A CPU Scheduling Algorithm Simulator

Sukanya Suranauwarat
School of Applied Statistics, National Institute of Development Administration,
118 Seri Thai Rd., Bangkapi, Bangkok 10240, Thailand sukanya@as.nida.ac.th

Abstract - **This paper presents a simulator that uses graphical animation to convey the concepts of various scheduling algorithms for a single CPU. The simulator is unique in a number of respects. First, it uses a more realistic process 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. 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 time. A third unique feature of the simulator is that it allows the user to test and increase his understanding of the concepts studied by making his own scheduling decisions, through the very easy-to-use graphical user interface of the simulator. The simulator can be used by students in operating system courses or by anyone interested in learning CPU scheduling algorithms in an easier and a more effective way.**

Index Terms - Algorithm animation, Computer science education, CPU scheduling algorithms, Operating systems

INTRODUCTION

Scheduling is a fundamental operating-system function. The concept is to have computer resources shared by a number of processes. Almost all computer resources are scheduled before use. The CPU is, of course, one of the primary computer resources. Thus, its scheduling is central to an operating-system's design and constitutes an important topic in the computer science curriculum. However, the main difficulty in learning CPU scheduling from a textbook is the lack of interactivity inherent in its static representation. Moreover, textbooks often simplify the illustration of various CPU scheduling algorithms by using an unrealistic process model. For example, although almost all processes alternate CPU bursts with I/O bursts (i.e., alternate bursts of computing with I/O requests), only one CPU burst per process is used in most textbook examples. As a result, students are not able to gain insight into exactly how the algorithms work in real operating systems.

Studies have shown that, for many types of computer science algorithms, animation can do much to enhance learning and understanding [1][2]. Therefore, the author has developed a simulator that uses graphical animation to convey the concepts of various scheduling algorithms for a single CPU. The simulator is unique in a number of respects. First, it uses a more realistic process model that can be configured easily by the user. Second, it 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 state; the simulator shows these events. 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.

A third unique feature of the simulator is that it allows the user to test and increase his understanding of the concepts he has learnt through its very easy-to-use graphical user interface. More specifically, the simulator has two operating modes: *simulation mode* and *practice mode*. In simulation mode, the user can watch virtually step-by-step how an algorithm works or watch it straight through from the beginning until the end, to achieve a better conceptual understanding of the algorithm. In practice mode, the user can reinforce the concepts studied by making his own scheduling decisions, that is, deciding when and for how long each process runs. The author believes that it is important for the simulator to have not only a simulation mode but also a practice mode, since it has been reported that learners who are actively engaged with visualization technology have consistently outperformed learners who passively view graphics [3][4]. For example, Byrne et al. [5] conducted an experiment in which viewers were forced to make predictions about what they would see during an animation. These viewers scored significantly better on a post-test than others who merely watched identical animation without making such predictions.

The remainder of this paper is organized as follows: the next section is a brief overview of the process state and scheduling algorithms used in the simulator, the section after that gives a description of the simulator, followed by a section that discusses related work, and the final section draws some conclusions.

OVERVIEW

The CPU scheduler, which is part of the operating system of a computer, manages the allocation of the CPU among processes. 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 an I/O completion event, before it can proceed. Various events can cause a process to change state. 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 may be running at a time, but several processes may be ready and several may be blocked. All ready processes are kept on a ready queue. All blocked processes are placed on an I/O queue for the requested I/O device. The scheduler uses a scheduling algorithm to decide which process from the ready queue to run when and for how long. There are various scheduling algorithms. The simulator uses the algorithms listed below (which are discussed in [6][7]).

- **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 scheduler 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 scheduler will preempt the currently running process.
- **Multilevel Feedback Queues (MLFQ)**: There are several ready queues, each with different priority. When the CPU is available, the scheduler selects a process from the highest-priority, non-empty ready queue. Within a queue, it uses RR scheduling. The scheduler adjusts the priority of a process dynamically, for example, to reflect resource requirements (e.g., being blocked awaiting an event) and the amount of resources consumed by the process (e.g., CPU time). Processes are moved between ready queues based on changes in their priority. When a process other than the currently running process attains a higher priority, the scheduler will preempt the currently running process and add it to the appropriate ready queue.

