shortest Job First (SJF)** is an algorithm in which the process having the smallest execution time is chosen for the next execution. This scheduling method can be pre-emptive or non-pre-emptive. It significantly reduces the average waiting time for other processes awaiting execution. It is associated with each job as a unit of time to complete. This algorithm method is helpful for batch-type processing, where waiting for jobs to complete is not critical. It can improve process throughput by making sure that shorter jobs are executed first, hence possibly have a short turnaround time. It improves job output by offering shorter jobs, which should be executed first, which mostly have a shorter turnaround time.

**Shortest Remaining Time Next**

Shortest remaining time next scheduling algorithm is also referred to as pre-emptive SJF scheduling algorithm. When a new process arrives at the ready queue while one process is still executing then the SRTN algorithm is performed to decide which process will execute next. This algorithm compares CPU burst time of newly arrived process remaining (left) CPU burst time to currently executing process. If the CPU burst time of a new process is less than the remaining time of current process, then SRTN algorithm pre-empts current process execution and starts executing new process.

**First Come First Serve**

First Come First Serve (FCFS) is an operating system scheduling algorithm that automatically executes queued requests and processes in order of their arrival. It is the easiest and simplest CPU scheduling algorithm. In this type of algorithm, the process which requests the CPU first gets the CPU allocation first. This is managed with a FIFO queue. As the process enters the ready queue, its PCB (Process Control Block) is linked with the tail of the queue and, when the CPU becomes free, it should be assigned to the process at the beginning of the queue.

It supports non-pre-emptive and pre-emptive scheduling algorithms. Jobs are always executed on a first come, first-serve basis. It is easy to implement and use. This method is poor in performance, and the general wait time is quite high.

**Program Code**

INT\_MAX = 10000

class Process:

    no\_process = '#'

    def \_\_init\_\_(self, name, arrival\_time, cpu\_time):

        self.name = name

        self.AT = arrival\_time

        self.CT = cpu\_time

        self.RT = self.CT

        self.TAT = 0

        self.WT = 0

        self.executed = False

    def resetRT(self):

        self.RT = self.CT

    def turnaround(self, completion\_time):

        self.TAT = completion\_time - self.AT

        self.WT = self.TAT - self.CT

def FCFS(process):

    process.sort(key = lambda p: p.AT)

    cur\_time = 0

    schedule = []

    while cur\_time < process[0].AT:

        schedule.append(Process.no\_process)

        cur\_time += 1

    for cur\_process in range(len(process)):

        for t in range(process[cur\_process].CT):

            cur\_time += 1

            schedule.append(process[cur\_process].name)

        process[cur\_process].turnaround(cur\_time)

    return schedule

def SJF(process):

    schedule = []

    executing\_process = None

    process\_executed = 0

    cur\_time = 0

    sj = INT\_MAX

    n = len(process)

    while process\_executed < n:

        if executing\_process == None:

            sj = INT\_MAX

            for p in process:

                if p.executed:

                    continue

                if p.AT <= cur\_time:

                    if p.CT < sj:

                        sj = p.CT

                        executing\_process = p

        if sj == INT\_MAX:

            schedule.append(Process.no\_process)

        else:

            schedule.append(executing\_process.name)

            executing\_process.RT -= 1

            if executing\_process.RT == 0:

                executing\_process.turnaround(cur\_time + 1)

                executing\_process.executed = True

                process\_executed += 1

                executing\_process = None

        cur\_time += 1

    return schedule

def SRTN(process):

    schedule = []

    executing\_process = None

    process\_executed = 0

    cur\_time = 0

    rt = INT\_MAX

    n = len(process)

    while process\_executed < n:

        rt = INT\_MAX

        for p in process:

            if p.executed:

                continue

            if p.AT <= cur\_time:

                if p.RT < rt:

                    rt = p.RT

                    executing\_process = p

        if rt == INT\_MAX:

            schedule.append(Process.no\_process)

        else:

            schedule.append(executing\_process.name)

            executing\_process.RT -= 1

            if executing\_process.RT == 0:

                executing\_process.turnaround(cur\_time + 1)

                executing\_process.executed = True

                process\_executed += 1

                executing\_process = None

        cur\_time += 1

    return schedule

def RR(process, timeQuanta):

