MySIM: A Light-Weight Tool for the Simulation of Probabilistic CPU Process Scheduling Algorithm

Fatai Adesina Anifowose, *Member, IACSIT*

***Abstract*—Students taking their first course in the Principles of Operating Systems usually have textbooks as their only resource material. In order to assist them to better understand the basic concepts of process scheduling algorithms as well as their respective advantages and disadvantages, there is the need for a tool to show experimentally the behaviour of these algorithms through comparative performance studies and analyses. A lot of commercial and educational software programs are available but each has its drawbacks ranging from high cost of acquisition for the commercial systems and non-flexibility for the educational systems. Others find their place between these two extremes such as non-availability and inefficiency. This paper presents the implementation of a lightweight, simple, robust and flexible tool for the comparative and experimental study of two existing as well as an innovative probabilistic CPU process scheduling algorithm, using Average Waiting Time (AWT) and Average Turn-around Time (ATT) as the criteria for performance evaluation. The tool was used to simulate the three algorithms using eight datasets representing different scenarios of processes with their burst times and respective locations on a virtual queue. The results showed the robustness of this software, especially for academic and experimental use, as well as proving the desirability and efficiency of the probabilistic algorithm over the other existing techniques.**

***Index Terms*—Process scheduling algorithms, probabilistic algorithm, average waiting time; average turn-around time.**

I. Introduction

Simulation programs are usually needed to mimic and study the behaviour of process scheduling algorithms. There are a number of such algorithms with each having its respective advantages and drawbacks. In order to determine the comparative and competitive advantages and disadvantages of these algorithms, they need to be simulated and their performance indices studied and used for better understanding of operating system principles especially by entry-level students and for possible choice of implementation in real-life operating environments. Some of these algorithms would show promising results in terms of ease of implementation but perform poorly in terms of starvation and vice versa. Comparative studies need to be carried out on them in order to determine their average performance while noting the costs and benefits of implementing each of them using simple evaluation criteria.

Some of the available simulators are too rigid and cumbersome to use. They often prescribe certain ways for users to present data. Any deviation from such prescribed formats would result in the program not working properly. Most of them are only available online. PCs would need to be connected to the internet before they can be used. Because of the internet connectivity requirement, some of them are very slow as they require and consume a lot of bandwidth. Majority of them are programmed in Java and hence would require that Java Runtime Environment be downloaded, installed and updated before such programs can be used. Those that are available as packaged programs that would not need any special requirement and updates are not cost-effective to use, especially in academic and other non-commercial environments.

In order to overcome these bottlenecks, this paper highlights the development of a small, light-weight and efficient software that would be free of such overheads as internet connectivity; and offer the required flexibility and freedom of choices for the user to prepare and present data to the program for smooth running of the algorithms. This paper presents MySIM, a tiny and light-weight simulation software that can be used to study the behavior of two commonly studied process scheduling algorithms: First-Come, First-Served (FCFS) and Shortest Job First (SJF). It also presents the promising behaviour of a probabilistic algorithm. It presents, in addition to the afore-mentioned, a simple implemented proof and confirmation of the efficiency of probabilistic algorithms. The outputs of the software have created a good premise to recommend this technique for implementation in real-life systems in addition or in replacement of their present deterministic techniques.

The rest of this paper is organized as follows: Section 2 presents an overview of some of the simulators that are available and their respective drawbacks. A brief overview and drawbacks of some of the existing process scheduling algorithms are discussed in Section 3. A detailed overview of probabilistic algorithm and the mathematical proof of its efficiency are presented in Section 4. Section 5 describes the datasets, design issues, mode of operation, and the details of implementation of this software on the algorithms. Results showing the comparative performance of the three algorithms are presented and discussed in Section 6 while conclusions are drawn from the study in Section 7.

II. Existing Simulators

A good number of simulators exist in literature. Reference [1] is a java-based web application that implements FCFS, SJF, Priority SJF and Round Robin. Each input in the system is characterized by its arrival time, CPU burst and I/O bursts. It claims to be very efficient but a sample run disclosed that it

Manuscript received September 7, 2011; revised November 23, 2011.

F. Anifowose was with the Information and Computer Science department when this experiment was carried out. He is a Research Engineer at the Center for Petroleum and Minerals, King Fahd University of Petroleum and Minerals, Saudi Arabia (e-mail: anifowo@kfupm.edu.sa).

is very slow, since it requires a high-speed internet connection to load the applet, and also requires that Java software be either installed or updated.

