A Visual and Interactive Learning Tool for CPU Scheduling Algorithms

Sukanya Suranauwarat

Graduate School of Applied Statistics, National Institute of Development Administration, Bangkok 10240, Thailand

Abstract

CPU scheduling is an important topic in operating systems courses. In this paper, a tool implemented as a Java application and designed as an auxiliary instrument for both classroom teaching and independent study of CPU scheduling algorithms is presented. This tool uses graphical animation to convey the concepts of various CPU scheduling algorithms. The tool is unique in a number of respects. First, it uses a more realistic process execution model that can be configured easily by the user. Second, it graphically depicts each process in terms of what the process is currently doing against time. By using this representation, it becomes much easier to understand what is going on inside the system and why a different set of processes is a candidate for the allocation of the CPU at different times. Third, the tool allows the user to test and increase his understanding of the concepts studied by making his own scheduling decisions and receive immediate feedback on the test problems.

Keywords: *Educational Software, Animation Tool, Computer Science Education, CPU Scheduling Algorithms, Operating System.*

1. Introduction

In the past two decades, a number of visualization and animation tools have been developed and used in many areas of computer science and engineering education [1]-[8]. Experiments carried out with various visualization and animation tools have provided evidence indicating that carefully designed visualizations and animations can have beneficial learning effects. For example, engagement of the learners attention [9]-[11] and the ability to control the pace of the visualization [12] appear to be key factors in building effective visualization and animation tools. Keeping these in mind, the author has developed an interactive Java-based simulator that uses graphical animation to convey the concepts of various CPU scheduling algorithms for a single CPU. CPU scheduling can be defined as the art of determining when and for how long each process runs on the CPU when there are multiple runnable processes. It is central to an operating-system's design and constitutes an important topic in the computer science curriculum.

In addition to providing a visual and animated view as an alternative to a static representation provided by textbooks, the simulator is unique in a number of respects. First, it uses a more realistic process execution model — the execution of a process consists of alternating CPU bursts and I/O bursts, as opposed to a simplified model used in textbooks examples — only one CPU burst per process. Through a graphical user interface of the simulator, the user can configure several sets of processes easily and use them in observing simulations of various CPU scheduling algorithms. By using a more realistic process execution model, users will be able to gain insight into exactly how the algorithms work in real operating systems. Second, the simulator graphically depicts each process' state versus time. The state of a process describes the current activity of that process such as "the process is waiting for an I/O operation to complete" or "the process is currently using the CPU". Various events can cause a process to change states; the simulator shows these events. By using this representation, it becomes much easier to understand what is going on inside the system, why, at any given time, some processes are candidates for the allocation of the CPU and some are not, and why the currently running process can continue using the CPU or why it cannot. Third, the simulator allows the user to practice and test his understanding of the concepts studied by making his own scheduling decisions (i.e., by deciding when and for how long each process runs) through an easy-to-use graphical user interface of the simulator, and receive immediate feedback on the test problems.

The simulator can be used as an auxiliary instrument for both classroom teaching and independent study of CPU scheduling algorithms.

The remainder of this paper is organized as follows: section 2 is a brief overview of the process state and scheduling algorithms used in the simulator, section 3 gives a description of the simulator, section 4 discusses versions and availability of the simulator, section 5 discusses related work, and section 6 draws some conclusions.

2. Overview

A process has three basic states namely running, ready, and waiting. A process is said to be running in the running state if it is currently using the CPU. A process is said to be ready in the ready state if it could use the CPU if it were available. A process is said to be blocked in the waiting state if it is waiting for some event to happen, such as the completion of an I/O operation, before it can proceed. Various events can cause a process to change states. For example, when the currently running process makes an I/O request, it will change from running state to waiting state. When its I/O request completes, an I/O interrupt is generated and then that process will change from waiting state to ready state. For a single CPU system, only one process can run at a time, but several processes may be ready. When more than one process is ready, the operating system must then use a CPU scheduling algorithm to decide which one is to run first and for how long. There are various scheduling algorithms. The simulator uses the algorithms listed below (which are discussed in [13]-[15]).

