SOLARIS TS SCHEDULING AND LOCALITY OF REFERENCE:
TWO NEW ANIMATIONS 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

Khyati Desai

FALL
2012
SOLARIS TS SCHEDULING AND LOCALITY OF REFERENCE:
TWO NEW ANIMATIONS TO OSCAL

A Project

by

Khyati Desai

Approved by:

__________________________________, Committee Chair
Du Zhang, Ph.D.

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

__________________________________
Date
Student: Khyati Desai

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

SOLARIS TS SCHEDULING AND LOCALITY OF REFERENCE:
TWO NEW ANIMATIONS TO OSCAL

by

Khyati Desai

OSCAL, the Operating System Concept Animation Library (http://gaia.ecs.csus.edu/~zhangd/oscal/oscal.htm), contains a set of animated algorithms as Java applets for concepts in operating systems. OSCAL has been used as a teaching and learning tool for the operating system principles course both at CSUS and in universities around the world. Currently, OSCAL consists of animations for: various processor scheduling algorithms (FCFS, RR, SPN, SRT), real-time scheduling algorithms (RMS, ATSD, PTCD), algorithms for critical section problem (Eisenberg-McGuire, semaphore, message-passing, monitor), Solaris RW Lock, classic concurrency control problems (producer-consumer problem, readers-writer problem, barbershop problem), Banker’s algorithm for deadlock avoidance, various page replacement algorithms in virtual memory (random, FIFO, LRU, MFU, Clock), disk scheduling algorithms (FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK), RAID, cache coherency algorithm, and various data link error control (Stop-and-Wait ARQ, Go-Back-n ARQ, Selective-Reject ARQ).
This project augments OSCAL with two new animations: (1) the Solaris Time Sharing (TS) scheduling algorithm; and (2) demonstration of the concept for the principle of locality (temporal locality and spatial locality). Solaris TS scheduling is based on the dynamic priority a process is assigned. Each process starts with an initial priority and an associated time quantum to execute. The higher the priority the smaller the time quantum will be. If the process uses up its entire time quantum, then it is re-assigned to a lower priority, as it is deemed CPU intensive, but with a larger time quantum. When a process returns from sleep its priority will be escalated. Temporal locality exhibits that if a particular memory location is referenced, then it is likely that the same location will be referenced again in the near future. On the other hand, spatial locality indicates that if a particular memory location is accessed, the likelihood that its nearby locations will be referenced in the near future is higher.

The addition of these two animations will enhance the capabilities of OSCAL, and will help students gain insight, through visualization and interaction, to a working scheduling algorithm in a popular industry-strength operating system (Solaris), and to the fundamental concept of locality of reference. The new animations will prove to be useful tools for both the instructor and students.

_______________________, Committee Chair
Du Zhang, Ph.D.

_______________________
Date
DEDICATION

To my husband
ACKNOWLEDGEMENTS

I would like to thank my project advisor Dr. Du Zhang for giving me an opportunity to work on this project. I sincerely extend my gratitude to him for guiding and supporting me throughout the completion of this project. He constantly gave me valuable input at every stage in the project. His immense knowledge in the field of operating systems helped me to gain insight into this project. I am extremely grateful to Dr. Chung-E Wang for agreeing to be my second reader and taking time to review my project report.

I would like to thank my God, my parents and my parents in law for always inspiring me to achieve the best in life. While pursuing Masters, my friends were always ready to extend their kind help whenever I needed. Lastly, I would like to dedicate my Master’s Degree to my husband without whom I would not have even dreamed of pursuing one. It was his consistent support and encouragement that has brought me this far and of which I can be proud of.
# TABLE OF CONTENTS