Another simulator called CPU Scheduling Simulator (CPUSS) was presented in [2]. CPUSS is a framework that allows users to quickly and easily design and gather metrics for custom CPU scheduling strategies including FCFS, Round Robin, SJF, Priority First, and SJF with Priority Elevation rule. It proves to be very robust and comprehensive software with capabilities to gather metrics such as Average Wait Times, Idle CPU Time, Busy CPU Time, Wait Time Mean, Wait Time Standard Deviation, Mean Response Time, Response Time Standard Deviation, Turnaround Time Mean, Turnaround Time Standard Deviation, etc. The long list of the capabilities it can handle makes it too complex and complicated for simple academic demonstrations and use by non-computer geeks such as fresh students that are just taking their first course in Computer Science. Above all, it runs in the windows-DOS environment which is characterized by unattractive user interface and hence, lacks user-friendliness.

A project that is very close to our work is a simulator presented by [3]. However, this simulator was designed for a software project scheduling rather than CPU process scheduling, hence not relevant for our consideration in this study.

MOSS, Modern Operating Systems Simulators, was found in [4]. It is a collection of Java-based simulation programs which illustrate key operating system concepts presented in a textbook by Tanenbaum (2001) for university students using the text. Though a very robust application, but since it is targeted at the users of a textbook for a university course, it thus requires that all users must have taken a full course in Operating Systems using exactly the same textbook for which it was designed. This does not fit in to independent software that can be used freely without any such constraint.

The best simulator we could find, so far, during our survey of previous related work was presented by [5]. It was developed in Visual Basic 6.0 and implemented the Round Robin as a non-preemptive scheduling algorithm. It uses Average Completion Time (ACT) and Average Turn-around Times (ATT) as the criteria for performance evaluation. However, it is not as robust as ours in the sense that we implemented a probabilistic algorithm in addition to FCFS and SJF algorithms. Our major objective is to introduce the new probabilistic algorithm while using its excellent comparison with FCFS and SJF as a proof of the new algorithm's efficiency and desirability for academic demonstrations and possible implementation in real-life systems.

A recent effort by [6] focused on the development of a simulator to implement a comparative study between three distributed process scheduling algorithms; sender-initiated, receiver-initiated and hybrid sender-receiver-initiated algorithms. This is a clear deviation from our objective of presenting the capabilities of a probabilistic scheduling algorithm and commonly-studied algorithms.

Other related works are limited to small functions and modules for very small classroom demonstrations. Hence, this software is unique in being the first to implement a probabilistic algorithm in addition to two other CPU process scheduling algorithms based on our comprehensive survey of literature. The non-preemptive CPU process scheduling algorithms discussed in this paper are limited to First-Come First-Served (FCFS), Shortest Job First (SJF), and Priority-based algorithms.

III. Conventional Process Scheduling Algorithms

A. First-Come, First-Served

In First-Come First-Served (FCFS), the process that arrives first is allocated the CPU first while the ready queue is based on a First In, First Out (FIFO) policy. Though, it is simple and easy to implement, but since scheduling is done at constant time independent of the number of processes in the ready queue, it is often associated with a long Average Waiting, Response and Turnaround Times, especially if processes with long burst times are the first to come on the queue. In this case, processes with short burst times at the back of the queue will have to wait until it is their turn [7, 8].

B. Shortest Job First

In Shortest Job First (SJF), the next process to be scheduled is selected based on the shortest burst time, which is linear with respect to the number of processes in the ready queue. Though, it gives the minimum Average Waiting Time for a given set of processes but selection is more complex as it is accompanied by more overhead than in FCFS with the need to estimate the duration of a job (the burst time) prior to its scheduling. This requires the need for predictive algorithms, which will further increase the complexity of implementation. Also, starvation is possible, since if new and short processes keep on arriving, old and long processes may never be served [7], [8]. The bias here is towards processes with the least burst times.

C. Priority-Based

In priority-based CPU process scheduling algorithm, a priority number is associated with each process. The CPU is allocated to the process with the highest priority. In the non-preemptive version of Priority-based scheduling policy, the CPU is allocated to the process with the highest priority and equal priority processes are scheduled in FCFS order. Also, SJF is a special type of priority-based scheduling where priority is the shortest of the next predicted CPU burst time. This policy is associated with the problem of starvation as low priority processes may never be executed or will be executed last. The bias here is the priority given to some processes while others are denied. It is interesting to note that priority-based policy boils down eventually to FCFS [7, 8].

