ANIMATION AND ADDITION OF SSA AND PFH TO OSCAL

A Project

Presented to the faculty of the Department of Computer Science

California State University, Sacramento

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

by

Akshith Vasanth

FALL
2012
ANIMATION AND ADDITION OF SSA AND PFH TO OSCAL

A Project

by

Akshith Vasanth

Approved by:

__________________________________, Committee Chair
Du Zhang, Ph.D.

__________________________________, Second Reader
Chung-E Wang, Ph.D.

____________________________
Date

ii
Student: Akshith Vasanth

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the Project.

__________________________, Graduate Coordinator
Nikrouz Faroughi, Ph.D.          Date

Department of Computer Science
Abstract

of

ANIMATION AND ADDITION OF SSA AND PFH TO OSCAL

by

Akshith Vasanth

OSCAL stands for Operating Systems Concepts Animation Library, a collection of animations to help explain many of the Operating System Concepts. These animations are developed as web applets so that people could remotely use it from anywhere with internet access.

This project involves adding two web applets to OSCAL for animation of complex operating system concepts of, page fault handling and Solaris processor scheduling algorithm in order to make it simpler to understand for the users.

The planned objective of this project is to aid students in understanding of complex operating system concepts easily. This project is an attempt to visualize these concepts and make it easier for a user to understand. It ensures that the users are allowed to interact
with the applet by giving inputs and observing the output changing respectively. The system is developed using Java language with the NetBeans IDE.

________________________, Committee Chair
Du Zhang, Ph.D.

________________________
Date
ACKNOWLEDGEMENTS

This space provides me a great honor to thank all the people with whose support this project and my masters have been a success. I would take this opportunity to convey my sincere thank you to all.

Firstly, I would like to thank Dr. Du Zhang, his help and supportive guide throughout this project in highs and lows has been commendable. He took extra effort to review the report and kept on giving his pieces of advice during my course of project completion.

I would like to also thank Dr. Chung-E Wang for his extended support. He provided me with his valuable suggestions when they are really needed.

Furthermore, I would like to thank the Department of Computer Science at California State University, Sacramento for extending this opportunity for me to pursue my Masters degree and guiding me all the way to become a successful student.

Last but not the least; I am thankful to my parents Vasantha H N and Savithramma M and my sister Arpitha V for their constant support and belief in me, their words of wisdom and moral support helped me overcome all the challenges and through their guidance I was able to successfully complete my project and earn my Masters Degree. I would also like to remember and thank my friends Navin, Ananth, Ravi, Manoj, Rakesh and Prashanth for their support.
TABLE OF CONTENTS

Acknowledgements........................................................................................................ vi

List of Tables ..................................................................................................................... viii

List of Figures .................................................................................................................... ix

Chapter

1. INTRODUCTION ........................................................................................................ 1

2. BACKGROUND ............................................................................................................ 9

  2.1. Page Fault ............................................................................................................. 9

  2.2. Processor Scheduling .......................................................................................... 9

3. DESIGN ....................................................................................................................... 13

  3.1. Design of Page Fault Applet .............................................................................. 13

  3.2. Design of Solaris Scheduling Applet ................................................................. 20

4. IMPLEMENTATION ................................................................................................... 30

  4.1. Page Fault Implementation ................................................................................ 30

  4.2. Solaris Scheduling Implementation ................................................................. 31

5. PERFORMANCE ........................................................................................................ 39

  5.1. Analysis of Page Fault Applet .......................................................................... 39

  5.2. Analysis of Solaris Scheduling Applet ............................................................. 45

6. CONCLUSION AND FUTURE WORK ................................................................. 54

Bibliography .................................................................................................................... 55
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Tables</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Table 4.1: Thread dispatch table</td>
<td>31</td>
</tr>
</tbody>
</table>

viii
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figures</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 3.1 Page Fault algorithm flow</td>
<td>14</td>
</tr>
<tr>
<td>Figure 3.2 Page Fault applet’s input area</td>
<td>15</td>
</tr>
<tr>
<td>Figure 3.3 Page Fault applet’s buttons and delay input</td>
<td>16</td>
</tr>
<tr>
<td>Figure 3.4 Page Fault applet components</td>
<td>17</td>
</tr>
<tr>
<td>Figure 3.5 Page Fault applet’s report area</td>
<td>17</td>
</tr>
<tr>
<td>Figure 3.6 Scheduling applet overview</td>
<td>20</td>
</tr>
<tr>
<td>Figure 3.7 Scheduling applet’s input panel</td>
<td>20</td>
</tr>
<tr>
<td>Figure 3.8 Scheduling applet showing summary of priority changes in every process’s lifecycle</td>
<td>22</td>
</tr>
<tr>
<td>Figure 3.9 Scheduling applet’s dispatch table</td>
<td>23</td>
</tr>
<tr>
<td>Figure 3.10 Scheduling applet’s report area</td>
<td>24</td>
</tr>
<tr>
<td>Figure 3.11 Scheduling applet’s graph area</td>
<td>25</td>
</tr>
<tr>
<td>Figure 3.12 Scheduling applet’s buttons and delay input area</td>
<td>25</td>
</tr>
<tr>
<td>Figure 3.13 Page Fault applet usecase</td>
<td>27</td>
</tr>
<tr>
<td>Figure 3.14 Scheduling applet usecase</td>
<td>28</td>
</tr>
<tr>
<td>Figure 5.1 Page Fault applet started with no pages in physical memory</td>
<td>39</td>
</tr>
<tr>
<td>Figure 5.2 Page Fault applet started with 3 pages in physical memory</td>
<td>40</td>
</tr>
<tr>
<td>Figure 5.3 Page Fault applet scenario: NO TRAP to operating system</td>
<td>41</td>
</tr>
<tr>
<td>Figure 5.4 Page Fault applet scenario: TRAP to operating system</td>
<td>42</td>
</tr>
</tbody>
</table>
Figure 5.5 Page Fault applet scenario: Handling of TRAP when free frames are found in physical memory........................................42