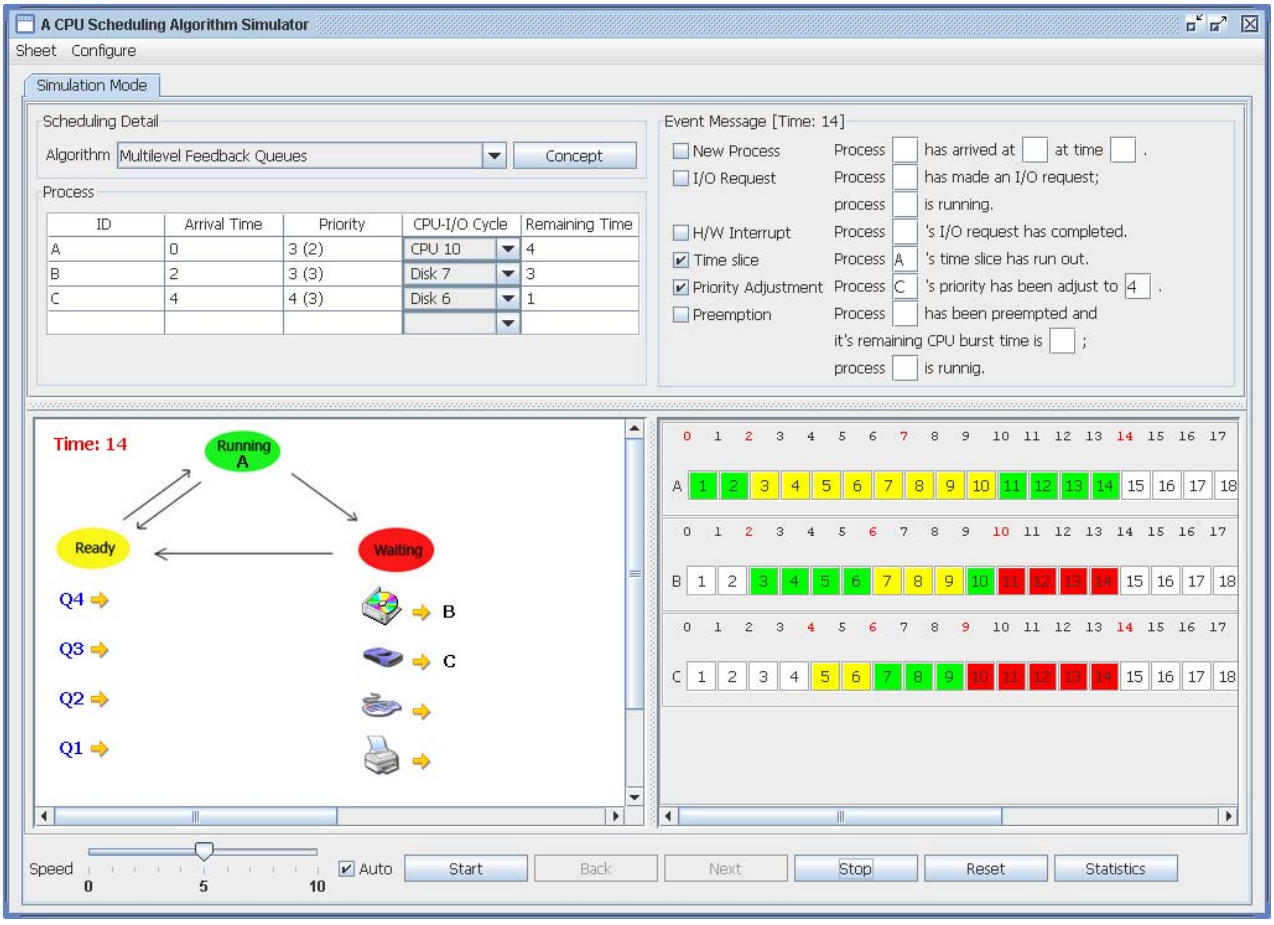

FIGURE 1
SNAPSHOT OF THE SIMULATOR IN SIMULATION MODE AT TIME 14.

DESIGN AND FEATURES

The simulator is written using Java 5.0. It has two operating modes: simulation and practice modes. In simulation mode, the user can watch virtually step-by-step how an algorithm works or watch it straight through from the beginning until the end. 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. Each mode is described below.

I. Simulation Mode

Figure 1 is a snapshot of the simulator in simulation mode. The user can select this operating mode by clicking the "Sheet" menu, pointing to "New Sheet", and then clicking "Simulation Mode". This will cause a new sheet of the corresponding mode to be created. Within a simulation-mode sheet, 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 scheduling parameters and the predefined set of processes will be loaded. The user can view or change the predefined scheduling parameters by clicking "Scheduling Parameters" from the "Configure" menu. Figure 2 shows a scheduling-parameter window when the MLFQ algorithm is selected. The simulator also gives the user the option to replace the predefined scheduling parameters with the user's own parameters by clicking on the "Save as Default" checkbox. 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's priority, while the length of time slice is the only parameter used by the RR algorithm.