IV. Overview of Probabilistic Algorithms

Some of the existing CPU process scheduling algorithms such as FCFS, SJF, Shortest Remaining Time First (SRTF), Round Robin (RR), and Priority-based, have certain drawbacks associated with them. These drawbacks include starvation (very common in SJF and Priority-based scheduling algorithms), complexity of implementation such as in SJF, long waiting time and high average turn-around time. Fairness, efficiency, high throughput, low turnaround time, low waiting time, low response time, low frequency of context switches, and simplicity of the scheduling algorithm

are the goals of any efficient CPU process scheduling algorithm.

Randomness has been known for its fairness, on the average and faster speed [9]. A process among a set of processes in the ready queue will be selected and dispatched to the CPU by random sampling while assuming that the default quantum time is optimal. Previous studies [9]-[13] have suggested that probabilistic scheduling algorithms give fair selection of processes without any of them having the tendency to be starved. They are characterized by improved average throughput and minimum average turn-around and response times.

Some of commercial operating systems, such as PicOS [14], are based on event-driven process scheduling, only when there are events and processes awaiting such events. At a point, when there is no event being awaited or when there is no process awaiting any event, scheduling becomes FCFS which is already known to be associated with a long ATT and AWT. Application of this algorithm in such a case will offer better performance.

A probabilistic algorithm employs a degree of randomness as part of its logic. This means that the machine implementing the algorithm has access to a random number generator. The algorithm typically uses the random bits as an auxiliary input to guide its behavior, in the hope of achieving good performance in the "average case". If there are 10 processes, a number will be generated between 1 and 10 inclusive. If that number has not been selected before, the process at the location corresponding to that number is selected. Otherwise, it will be ignored and another one will be generated while excluding the previously selected number. Formally, the algorithm's performance will be a random variable determined by the random bits, with good expected value. The "worst case" is typically so unlikely to occur that it can safely be ignored [15].

Hence, a probabilistic algorithm can be defined as one that receives, in addition to its input, a stream of random bits that it can use in the course of its action for the purpose of making random choices. It may give different results when applied to the same input in different runs. It is recognized now that, in a wide range of applications, randomization is an extremely important tool for the construction of algorithms. Probabilistic algorithms are usually characterized by smaller execution time or space requirement than that of the best deterministic algorithm for the same problem and are often simple and easy to implement [11].

A. Review of Previous Work

In literature, [11] proved lower bounds on the competitive ratio of probabilistic algorithms for several on-line scheduling problems. Their results showed that a probabilistic algorithm is the best possible and most applicable for the problem. Reference [9] proposed a suite of probabilistic algorithms for input-queued (IQ) switches for finding a good matching between inputs and outputs to transfer packets at high line rates or in large switches, which is usually a complicated task. They reported that the performance of the probabilistic algorithms is comparable to that of well-known, effective matching algorithms, yet are simple to implement.

Reference [10] provided a methodology for analyzing the impact of perturbations on performance for a generic Lebesgue-measurable computation. They concluded that the associated robustness problem, whose solution is computationally intractable, could be nicely addressed with a polynomial-time procedure based on probabilistic algorithms. In order to satisfy given real-time constraints, maximize reliability and minimize inter-communication costs, [12] proposed and developed an objective function approach to schedule real-time task graphs that need to satisfy multiple criteria simultaneously. They used two different searching techniques, a heuristic and a random search technique. They concluded that the random search technique, which is probabilistic in nature, achieves better performance.

This work is aimed at introducing this probabilistic algorithm to students of operating system principles as well as its promising performance as compared with other common algorithms.

B. Mathematical Proof of Probabilistic Algorithms

Computational complexity theory models randomized algorithms as probabilistic Turing Machines. Both Las Vegas and Monte Carlo algorithms are considered, and several complexity classes are studied. The most basic randomized complexity class is RP, which is the class of decision problems for which there is an efficient (polynomial time) randomized algorithm or probabilistic Turing Machine which recognizes NO-instances with absolute certainty and recognizes YES-instances with a probability of at least 1/2.

