A Critical Simulation of CPU Scheduling Algorithm using Exponential Distribution

Maria Abur[1], Aminu Mohammed[2], Sani Danjuma[3] and Saleh Abdullahi[4]
[1]Iya Abubakar Computer Centre, Ahmadu Bello University, Zaria,

[2]Department of Mathematics, Ahmadu Bello University, Zaria,

[3]Sa'datu Rimi College of education, Kumbosto Kano state Nigeria

[4]Department of Mathematics, Ahmadu Bello University, Zaria,

Abstract
A multiprogramming operating system allows more than one process to be loaded into the executable memory at a time and for the loaded process to share the CPU using time-multiplexing. Part of the reason for using multiprogramming is that the operating system itself is implemented as one or more processes, so there must be a way for the operating system and application processes to share the CPU. The assignment of physical processors to processes allows processors to accomplish work. The problem of determining when processors should be assigned and to which processes is called CPU scheduling. How do we select a CPU Scheduling algorithm for a particular system? Since we have different scheduling algorithm with its own parameter selection can be difficult. To select an algorithm we must first define the relative importance of CPU Scheduling criteria. Next we use an evaluation method. This paper presents an algorithm and a life simulation of the CPU Scheduling algorithms using exponential distribution to generate the random numbers for the burst times, arrival times and processes with Ms Visual Basic 2010 for the Scheduling algorithms and comparing their average waiting time to know which has the least average waiting time.

Keyword: *CPU Scheduling, Exponential distribution, Multiprogramming, Processors, Simulation*

1. 0 Introduction

A multiprogramming operating system allows more than one process to be loaded into the executable memory at a time and for the loaded process to share the CPU using time-multiplexing. Part of the reason for using multiprogramming is that the operating system itself is implemented as one or more processes, so there must be a way for the operating system and application processes to share the CPU. Another main reason is the need for processes to perform I/O operations in the normal course of computation. Since I/O operations ordinarily require orders of magnitude more time to complete than do CPU instructions, multiprogramming systems allocate the CPU to another process whenever a process invokes an I/O operation.

The assignment of physical processors to processes allow processors to accomplish work. The problem of determining when processors should be assigned and to which processes is called processor scheduling or CPU scheduling. When more than one process is runable, the operating system must decide which one first. The part of the operating system concerned with this decision is called the scheduler, and algorithm it uses is called the scheduling algorithm.

2.0 CPU scheduling

- The scheduling problem:
  - Have K jobs ready to run
  - Have N _ 1 CPUs
  - Which jobs to assign to which CPU(s)
- When do we make decision? Scheduling decisions may take place when a process:
  - Switches from running to waiting state
  - Switches from running to ready state
  - Switches from waiting to ready
  - Exits.

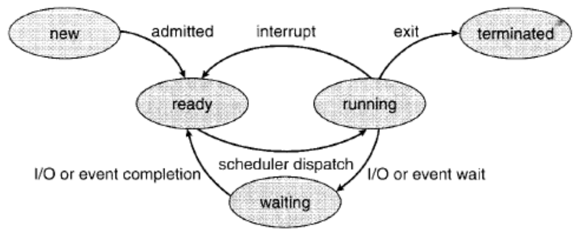


Fig. 1: A diagram illustrating process state.

2.1 Goals for CPU Scheduling

To make sure that scheduling strategy is good enough with the following criteria:

- Utilization/Efficiency: keeps the CPU busy 100% of the time with useful work.
- Throughput: maximizes the number of jobs processed per hour.
- Turnaround time: from the time of submission to the time of completion and minimize the time batch users must wait for output.
- Waiting time: Sum of times spent in ready queue.
- Response Time: time from submission till the first response is produced and minimize response time for interactive users.
- Fairness: make sure each process gets a fair share of the CPU.

2.2.0 Pre-emptive Vs Non pre-emptive Scheduling

The Scheduling algorithms can be divided into two categories with respect to how they deal with clock interrupts:

2.2.1 Non pre-emptive Scheduling

A scheduling discipline is non pre-emptive if, once a process has been given the CPU, the CPU cannot be taken away from that process. The following are some characteristics of non pre-emptive scheduling:

- Short jobs are made to wait by longer jobs but the overall treatment of all processes is fair.
- Response times are more predictable because incoming high priority jobs cannot displace waiting jobs.
- In non pre-emptive scheduling, a scheduler executes jobs in the following two situations.
  - When a process switches from running state to the waiting state.
  - When a process terminates.

