IMPACT: International Journal of Research in Applied, Natural and Social Sciences (IMPACT: IJRANSS)
ISSN(E): 2321-8851; ISSN(P): 2347-4580
Vol. 3, Issue 1, Jan 2015, 121-132
© Impact Journals

A COMPARATIVE ANALYSIS OF SCHEDULING ALGORITHMS

AJALA FUNMILOLA A, FENWA OLUSAYO D & ALADE MODUPE O

Department of Computer Science and Engineering, Ladoke Akintola University ofTechnology, Ogbomoso, Nigeria

ABSTRACT

Time management is an important factor highly considered in any organization or system because it goes a very long way in determining productivity. In the light of this, software engineers using the scheduler have taken series of measures in designing systems that will process and complete jobs assigned to them in a predictably manageable time in order to increase the number of jobs processes per unit time. Decisions that the scheduler makes, concerning the sequence and length of time that processes may run are not easy ones, as the scheduler has only a limited amount of information about the processes that are ready to run. However, with the use of appropriate scheduling algorithms, important goals such as interactivity, effectiveness, cost and most importantly time can be balanced. In this work, three scheduling algorithms were considered, first-in-first-out (FIFO), round robin and shortest job First algorithms. A theoretical analysis that subject the algorithms to the same condition is performed through the use of exemplary job processes to determine the best among the algorithms. Job completion time, response time and permutation time were evaluated and it was discovered that shortest job first gives the optimal performance of scheduling processes followed by round robin and lastly FIFO.

KEYWORDS: Comparative Analysis, Scheduling, Single Tape Case, Completion Time

1. INTRODUCTION

In computer science, looking at some computing process, it will be noticed that it spends some time executing instructions (computing) and then makes some I/O request, for example, read or write data unnecessarily longer. After that, it executes more instructions and then, again, waits on I/O. The period of computation between I/O requests is called the CPU burst. Some processes perform a lot of input/output operations but use little CPU time (examples are web browsers, shells and editors). They spend much of their time in the blocked state in between little bursts of computation. The overall performance of these I/O bound processes is constrained by the speed of the I/O devices. CPU-bound processes and spend most of their time computing and execution time is largely determined by the speed of the CPU and the amount of CPU time they can get. [1]. As work goes on in a computer system there are series of other processes running too, e.g. VLC media player, some few games and some online search. Most of the time, these processes collectively are using less than 3% of the CPU. This is not surprising since most of these programs are waiting for user input like the games, a network message like the search, or sleeping and waking up periodically to check some states.[4]

Consider a 2.4 GHz processor. It executes approximately 2,400 million instructions per second. Hence, the process of switching the processor to run a particular process when another one has to do an I/O is multiprogramming. The process of running many processes at the same time is multitasking [1]. 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 are kept on a ready list also known as a run queue. Entries in this list are pointers to the process control block, which stores all information and state about a process. SCHEDULING therefore refers to a set of policies and mechanisms built into the operating system that govern the orders in which the work to be done by a computer system is completed. It can be said to be the method by which threads, processes or data flows are given access to system resources e.g, processor time, and communicator bandwidths, among others. There are lots of ways to do this and they depend on the goals of the designers. 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. A scheduler is preemptive, if it has the ability to get invoked by an interrupt and move a process out of a running state and let another process run. The last two events in the above list may cause this to happen. If a scheduler cannot take the CPU away from a process then it is a cooperative or non-preemptive scheduler. Old operating systems such as Windows 3.1 or Mac OS before OS X are examples of such schedulers. Even older batch processing systems had run-to-completion schedulers where a process ran to termination before any other process would be allowed to run.

However, only the First in First Out algorithm (FIFO), Round Robin Policy (RRB) and the Shortest Job First (SJF) is examined in this paper. In section 2, each algorithm was briefly discussed and a step by step procedure to implement each with their respective flowcharts was shown. In section 3, a theoretical analysis of the algorithms is performed through the use of exemplary job processes to determine the best among them and conclusion was drawn in section 4.