Indicator random variables are useful for analyzing situations in which repeated random trials are performed as in the case of selecting a process to schedule at random. We can let X_i be the indicator random variable associated with the event in which the i^{th} process is selected. Letting Y_i be the random variable denoting the outcome of the i^{th} selection, we have that:

$$X_i = I\{Yi = S\}$$

Let X be the random variable denoting the total number of selected processes in the n-long ready queue, so that

$$X = \sum_{i=1}^{n} X_i$$

Attempting to avoid too much mathematical details, it has been established in [6] that the expected cost of a typical randomized algorithm is:

$$O(\ln n)$$

where n is the number of processes [16].

It has also been established by [16] that in a randomized algorithm, each variable has an equal chance of being selected. This is the most important feature of a randomized algorithm that makes it most desirable for scheduling tasks where each process will have an equal chance of being selected instead of the biasness associated with assigning priorities, using burst-times or a FIFO queue.

V. Design, Implementation and Data Description

A. Software Design

The simulator was designed and developed using the Microsoft Visual Basic 6.0 Professional Edition's Integrated

Developed Environment (IDE). The input data were created as an ASCII file and arranged as a 1-column vector from where they are read using the Open menu command. There is no imposed naming policy for these input files. Any name is allowed since the user is free to select the necessary file from an Open dialog box from any convenient location whether on a local disk, an external or a network drive.

Based on the selected input data, the FCFS, SJF and the probabilistic algorithms were computed and written to another ASCII file that have been automatically created at runtime for the purpose. The output file, named output.txt, is created if it does not exist or appended to, otherwise. The result of each algorithm is also displayed on a window for the user to view. A summary of the probabilistic algorithm is shown in the flowchart in Fig. 1.

The MySIM software was designed as a simple, light-weight system for academic purpose for the simulation of the behaviour of FCFS, SJF and probabilistic CPU process scheduling algorithms. Fig. 2 to 7 shows the main screen and some of the windows of the software including the output window and the File-Open Dialog box.

The use of ASCII files for input and output in addition to the choice of Microsoft Visual Basic for this work made this simulator really light-weight. This quality is further strengthened with the fact that the entire software does not exceed 5MB in size.

The user interfaces are simple, concise, unambiguous and easy to use but replete with only the relevant information.

The input and output files are re-useable: they can be deleted and new ones created. The output file is created automatically by the simulator at runtime so no harm will be caused by deleting an existing copy of it. The innovative probabilistic algorithm is well implemented and its mode of operation was clearly shown and presented in the simulator.

A university student who is just taking his/her first course in Operating Systems can easily use this simulator to understand the concepts of Probabilistic Algorithm in addition to the FCFS and the SJF algorithms implemented in the simulator. All these account for the robustness and flexibility of this tool.

B. Implementation

The software was implemented to simulate the operations of FCFS, SJF and probabilistic CPU process scheduling algorithms. These algorithms were implemented in order to establish a valid premise for effective comparison. The simulator takes process IDs as integer input, estimated burst times and their respective positions in terms of their order like in a virtual queue. Each of the datasets was used for the three scheduling algorithms in order to observe the behaviour of each algorithm under the same data conditions.

For simplicity, the simulator was built on two major assumptions:

- The scheduling policies are non-preemptive and
- The processes arrive at the same time.

The simulator was run on eight different datasets containing processes with different burst times that have been positioned in different ways in order to create different process arrival scenarios. This was done to determine, as part of the experiment, whether the location of a process in a queue will affect the results of the simulation, especially, that of the probabilistic algorithm.

The simulation was run several times to ensure fairness to all datasets and the averaged final results were recorded and presented for each algorithm using Average Waiting and Turn-around Times (AWT and ATT) as the performance evaluation indices.

C. Description of Data

Tables I to VIII show the datasets representing processes that are identified by their IDs, with their estimated burst times as input to the simulator and their respective locations representing the arrival queue.

The different arrangement of the jobs was meant to represent the different real-world scenarios jobs can take with different estimated burst times and arrival on the waiting queue.

The number of processes can be extended to any length as desired. For demonstration purpose, a maximum of 10 jobs was implemented and reported in this paper. It was tested with 15 and 20 jobs during testing. However, the maximum attainable number of jobs was not determined and it could not be stated that the number of jobs could go to infinity.