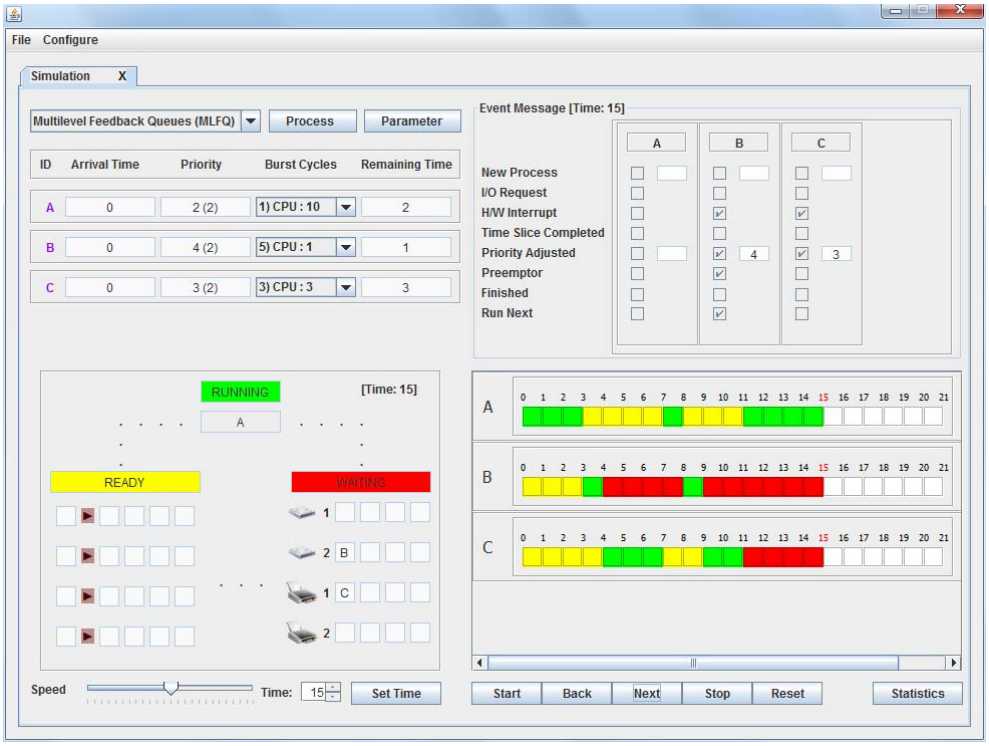
- **First-Come, First-Served (FCFS)**: Processes are assigned the CPU in the order they request it.
- **Round-Robin (RR)**: Each process is given a limited amount of CPU time, called a time slice, to execute. If the required CPU burst of the process is less than or equal to the time slice, it releases the CPU voluntarily. Otherwise, the operating system will preempt the running process after one time slice and put it at the back of the ready queue, then dispatch another process from the ready queue.
- **Shortest-Job-First (SJF)**: When the CPU is available, it is allocated to the process that has the smallest next CPU burst.
- **Shortest-Remaining-Time-First (SRTF)**: When the CPU is available; it is allocated to the process that has the shortest remaining CPU burst. When a process arrives at the ready queue, it may have a shorter remaining CPU burst than the currently running process. Accordingly, the operating system will preempt the currently running process.
- **Priority Scheduling**: There are several ready queues, each with different priority. When the CPU is available, the operating system selects a process from the highest-priority, non-empty ready queue. Within a queue, it uses RR scheduling.
- **Multilevel Feedback Queues (MLFQ)**: This scheduling algorithm is a variant version of priority scheduling algorithm designed to prevent high-priority processes from running indefinitely. Rather than giving a fixed priority to each process like priority scheduling algorithm, MLFQ varies the priority of a process based on its observed behavior. If, for example, a process repeatedly relinquishes the CPU while waiting for input from the keyboard, MLFQ will keep its priority high, as this is how an interactive process might behave. If, instead, a process uses the CPU intensively for long periods of time, MLFQ will reduce its priority. In this way, MLFQ will try to learn about processes as they run, and thus use the history of the process to predict its future behavior.

3. Description of the Simulator

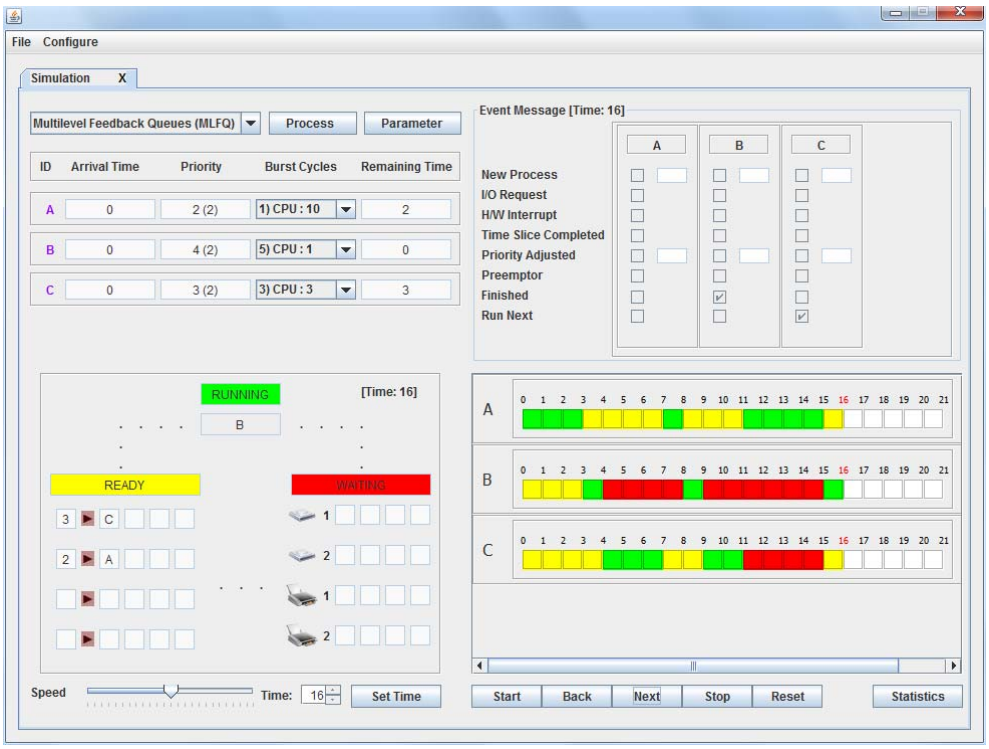
The simulator is written using Java 6 and has two operating modes: simulation and practice modes. In simulation mode, the user can watch the animation of how an algorithm works or trace the algorithm step by step. In practice mode, the user can reinforce concepts studied by making his own scheduling decisions, that is, by deciding when and for how long each process runs. Both modes are described below.