The user can change the predefined set of processes by clicking the "Configure" menu, pointing to "Processes" and then clicking "Customize", which causes the window shown in Figure 3 to appear. By clicking the "Add Process" button in this window, the user can add one process at a time to a user-defined set up to a predefined user adjustable limit of 4. However, before adding a process, the user needs to specify the following information about the process: ID, arrival time, and priority. For each added process, the user needs to specify how many CPU-I/O burst cycles the process will have and how long its CPU and I/O burst times are. As an example, Figure 3 shows that process A, which has the priority of 1 and arrives at time 0, is added into the user-defined set of processes, and its first CPU and I/O bursts are respectively set to 3 and 7 units of time. Note that, unless the option to share I/O devices with FCFS is selected, the user does not need to specify the I/O devices for each process since all the processes will be allocated unique I/O devices. This option can be selected through the window that pops up when "Simulator Parameters" from the "Configure" menu is clicked, and 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 also an important topic in the computer science curriculum. The user can save the user-defined set of processes for later use by clicking on the "Save" button and then entering the filename. Otherwise, the user-defined set of processes will be lost when the user animates a new algorithm or when the user exits the program.

In addition to constructing a set of processes from scratch, the user can tell the simulator to generate one for him by clicking the "Configure" menu, pointing to "Processes", and then clicking "Randomly Generate". This will cause a window that looks slightly different from the one shown in Figure 3 to appear. Through this window, the user can be more specific about the set of processes he wants the simulator to automatically generate by specifying the number of processes and the range for the processes' CPU-I/O burst cycles. Also, the generated set of processes can be modified and/or saved using the interfaces that look and function exactly the same as those shown in the window in Figure 3. After the user-defined set of processes is constructed (either by clicking on "Customize" or "Randomly Generate"), the "Process" panel located under the drop-down list box of the algorithms will be updated to respond to the changes. Note that, during the animation, the "CPU-I/O Cycle" drop-down list box of each process 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 bottom area of the snapshot screen in Figure 1 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. He can choose a step-by-step approach, in order to understand the details of the algorithm, by repeatedly clicking the "Next" button once the animation has started. Alternatively, he can choose to watch the animation straight through from the beginning until the end by clicking the "Auto" checkbox before the animation starts. Also, the speed of the animation can be changed using the slider.

The bottom half of the snapshot screen in Figure 1 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; it graphically shows the transition from one state to another. 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 along the line of the transition edge from one state to another. Similarly, the process moves smoothly to or from a ready and an I/O queue. Also, the state diagram has been designed to look as much like the traditional diagram found in textbooks as possible. In addition, any event that may cause a process to change its state is displayed. 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, if a number on a timeline turns red then at least one event has occurred for that process. Details about the event can be viewed in the "Event Message" panel. For example, Figure 1 shows that some events have occurred at time 0, 2, 4, 6, 7, 9, 10, and 14. The user can view the information about any events that occurred at a particular time by clicking on the corresponding number on the timeline.

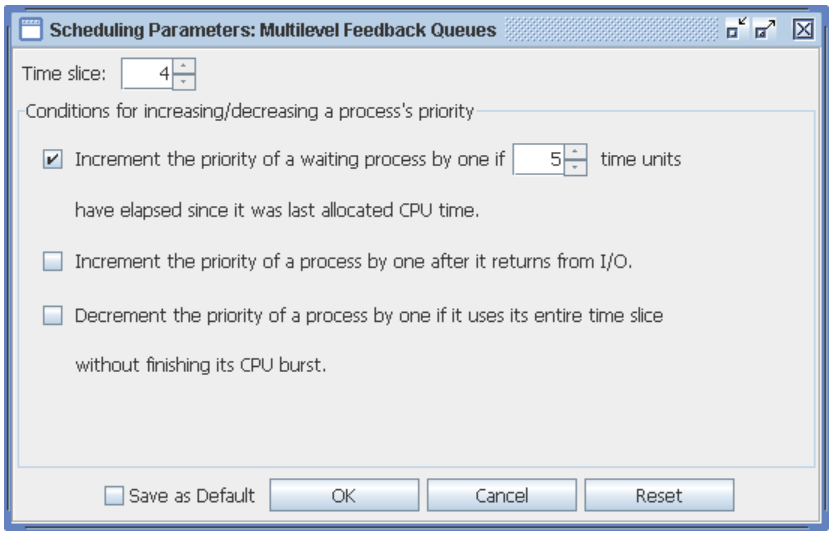

FIGURE 2
AN EXAMPLE OF A SCHEDULING-PARAMETER WINDOW.