Figure 5.6 Page Fault applet scenario: TRAP when NO free frame is found in physical memory .....................................................43

Figure 5.7 Page Fault applet scenario: Handling of TRAP when NO free frame is found in physical memory .............................44

Figure 5.8 Scheduling applet input example..........................................................................................................................45

Figure 5.9 Scheduling applet scenario: Comparing priority changes of Process “E” with Process “D” ........................................46

Figure 5.10 Scheduling applet: Watching Process “E”’s turnaround time graph when it does not have any sleep interval ..................47

Figure 5.11 Scheduling applet: Watching Process “E”’s turnaround time graph when it has sleep intervals. ...................................47

Figure 5.12 Scheduling applet: Comparing Process “E” priority changes when it has three sleep intervals with Figure 5.9 .................48

Figure 5.13 Scheduling applet: Process with 3 sleep intervals ............................................49

Figure 5.14 Graph showing priority hikes of a process with 3 sleep intervals .................49

Figure 5.15 Scheduling applet: Process with 2 sleep intervals ........................................50

Figure 5.16 Graph showing priority hikes of a process with 2 sleep intervals ...............50

Figure 5.17 Scheduling applet: Process with 1 sleep intervals ........................................51

Figure 5.18 Graph showing priority hike of a process with 1 sleep interval ...............51
Figure 5.19 Scheduling applet: Process with NO sleep interval ..........................52

Figure 5.20 Graph showing NO priority hike for

a process with NO sleep interval ..............................................................52

Figure 5.21 Combined graphs to compare the priority changes of

a process having 3, 2, 1 and no sleep intervals.................................53
Chapter 1

INTRODUCTION

OSCAL stands for Operating System Concept Animation Library [10]. As the name suggests, the library consists of many modules which animates the Operating System Concepts like:

- Processor Scheduling,
- Solutions and mechanisms for Critical Section Problems.
- Classic problems of Concurrency control and Synchronization.
- Deadlock avoidance algorithm
- Virtual Memory
- Disk Scheduling and RAID
- SMP Cache coherency Algorithm
- Data link error control.

Under each module resides many animated Operating System concepts. OSCAL will soon be a place for students who are thriving to understand Operating System concepts should visit to get a clear idea on how each of the concepts works.

The Solaris scheduling system that I develop will be an addition to the “Processor Scheduling” module. Whereas, the Page Fault handling system will be an addition to the “Virtual Memory” module.
**Java and the web:**

Java’s level of interactivity is possible because of how Web browsers that can interpret Java programs for the Web (called applets) operate. In a non-Java-enabled Web browser, information content is defined in terms of Multipurpose Internet Mail Extensions (MIME) specifications, which define a variety of multimedia document formats. This content is specified so that it can be displayed in the browser or in a helper application (such as images, sound, and video). The result is that the user chooses and then observes content. A Java-enabled browser also downloads content defined by MIME specifications and displays it [4] [5].

When a Java-enabled browser downloads a Web page containing a Java applet, indicated by the hypertext tag (``APPLET``), the browser knows that a Java program (applet) is associated with that tag. The Java-enabled browser then can download that applet from the information provider’s Web server. The applet is in a special format called byte codes. The Java-enabled browser can interpret these byte codes and run them as an executable program on the user’s computer host. The result is that the user downloads and runs an executable program—not just content displayed in the Web browser. Java also uses this same scheme to support programmer-defined protocols and special document formats [4] [5].
**How applet works:**

The user sends a request for an HTML document to the information provider’s Web server. The Web server returns an HTML document to the user’s browser. The document contains the `<APPLET>` tag that identifies the applet. The byte code corresponding to that applet is transferred to the user’s host. This byte code was created previously by the Java compiler using the Java source code file for that applet. The Java runtime system on the user’s host interprets the byte codes and provides display. The user then can use the applet with no further downloading from the provider’s Web server. This is because the byte code contains all the information necessary to run the applet. [4][5]

A Java applet runs in the context of a browser. The Java Plug-in software in the browser controls the launch and execution of Java applets. The browser also has a JavaScript interpreter, which runs the JavaScript code on a web page. [6]

**Java Plug-in [6]**

The Java Plug-in software creates a worker thread for every Java applet. It launches an applet in an instance of the Java Runtime Environment (JRE) software. Normally, all applets run in the same instance of the JRE. The Java Plug-in software starts a new instance of the JRE in the following cases:

- When an applet requests to be executed in a specific version of the JRE.
When an applet specifies its own JRE startup parameters, for example, the heap size. A new applet uses an existing JRE if its requirements are a subset of an existing JRE, otherwise, a new JRE instance is started.

An applet will run in an existing JRE if the following conditions are met:

- The JRE version required by the applet matches an existing JRE.
- The JRE's startup parameters satisfy the applet's requirements.

**JAVA SWING [7]:**

The Swing toolkit includes a rich set of components for building GUIs and adding interactivity to Java applications. Swing includes all the components you would expect from a modern toolkit: table controls, list controls, tree controls, buttons, and labels.

Swing is far from a simple component toolkit, however. It includes rich undo support, a highly customizable text package, integrated internationalization and accessibility support. To truly leverage the cross-platform capabilities of the Java platform, Swing supports numerous look and feels, including the ability to create your own look and feel. The ability to create a custom look and feel is made easier with Synth, a look and feel specifically designed to be customized. Swing wouldn't be a component toolkit without the basic user interface primitives such as drag and drop, event handling, customizable painting, and window management.
Swing is part of the Java Foundation Classes (JFC). The JFC also include other features important to a GUI program, such as the ability to add rich graphics functionality and the ability to create a program that can work in different languages and by users with different input devices.

Java Swing GUI components used in this project are listed below [8]:

- **JLabel:**
  - A display area for a short text string or an image, or both. A label does not react to input events.

- **JButton:**
  - An implementation of a "push" button.