D. Experimental Computing Environment

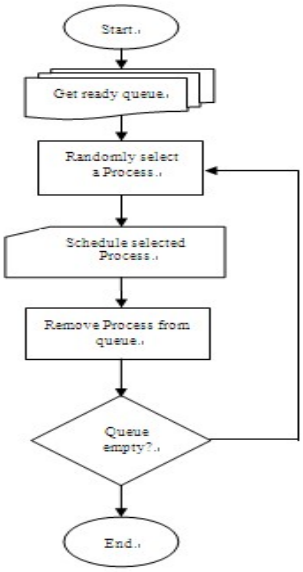

Fig. 1. Flowchart of the probabilistic algorithm.

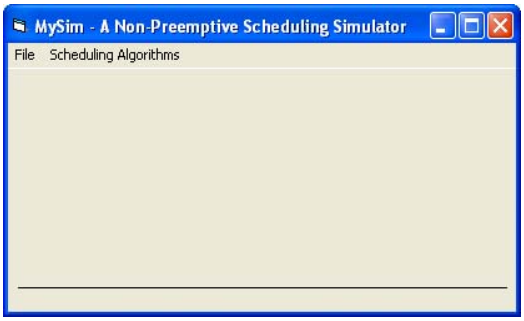

Fig. 2. The main menu of the mysim simulation software.

The computing environment used for this simulation study consists of Microsoft Visual Basic 6.0 Professional Edition that runs on a laboratory Personal Computer with the Service Park 2 update of Windows XP Professional Edition version 2002. The processor is based on Intel Pentium M technology with a speed of 1.8 GHz and a RAM size of 512 MB. Hence,

the AWT and ATT used as criteria for performance evaluation are based on this system configuration. Therefore, the AWT and ATT are expected to be smaller when the simulator is run on a Personal Computer with a higher configuration in terms of processing speed and RAM size.

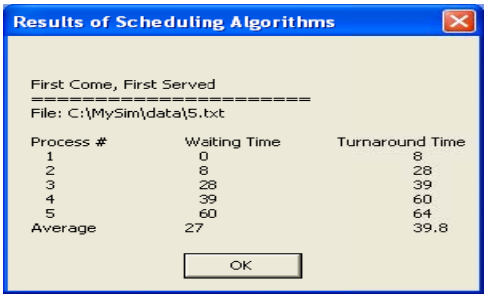

Fig. 3. The result window for fcfs algorithm.

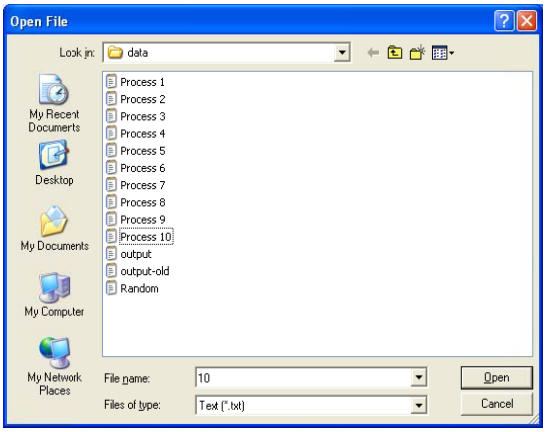

Fig. 4. The open file dialog box of MYSIM.

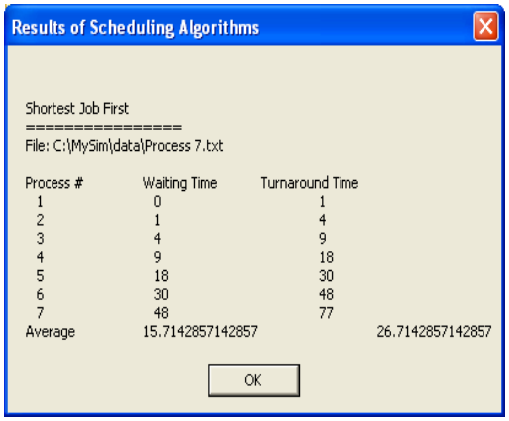

Fig. 5. The result window for SJF algorithm.

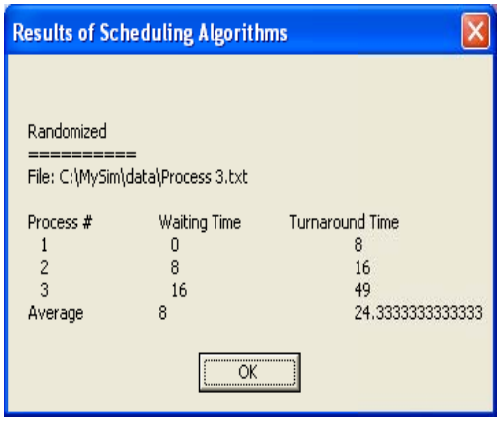

Fig. 6. The result window for probabilistic algorithm.