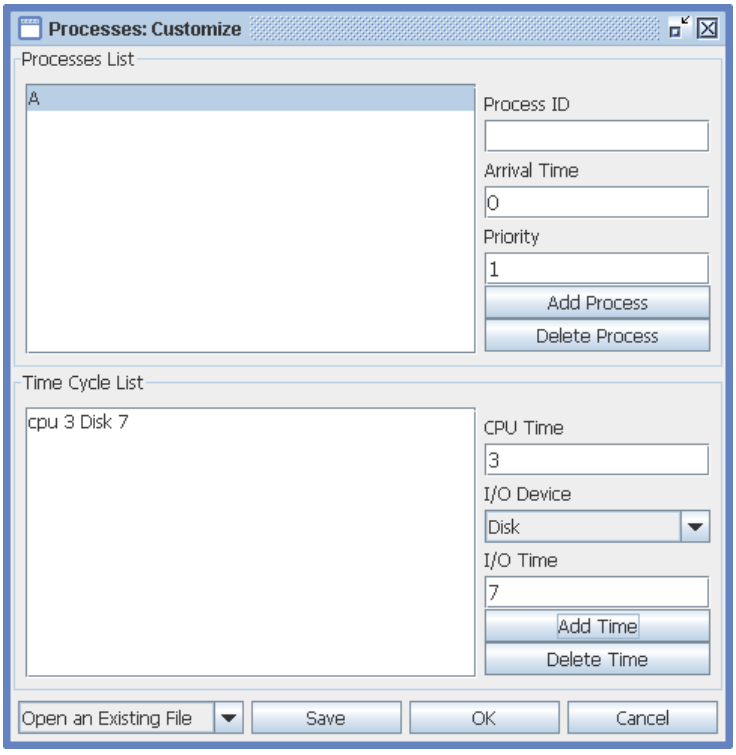

FIGURE 3
A WINDOW THAT IS USED TO CONSTRUCT A USER-DEFINED SET OF PROCESSES.

In the simulation of Figure 1, the predefined scheduling parameters shown in Figure 2 and the predefined set of processes which is summarized in the "Process" panel were used. As shown in Figure 2, time slice is set to 4 time units and any process that has not used the CPU for 5 time units or more since it was last allocated CPU time will have its priority raised by one. The predefined set of processes contains processes A, B, and C. Process A has the priority of 2, arrives at time 0, and requests only one burst of 10 units of CPU time. Processes B and C both have the priority of 3 and arrive at time 2 and 4, respectively. Process B requests a burst of 5 units of CPU time, then blocks on I/O for 7 units of time, and then requests one last burst of 6 units of CPU time. Process C requests a burst of 3 units of CPU time, then blocks on I/O for 6 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.

Figure 1 gives a snapshot of the scenario at time 14. The state-diagram view shows that process A is in the running state while processes B and C both are in the waiting state. The timeline view shows that processes A, B, and C have been in the current states since time 10, 10, and 9, respectively. It also shows that two events have occurred at the current time (i.e., time 14); as reported in the "Event Message" panel, the two events are process A has used up its time slice and the priority of process C has been raised to 4. As shown in the "Process" panel, the remaining times of the current bursts of processes A, B, and C are 4, 3, and 1, respectively.

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.

II. Practice Mode

The user can get a practice-mode sheet by clicking the "Sheet" menu, pointing to "New Sheet", and then clicking "Practice Mode". To reduce the time the user has to devote to learn how to use the simulator, a practice-mode sheet has been designed to look as much like a simulation-mode sheet as possible. The only differences between a simulation-mode sheet and a practice-mode sheet are as follows. First, there is no state-diagram view in a practice-mode sheet. Second, the timeline view of a practice-mode sheet does not contain the buttons that are used to control the animation and the speed-control slider, but instead, it contains the following buttons: "Running", "Ready", "Waiting", "Check" and "Statistics" buttons. Third, "OK", "Cancel", and "Reset" buttons are added in the bottom section of the "Event Message" panel. Within a practice-mode sheet, the user can decide which CPU scheduling algorithm he wants to test his understanding on through a drop-down list box located in the top left section. As in simulation mode, for each selected algorithm, the user can use the predefined scheduling parameters and the predefined set of processes, or the user can customize them using the same interfaces (See Figures 2 and 3).