- **JTextArea:**
  - A JTextArea is a multi-line area that displays plain text. It is intended to be a lightweight component that provides source compatibility with the java.awt.TextArea class where it can reasonably do so.

- **JTextField:**
  - JTextField is a lightweight component that allows the editing of a single line of text
- **JTable:**
  - The JTable is used to display and edit regular two-dimensional tables of cells. The JTable has many facilities that make it possible to customize its rendering and editing but provides defaults for these features so that simple tables can be set up easily.

- **JTextPane:**
  - A text component that can be marked up with attributes that are represented graphically. This component models paragraphs that are composed of runs of character level attributes. Each paragraph may have a logical style attached to it which contains the default attributes to use if not overridden by attributes set on the paragraph or character run. Components and images may be embedded in the flow of text.

- **JPanel:**
  - JPanel is a generic lightweight container

- **JList:**
  - A component that displays a list of objects and allows the user to select one or more items. A separate model,ListModel, maintains the contents of the list.
Related work:

Page Fault:

Several simple animations of the page fault handling could be found over the internet [11]. But, in this system:

- A user can select what the initial physical memory holds prior to the start of execution.
- User can give the page access order as an input so that he could see which page access causes the page fault and which page access does not.
- User can see how the frame number in the page table is updated
  - When a new page is brought into physical memory or
  - When a page is taken out of physical memory.
- User can see “first in first out” page replacement algorithm, when a page needs to be brought in to physical memory, but, no free frames are available in the physical memory.
- User can pause and continue the execution at his own convenience.
- User can provide the delay (the pace at which each step is executed).
- The report/summary of the execution is printed during the execution.
In this animation of Solaris Scheduling algorithm:

- A user can type in Process name and give it a priority, time to complete and up to three sleep intervals.
- User can pause and then continue the execution at his own convenience.
- User can provide the delay (the pace at which each step is executed).
- During execution, user can see which process is executing currently, what its priority is, if there are any priority changes and the reason.
- At the end of execution, a list of priority changes for each process with reason for the priority change is displayed.
- Graph will be shown consisting of all the process plotted against time is shown.

There is no other system which has so many features. After using these systems couple of times, a user without doubt will get a vast idea of how these (page fault handling and Solaris scheduling algorithm) operating system concepts work.
Chapter 2

BACKGROUND

2.1 Page Fault:

The concept of virtual memory is a memory management technique developed for multitasking kernels. Virtual memory allows a program to be designed as though there is only one kind of memory, "virtual" memory, which behaves like directly addressable read/write memory (RAM). This technique lets user design program which are not limited by the size of physical memory. Using this concept, a part of the secondary memory such as hard disk could also be used as physical memory. Virtual memory makes application programming easier by hiding fragmentation of physical memory. Primary memory is divided into similar sized parts called as page frames and the secondary memory is divided into equal sized parts called as pages. [1]

Paging is one of the memory-management schemes by which an operating system can store and retrieve data from secondary storage in same-size blocks called pages for use in main memory. The main advantage of paging over memory segmentation is that paging allows the physical address space of a process to be noncontiguous. Before paging came into use, systems had to fit whole programs into the memory contiguously, which caused various storage and fragmentation problems.
**Page tables:**

Page tables are used to translate the virtual addresses seen by the application into physical addresses used by the hardware to process instructions; such hardware that handles this specific translation is often known as the memory management unit. Each entry in the page table holds a flag indicating whether the corresponding page is in real memory or not. If it is in real memory, the page table entry will contain the real memory address at which the page is stored. When a reference is made to a page by the hardware, if the page table entry for the page indicates that it is not currently in real memory, the hardware raises a page fault exception, invoking the paging supervisor component of the operating system. [1]

Until there is not enough RAM to store all the data needed, the process of obtaining an empty page frame does not involve removing another page from RAM. If all page frames are not empty, obtaining an empty page frame requires choosing a page frame containing data to empty. If the data in that page frame has been modified since it was read into RAM (i.e., if it has become "dirty"), it must be written back to its location in secondary storage before being freed; otherwise, the contents of the page's page frame in RAM are the same as the contents of the page in secondary storage, so it does not need to be written back to secondary storage. If a reference is then made to that page, a page fault will occur, and an empty page frame must be obtained and the contents of the page in secondary storage again read into that page frame.
The main functions of paging are performed when a program tries to access pages that are not currently mapped to physical memory (RAM). This situation is known as a page fault. The operating system must then take control and handle the page fault, in a manner invisible to the program. Therefore, the operating system must: [1]

- Determine the location of the data in auxiliary storage.
- Obtain an empty page frame in RAM to use as a container for the data.
- Load the requested data into the available page frame.
- Update the page table to show the new data.
- Return control to the program, transparently retrying the instruction that caused the page fault.

2.2 Processor Scheduling:

Processor scheduling divides a computer processor's work between multiple programs so that it is continually switching from one open application to another. This gives the appearance that the computer is running a number of different programs simultaneously. [9]. Scheduling is a technique in which processes are given access to system resources (e.g. Processor time). CPU (Central processing unit) scheduling is a method of allocating processor time to processes. CPU scheduling is the fundamental to multiprogramming operating system. This process is done by the scheduler. All the processes that are ready to run are put in a list called as ready list. The scheduler job is to intelligently pick a
process from this ready list which runs/executes next. The scheduling decision may take place when a process switches its state as shown below:

- A process switching from running to wait state.
- A process switching from running to ready state.
- A process switching from waiting to ready state.
- A process terminates.

The scheduler will mainly consider the factors mentioned below:

- **CPU utilization:** The CPU should always be kept busy executing ready processes.
- **Throughput:** Total number of processes that complete their execution per time unit.
- **Turnaround time:** total time taken between entry of a process and its completion.
- **Response time:** amount of time it takes from when a request was submitted until the first response is produced.
- **Waiting time:** Amount of time a process waits in the ready queue.