Dedication....................................................................................................................... vii
Acknowledgements.......................................................................................................... viii
List of Tables ..................................................................................................................... xi
List of Figures ..................................................................................................................... xii
Chapter
1. INTRODUCTION........................................................................................................... 1
   1.1 Problem Statement ................................................................................................. 2
   1.2 Motivation ................................................................................................................ 3
   1.3 Goal ......................................................................................................................... 4
   1.4 Report Organization ............................................................................................... 5
2. BACKGROUND ............................................................................................................. 6
   2.1 Related Work .......................................................................................................... 6
3. DESIGN AND IMPLEMENTATION ........................................................................... 8
   3.1 Java Applets ........................................................................................................... 8
   3.2 Applet Methods ..................................................................................................... 10
   3.3 Applet Life Cycle ................................................................................................... 11
   3.4 Solaris Time Sharing (TS) Scheduling ................................................................. 13
      3.4.1 Solaris TS Algorithm ...................................................................................... 13
      3.4.2 Applet Design ................................................................................................. 15
      3.4.3 Applet Animation ........................................................................................... 17
      3.4.4 Applet Implementation ................................................................................... 19
   3.5 Principle of Locality .............................................................................................. 31
      3.5.1 Concept of Principle of Locality ..................................................................... 31
      3.5.2 Applet Design ................................................................................................. 33
      3.5.3 Applet Implementation ................................................................................... 35
4. PERFORMANCE EVALUATION .............................................................................. 42
   4.1 Solaris TS Applet ................................................................................................. 42
   4.2 Locality of Reference ............................................................................................ 44
4.2.1 Temporal Locality.........................................................44
4.2.2 Spatial Locality............................................................46

5. FUTURE WORK........................................................................48

Bibliography..............................................................................49
<table>
<thead>
<tr>
<th>Tables</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Dispatch Table</td>
<td>14</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figures</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Java Applets</td>
<td>9</td>
</tr>
<tr>
<td>2. Life Cycle of an Applet</td>
<td>11</td>
</tr>
<tr>
<td>3. Event Driven Applet</td>
<td>12</td>
</tr>
<tr>
<td>4. Applet Design</td>
<td>15</td>
</tr>
<tr>
<td>5. Design of Solaris TS Applet</td>
<td>17</td>
</tr>
<tr>
<td>6. Relationships Among Classes</td>
<td>19</td>
</tr>
<tr>
<td>7. Flow Diagram of Applet Animation</td>
<td>23</td>
</tr>
<tr>
<td>8. Run Button Example</td>
<td>29</td>
</tr>
<tr>
<td>9. Step in Example</td>
<td>31</td>
</tr>
<tr>
<td>10. Graph Representation of Temporal and Spatial Locality</td>
<td>32</td>
</tr>
<tr>
<td>11. Graph Image</td>
<td>33</td>
</tr>
<tr>
<td>12. Principle of Locality Screen Shot</td>
<td>35</td>
</tr>
<tr>
<td>13. Class Relationships</td>
<td>35</td>
</tr>
<tr>
<td>14. Result 1 of Solaris TS Applet</td>
<td>42</td>
</tr>
<tr>
<td>15. Result 2 of Solaris TS Applet</td>
<td>43</td>
</tr>
<tr>
<td>16. Result 1 of Temporal Locality</td>
<td>44</td>
</tr>
<tr>
<td>17. Result 2 of Temporal Locality</td>
<td>45</td>
</tr>
<tr>
<td>18. Result 1 of Spatial Locality</td>
<td>46</td>
</tr>
<tr>
<td>19. Result 2 of Spatial Locality</td>
<td>47</td>
</tr>
</tbody>
</table>
Chapter 1
INTRODUCTION

Operating system forms the basis of a computer system that manages software and hardware components. Operating system is between the hardware and application programs. Some of the responsibilities of operating systems include scheduling of processes in a real time system, managing memory space, input output processing, and concurrency control. One of the crucial tasks is to allocate resources to processes and to make decision on which process to execute at a time. This should be done in an efficient manner and each process should get a fair share to run. Scheduler performs the task of scheduling processes that are ready to execute and ensures that each process gets an equal chance to run.