3.1 Simulation Mode

In Fig. 1, two snapshots of the simulator during a simulation in simulation mode are shown. By default, the simulator will start with a simulation-mode tab being opened as shown in Fig. 1(a). A new simulation-mode tab can also be opened by clicking the "File" menu and then clicking "New Simulation". Within a simulation-mode tab, the user can select which algorithm to be animated through a drop-down list box located in the top left section. For each selected algorithm, the predefined set of processes and the predefined scheduling parameters will be loaded so that the user can start watching the animation instantly.



(a) The simulation at time 15.



(b) The simulation at time 16.

Fig. 1 Two snapshots of the simulator during a simulation using the MLFQ algorithm.

The user can modify the predefined set of processes by clicking the "Process" button located next to the drop-down list box of the algorithms, which causes the window shown in Fig. 2 to appear. In the "List of Processes", the information about each process (i.e., its ID, arrival time, and priority) in the predefined set is shown, and can be changed by clicking the "Edit" button after selecting which process' information is to be changed. The user can also add a new process or remove a current one by clicking the "Add" or the "Remove" button. The maximum number of processes in a set is 4. This number is set based on observations that students are able to trace algorithms without tiring them too much when the number of processes is no more than four.

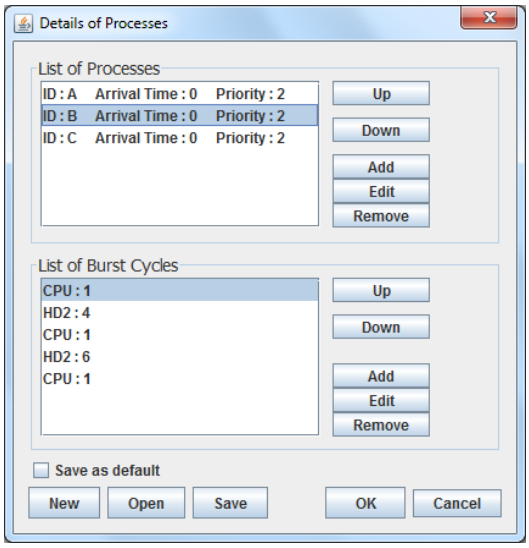


Fig. 2 A window interface that is used to construct or modify a set of process.

In the "List of Burst Cycles" (see Fig. 2), the lengths of the CPU and I/O burst times in each CPU-I/O burst cycle of a process from the "List of Processes" — process B in this example — are shown. In the same way, through the "Add", "Edit", and "Remove" buttons, the information about the CPU and I/O burst times can be manipulated. When adding an I/O burst time, the user needs to specify which I/O device the process will use: Hard disk 1 (HD1), Hard disk 2 (HD2), Printer 1 (PT1), or Printer 2 (PT2). This makes it possible to configure processes in a particular set to share I/O devices or not. When I/O devices are shared among processes, they will be scheduled on a First-Come-First-Served (FCFS) basis. The ability to share I/O devices is included in order to help the user to understand the relationship between CPU and I/O scheduling and can be used to help introduce users to I/O scheduling which is a standard topic in operating systems courses. The user can save the modified set of processes for later use by clicking on the "Save" button and then entering the file name. Otherwise, the modified set of processes will be lost when the user exits the program.

The user can also create a new set of processes easily by clicking the "New" button in the window shown in Fig. 2 and then telling the simulator to automatically generate a new set of processes for him. This will cause a window shown in Fig. 3 to appear. Through this window, the user can be more specific about the set of processes he wants the simulator to automatically generate. Also, the generated set of processes can be modified and/or saved using the same window interface shown in Fig. 2.

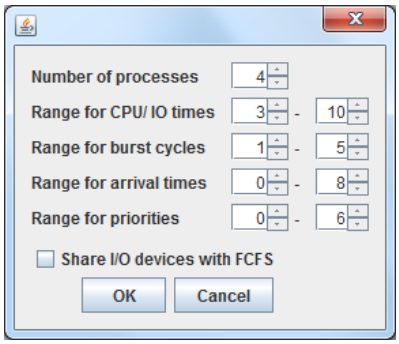


Fig. 3 A window interface that is used to give specific details about the set of processes to be automatically generated by the simulator.