The optimization criteria would be to have maximum CPU utilization, maximum throughput, minimum turnaround time, minimum waiting time and minimum response time.
Chapter 3

DESIGN

3.1 Design of Page Fault Applet

The system takes initial physical memory and page sequence as input. Initial physical memory sequence asks user to input pages that they desire to see initially in the physical memory before the system starts. As soon as the user presses the start button, the valid bit in the page table for the corresponding bit is set and frame number is updated.

The initial setup is to update the physical memory and the page table using the values entered by a user. After which the actual processing of the pages from the page sequence starts. When a page is being accessed from the physical memory, first the page table is referenced and if the valid bit is set, and then the execution flows smoothly without any interrupt. If the referenced page has an invalid bit in the page table, then the instruction execution is stopped and page fault interrupt occurs. This will send a trap to the operating system asking to get the page from the secondary memory to the physical memory. The operating system will then, gets the requested page from secondary memory. If the physical memory has free frame available to accommodate the new (requested) page, no further calculations are needed and the page is put into free frame of the physical memory. If there is no free frame available, then there comes the need for page replacement policies like, first in first out, least recently used, second chance aging and many more. In this system, first in first out (FIFO) page replacement policy is used to
show the page replacement. FIFO algorithm is a low overhead and requires minimal tracking on the part of operating system. As the name suggests, the oldest page is chosen to be replaced. Once the page is chosen, the old page is swapped out of the physical memory and one free frame is created. To that free frame, the requested page is swapped in to the physical memory. And the page table is updated accordingly with the following:

- New page entry and its frame number are updated.
- Validity bit of the page brought in is set to valid.
- Validity bit of the old page which was swapped out is set to invalid.

![Figure 3.1 Page Fault algorithm flow](image-url)
The above diagram shows a scenario of accessing page “A” which is currently not in physical memory. Hence, its corresponding valid bit entry in the page table will be “invalid”.

1. Once the page A is requested, the page table is checked for valid bit. If it is valid, the instruction continues.
2. If the entry in the page table of the requested page is invalid, then, trap is raised.
3. The operating system then brings the requested page from the secondary memory to the physical memory.
4. Operating system searches for a free space in the physical memory and puts the requested page in the free frame.
5. Sets the valid bit of the page and also the frame number where the page resides.
6. Then, the instruction which caused the page fault is restarted.

Design:

![Diagram](image)

**Figure 3.2 Page Fault applet’s input area**

Above picture shows the input boxes for the users to enter the following:
• Initial Physical Memory: The initial pages contained in the physical memory before the start of execution of the system. In order to show page replacement algorithm, size is limited to 4 (meaning physical memory could contain a maximum of 4 pages at a time).

• Sequence: Sequence in which the pages are accessed from the memory.

![Figure 3.3 Page Fault applet’s buttons and delay input](image)

The above picture shows the buttons and the delay input present on the applet and its functionalities are explained below.

• Delay: Takes integer input form the user. It determines how fast the system should run. The integer entered would be in milliseconds. In the above example, 2000 millisecond means, 2 second will be the delay between each step of execution shown on the applet.

• Clear: Clear button resets the contents of page table Physical memory and hard disk. Clears report area.

• Start: Start button starts the execution.
• Play/Pause: During execution, a user could stop the execution at any point by clicking this button and again resume the execution by clicking the same button.

**Figure 3.4 Page Fault applet components**

The above picture shows the following:

- **Page table**: Contains the page name, Frame number and the valid/invalid bit.
- **Physical Memory**: Contains frame number and the page name.
- **Secondary memory**: Contains all the pages.

**Figure 3.5 Page Fault applet’s report area**
Above picture shows the report area where all the events/steps that happened till then are noted.

**SOLARIS Scheduling Algorithm:**

Outline [3]:

- Priority based scheduling algorithm
- It uses four or more scheduling classes. Four of them are mentioned below and almost every class uses different scheduling algorithm.
  - Real time
  - System
  - Interactive and
  - Time sharing
- Multilevel feedback queue is used for Interactive and timesharing classes.
- Lower the priority in number, higher the time quantum. For example, timesharing class are assigned a priority number from 0 to 59 with 0 being the lowest (high time quantum) and 59 being the highest priority (very less time quantum).
- Process with high priority gets access to CPU more frequently than a process with low priority. Hence, high priority process will get very less time quantum and low priority process will get a large time quantum.
- Each priority level is assigned a time quantum with short time quanta for high priority and log time quanta for low priority levels.
- As and when the given time quantum expires, the priority of the ready thread is lowered by 10 until it reaches 0 (lowest).
- When the thread switches from Waiting to ready, its priority will rise between 50 and 59 depending on the old priority.

In this system efforts have been made to show the working of priority changes of the threads in the time sharing class of scheduling. All the priority changes that a single thread undergoes in its lifecycle will be captured and shown to the user. As mentioned above, user can see the changes in the priority like, for example in a decreasing fashion if the thread does not go to sleep and always craving for CPU time where as if a thread goes to sleep and wakes up, its priority would have been hiked. The main reason is to avoid starvation. Starvation means the process with highest priority will keep on using the CPU until it terminates whereas other threads with lower priorities will keep on waiting for the CPU. In order to avoid this, thread with the highest priority will be given less time quantum whereas the thread with lowest priority will be given maximum time quantum. If we have a look at the dispatch table, thread with priority 0 (lowest) will be having the time quantum of 200 and the thread with priority 59 (highest) will be having the time quantum of 20.
3.2 Design of Solaris Scheduling Applet:

The above picture shows the input pane. It takes the following as input:

![Figure 3.6 Scheduling applet overview](image1)

![Figure 3.7 Scheduling applet’s input panel](image2)

The above picture shows the input pane. It takes the following as input:
• Process Name: In order to distinguish, every process is given a name. It can be a combination of letters and digits.

• Priority: User should input the priority of the process. This system is designed to take one among the 19 priorities that are displayed in the Dispatch table.