There are many algorithms for scheduling a process. These algorithms are complex in nature and to understand them just by reading is a difficult task. It is critical for teachers to explain the rules of algorithm using white board. Animation of algorithms will lighten the task of teachers to explain them with ease and makes it easier for students to learn by visualizing them. OSCAL is one such library of animations of operating system algorithms [1]. It consists of various categories of algorithms like Processor scheduling, Critical section problem, Concurrency control and synchronization, Deadlock avoidance, Virtual memory, Disk scheduling and RAID, SMP cache coherency algorithms and Data
link error control. This project will add two new algorithms that fall under Processor scheduling and Virtual memory.

1.1 Problem Statement

In olden days, a computer could execute only one program at a time. With the advent of technology and rising needs, it was required to build a system that could run multiple programs and execution was fast. To overcome this limitation, concept of multiprogramming was introduced. This concept allowed multiple programs to run at a time by a single processor. Multiple programs run seamlessly without the user’s knowledge that CPU can execute only one process at a time. This is done by sharing the resources of CPU among processes. All the processes compete to get the CPU and each process should get a control of CPU. There are many scheduling mechanisms to achieve this type of scheduling [2].

Another important concept of operating system is Principle of Locality for Virtual memory.

“The locality principle guided us in designing robust replacement algorithms, compiler code generators, and thrashing-proof systems. It transformed virtual memory from an unpredictable to a robust technology that regulated itself dynamically and optimized throughput without user intervention. Virtual memory faded into the background of every
operating system, where it performs so well at managing memory with multithreading and multitasking that no one notices”[5].

“The locality principle found application well beyond virtual memory. Today it directly influences the design of processor caches, disk controller caches, storage hierarchies, network interfaces, database systems, graphics display systems, human-computer interfaces, individual application programs, search engines, Web browsers, edge caches for Web based environments, and computer forensics”[5].

One of the algorithms for this project based on scheduling mechanisms called as Solaris Time sharing algorithm. Animation of this algorithm will show how processes are scheduled based on the priority. This will help students to understand the algorithm in a better way. It has a step in feature by which students can run the algorithm systematically and understand it thoroughly.

Another animation implemented is of Principle of Locality. It shows both temporal and spatial locality. I have used an example of “for loop” to display temporal locality. For spatial locality, I have used an example of an array.

1.2 Motivation
We are always looking for new ways to learn and educate ourselves. Students tend to learn faster and retain information quickly when the medium of communication is visual. Animations will increase the skills they acquire and make difficult content easier to
understand. Animation makes learning process fun and interactive for all students. Animation act as an excellent teaching tool for instructors, attracts attention of a learner, nurtures motivational skills, emphasizes on development of student abilities, and enables them to explore a problem [6].

OSCAL has several animation algorithms, which are aimed to help students to understand operating system principles with ease. Two new algorithms, Solaris TS and Principle of Locality will enhance the capability of OSCAL. Both of these algorithms are interactive, so students can enter different values and animation will react in different ways. Learners can execute the algorithms repeatedly as many times as they want and get complete perception. Animation has proven to be very useful as a learning tool and aids learners understanding. This will be an online animation tool, so students can access it from anywhere and at any time of the day wherever internet facility is available [6].

1.3 Goal
The goal of this project is to design and implement animation of Solaris TS scheduling and Principle of Locality. These animations should be easy to use and explain the workings of both the concepts. Features of Solaris TS scheduling are status of processes executing at a particular time, display of graph, statistics of changes of priority, step in feature, speed bar to increase/decrease the speed of animation. Some of the features provided by Locality of reference are speed bar; graph display to show which page address is accessed at a time. The animations are implemented using Java applets.
1.4 Report Organization

Chapter 2 will discuss the related works of what has been done previously. Chapter 3 describes in detail the design and implementation of animations. This chapter will also include some code snippets to explain its working. Chapter 4 will concentrate on performance evaluation by depicting some scenarios and their results. Chapter 5 will describe future work, which will increase its capability.
Chapter 2

BACKGROUND

2.1 Related Work

There is a website on Introduction to Animations for Operating systems by William Stallings [7]. Most of the animations given on this website do not provide an interactive mode. There are buttons to start, stop and reset the algorithm. Learners cannot input different values and they will not be able to see a different behavior of the algorithm. One of the algorithms is based on priority scheduling. It allows creating some processes to submit, but users cannot input any values. There is no animation that is based on Principle of Locality.