After the predefined set of processes is modified or the user-defined set of processes is constructed, the process table located under the drop-down list box of the algorithms (See Fig. 1(a)) will be updated to respond to the changes. Note that, during the animation, the "Burst Cycles" drop-down list box of each process in the process table shows the current burst time (either CPU or I/O burst time) of the process. Also, the figure in the parenthesis of the "Priority" field of each process represents the original priority while the one outside the parenthesis represents the current priority. The information in the "Priority" field will be shown only when the algorithms that use priorities are selected.

The user can view or change the predefined scheduling parameters by clicking the "Parameter" button located next to the "Process" button (See Fig. 1(a)). Fig. 4 shows a scheduling-parameter window when the MLFQ algorithm is selected. Note that the number and the type of scheduling parameters vary from algorithm to algorithm. For example, the scheduling parameters used by the MLFQ algorithm are the length of time slice and the conditions for increasing/decreasing a process' priority, while the length of time slice is the only parameter used by the RR algorithm.

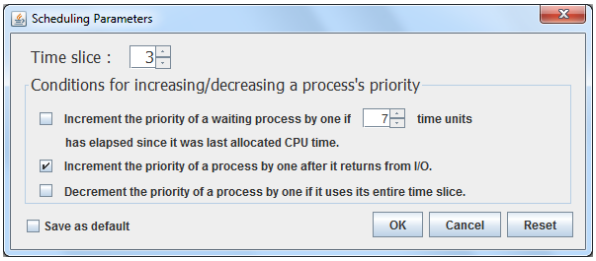


Fig. 4 An example of a scheduling-parameter window.

The bottom area of the snapshot in Fig. 1(a) contains the buttons that allow the user to control the animation. The user can start and stop the animation whenever he wishes by clicking on the "Start" and "Stop" buttons. Alternatively, the user can choose to trace the algorithm step by step, in order to understand the details of the algorithm, by repeatedly clicking the "Next" button. The user does not need to watch the animation or trace the algorithm from the beginning. Rather, he can choose to start from any time by setting the desired time in the "Time" field and then clicking "Set Time" button. Also, the speed of the animation can be changed using the slider and through the "Configure" menu when a fine-grained control is needed.

The bottom half of the snapshot in Fig. 1(a) shows the *display area* that accommodates the animation that demonstrates how the selected algorithm works. The left side of the display area is the *state-diagram view* which displays the different states in which processes can be at different times. To ensure that user learning is enhanced, rather than jumping instantaneously to the next state during a state transition, a process (which is represented by its ID) moves smoothly from one state to another. Similarly, the process moves smoothly from an I/O queue to a ready queue. For the algorithms that use priorities, the priority levels of the ready queues are displayed in front of the queues and the ready queue with higher priority level is placed above the one with lower priority level. As an example, the state-diagram view of the snapshot in Fig. 1(b) shows that there are two ready queues representing the priority levels of 2 and 3. At any time during a simulation, the number of the ready queues being used and the priority levels the queues represent are determined by the number of the ready processes and the priority values that the ready processes have at that time. For example, if there are three ready processes, two of which have the priority of 2 and one of which has the priority of 3, then there will be two ready queues representing the priority levels of 2 and 3. Since the maximum number of the processes is limit to four, the maximum number of the ready queues is four.

The right side of the display area is the *timeline view* which displays a colored block for each unit of time a process spends in any state. The color of the block, which corresponds to one of the colors in the state-diagram view, is determined by which state the process is in. During the animation, various events may occur and cause a process to change its state. Details about the event can be viewed in the "Event Message" panel, which is located above the timeline view.

In the simulation of Fig. 1, a user-defined set of processes, which is summarized in the process table located under the drop-down list box of the algorithms, was used. The user-defined set of processes contains processes A, B, and C, all of which have the same priority of 2 and arrive at time 0. When two or more processes have the same priority, the simulator puts them in the ready queue for that priority in alphabetical order. Process A requests only one burst of 10 units of CPU time. Processes B requests a burst of 1 unit of CPU time, then blocks on I/O for 4 units of time, then requests a burst of 1 unit of CPU time, then blocks on I/O for 6 units of time, and then requests one last burst of 1 unit of CPU time. Process C requests a burst of 5 units of CPU time, then blocks on I/O for 4 units of time, and then requests one last burst of 3 units of CPU time. Note that processes B and C are using different I/O devices in this simulation. In the simulation of Fig. 1, the scheduling parameters are set as shown in Fig. 4. That is, time slice is set to 3 time units and any process that has just returned from its I/O will have its priority raised by one.

Fig. 1 gives two snapshots of the scenario. The snapshots in Figs. 1(a) and (b) are at time 15 and time 16 respectively. The state-diagram view of the snapshot in Fig. 1(a) shows the state each process is in at the beginning of time 15. That is, process A is in the running state while processes B and C both are blocked in the waiting state for their I/O requests to complete. The timeline view shows that processes A, B, and C have been in the current states since time 11, 9, and 11, respectively. As reported in the "Event Message" panel, at time 15, two hardware interrupts have been generated indicating that the I/O operations requested by processes B and C have been completed, and the priority of processes B and C has been raised to 4 and 3 respectively. At this point, process B becomes the process with the highest priority; therefore, it preempts the CPU from process A and runs next. Various events occurring at time 15 cause all the processes to change their states; the state-diagram view will show such transitions. As shown in the timeline view of the snapshot in Fig. 1(b), all the processes spend a unit of time in their new states.