2.2.3 Pre-emptive Scheduling: A scheduling discipline is pre-emptive if, once a process has been given the CPU can taken away. The strategy of allowing processes that are logically runable to be temporarily suspended is called pre-emptive Scheduling and it is contrast to the "run to completion" method.

2.3 Basic CPU Scheduling Algorithm: CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU. The Basic CPU Scheduling algorithms are First-Come, First-Served, Shortest Job First, Round Robin and Priority Based Scheduling.

2.3.1 FCFS - First-Come, First-Served: It is non-pre-emptive, Ready queue is a FIFO queue, Jobs arriving are placed at the end of queue, Dispatcher selects first job in queue and this job runs to completion of CPU burst. The advantages of FCFS is that it is simple and has low overhead. And has disadvantages of inappropriate for interactive systems and large fluctuations in average turnaround time are possible.

2.3.2 SJF - Shortest Job First: It is non-pre-emptive, Ready queue treated as a priority queue based on smallest CPU time requirement, arriving jobs inserted at proper position in queue, dispatcher selects shortest job (1st in queue) and runs to completion. Its advantage is that it is provably optimal from turnaround/waiting point of view. The disadvantages of SJF are that in general, it cannot be implemented, also starvation is possible, Can do it approximately: use exponential averaging to predict length of next CPU burst.

2.3.3 RR - Round Robin: It is the pre-emptive version of FCFS, treat ready queue as circular, arriving jobs are placed at the end, dispatcher selects first job in queue and runs until completion of CPU burst, or until time quantum expires if quantum expires, job is again placed at end. The advantages of Round Robin are that it is simple, low overhead, works for interactive systems and has the following disadvantages if quantum is too small, there will be too much time wasted in context switching and if too large (i.e., longer than mean CPU burst), it approaches FCFS.

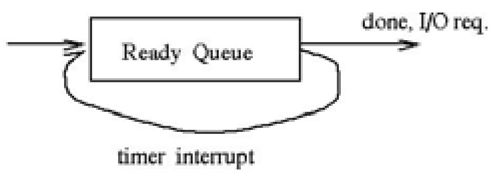


Fig. 2: Job execution in Round Robin

2.3.4 Priority Based Scheduling: Run highest-priority processes first, use round-robin among processes of equal priority. Re-insert process in run queue behind all processes of greater or equal priority. Allows CPU to be given preferentially to important processes. Scheduler adjusts dispatcher priorities to achieve the desired overall priorities for the processes, e.g. one process gets 90% of the CPU. The disadvantage of the Priority Based Scheduling is that it may cause low-priority processes to starve.

2.4 Algorithm Evaluation

How do we select a CPU Scheduling algorithm for a particular system? Since we have different scheduling algorithm with its own parameter selection can be difficult. To select an algorithm we must first define the relative importance of CPU Scheduling criteria (which have been discussed above). Next we use an evaluation method. The various evaluation methods for evaluating CPU Scheduling algorithms are discussed below:

2.4.1 Deterministic Modelling: one major class of evaluation methods is analytic evaluation. Analytic evaluation uses the given algorithm and the system workload to produce a formula or number that evaluates the performance of the algorithm for that workload. One type of analytic evaluation is deterministic modelling. This method takes a particular predetermined workload and defines the performance of each algorithm for that workload.

2.4.2 Queueing Models: On many systems, the processes that are run vary from day to day, so there is no static set of processes (or times) to use for deterministic modelling. What can be determined; however, is the distribution of CPU and I/O bursts. These distributions can be measured and then approximated or simply estimated. The result is a mathematical formula describing the probability of a particular CPU burst. Commonly, this distribution is exponential is described by its mean. Similarly, we can describe the distribution of times when processes arrive in the system (the arrival-time distribution). From these two distributions, it is possible to compute the average throughput, utilization, waiting time, and so on for most algorithms. The computer system is described as a network of servers. Each server has a queue of waiting processes. The CPU is a server with its ready queue, as is the I/O system with its device queues. Knowing arrival rates and service rates, we can compute utilization, average queue length, average wait, and so on. This area is called queueing-network analysis.