• TTC: TTC stands for “time to complete”. TTC is the amount of time a process takes to complete its execution.

• Sleep1, Sleep2 and Sleep3: These are the 3 sleep time intervals meaning that, when the process is running and if it reaches any of the sleep time interval, it should go to sleep and the priority of that process is again recalculate according to the “return from sleep time” shown in the dispatch table. If negative number is entered in any of the sleep interval input, then, it is considered as “no sleep interval” for that particular process.
Figure 3.8 Scheduling applet showing summary of priority changes in every process’s lifecycle

The above picture shows the priority changes that a process undergoes throughout its lifecycle. With the priority, it also shows the reason for the priority change (whether its sleep or Time Quantum expired), global time and also the time elapsed.

- The first column displays the process name.
- Second column shows the priority change. (old priority ➔ new priority)
- Third column shows the time elapsed. Meaning how long the process executed so far.
Fourth column displays the reason for the priority change.

Fifth column displays what the global time was, when this event occurred.

**Figure 3.9 Scheduling applet’s dispatch table**

The above figure shows the Dispatch table as displayed in the applet. This table shows the following:

- **Proc**: process name with the global time will be displayed dynamically as and when a process undergoes a priority change and the whole row is highlighted.
- **Pri**: Priority of a process.
- **TQ**: Time Quantum is the amount of CPU time allocated for a given process with particular priority in the second column.

- **TQ_E**: Time Quantum Expired is the new priority that will be assigned to a process which has used all of its assigned time quanta in the third column.

- **RFS**: “Return From Sleep” is the new priority which will be assigned to a process that will be returning from sleep having the priority matching the second column.

**Figure 3.10 Scheduling applet’s report area**

The above picture shows the report part of the applet. In the beginning the initial ready queue contents are displayed. Ready queue is sorted according to the priority of a process. Process with the highest priority is executed first. And hence, as shown in the picture above, process E with priority 53 will start executing at time 0.
The report part is dynamic. As and when something changes, like priority change, sleep, time quantum expired or a process is completed, the data gets appended to the report and lets the user know what exactly is happening in that instance.

![Image of scheduling applet's graph area]

**Figure 3.11 Scheduling applet’s graph area**

The above figure shows the chart plotted after the execution part is over. The chart shows which process ran at what time and the context switching over the time. X-axis shows the time whereas Y-axis shows the process name. In this system, the graph can accommodate up to 9 processes.

![Image of scheduling applet’s buttons and delay input area]

**Figure 3.12 Scheduling applet’s buttons and delay input area**

The above picture shows the buttons and the delay input present on the applet and its functionalities are explained below.
• Delay: Takes integer input form the user. It determines how fast the system should run. The integer entered would be in milliseconds. In the above example, 1000 millisecond means, 1 second will be the delay between each step of execution shown on the applet.

• Clear: Clear button clears the graph, report and the selections on the dispatch table in order to start new execution.

• Start: Start button starts the execution.

• Graph: Graph button will be activated only after the execution completes. When a user clicks on the graph button, the graph will be plotted.

• Play/Pause: During execution, a user could stop the execution at any point by clicking this button and again resume the execution by clicking the same button.
Page Fault Applet Use case:

Start: The page fault execution starts once a user clicks the start button on the applet.

Play/pause: User can stop the running execution by pressing the “Play/Pause” button. Again, he can continue the execution by pressing the same button. Play/Pause button is as shown in the figure 3.3.

View Report: User can view the summary of events that happened during the execution of page fault algorithm in the report area as shown in the figure 3.5. The report area is dynamic and keeps updating as and when an event occurs.

Input Phy mem contents: User can decide what resides in the physical memory before the execution of page fault starts, by entering page numbers (between 1 and 6) in the input box as shown in figure 3.2.
**Input sequence:** User enters the page numbers between 1 and 6 in the sequence input box as shown in the figure 3.2. The order in which a user enters page numbers, is the sequence how the page fault algorithm accesses the pages.

**Clear:** Once an execution is over, user clicks on this button for resetting all the contents (Page table, Physical memory contents, report area, Initial memory contents, sequence and steps shown on the applet during last execution) and makes place for a new start of page fault algorithm.

**Solaris Scheduling Applet Use Case:**

![Solaris Scheduling Applet Use Case Diagram](image)

**Figure 3.14 Scheduling applet usecase**

**Start:** Execution of scheduling applet starts when a user clicks the start button on the applet.
**Play/pause:** Users can stop the running execution by pressing the “Play/Pause” button. Again, he can continue the execution by pressing the same button. Play/Pause button is as shown in the figure 3.12.

**View Graph:** Users can view the graph plotted with process vs. turnaround time after the execution of Solaris scheduling algorithm by clicking the graph button on the applet as shown in the figure 3.12. Sample graph in the graph area will be as shown in the figure 3.11.

**Clear:** clicking on this button will clear the report area, graph, clears selection on the dispatch table and also clears data present in “Priority changes in process life cycle” section, making way for the start of new execution.

**Input process parameters:** Users can enter the name, priority, time to complete and up to 3 sleep intervals for each process. The input area is as shown in the figure 3.7.

**View Report:** User can view the summary of events that happened during the execution of Scheduling algorithm in the report area as shown in the figure 3.10. The report area is dynamic and keeps updating as and when an event occurs.