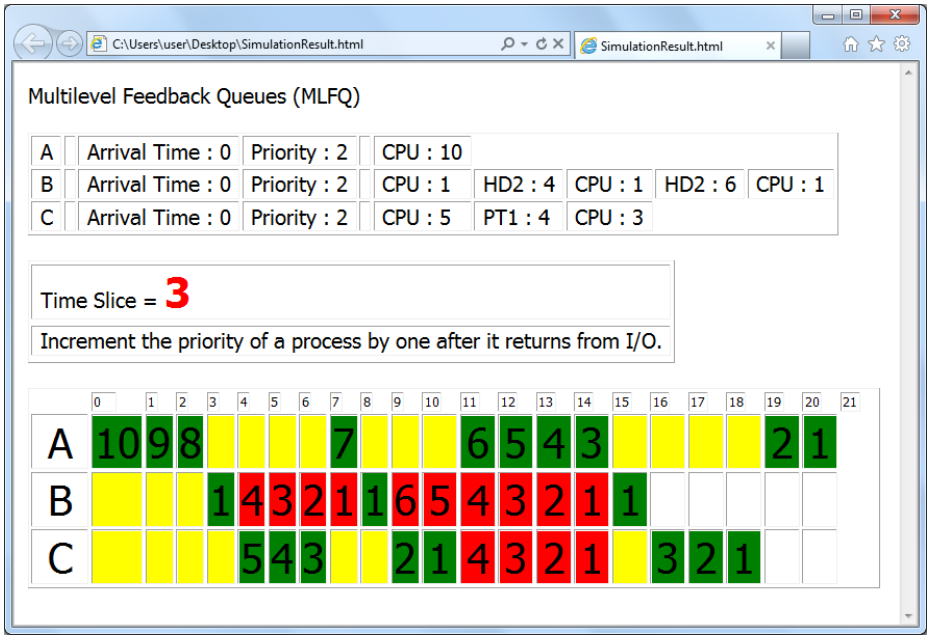


Fig. 5 The simulation result in an HTML file.

When the simulation is over, the user can view the performance statistics, which include the response time, the waiting time, and the turnaround time of each process; the average response time, the average waiting time, the average turnaround time, and the CPU utilization, by clicking on "Statistics" button. Also, by clicking "Save as HTML" on the "File" menu, simulation result can be saved in an HTML (HyperText Markup Language) file that can be displayed and printed by a standard browser. Fig. 5 shows the simulation result of the above scenario in an HTML file.

3.2 Practice Mode

Fig. 6 shows a snapshot of the simulator in practice mode. The user can open a practice-mode tab by clicking the "File" menu and then clicking "New Practice". To reduce the amount of time the user has to devote to learn how to use the simulator, a practice-mode tab has been designed to look as much like a simulation-mode tab as possible. The major differences between a simulation-mode tab and a practice-mode tab are as follows. First, there is no state-diagram view in a practice-mode tab. Second, a practice-mode tab does not contain the speed-control slider, but instead, it contains the "Display Answer" button. Third, the process table becomes editable so that it can be used as an interface for the user to enter the information about each process at any particular time. Fourth, the timeline view and the "Event Message" panel become editable so that they can be used as interfaces for the user to predict when and for how long each process is in a particular state and why it is in that state. As in simulation mode, the user needs to specify which CPU scheduling algorithm will be used, and for each selected algorithm, the user can use the predefined set of processes and the predefined scheduling parameters, or the user can customize them using the same interfaces (See Figs. 2 and 4).

Since the timeline view is editable in practice mode, clicking the blocks under the timeline will change the color of the blocks. The user can predict which state each process is in for each block under the timeline by repeatedly clicking each block until the color corresponding to the predicted state is displayed. The color green, yellow, and red are used to represent running, ready, and waiting states respectively. The color of the blocks will be changed in the following cyclic order: from no color to green, from green to yellow, from yellow to red, and from red to no color. For the sake of user convenience, the color of each block will start with the previously selected color, since processes generally spend a certain amount of time in each state before making a transition to another state. As an example, to indicate that process B is in the ready state for 3 units of time since time 0, the user needs to repeatedly click on the first block until the yellow color representing the ready state is displayed, and then one-click on the second and the third blocks, which also changes these two blocks to yellow. This allows the user to visually predict which state each process is in for each block under the timeline.