Fig. 7. The main screen showing the algorithms menu.

TABLE I: DATA 1 WITH 3 PROCESSES.

P1	P2	P3
25	33	2

TABLE II: DATA 2 WITH 3 PROCESSES.

P1	P2	P3
33	8	8

TABLE III: DATA 3 WITH 3 PROCESSES.

P1	P2	P3
8	8	33

TABLE IV: DATA 4 WITH 3 PROCESSES.

P1	P2	P3
8	33	8

TABLE V: DATA 5 WITH 5 PROCESSES.

P1	P2	P3	P4	P5
8	20	11	21	4

TABLE VI: DATA 6 WITH 5 PROCESSES.

P1	P2	P3	P4	P5
15	5	25	2	11

TABLE VII: DATA 7 WITH 7 PROCESSES.

P1	P2	P3	P4	P5	P6	P7
18	29	3	5	9	11	12

TABLE VIII: DATA 8 WITH 10 PROCESSES.

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
8	29	3	5	9	1	12	20	7	18

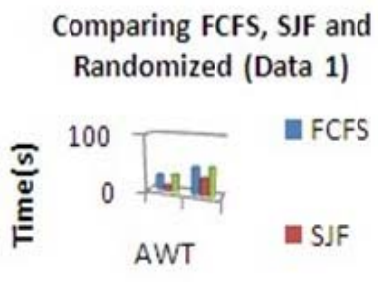

Fig. 8. Waiting and turnaround times for data 1

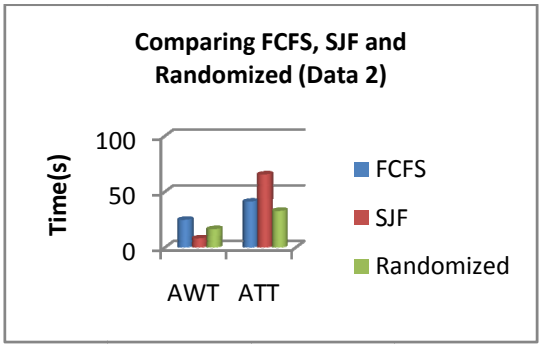

Fig. 9. Waiting and turnaround times for data 2

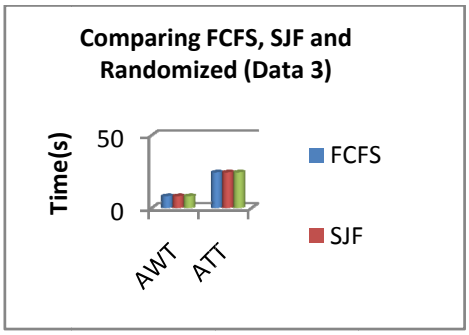

Fig. 10. Waiting and turnaround times for data 3

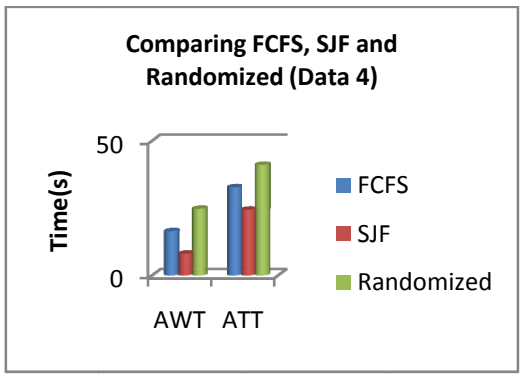

Fig. 11. Waiting and turnaround times for data 4.

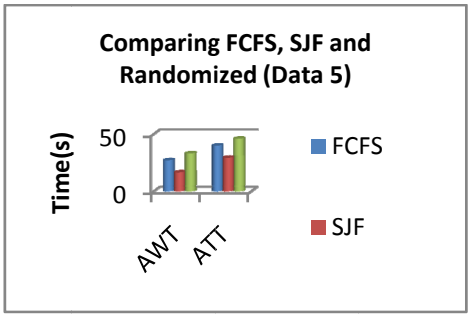

Fig. 12. Waiting and turnaround times for data 5.