2. LITERATURE REVIEW

Scheduling algorithms are the systematic methods used in determining the order in which series of tasks will be performed by a computer system. When incorporated into the operating system, it balances important goals such as interactivity and dictates how much CPU time is allocated to the above said processes, jobs; threads etc. [2]. A good scheduling algorithm should:

- Be fair-give each process a fair share of the CPU, allow each process to run in a reasonable amount of time.
- Be efficient-keep the CPU busy all the time.
- 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.
- Minimize response time-interactive users should see good performance.
- Be predictable; a given job should take about the same amount of time to run when run multiple times. This keeps users sane.
- Minimize overhead; don’t waste too many resources. Keep scheduling time and context switch time at a minimum.
- Maximize resource use; favor processes that will use underutilized resources.
- Avoid indefinite postponement; every process should get a chance to run eventually.
- Enforce priorities; if the scheduler allows a process to be assigned a priority, it should be meaningful and enforced.

- Degrade gracefully-as the system becomes more heavily loaded, performance should deteriorate gradually, not abruptly. [6]

It is clear that some of these goals are contradictory. To make matters even more complex, the scheduler does not know much about the behavior of each process. Therefore, to help the scheduler monitor processes and the amount of CPU time that they use, a programmable interval timer interrupts the processor periodically (typically 50 or 60 times per second). This timer is programmed when the operating system initializes itself. At each interrupt, the operating system's scheduler gets to run and decide whether the currently running process should be allowed to continue running or whether it should be suspended and another ready process allowed running. This is the mechanism used for preemptive scheduling. There are three main categories of scheduling algorithms which are:

- The Interactive Scheduling Algorithms, this include, the Round Robin Scheduling Algorithm, Priority Based Scheduling Algorithm, Shortest Process Next Algorithm and the Lottery Scheduling Algorithm.
- The batch scheduling algorithms, this include, First Come First Serve scheduling algorithm, Shortest Job First Algorithm (SJF), Shortest Remaining Time Next Algorithm, Lightest Response Ratio Algorithm and;
- The real time scheduling algorithms this include, The Rate Monotonic Scheduling Algorithm, Earliest Deadline First Algorithm. [5]

This section reviewed the three basic algorithms considered in this work.

2.1 First in First out Algorithm

A non-preemptive scheduling policy FIFO also known as FIRST COME FIRST SERVED is a queues processes in the order that they arrive in the FIFO ready queue. Basically there is a single queue of ready processes. Relative importance of jobs is measured only by arrival time. When a process enters the ready queue, its **PCB** is linked unto the tail of the queue. Processes that arrive while another is being served wait in line in the order of arrival. The order is very important for the turnaround-time.

With first-come, first-served scheduling, a process with a long CPU burst will hold up other processes. Moreover, it can hurt overall throughput since I/O on processes in the waiting state may complete while the CPU bound process is still running. Now devices are not being used effectively. For increasing throughput, it would have been great if the scheduler instead could have briefly run some I/O bound process that could request some I/O and wait. Because CPU bound processes don't get preempted, they hurt interactive performance because the interactive process won't get scheduled until the CPU bound one has completed.

A general algorithm for First in first out policy is described below;

Step 1: Input n, the number of jobs in the array

Step 2: Input all the times i.e. t1, t2….tn into the array

Step 3: Set the index I to 1 i.e. I ← j

Step 4: Repeat the following steps as long as i= n

Step 5: i← i + 1

Step 6: Print out t[i]

Advantages of First in First out Algorithm

- It is also intuitively fair (the first one in line gets to run first)
- It is the simplest form of scheduling algorithm.
- Since context switching only occur upon processes termination and no recognition of the process queue is required, scheduling overhead is minimal.
- The lack of prioritization means that as long as every process eventually completes, there is no starvation.
- It is more predictable than most other schemes because it offers time.
- The code for FIFO is simple to write and understand.