**View priority changes:** After the execution of the scheduling algorithm, priority changes for each process throughout its life cycle is displayed in “Priority changes in process life cycle” section as shown in the figure 3.8.
Chapter 4

IMPLEMENTATION

4.1 Page Fault Implementation:

Page Fault algorithm that is used in developing this system is as shown below:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Get the instruction sequence from the user and store it in an array</td>
</tr>
<tr>
<td>2.</td>
<td>Check the instruction with the Page table:</td>
</tr>
<tr>
<td></td>
<td>a. <strong>IF (valid):</strong></td>
</tr>
<tr>
<td></td>
<td>1. No TRAP</td>
</tr>
<tr>
<td></td>
<td>2. Continue</td>
</tr>
<tr>
<td></td>
<td>b. <strong>IF (Invalid):</strong></td>
</tr>
<tr>
<td></td>
<td>1. OS TRAP</td>
</tr>
<tr>
<td></td>
<td>2. OS searches the page on the HD</td>
</tr>
<tr>
<td></td>
<td>3. <strong>IF (free frame found on the PHY_MEM):</strong></td>
</tr>
<tr>
<td></td>
<td>I. Put the page into the frame</td>
</tr>
<tr>
<td></td>
<td>II. SET the Page table bit to Valid</td>
</tr>
<tr>
<td></td>
<td>III. Continue execution</td>
</tr>
<tr>
<td></td>
<td>4. <strong>Else (No free page frame):</strong></td>
</tr>
<tr>
<td></td>
<td>I. See which is the <strong>OLDEST</strong> Page in the <strong>PHY_MEM</strong></td>
</tr>
<tr>
<td></td>
<td>II. SWAP the PAGES (HD ↔ PHY_MEM)</td>
</tr>
<tr>
<td></td>
<td>III. Update the page table (Setting bit to valid/invalid)</td>
</tr>
<tr>
<td></td>
<td>IV. Continue</td>
</tr>
</tbody>
</table>

**Page Fault algorithm**
4.2 Solaris Scheduling Implementation:

The system developed will take the following inputs:

- Process Name
- Process Priority
- CPU time required for the Process to complete (TTC)
- Three sleep intervals.

The dispatch table that I have used to calculate priorities and time quantum is as shown below.

**TABLE 4.1 Thread dispatch table**

<table>
<thead>
<tr>
<th>Priority</th>
<th>Time Quantum</th>
<th>Time Quantum Expired</th>
<th>Return from sleep</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>200</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>200</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>10</td>
<td>160</td>
<td>0</td>
<td>51</td>
</tr>
<tr>
<td>15</td>
<td>160</td>
<td>5</td>
<td>51</td>
</tr>
<tr>
<td>20</td>
<td>120</td>
<td>10</td>
<td>52</td>
</tr>
<tr>
<td>25</td>
<td>120</td>
<td>15</td>
<td>52</td>
</tr>
<tr>
<td>30</td>
<td>80</td>
<td>20</td>
<td>53</td>
</tr>
<tr>
<td>35</td>
<td>80</td>
<td>25</td>
<td>54</td>
</tr>
<tr>
<td>40</td>
<td>40</td>
<td>30</td>
<td>55</td>
</tr>
<tr>
<td>45</td>
<td>40</td>
<td>35</td>
<td>56</td>
</tr>
<tr>
<td>50</td>
<td>40</td>
<td>40</td>
<td>58</td>
</tr>
</tbody>
</table>
Every process has a priority and in this system the total number of priorities is limited to nineteen priorities. First column is the priority of the process. Second column is the Time quantum that a process gets at that particular priority. Third column is the new priority that will be assigned to a process which has used all of its assigned time quanta in the second column. Fourth column is the priority which will be assigned to a process that will be returning from sleep. As we can notice, if a process did not go to sleep and just keeps on executing, its priority will keep on reducing to the least priority. Whereas when a process is getting up from sleep, its priority will be increased.

<table>
<thead>
<tr>
<th>Priority</th>
<th>Time Quantum</th>
<th>New Priority</th>
<th>Return Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td>40</td>
<td>40</td>
<td>58</td>
</tr>
<tr>
<td>52</td>
<td>40</td>
<td>40</td>
<td>58</td>
</tr>
<tr>
<td>53</td>
<td>40</td>
<td>40</td>
<td>58</td>
</tr>
<tr>
<td>54</td>
<td>40</td>
<td>40</td>
<td>58</td>
</tr>
<tr>
<td>55</td>
<td>40</td>
<td>45</td>
<td>58</td>
</tr>
<tr>
<td>56</td>
<td>40</td>
<td>45</td>
<td>58</td>
</tr>
<tr>
<td>58</td>
<td>20</td>
<td>50</td>
<td>58</td>
</tr>
<tr>
<td>59</td>
<td>20</td>
<td>50</td>
<td>59</td>
</tr>
</tbody>
</table>

Depending on the priority, each process is assigned time quantum by looking upon the dispatch table. Time quantum is the amount of CPU time allocated for a given process with particular priority. Once the input has been provided, and the start button has been
pressed, the actual calculation starts. The process with the highest priority will be the first to run (that is to get the CPU). A process can run until its assigned time quantum expires in which case the new time quantum is recalculated and put back to the ready queue. Or, the process may run until it reaches its sleep interval in that case, it will be put back to the sleep queue. Once a process wakes up from sleep, again a new priority is calculated and inserted into the ready queue. The ready queue is sorted according to the priority.

Main algorithm that is used in developing this scheduling system is as shown below:
1. While (run == TRUE)
   
a. RunningProcess = get last element (Highest priority) from Ready List

b. If (RunningProcess == Null )
   
   I. Completed. Run == false

c. Get the TQ from Dispatch table (according to Priority)

d. While (Counter < TQ & Counter < TTC)
   
   I. If (sleep_time)
      
      i. Recalculate the priority using Dispatch table
      ii. Insert to ready queue according to Priority
      iii. Break

   II. Decrement Counter and TTC

e. If (Counter >= TTC):
   