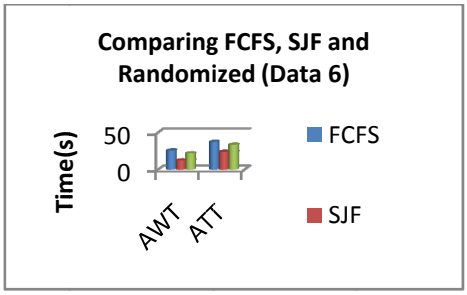

Fig. 13. Waiting and turnaround times for data 6

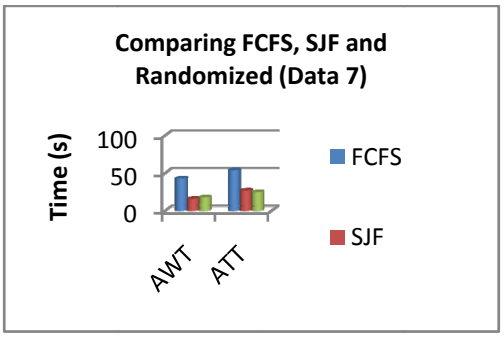

Fig. 14. Waiting and turnaround times for data 7

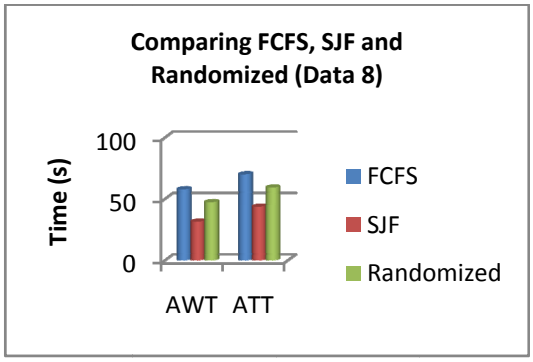

Fig. 15. Waiting and turnaround times for data 8.

VI. Results and Discussion

The results of the simulation run on the eight datasets, with their corresponding waiting times, turnaround times, as well as their respective averages are displayed in Fig. 8 to 15. Out of the eight different cases, probabilistic scheduling algorithm performed better than or equal to FCFS in five cases in terms of AWT and ATT. The other cases where FCFS performed better than probabilistic algorithm were with very little margin. This is the reason for describing the good performance of probabilistic algorithms as "average-case". It can be inferred from the results that the probabilistic algorithm, on the average, performs better than FCFS (except with datasets 1, 4 and 5 in Fig. 8, 11 and 12) while SJF remains the best. Even though SJF gave the optimal average waiting and turnaround times, that does not make it attractive and appealing for implementation in real-life due to the starvation problems associated with it and its complexity and difficulty of implementation. The real difficulty with the SJF algorithm is in estimating the length of the next CPU request. Consequently, despite the optimality of the SJF algorithm, it cannot be implemented at the level of short-term CPU scheduling. Also, there is the problem of starvation, since in real life, long processes will be starved and may never be executed, if short jobs keep coming. In some cases, short jobs could be more important than longer ones.

From the observations of this study, it could not be determined exactly the nature of conditions of processes that MySIM will achieve the best outcome. This is because, since the algorithm is probabilistic, the next processes are chosen randomly and can only be predicted with certain degree of probability.

This is the beauty of this algorithm as it operates on

fairness rather than being influenced by certain user's actions. However, further studies will be carried in order to determine the possibility of certain conditions of processes that could influence the performance of this innovative algorithm.

VII. Conclusion

This paper has presented a tiny, light-weight and custom-built CPU process scheduling simulation software, comparing the efficiency of a probabilistic scheduling algorithms in terms of Average Waiting and Turnaround Times with FCFS and SJF algorithms. Probabilistic algorithm has proven, on the average, to be very fair to the process to be selected from a ready queue, and fast in terms of execution time. Each process, having equal chances, is scheduled by random sampling from among waiting processes in the ready queue. From the output of the software, FCFS has been shown to be characterized by long average waiting and turnaround times while SJF, though desirable in terms of lowest AWT and ATT but still have the tendency of starvation.

This paper, along with the software, will serve to assist students of operating system basic principles to gain better understanding of the basic concepts of these algorithms and especially to appreciate the efficiency of probabilistic algorithms.

The limitation of this software is that it is meant for non-preemptive processes, hence not suitable for real-time applications. Real-time applications give no room for a queue as processes are handled as soon as they arrive. The probabilistic algorithm is also affected by this limitation.