    schedule = []

    executing\_process = []

    process\_executed = 0

    cur\_time = 0

    n = len(process)

    rtq = timeQuanta

    while process\_executed < n:

        for p in process:

            if p.executed or p in executing\_process:

                continue

            if p.AT <= cur\_time:

                executing\_process.append(p)

        if len(executing\_process) == 0 :

            schedule.append(Process.no\_process)

        else:

            rtq -= 1

            executing\_process[0].RT -= 1

            schedule.append(executing\_process[0].name)

            if executing\_process[0].RT == 0:

                p = executing\_process.pop(0)

                p.turnaround(cur\_time + 1)

                p.executed = True

                process\_executed += 1

                rtq = timeQuanta

            elif rtq == 0:

                lep = executing\_process.pop(0)

                executing\_process.append(lep)

                rtq = timeQuanta

        cur\_time += 1

    return schedule

def printSchedule(schedule):

    for i in range(len(schedule)):

        print(i,': ', schedule[i])

def calcAvgTimes(process):

    avgTAT = 0

    avgWT = 0

    n = len(process)

    for i in range(n):

        avgTAT += process[i].TAT

        avgWT += process[i].WT

    avgTAT = avgTAT / n

    avgWT = avgWT / n

    return (avgTAT, avgWT)

def printProcess(process):

    print('{0: <10} {1: <15} {2: <10} {3: <10} {4:<10}'.format('Process','Arrival Time','CPU Time','TAT','WT'))

    for p in process:

        print('{0: <10} {1: <15} {2: <10} {3: <10} {4:<10}'.format(p.name,p.AT,p.CT,p.TAT,p.WT))

def getProcessesFromUser():

    process = []

    print('Enter the number of processses: ',end = ' ')

    n = int(input())

    for i in range(n):

        print('Process name: ',end = ' ')

        name = input()

        print('Arrival time: ',end = ' ')

        at = int(input())

        print('CPU execution time: ',end = ' ')

        ct = int(input())

        process.append(Process(name,at,ct))

        print(' ')

    return process

def getSchedulingAlg():

    print('Enter the scheduling algorithm to use: ')

    print('1.FCFS \t2.SJF \t3.SRTN \t4.RR ',end = '\t')

    n = int(input())

    if n == 1:

        return 'FCFS'

    elif n == 2:

        return 'SJF'

    elif n == 3:

        return 'SRTN'

    elif n == 4:

        return 'RR'

    else:

        return None

if \_\_name\_\_ == '\_\_main\_\_':

    process = getProcessesFromUser()

    salg = getSchedulingAlg()

    schedule = []

    if salg == 'RR':

        print('time Quanta: ',end = ' ')

        tq = int(input())

    print(f'\nScheduling by {salg}')

    if salg == 'FCFS':

        schedule = FCFS(process)

    elif salg == 'SJF':

        schedule = SJF(process)

    elif salg == 'SRTN':

        schedule = SRTN(process)

    elif salg == 'RR':

        schedule = RR(process,tq)

    printSchedule(schedule)

    avgtime = calcAvgTimes(process)

    printProcess(process)

    print('\navgTAT: ',avgtime[0],' avgWT: ',avgtime[1])

    input()

**Output:**

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**DISCUSSION AND CONCLUSION**

Process Scheduling is used to manage the order of execution of multiple processes and every process scheduling algorithm uses all the CPU time slots. The total CPU execution time for all the processes is the same theoretically, however, in practice there might be slight variations in time due to context switching which is usually higher in RR. The performance of the scheduling algorithm is highly dependent on the nature of the processes to be scheduled and therefore, a certain method cannot be declared superior to others. The choice of appropriate scheduling algorithm varies on the nature of each process in the queue. Some process scheduling algorithms such as SJF and SRTN require early knowledge of CPU execution time of the process which means batch process may be efficiently executed whereas interactive processes requiring I/O and thus having unpredictable nature may not be as efficiently executed. Therefore, the selection of process scheduling algorithm is completely process dependent. Hence there are different types of process scheduling algorithms for managing the order of execution of processes. Each of these processes has its own merits and demerits. The strategy for choosing the appropriate algorithm depends upon the requirement.