Disadvantages of First in First out Algorithm

- The greatest drawback of first-come, first-served scheduling is that it is not preemptive. Because of this, it is not suitable for interactive jobs.
- The importance of a job is measured only by the arrival tome (poor choice).
- Turnaround time, average waiting time and response time can be high for the same reason above.
- No prioritization occur, thus this system has trouble meeting processes' deadlines.
- It is unfair in the sense that long jobs make short jobs wait and unimportant jobs make important ones wait.

The first in first out algorithm is rarely used as a master scheme in modern operating systems but it is often embedded within other schemes.

2.2 Round Robin Scheduling Algorithm

The Round Robin (RR) scheduling algorithm is designed specifically for time-sharing systems. It is a preemptive version of first-come, first-served scheduling. Processes are dispatched in a first-in-first-out sequence but each process is allowed to run for only a limited amount of time. This time interval is known as a time-slice or quantum. It is similar to FIFO scheduling but pre-emption is added to switches between processes. [3]. Typical quantum varies from 100 milliseconds to 2 seconds. To implement RR scheduling,

- The ready queue is kept as a FIFO queue of processes.
- New processes are added to the tail of the ready queue.
- The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after the time quantum, and dispatches the processes.
- The process may have a CPU burst of less than 1 time quantum. In this case, the process itself will release the CPU voluntarily. The scheduler will now proceed to the next process in the ready queue.

- Otherwise, if the CPU burst of the currently running process is longer than the 1 time quantum, the timer will go off and will cause an interrupt to the operating system. A context switch will be executed and the process will be put at the tail of the ready queue. The CPU scheduler will now select the next process in the ready queue.

The size of the quantum time in relation to the average CPU cycle is crucial to the performance of the system. [4]. with round robin scheduling, interactive performance depends on the length of the quantum and the number of processes in the run queue. A very long quantum makes the algorithm behave very much like first come, first served scheduling since it's very likely that a process with block or complete before the time slice is up. A small quantum lets the system cycle through processes quickly. This is wonderful for interactive processes. Unfortunately, there is an overhead to context switching and having to do so frequently increase the percentage of system time that is used on context switching rather than real work. The overhead associated with a context switch can be expressed as:

Context switch overhead = $C / (Q+C)$.

Where Q is the length of the time-slice and C is the context switch time. An increase in Q increases efficiency but reduces average response time.

A general algorithm for Round robin policy is described below;

```
Step 1:  Input n, the number of element in the array
Step 2:  Input all the numbers into array job
Step 3:  Set the index I to n do
Step 4:  Input job[i] period
Step 5:  Job [i] sum: =0
Step 6:  // supply the round robin time //
             •  Input (r time)
             •  Count: = 0
Step 7:  Repeat for i= 1 to n do
             •  Job [i]
             •  (2) If [(sum=period) and (done<>true), then begin.
Step 8:  // 'Job', 'I', 'of time', 'period', 'has been executed' //
   •  Count: = count + 1;
   •  Done: = true
         Else
Step 9:  Sum: = sum + r time;
Until
```

Step 10: Count = n

Step 11: Stop.

Advantages of Round Robin Algorithm

- Round robin scheduling is fair in that every process gets an equal share of the CPU.
- It is easy to implement and, if we know the number of processes on the run queue, we can know the worst-case response time for a process.
- The algorithm is simple in nature
- It has a strict first come first serve nature
- It is a fast improvement over FIFO

Disadvantages of Round Robin algorithm

- Giving every process an equal share of the CPU is not always a good idea. For instance, highly interactive processes will get scheduled no more frequently than CPU-bound processes.
- A process may block itself before it time slice expires.
- Poor average waiting time when the job lengths are identical.
- Absence of priority system which means lots of low privileged processes may starve one high privileged one.

2. 3 The Shortest Job First Algorithm

A different approach to CPU scheduling is the SHORTEST JOB FIRST ALGORITHM. This algorithm associates with each process, the length of the process' next CPU burst i.e. the running time. When the CPU is available, it is assigned to the process that has the smallest CPU burst. If the next CPU bursts of two processes are the same, FIFO IS USED. The process that arrived first out of the two is executed first. [5]. Shortest job first scheduling runs a process to completion before running the next one. The queue of jobs is sorted by estimated job length, so that short programs get to run first and not be held up by long ones.