2.4.3 Simulations: This is the imitation of the operation of a real-world process or system over time. Whether done by hand or on a computer, simulation involves the generation of an artificial history of a system and the observation of that artificial history to draw inferences concerning the operating characteristics of the real system (Bank et al 2005). To get a more accurate evaluation of scheduling algorithms, we can use simulations (Silberschatz et al 2002).

2.4.4 Implementation: This approach puts the actual algorithm in the real system for evaluation under real operating conditions. The major difficult y with this approach is the high cost. The expense is incurred not only in coding the algorithm and modifying the operating system to support it (along with its required data structures) but also in the reaction of the users to a constantly changing operating system. Another difficultly is the environment in which the algorithm is used will change. The environment will change not only in the usual way, as new programs are written and the types of problems change, but also as a result of the performance of the scheduler. If short processed are given priority, then users may break larger processes are given priority, then users may break larger processes into sets of smaller processes. If interactive processes are given priority over non interactive processes, then users may switch to interactive use.

2.5 Exploiting the Simulation Approach:

In this paper to get a more accurate evaluation of scheduling algorithms, we decided to use simulations. Running simulations involves programming a model of the computer system. Software data structures represent the major components of the system. The simulator has a variable representing a clock; as this variable value is increased, the simulator modifies the system state to reflect the activities of the devices, the processes and the scheduler. As the simulation executes, statistics that indicate algorithm performance are gathered and printed.

The data to drive the simulation can be generated in several ways. The most common method uses a random-number generator, which is programmed to generate processes; CPU burst times, arrivals, departures and so on, according to probability distributions. The distributions can be defined mathematically (uniform, exponential, poisson) or empirically (Silberschatz et al 2002). If a distribution is to be defined empirically, measurements of the actual system under study are taken. The results define the distribution of events in the real system; this distribution can then be used to drive the simulation.

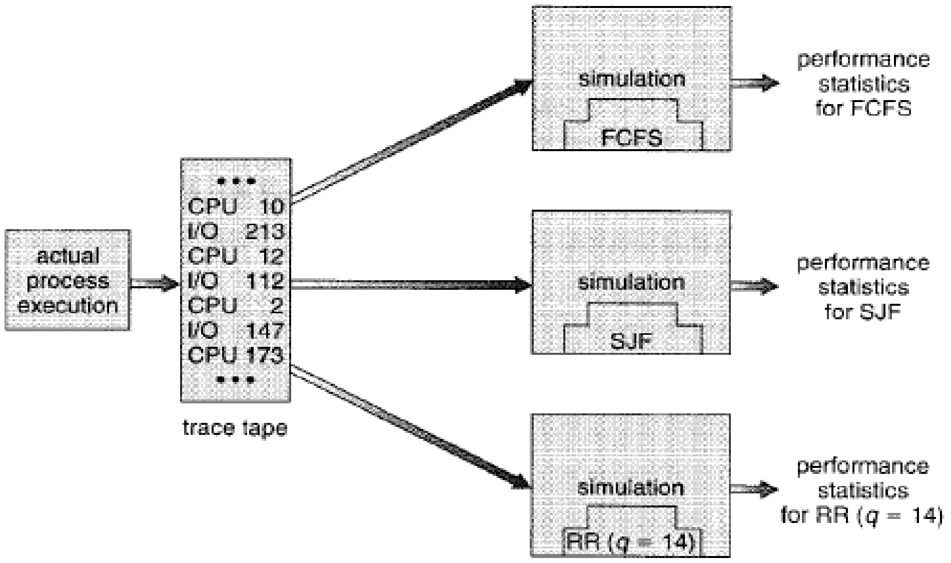


Fig. 3: Evaluation of CPU Schedulers by Simulation method.

3.0 Using Exponential Distribution to Simulate the CPU Scheduling Algorithms

In this paper we decided to use the Simulation approach of algorithm evaluation to simulate CPU scheduling algorithms, since this approach gives more accurate evaluation of scheduling algorithms compared to the others discussed above. The data to drive the simulation is generated using a random-number generator which is programmed to generate processes, CPU burst times and arrival times, according to the exponential probability distributions.