There is the plan in the future to expand the software to include other common scheduling algorithms such as Round Robin and Shortest Remaining Time First (SRTF), for the purpose of comparative studies. A hybrid of probabilistic and round robin algorithms, to be named Prob-RoundRobin (PRR) is also in the plan.

This software will be freely available to the academic community as free and open-source software for academic and class demonstration purposes.

Acknowledgment

We would like to thank King Fahd University of Petroleum and Minerals (KFUPM), the Research Institute, and the College of Computer and Systems Engineering for providing the computing facilities and a conducive research atmosphere.

References

[1] http://vip.cs.utsa.edu/classes/cs3733s2004/notes/ps/runps.html(Accessed June 11, 2010).
[2] http://www.codeplex.com/cpuss (Accessed June 11, 2010).
[3] F. Padberg, A Software Process Scheduling Simulator, in *Proc. of the 25th IEEE International Conference on Software Engineering (ICSE'03)*, 0270-5257/03, 2003.
[4] http://www.ontko.com/moss/ (Accessed June 11, 2010).
[5] D. A. Cardella, A Simulator of Operating System Job Scheduling, Visual *Basic 6*. Available for download at http://www.freevbcode.com/ShowCode.asp?ID=4079,2002 (Accessed June 11, 2010).
[6] S. H. Nazleeni, H. M. A. Anang, M. H. Hasan, A. A. Izzatdin, and M. W. Wirdhayu, "Time comparative simulator for distributed process scheduling algorithms," *World Academy of Science, Engineering and Technology 19*, pp. 84-89, 2006.
[7] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, John Wiley & Sons Inc, 3rd Edition, pp. 158-164, 2005.
[8] A. S. Tanenbaum, *Modern Operating Systems*, 2nd Edition, ISBN-10-0130313580, Prentice Hall, Netherlands, pp. 976, 2001.
[9] D. Shah, P. Giaccone, and B. Prabhakar, "An efficient probabilistic algorithm for input-queued switch scheduling," *IEEE Transactions on Microprocessors*, Vol. 22 (1), pp. 19-25, 2002.
[10] C. Alippi, "probabilistic algorithms: a system-level, poly-time analysis of robust computation," *IEEE Transactions on Computers*, Vol. 51, pp. 740 - 749, 2002.
[11] L. Stougie and A. P. A. Vestjens, "Probabilistic algorithms for on-line scheduling problems: how low can't you go?" *Operations Research Letters*, Volume 30, Issue 2, Pp. 89-96, 2002.
[12] A. Amin, R. Ammar, and S. Rajasekaran, "Probabilistic algorithm to schedule real-time task graphs to satisfy a multi-criteria objective function," in *Proc. of the Fourth IEEE International Symposium on Signal Processing and Information Technology*, pp. 381-386, 18-21, 2004.
[13] Y. Jiang and Y. He, "Pre-emptive online algorithms for scheduling with machine cost," *Acta Informatica 41*, pp. 315–340, 2005.
[14] E. Akhmetshina, P. Gburzynski, and F. S. Vizeacoumar, "PicOS: A Tiny Operating System for Extremely Small Embedded Platforms," in *Proc.* of *the International Conference on Embedded Systems and Applications, (ESA '03*), June 23 - 26, Las Vegas, Nevada, USA. CSREA Press 2003, ISBN 1-932415-05-X, pp. 116-122, 2003.
[15] M. H. Alsuwaiyel, "Algorithms Design Techniques and Analysis, *Lecture Notes Series on Computing*, Vol. 7, pp. 371-392, 1999.
[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C.S. Stein, *Introduction to Algorithms*, The MIT Press, Cambridge,Massachusetts London, England, pp. 301-309, 2001.

Fatai Adesina Anifowose is formerly a Systems Programmer at the Computer Centre, Obafemi Awolowo University, Ile-Ife, Nigeria and currently a Research Engineer in the Center for Petroleum and Minerals, Research Institute, King Fahd University of Petroleum and Minerals, Saudi Arabia. He now specializes in the application of Artificial Intelligence in the modeling and characterization of oil and gas reservoir properties. He obtained a Bachelor of Technology (B. Tech.) in Computer Science with special interest in software development; and a Master degree in Information and Computer Science with special interest in Artificial Intelligence. He is a member of the Society of Petroleum Engineers (SPE), type2fuzzylogic.org, UK and the Nigeria Computer Society (NCS). He is on the Editorial Board of the International Association of Computer Science and Information Technology (IACSIT).