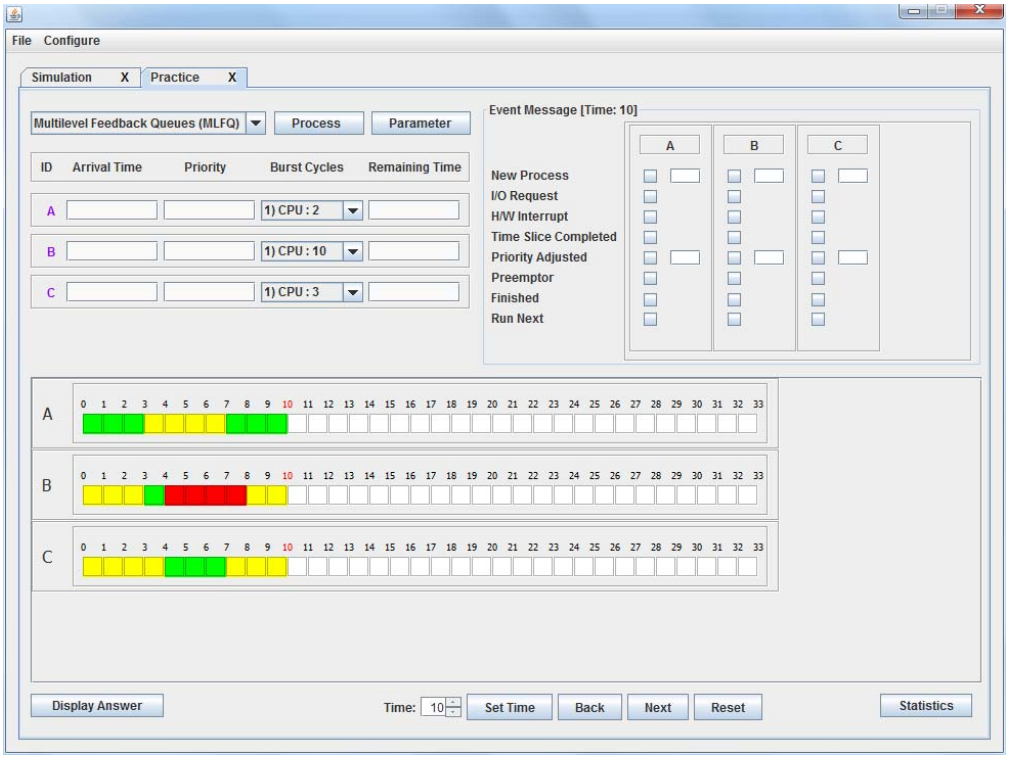


Fig. 6 A snapshot of the simulator in practice mode.

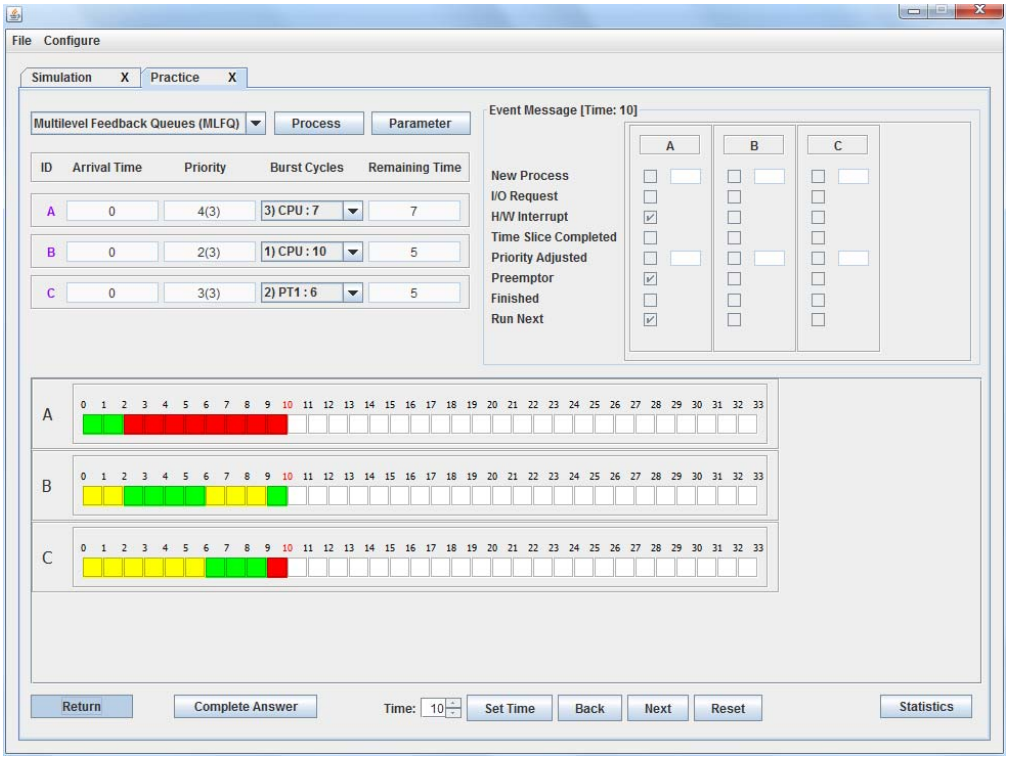


Fig. 7 A snapshot of the simulator in practice mode after the "Display Answer" button is clicked.