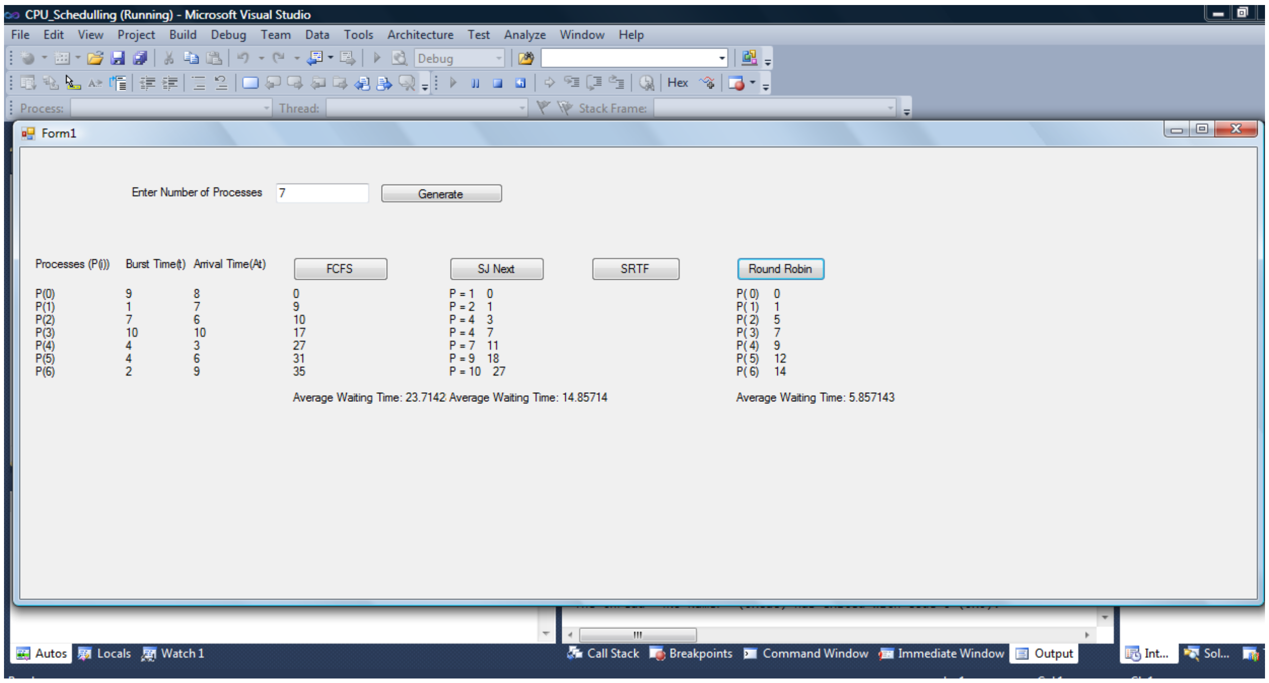
The objective of this paper is to use exponential distribution function to generate the arrival, and assumed burst time. Compute waiting time and average waiting of each algorithm (FCFS, SJN, and RR with quantum of 2). Compare the results of each of the algorithms.

The job execution times are assumed to be drawn from a common distribution using exponential realistic for execution times. An exponentially-distributed random variable with parameter > 0 was used. The algorithm that generated the random number is shown below as it was been executed in Microsoft Visual Basic 2010.

```
1.  Public Function rndom() As Integer
2.  Dim Lambda As Integer = 1
3.  Dim seed As Single = 0
4.  Dim X As Integer = 0
5.  Randomize()
6.  seed = Rnd()
7.  X = seed * Lambda
8.  rndom = Int(1 - Exp(-Lambda * X) * 10)
9.  Debug.Print(Exp(-Lambda * X))
10. Debug.Print(Exp(1))
11. End Function
```

3.1 Implementation of the CPU Scheduling using Ms Visual Basic 2011.

In order to evaluate the CPU Scheduling Algorithms – FCFS, SJF and Round Robin, we used Ms Visual Basic 2010 to create the User interface and the codes. Once the number of processes is entered the burst, arrival times waiting times and average waiting times are generated by the random number generator for each of the following CPU Scheduling Algorithms (FCFS, SJF and Round Robin) respectively. By comparing the average waiting times of the Scheduling Algorithms we can tell from this that the Round Robin Scheduling Algorithms has least average waiting time; this is illustrated in figure 4 below:


CPU_Schedulling (Running) - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze Window Help
Debug
Hex
Process: Thread: Stack Frame:
Form1
Generate
FCFS
SJ Next
SRTF
Round Robin
Autos Locals Watch 1
Call Stack Breakpoints Command Window Immediate Window Output
Int... Sol...

```
Enter Number of Processes  7

Processes (P(i))  Burst Time(t)  Arrival Time(At)

P(0)   9    8     0       P = 1   0      P( 0)  0
P(1)   1    7     9       P = 2   1      P( 1)  1
P(2)   7    6     10      P = 4   3      P( 2)  5
P(3)   10   10    17      P = 4   7      P( 3)  7
P(4)   4    3     27      P = 7   11     P( 4)  9
P(5)   4    6     31      P = 9   18     P( 5)  12
P(6)   2    9     35      P = 10  27     P( 6)  14

Average Waiting Time: 23.7142 Average Waiting Time: 14.85714     Average Waiting Time: 5.857143
```

Fig 4: A diagram showing the Implementation of the CPU Scheduling using Ms Visual Basic 2010.

4.0 Conclusion

After running and comparing the waiting times, and average waiting time of each scheduling algorithm (FCFS, SJF and RR) using exponential distribution, we noticed that RR resulted in a minimal average waiting time, though we encountered some difficulties in generating index numbers for SJF.
Finally, simulations can be expensive, often requiring a lot of hours. Simulation approach provides more accurate results in evaluating the Scheduling Algorithms.

5.0 References

A Silberschatz, B. P. Galvin & G. Gagne, (2002). Operating system concepts, seventh Edition.

H. Casanova, "Simgrid: a Toolkit for the Simulation of Application Scheduling", 3rd IEEE/ACM International

H. Y. Low, "Survey of Languages and Runtime Libraries for Parallel Discrete-Event Simulation", IEEE

J. Banks, J. S. Carson J.S., Nelson B. L., Nicol D.M., (2005). Discrete-Event System Simulation Fourth Edition.

Kelliher T. P. CI 318 1998

L. F. Góes, C. A. Martins: Proposal and Development of a Reconfigurable Parallel Job Scheduling

R. Buyya, and M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing", The Journal of Concurrency and Computation: Practice and Experience, Volume 14, Issue 13-15, Pages: 1175-1220, Wiley Press, USA, November - December 2002. Symposium on Cluster Computing and the Grid, Los Angeles, 2001

R. S. Murray, J.J. Schiller, R. A Srinivasan, (2009) Probability and Statistics, third edition.

Microsoft Visual Basic 2010 edition.

First Author Abur, Maria Mngohol is a Software programmer working with Iya Abubakar Computer Centre, Ahmadu Bello University, Nigeria. She obtained a BSc. Degree in Computer Science at the University of Abuja, Nigeria. Currently, she is undergoing a master's degree program in Computer Science at ABU, Nigeria, She has written many papers among them is the paper titled "Adaptation of Semantic Web to Rural Healthcare delivery". She contributed on the paper "Process and Database Modelling of a University Bursary System-A Perspective of Cash Office, Volume 8, Issue 4, July 2011". For more information about

Abur Mngohol Maria and papers she has written check this site http://www.abu.edu.ng/iacc/abur also she is a member of a professional body, the Nigerian Computer Society.

Second Author Mohammed Aminu Umar is the Regional IT Coordinator, Mainstreet Bank Limited. Kaduna Regional office Plot 1472, Mogadishu layout, Ahmadu Bello Way, Kaduna. He obtained a BSc. Degree in Computer Science at the University of Abuja, Nigeria and currently, he is undergoing a master's degree program in Computer Science at ABU, Nigeria.

Third Author Sani Dajuma is working with Sa'datu Rimi College of education, Kumbosto Kano state Nigeria. He did his first degree in computer science and currently, he is undergoing a master's degree program in Computer Science at ABU, Nigeria.

Fourth Author Saleh Abdullahi is the Ag. MD/Chief Executive Officer, of Mtel; he did his first, second and doctorate degrees in computer science. He is a visiting Senior lecturer at the Ahmadu Bello University, Nigeria.