The SJF algorithms can either be preemptive or non-preemptive. The choices arise when a new process arrives at the ready queue while a previous process is still executing. In some cases, the next CPU burst of the newly arrived process may be shorter than what is left of the currently running process. When a preemptive SJF is the choice, it will preempt the currently running process i.e. Stop it till the execution of the newly arrived one while a non-preemptive SJF will allow the old process to finish before the execution of the new one. The preemptive SJF is sometimes called the shortest-remaining-time-first algorithm.

A general algorithm for shortest job first policy is described below;

Step 1: Input n, the number of jobs in the array

Step 2: Input all the times i.e. t1, t2….tn into the array

Step 3: initialize i $\leftarrow$ 0

```
        Step 4:  Repeat the following steps as long as I < n-1
    •   Set the index j to i + 1 i.e. j ← i+1
        Repeat the following steps as long as j< n
        If x[j]< x[i]
        Temp ← x[j]
        X[j] ← x[i]
        X[i] ← temp
        J ← j+1
    •   I ← i+1
    Step 6: Stop.
```

Advantages of Shortest-Job-First Scheduling Algorithm

- This scheduling always produces the lowest mean response time Processes with short CPU bursts run quickly (are scheduled frequently).
- It helps in knowing in advance, the minimum average waiting time for a given set of processes.
- The SJF algorithm has an optimal turnaround time especially if all jobs and processes are available simultaneously.
- Moving a short process before a long one decreases the waiting time of the short process more than it increases that of a long process. Consequently, the average waiting time is decreased.

Disadvantages of Shortest-Job-First Scheduling Algorithm

- Long-burst (CPU-intensive) processes are hurt with a long mean waiting time. In fact, if short-burst processes are always available to run, the long-burst ones may never get scheduled. Moreover, the effectiveness of meeting the scheduling criteria relies on our ability to estimate the CPU burst time.
- It cannot be implemented at the level of short-term scheduling due to the reason stated above.

In all, scheduling can be summarized to cover the question of when to introduce new processes into the system, the order in which they should run and for how long. The part of the operating system concerned with making these decisions is called the SCHEDULER and the algorithms it uses are called the SCHEDULING ALGORITHMS.

3. RESULTS AND DISCUSSIONS

This section examined the algorithms using a particular job process. A way to minimize their mean completion times from a single-tape case was determined using the permutation method and then compared the results.

- Average mean completion time case

Supposing some jobs J1 J2 ……Jnare given, all with known running times t1 t2…..tn respectively with a single processor serving them. The question then arises as to the best way to schedule this job in order to minimize the average completion time. In this entire section, non-preemptive scheduling shall be assumed i.e. once a job is started, it must run to completion.

The assumed four jobs with their associated running times are shown in the table below:

Table 3: Four Jobs and their Associated Running Times

Job	Run Time
J1	15
J2	8
J3	3
J4s	10

Table 4: FIFO Queue

Job	Run Time	Execution
J1	T1	First
J2	T2	Second
J3	T3	Third
J4	T4	Fourth

It is known from the above table that provided J1 comes before J2 and J3 comes before J4, then J1 is executed first, followed by J2, then J3 and lastly J4.

Table 5: Round Robin Queue

	J1	J2	J3	J4
First Round	t1 – 1 - - -	- t2 – 1 - -	- - t3 – 1 -	- - - t4 – 1
Second Round Repeat until	t1 – 2 - - - t1 = 0	- t2 – 2 - - t2 = 0	- - t3 – 2 - t3 = 0	- - - t4 – 2 t4 = 0

Round Robin services a process only for a single quantum of time although in the order of arrival. Suppose each process is to be serviced for 1 second before it is interrupted, then the procedure is as stated in table below.

Table 6: SJF Queue

Job	Run Time
J3	3
J2	8
J4	10
J1	15

The above table shows that when a process is to be selected from a ready queue the one with the shortest service time is chosen. t3 is executed first, followed by t2, then t4 and lastly t1.Having considered these, the total cost was determined(i.e. the completion time of the schedules) using equation 1.