In practice mode, the user can predict why the processes are in the states he predicted by checking relevant checkboxes and filling in the missing information in the "Event Message" panel for each time. To be more flexible, this type of practice as well as practicing by entering data in the process table are optional.

Once the user is done making a scheduling decision for a particular time, he can then click the "Next" button, which causes the simulator to remember his input for that time and then wait for new input for the next time. Alternatively, the user can have the simulator remember all the scheduling decisions he has made up to a particular time by setting that time value in the "Time" field and then clicking "Set Time" button. At any time while in practice mode, the user can check whether his answer is correct or not by clicking on the "Display Answer" button, which causes the results up to where he has finished to be displayed, as shown in Fig. 7. By clicking the "Return" button (see Fig. 7), the user can go back to continue practicing from where he has left off. The user can also see all the results by clicking on the "Complete Answer" button (see Fig. 7).

4. Versions and Availability

This paper describes version 2 of the author's simulator for learning CPU scheduling algorithms. Version 2 of the simulator is completely redesigned and rewritten from the ground up in order to overcome the design flaws of the initial version [16].

The first major design flaw of the initial version is that the simulator is not able to handle when more than one event of the same type occurs at the same time. Examples of such situations are when more than one process arrives at the same time, when more than one hardware interrupt occurs at the same time, and when more than one process is assigned a new priority (due to the conditions for increasing/decreasing a process' priority) at the same time.

The second major design flaw is that the ready queues used in the MLFQ algorithm cannot represent all the priority levels that the process can have. In the initial version of the simulator, there is a limit on the number of ready queues and each of the queues represents a fixed priority level. Since there is no limit on the number of processes and the priorities of processes are dynamically adjusted, it is possible that the priority of a process is higher (or lower) than the highest (or lowest) priority level of the ready queues. Such a process will be put in the ready queue with the highest (or lowest) priority level. As a result, the process selected to run next may not be the process with the highest priority.

The third major design flaw is the way the "Back" and the "Next" buttons that are used to control the animation work. Clicking the "Back" button on the initial version will bring the user back to the previous state of the currently running process, rather than its previous time. Similarly, clicking the "Next" button will bring the user to the next state to which the currently running process will make its transition, rather than its next time.

Other significant improvements from its initial version are easier-to-use user interfaces, Priority Scheduling algorithm being added, and facility to save the simulation result in an HTML file.

Version 2 of the simulator will be made available to any interested instructor or student who sends a request by email.

5. Related Work

In this section, some animation tools for learning CPU scheduling algorithms that others have developed are discussed.

English and Rainwater [17] developed several animations using Adobe Flash and used them as part of their lecture in an operating system course. Among these animations, four of them are used in teaching FCFS, RR, SJF, and Priority Scheduling algorithms. Since these animations were designed to be closely aligned with the content in a traditional operating systems textbook, each process consists only one CPU burst. The animations are also accessible through the web [18] for anyone to use. However, they do not allow the users to interact with them that much; only one data set is used, and the same animation plays over and over again.

The HyperLearning Center [19] at George Mason University provides a set of Java applets to illustrate several algorithms of computer science including RR and Priority Scheduling algorithms. The applets allow the user to create up to twelve processes, but only in a restricted manner. That is, each process can consists only one CPU burst and the length of the burst is randomly specified by the applets.

The Tran's Scheduling Algorithm Simulator [20] supports all the algorithms the author's simulator supports. Although it lets the user create a personal set of processes, only one CPU burst per process can be specified. Also, time slice is program coded and taken as 1 or 4 for each set of processes.

The MLFQ Scheduling Algorithm Simulator [21] is a tool that supports only one scheduling algorithm, as the name implies. It allows each process to alternate CPU bursts with I/O bursts. However, unlike the author's simulator, all I/O bursts are fixed to one value.

The animation demonstrated by the Java applets of the HyperLearning Center is similar to that provided by the state-diagram view of the author's simulator in the sense that it shows how a process is put into a ready queue and how it is removed from the ready queue and assigned the CPU. However, there is no animation of I/O activity since the applets adopt the simplified process model (i.e., one CPU burst per process). On the other hand, the rest of the above simulators use a Gantt chart to animate which process is using the CPU at what time. This approach is fine when the process model is the simplified one. Since the MLFQ Scheduling Algorithm Simulator allows processes to alternate CPU bursts with I/O bursts, it also uses a Gantt chart to report I/O usage. However, it reports I/O usage in a composite Gantt chart with little hint as to how multiple simultaneous I/O requests are handled. This can confuse the user. On the other hand, the author's simulator uses a different approach to represent the animation. Rather than focusing on the resource usage, the author's simulator focuses on the processes' states and the events that cause them to change their states. Using this approach, the user will be able to gain insight into exactly how the algorithms work, that is, the user will be able to understand what is currently happening to the processes and why the currently running process can continue using the CPU or why it cannot. Also, the author's simulator gives the user the choice of sharing I/O devices with FCFS queues or using unique I/O devices for each process. In addition, the author's simulator animates I/O activity to help the user understand the specific outcome for multiple simultaneous I/O requests.

Finally, none of the existing tools provide any function that is similar to the practice mode of the author's simulator.