   I. Running Process has been COMPLETED

f. If (Counter >= TQ): // TQ expired
   
   I. Recalculate the priority using Dispatch table
   II. Insert to ready queue according to Priority

Scheduling applet algorithm

Ready queue is implemented using the Linked List package provided by JAVA. It is sorted in ascending order of priority. Code to create the initial ready list is:
public static void CreateReadyLL(int len) {
    globalT = 0;
    for (int i = 0; i < len; i++) {
        if (readyList.isEmpty() == true) {
            readyList.add(process[i]);
        } // if
        else {
            if (process[i].Priority >= readyList.getLast().Priority) {
                readyList.addLast(process[i]);
            } else {
                int lenlist = readyList.size();
                int j = 0;
                while (j < lenlist) {
                    ProcessClass temp = new ProcessClass();
                    temp = readyList.get(j);
                    if (process[i].Priority <= temp.Priority) {
                        readyList.add(j, process[i]);
                        j++;
                        break;
                    }
                }
            }
        }
    }
}
The code for inserting a running process to the ready list according to its priority:

```java
public static void insertByPriorityReady() {
    if (readyList.isEmpty() == true) {
        readyList.add(RunningProcess);
        RunningProcess = null;
    } //if
    else {
        if (RunningProcess.Priority >= readyList.getLast().Priority) {
            readyList.addLast(RunningProcess);
            RunningProcess = null;
        } else {
            int lenlist = readyList.size();
            int j = 0;
            for (j = 0; j < lenlist; j++) {
```
ProcessClass tempP = new ProcessClass();

tempP = readyList.get(j);

if (RunningProcess.Priority <= tempP.Priority) {
    readyList.add(j, RunningProcess);
    RunningProcess = null;
    return;
}
}
}
//else2
}
//else 1
}

Code for insert by priority

Algorithm for a process when it goes to go to sleep:

Function SLEEP:

1. Index: get the index to the dispatch table for the current priority of the process.
2. Using the above index, find the new priority for this process when it returns from sleep
3. Replace it with the current priority
4. And insert it to the sleep queue.
5. Once the process wakes up from sleep, it is inserted into the ready queue.
according to the new priority.

Sleep Algorithm

The class which is used to store each of the process data is as shown below:

```java
public class ProcessClass {
    public String ProcessName;
    public int Priority;
    public int TTC;
    public int Sleep;
    public int Sleep2;
    public int Sleep3;
    public int total;
    public String msg=" ";
    public String dmsg = "";
    public String phistory="";
    public LinkedList<Integer> startX = new LinkedList<>();
    public LinkedList<Integer> breadth = new LinkedList<>();
    public String testStr=" ";

    public static int [] Dpriority      =
    {0 , 5 , 10 , 15 , 20 , 25 , 30 , 35 , 40 , 45 , 50 , 51 , 52 , 53 , 54 , 55 , 56 , 58 , 59};

    public static int [] D_tq           =
    {200,200 , 160, 160, 120, 120, 80, 80, 40, 40, 40, 40, 40, 40, 40, 40, 40, 20, 20};

    public static int [] D_tq_expired   =
    {0 , 0 , 0, 5 , 10, 15 , 20, 25, 30, 35, 40, 40, 40, 40, 40, 45, 45, 50, 50};

    public static int [] D_ret_sleep    =
    {50 , 50 , 51 , 51 , 52 , 52 , 53 , 54 , 55 , 56 , 58 , 58 , 58 , 58 , 58 , 58 , 58 , 59};
}
```

Process class used in Scheduling applet
Chapter 5

PERFORMANCE

5.1 Analysis of Page Fault Applet:

If a user leaves “Initial Physical Memory” input box empty, it means that there is no page present in physical memory. Hence, all entries in the page table will be populated to “Invalid” and page column of the physical memory is left blank indicating empty physical memory. This will populate the page table values as shown below.

![Page Table and Physical Memory](image)

**Figure 5.1 Page Fault applet started with no pages in physical memory**

When a user enters the initial physical memory contents, both page table and physical memory are updated accordingly as show in the picture below.
Figure 5.2 Page Fault applet started with 3 pages in physical memory

When a page is present in the physical memory, its location that is the frame number and valid bit is set in the page table. Else if a page is not in physical memory, its corresponding validity bit is set to “Invalid”. As shown in the above picture, pages 1, 2 and 3 are present in physical memory. Hence, corresponding rows in page table are updated accordingly with the frame number and valid bit is set to “valid”. Whereas pages 4, 5 and 6 which are not in the physical memory. And hence, their corresponding validity bit is set to “Invalid”.

When a page is accessed and if its entry in the page table is valid, then, there will not be any trap to the operating system. This scenario is as shown below.
Figure 5.3 Page Fault applet scenario: NO TRAP to operating system

In the above picture, we also can notice that, when the applet is processing a page, its corresponding page table entry is highlighted. Here page 1 is referenced and its corresponding entry in page table validity bit is “Valid”. Hence, there will be no trap to the operating system.

When a page is referenced and its entry in the page table is invalid, then, there will be a trap to the operating system. This scenario is as shown in the picture.
Figure 5.4 Page Fault applet scenario: TRAP to operating system

Steps taken by the operating system in order to handle a page fault is also displayed on the applet as shown in the picture below.

Figure 5.5 Page Fault applet scenario: Handling of TRAP when free frames are found in physical memory
In the above scenario, as there were free frames available in physical memory, there was no need for page replacement algorithm. In the picture below, it exhibits the case, wherein a page needs to be brought in to physical memory but no free frames are available in physical memory.

**Figure 5.6 Page Fault applet scenario: TRAP when NO free frame is found in physical memory**

In this algorithm, “first in first out” page replacement algorithm is used. Hence, oldest page will be replaced when there is no free frame as shown in the picture below. It also shows the steps involved. Here page 1 will be the oldest page and it is selected to be replaced by the new page 5. Corresponding page table values are updated. As we can see,

- Page table entry of page 1 is set to “Invalid”
- Page table entry of page 5 is set to “Valid” and frame address is updated.
Figure 5.7 Page Fault applet scenario: Handling of TRAP when NO free frame is found in physical memory.
5.2 Analysis of Solaris Scheduling Applet:

Sample data entered by a user is as shown above. In the last row, we could notice that the sleep intervals (sleep1, sleep2 and sleep3) are -1 meaning, process “E” does not go to sleep. According to the algorithm, this means throughout its lifecycle, process “E” will not get a priority hike. After the execution, if we look at the priority changes in the process “E” lifecycle, the priority will be in a decreasing fashion. Whereas if we take any other process say for example, process “D”, there will be a priority hike for three times as it sleeps for 3 times(sleep intervals as shown in the above picture is 50, 100 and 150). The picture below shows the priority changes of Process “D” and “E”.