C = £ (n-k+1) ti (1)

Where n is the number of jobs, k is the number in order of arrival or execution and t is the runtime. Feasible permutation of the jobs was performed to determine the required ones.

First, let the permutation be within the range of δ = i1, i2, i3……..i24, where δ is the feasible solution as stated below, solving δ! , we have;

I1: t1,t2,t3,t4 i7: t2,t1,t3,i13;t3,t1,t2,t4

I2: t1, t2, t4, t3 i8: t2, t1, t4, i14: t3, t1, t4, t2

I3: t1, t3, t2, t4 i9: t2, t3, t1, t4 i15: t3, t2, t4, t1

I4: t1, t3, t4, t2 i10:t2, t3, t4, t1 i16: t3, t2, t1, t4

I5: t1, t4, t3, t2 i11: t2, t4, t3, t1 i17: t3, t4, t1, t2

I6: t1, t4, t2, t3 i12: t2, t4, t1, t3 i18: t3, t4, t2, t1

From above, i1: t1, t2, t3, t4is for FIFO and ROUNDROBIN which queue processes in order of arrival and the processes did arrived in the order J1, J2, J3 and J4 although RR assigns a fixed quantum of time for each process' execution, while i15: t3, t2, t4, t1 is for SJF which selects the process with the shortest service time for execution first. The processes had short service times in the order J3, J2, J4 and J1.

For FIFO and ROUND ROBIN, we get the average mean completion time using equation 2.

Cti= £ (n-k+1) ti (2)

Where k = 1, 2, 3 and 4 while ti = t1, t2, t3 and t4 respectively. Table 5 shows FIFO and Round robin with their cost.

Table 7: FIFO, Round Robin and Their Costs

Job	Time	Cost
J1	15	60
J2	8	24
J3	3	6
J4	10	10
Completion Time		**100**

The table above gives us the average mean completion time:

Ct1 = £ (4-1+1)15= 60

Ct2 = £ (4-2+1)8 = 24

Ct3 = £ (4-3+1)3 =6

Ct4= £ (4-4+1)10=10

The average mean completion time is

(ct1+ct2+ct3+ct4)/4

(60+24+6+10)/4 = 25

The table above shows that for SJF, The first job finishes in ti1, the second job finishes after ti1 + ti2, the third finishes after ti1 + ti2 + ti3 and the fourth finishes after ti1 + ti2 + ti3 + ti4.

Table 8: Shows SJF and Its Cost

Job	Time	Cost
J3	3	12
J2	8	24
J4	10	20
J1	15	15
Total Completion Time		**71**

Using equation 3

Ci15 = £ (n-k+1) ti (3)

Where k = 1, 2, 3 and 4 while ti = t3, t2, t1 and t4 respectively,

Ct3 = £ (4-1+1)3 = 12

Ct2 = £ (4-2+1)8 = 24

Ct1 = £ (4-3+1)10 =20

Ct4= £ (4-4+1)15=15

The average mean completion time is

(ct3+ct2+ct1+ct4)/4

(12+24+20+15)/4 =17.75

It has been shown from above that the shortest job first algorithm has the lowest mean completion time. And since completion time determines the response time, turnaround time and throughput; all which are very important qualities of a good scheduling algorithm, it can be concluded that the best of the three compared algorithm is the shortest job first scheduling algorithm in terms of time and cost

- **Single Tape Case**

Assuming there are n programs that are to be stored on a computer tape of length L. associated with each program I is a length li and $1 \leq i \leq n$.

Clearly, all programs can be stored on the tape provided the sum of the length of the programs does not exceed L. we shall assume that whenever a program is to be retrieved from this tape, the tape is initially positioned in front. Hence, if the programs are stored in the order I= i1, i2……in, the time t1 needed to retrieve a particular program ij is directly proportional to n (tj). To achieve the optimal storage on a tape problem, we are required to find a permutation for the n programs so that when they are stored on the tape in this order, the Mean Retrieval Time (MRT)is minimized. And minimizing the MRT is equivalent to minimizing C.