6. Conclusions

This paper presents an interactive Java-based simulator that demonstrates the concepts of various CPU scheduling algorithms through animation. There are two operating modes for the simulator; the first is simulation mode and the second is practice mode. In simulation mode, the user can watch the animation of how an algorithm works or trace the algorithm step by step. In practice mode, the user can predict when and for how long each process is in a particular state and why it is in that state through an easy-to-use graphical user interface, and check whether his answer is correct or not with the simulator at any time during practice. By using the simulator, the user could achieve a better conceptual understanding of the CPU scheduling algorithms.

Future work will include an extensive experiment with the simulator in a computer laboratory to determine its effect on student learning.

References

[1] C. A. Shaffer, M. L. Cooper, A. J. D. Alon, M. Akbar, M. Stewart, S. Ponce, and S. H. Edwards, "Algorithm Visualization: The State of the Field", *ACM Transactions on Computing Education*, Vol. 10, No. 3, Article 9, 2010.

[2] S. H. Rodger, E. Wiebe, K. M. Lee, C. Morgan, K. Omar, and J. Su, "Increasing Engagement in Automata Theory with JFLAP", *Proceedings of the 40th SIGCSE Technical Symposium on Computer Science Education*, 2009, pp. 403-407.

[3] D. Schweitzer and W. Brown, "Using Visualization to Teach Security", *Journal of Computing Sciences in Colleges*, Vol. 24, No. 5, 2009, pp. 143-150.

[4] W. S. Gilley, "Animations and Interactive Material for Improving the Effectiveness of Learning the Fundamentals of Computer Science, Master's Thesis, Department of Computer Science, Virginia Polytechnic Institute and State University, 2001.

[5] P. Bauer, J. Leuchter, V. Steklý, "Simulation and Animation of Power Electronics in Modern Education", *Proceedings of the 4th WSEAS International Conference on Applications of Electrical Engineering*, 2005, pp. 48-52.

[6] P. Marambeas, P. Stergiopoulos, S. Papathanasiou, P. Bauer, and S. Manias, "Interactive Multimedia Material for an Electrical Power Quality Course", *WSEAS Transactions on Advances in Engineering Education*, Issue 7, Vol. 4, 2007, pp. 141-146.

[7] M. G. Sánchez-Torrubia, M. A. Sastre-Rosa, V. Giménez-Martínez, C. Escribano-Iglesias, "Visualization on Learning Mathematics Concepts for Engineering Education", *Proceedings of the 4th WSEAS / IASME International Conference on Engineering Education*, 2007, pp. 232-235.

[8] M. G. Sánchez-Torrubia, C. Torres–Blanc, and S. Krishnankutty, "Mamdani's Fuzzy Inference eMathTeacher: a Tutorial for Active Learning", *WSEAS Transactions on Computers*, Issue 5, Vol. 7, 2008, pp. 363-374.

[9] M. Byrne, R. Catrambone, and J. Stasko, "Evaluating Animations as Student Aids in Learning Computer Algorithms", *Computers & Education*, Vol. 33, No. 4, 1999, pp. 253-278.

[10] C. Hundhausen, S. Douglas, and J. Stasko, "A Meta-Study of Algorithm Visualization Effectiveness", *Journal of Visual Languages and Computing*, Vol. 13, No. 3, 2002, pp. 259-290.

[11] S. Grissom, M. McNally, and T. Naps, "Algorithm Visualization in CS Education: Comparing Levels of Student Engagement", *Proceedings of the 2003 ACM Symposium on Software Visualization*, 2003, pp. 87-94.

[12] P. Saraiya, C. Shaffer, D. Mccrickard, and C. North, "Effective Features of Algorithm Visualizations", *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, 2004, pp. 382-386.
[13] A. Silberschatz, P. Galvin, and G. Gagne, *Operating System Concepts*, 8th ed., John Wiley & Sons, 2010.
[14] G. Nutt, *Operating Systems*, 3rd ed., Addison Wesley, 2004.
[15] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, *Operating Systems: Four Easy Pieces*, Version 0.4, May 2011.
[16] S. Suranauwarat, "A CPU Scheduling Algorithm Simulator", *Proceedings of the 37th ASEE/IEEE Frontiers in Education Conference*, 2007, pp. F2H-19-F2H-24.
[17] B. M. English and S. B. Rainwater, "The Effectiveness of Animations in an Undergraduate Operating Systems Course", *Journal of Computing Sciences in Colleges*, Vol. 26, No. 5, 2006, pp. 53-59.
[18] COSC 3355 Animations, http://cs.uttyler.edu/Faculty/Rainwater/COSC3355/Animations/index.htm
[19] The HyperLearning Center, http://cs.gmu.edu/cne/workbenches/index.html
[20] http://www.utdallas.edu/~ilyen/animation/cpu/program/prog.html
[21] S. Khuri and H. Hsu, "Visualizing the CPU Scheduler and Page Replacement Algorithms", *Proceedings of the 30th SIGCSE Technical Symposium on Computer Science Education*, 1999, pp. 227-231.