Through the new set of control buttons in the timeline view, the user can decide when and for how long each process is in a particular state. To help the user to clearly visualize what is happening with each process, the control buttons are each a different color. Clicking the "Running", the "Ready", or the "Waiting" button first and then clicking the blocks under the timeline will change the color of the blocks to the color that corresponds to the button. For example, to indicate that process A is in the running state for 2 units of time since time 0, the user needs to click on the "Running" button, which is green, and then click on the first and the second blocks, which changes the blocks to green. This allows the user to visually predict which state each process is in for each block under the timeline. Afterwards, the user can then click on any number on the timeline where he thinks an event will occur. When the user clicks on a number on the timeline, the "Event Message" panel becomes editable. Once the user agrees with the information he has edited, he can then click the "OK" button, which causes the simulator to remember this information and change the color of the corresponding number on the timeline to red. The "Event Message" panel is where the user can predict why the processes are in the states he predicted above. At any time while in practice mode, the user can check whether his answer is correct or not by clicking on the "Check" button, which causes the results to display in a pop-up window. Also, if the user wants to calculate the performance statistics (i.e., 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), a pop-up window where the user can enter all the calculated values can be summoned by clicking on the "Statistics" button. Afterwards, the user can check his answers with the simulator by clicking the "Check" button in the pop-up window.

RELATED WORK

There are many existing CPU scheduling algorithm simulators. They can be roughly divided into command-line programs and GUI programs. In addition to the author's simulator, examples of the GUI programs are CSCI 152 CPU Scheduling Algorithm Simulator [8], Tran's Scheduling Algorithm Simulator [9], and MLFQ Scheduling Algorithm Simulator [10]. The CSCI 152 CPU Scheduling Algorithm Simulator is a server-side program that allows the user to interact with it via its Web form. Unlike other programs, the purpose of this program is not to show how different scheduling algorithms work, but to provide a nice graphical comparison of the performance of three scheduling algorithms (i.e., FCFS, RR, and SJF scheduling algorithms) with the same set of processes. The Tran's Scheduling Algorithm Simulator supports all the algorithms the author's simulator supports. It also lets the user create a personal set of processes. However, this simulator uses a simple process model, that is, one CPU burst per process. The MLFQ Scheduling Algorithm Simulator supports only one scheduling algorithm, as the name implies. It is more flexible than the previous one in the sense that 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.

Each of the above simulators uses a Gantt chart to animate which process is using the CPU at what time. This approach is fine when the process model is one CPU burst per process. 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 simulators provide any function that is similar to the practice mode of the author's simulator.

CONCLUSION

This paper presents a simulator that uses graphical animation to convey the concepts of various scheduling algorithms for a single CPU. The simulator supports various types of CPU scheduling algorithms. For each algorithm, the user can use the predefined scheduling parameters and the predefined set of processes or the user can customize them. 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 start and stop the simulation whenever he wishes, and watch the simulation straight through from the beginning until the end, or watch it step-by-step. By using the simulator in simulation mode, the user could achieve a better conceptual understanding of the CPU scheduling algorithms. 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 a very easy-to-use graphical user interface, and check whether his answer is correct or not with the simulator at any time during practice.

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

This simulator is available through the web site located at: http://as.nida.ac.th/~sukanya/programs/cpu_simulator.html

REFERENCES

[1] Stasko, J., Badre, A., and Lewis, C., "Do Algorithm Animations Assist Learning?: An Empirical Study and Analysis", *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1993, pp. 61-66.

[2] Lawrence, A., Badre, A., and Stasko, J., "Empirically Evaluating the Use of Animations to Teach Algorithms", *Proceedings of the 1994 IEEE Symposium on Visual Languages*, 1994, pp. 48-54.

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

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

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

[6] Silberschatz, A., Galvin, P., and Gagne, G., *Operating System Concepts*, 7th ed., John Wiley & Sons, 2005.

[7] Nutt, G., *Operating System*, 3rd ed., Addison Wesley, 2004.

[8] http://www.capricorn.org/~akira/cgi-bin/scheduler/index.html

[9] http://www.utdallas.edu/~ilyen/animation/cpu/program/prog.html

[10] Khuri, S., Hsu, H., "Visualizing the CPU Scheduler and Page Replacement Algorithms", *Proceedings of the 30th SIGCSE Technical Symposium on Computer Science Education,* 1999, pp. 227-231.