For instance, let n i.e. the no of programs to be stored = 3 and (l1, l2, l3) are associated with (t1, t2, t3). There are n! = 6 possible orderings. These orderings and their respective Retrieval times (C) are on the table below:

Let the orderings of the permutation be i1, i2, i3 ….i6, we have the resulting values

I1=(1,2,3) , i2= (1,3,2) , i3= (2,1,3) , i4=(1,3,4) , i5= (3,1,2) and i6= (3,2,1).

The optimal ordering can then be determined since the one that has the minimum MRT can be calculated using C= £ (n-k+1) li

Table 9: Shows the FIFO (i1) Permutation Ordering

Length	Time	Cost
L1	5	15
L2	10	20
L3	3	3
TRT		38
MRT		38/3= 12 $_{2/3}$

Since FIFO groups jobs in order of arrival, and the jobs did arrived in the order 1, 2 and 3, i1 was chosen.

Table 10: Shows the SJF (i5) Permutation Ordering

Length	Time	Cost
L3	3	9
L1	5	10
L2	10	10
TRT		29
MRT		29/3 = 9 $_{2/3}$

Since SJF groups jobs by selecting those with the shortest execution time first, we have the order i5= 3, 1, 2 chosen.

Table 11: Shows the Round Robin (i1) Permutation Ordering

Length	Time	Cost
L1	5	15
L2	10	20
L3	3	3
TRT		38
MRT		38/3= 12 $_{2/3}$

Round Robin is just like FIFO, quantum time is neglected in this case. So i1 is chosen.

From the above, it can be seen that SJF gives the optimal ordering which is i5 resulting in the lowest MRT of 9$_{2/3.}$It is necessary to choose SJF in preference to others to increase the number of jobs i.e. programs to be retrieved per time and to minimize their turn-around times as the above tables have analyzed. The approach to build the required permutation would choose the next program based upon some optimization procedures. One possible measure would be the C value of the permutation constructed so far. The next program to be stored on the tape would be the one which minimizes the increase in C most. We trivially observed that the increase in C is minimized if the next program chosen is the one with the largest length from among the remaining programs

Table 11: Comparative Table of the Analysis

Algorithm Factors	SJF	FIFO	RRP
RESPONSE TIME	Low	High	Fair
Turn Around Time	Low	High	Fair
Throughput	Maximized	Minimized	A bit minimized
Completion Time	Low	High	High

4. CONCLUSIONS

As fascinating and as good as most of the scheduler's goals are, it is clear that some of these are contradictory. For example, minimizing overhead means that jobs should run longer, thus hurting interactive performance. Enforcing priorities means that high-priority processes will always be favored over low-priority ones, causing indefinite postponement. These factors make scheduling a task for which there can be no perfect algorithm. However, the Shortest Job First Preemptive scheduling allows the scheduler to control response times by taking the CPU away from a process that it decided has been running too long. It has more overhead than non-preemptive scheduling since it has to deal with the overhead of context switching processes instead of allowing a process to run to completion or until the next I/O operation or other system call. Therefore, the best algorithm among the three compared is the Shortest Job First Scheduling Algorithm, because it gives the optimal solution for scheduling jobs.

REFERENCES

1. Mark A. Wekss, (1991), ''Data Structure and Algorithms Analysis'', University Press, Florida, USA.
2. A.E Okeyinka, (1998), ''Introduction to Computer Technology'', LAUTECH Press, Ogbomosho.
3. Ellis Horoniz and SarjajSalini, (1991) ''Fundamental of Computer Algorithms'', online book data, www.ctl.com.
4. Perez et Roberts,(2002), Project scheduling, university of California, USA
5. Munger, R, Tesfai. K (2001). ''Scheduling with ease, an approach to utilizing specific algorithms''.
6. Friedman, M.A., Greene, E.J. (1972). ''Fundamentals of Operating system scheduling algorithms''