<table>
<thead>
<tr>
<th>Process Name</th>
<th>Priority</th>
<th>TTC</th>
<th>Sleep1</th>
<th>Sleep2</th>
<th>Sleep3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>30</td>
<td>100</td>
<td>50</td>
<td>100</td>
<td>150</td>
</tr>
<tr>
<td>B</td>
<td>50</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td>150</td>
</tr>
<tr>
<td>C</td>
<td>51</td>
<td>300</td>
<td>50</td>
<td>100</td>
<td>150</td>
</tr>
<tr>
<td>D</td>
<td>52</td>
<td>400</td>
<td>50</td>
<td>100</td>
<td>150</td>
</tr>
<tr>
<td>E</td>
<td>45</td>
<td>500</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Figure 5.8 Scheduling applet input example**
Figure 5.9 Scheduling applet scenario: comparing priority changes of Process “E” with Process “D”

In the above picture, we can notice that the priority of process “D” increases when it comes out of sleep.
The above picture shows the graph with process “E” not having any sleep interval. Let us see how it changes when we give sleep intervals for the process “E” (Sleep intervals at 50, 100 and 150). We can see the change in the pattern of the graph. In figure 5.10, we can see that process “E” ran from time 700 till time 780. Whereas if we compare this with figure 5.11 in that time range, process “E” ran from time 700 till time 970. This demonstrates the improved turnaround time of process “E”.

**Figure 5.10 Scheduling applet: watching Process “E”s turnaround time graph when it does not have any sleep interval.**

**Figure 5.11 Scheduling applet: watching Process “E”s turnaround time graph when it has sleep intervals.**
In the picture below, we could see the difference in priority changes of process E when compared with the previous scenario shown in figure 5.9.

![Priority Changes in Process Life Cycle](image)

**Figure 5.12 Scheduling applet: Comparing Process “E” priority changes when it has three sleep intervals with Figure 5.9**

The following pages show a comparison in the priority changes of a process with 3, 2, 1 and no sleep intervals. A line graph has been plotted to clearly show this scenario.
Figure 5.13 Scheduling applet: Process with 3 sleep intervals

Figure 5.14 Graph showing priority hikes of a process with 3 sleep intervals
Figure 5.15 Scheduling applet: Process with 2 sleep intervals

Figure 5.16 Graph showing priority hikes of a process with 2 sleep intervals
### Figure 5.17 Scheduling applet: Process with 1 sleep intervals

<table>
<thead>
<tr>
<th>Process Name</th>
<th>Priority</th>
<th>TTC</th>
<th>Sleep1</th>
<th>Sleep2</th>
<th>Sleep3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>30</td>
<td>1000</td>
<td>250</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

### Figure 5.18 Graph showing priority hike of a process with 1 sleep interval
Figure 5.19 Scheduling applet: Process with NO sleep interval

<table>
<thead>
<tr>
<th>Process Name</th>
<th>Priority</th>
<th>TTC</th>
<th>Sleep1</th>
<th>Sleep2</th>
<th>Sleep3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>30</td>
<td>1000</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

Figure 5.20 Graph showing NO priority hike for a process with NO sleep interval
Figure 5.21 Combined graphs to compare the priority changes of a process having 3, 2, 1 and no sleep intervals

Seeing these comparisons, we saw the patterns and behavior of dynamic priority changes involved in Solaris Scheduling algorithm. With the increase in priority, process is more likely to get processor time more often, thereby reducing the response and waiting time of a process. For a process with low priority, the response and waiting time will increase.
Chapter 6

CONCLUSION AND FUTURE WORK

This system is an attempt to help people who want to learn operating system concepts namely Solaris scheduling algorithm and page fault handling. While demonstrating these concepts, the system also exhibits many scenarios that a user can notice by giving different inputs as explaining in the previous chapters. With no doubt, this system could be used as a teaching supplement which really helps students to learn complex operating system concepts.

Future enhancements for “Page Fault Handling” system:

The page fault handling system developed could be further improved by the following:

- The system could be designed to take page names, the frame locations in which they reside in physical memory.
- The size of the physical memory could be taken as an input.
- Here in this system, the page replacement algorithm used is “First in first out”. In future, many other algorithms could be added and the choice of what replacement algorithm to use could be taken as an input by the users.

Future enhancements for “Scheduling” system:

- Plotting of the graph that shows the time and CPU usage for each process could be made dynamic.
BIBLIOGRAPHY

   http://en.wikipedia.org/wiki/Page_fault

   http://en.wikipedia.org/wiki/Scheduling_(computing)

[3] A presentation on “scheduling” by the course offered in University of Regina.
   Howard J. Hamilton [Online]

   http://www.computerschool.net/java/applet.html

   http://docs.oracle.com/javase/tutorial/deployment/applet/getStarted.html

   http://docs.oracle.com/javase/tutorial/deployment/applet/appletExecutionEnv.html

   http://docs.oracle.com/javase/tutorial/ui/overview/intro.html

[8] JAVA platform API specification Document, from Oracle. All the definition of
   JAVA Swing GUI components are taken from this website. [Online]
   http://docs.oracle.com/javase/6/docs/api/javax/swing/JComponent.html

[9] Online technical forum, from eHow tech [Online]
[10] Online link to Operating System Concept Animation Library, from Dr. Du Zhang. California State University, Sacramento [Online]
http://gaia.ecs.csus.edu/~zhangd/oscal/oscal.htm

http://cs.uttler.edu/Faculty/Rainwater/COSC3355/Animations/pagefaultsteps.htm