There is another website consisting of some animations on operating system concepts [8]. It contains a small set of animations, which are not interactive in nature. Animation for Page replacement algorithm is interactive and lets the user enter different values.

OSCAL consists of a rich set of interactive animation algorithms on operating system principles [1]. Each of these algorithms has decent graphical user interface. They are easy to use and some of them contain a help section to explain various terms used in the animation and working of algorithm. They have some features like graph representation, step by step execution, speed bar, status of executing process, statistical information, text area to enter values.
Solaris TS scheduling and Principle of Locality are designed and implemented in Java Applets. They will be an addition to ongoing part of OSCAL. These two applets will be a valuable contribution to OSCAL. It consists of fields of process name, priority, total units, arrival time and system call. When the algorithm is started, based on priority and arrival time the processes are scheduled. This scheduling can be a difficult to keep a track if it has to be done manually. Principle of Locality has fields like duration of loop, page address to be accessed and string of page addresses, speed bar. Algorithm will show which addresses are accessed in which range and the time range.
Chapter 3

DESIGN AND IMPLEMENTATION

This chapter discusses the design and implementation of both the algorithms in detail. Some code snippets are explained as needed. Java technology is used to design and implement these algorithms.

3.1 Java Applets

Applets are small applications that are accessed on the internet server, transported over the internet, automatically installed and run as a part of web document [10].

A Java applet is a special kind of Java program that a browser enabled with Java technology can download from the internet and run. An applet, is typically embedded inside a web page and runs in the context of a browser. An applet must be a subclass of the java.applet.Applet class. The Applet class provides the standard interface between the applet and the browser environment [9].

Swing provides a special subclass of the Applet class called javax.swing.JApplet. The JApplet class should be used for all applets that use Swing components to construct their graphical user interfaces (GUIs) [9].
The applet code is embedded in an HTML page via an `<APPLET>` tag that has a reference to the applet's .class file. Integrated byte code interpreter within the browser will download and execute the applet as shown in Figure 1.

Besides this, the `<APPLET>` tag also defines how large the applet's display area will be through the width and height parameters. The applet controls everything that happens within its own display area. It can create menus, scroll bars, push buttons, and other means of interacting with the user. It can also communicate with other applets running on the same page, or cause the browser to load a new page. However, an applet cannot affect any page contents that are specified by HTML tags [12].

Figure 1. Java Applets
3.2 Applet Methods

Applets have four methods, which are implemented as needed. These methods are used for applet execution and these methods can be over written in a class.

- **init method**
  This method is used for initialization of an applet, which does not take very long. The init method should be kept short so the applet can load quickly.

- **start method**
  Start method will start the execution of applet. The control must return from this method quickly, a new thread should be started in case of tasks that take long to complete.

- **stop method**
  Stop method will suspend the execution of an applet whenever the user is not viewing the applet’s page.

- **destroy method**
  Destroy method will perform tasks to close the applet’s execution. Additional resources will also be released [9].
3.3 Applet Life Cycle

An applet goes through the major events during its life cycle:

- Initializing the applet
- Start running
- Stopping the applet
- Cleanup after exit

![Applet Life Cycle Diagram](image)

**Figure 2. Life Cycle of an Applet**

init(): called (only once) when the applet is first loaded to initialize variables, resize the applet, setting up GUI components, etc.

start(): called by the browser after the init() to start the applet running. The start() method is run every time the applet becomes active after it has been inactive.

stop(): called when the user leaves the page on which the applet is running, reloads the page, or minimizes the browser, to terminate the applet's running threads.

destroy(): called when the applet is about to be purged from memory.
An applet will wait until an event has occurred and so they are event driven. Whenever an event occurs, event handler is called and Abstract Window Toolkit (AWT) notifies the applet. Applets should quickly take the required action and return control back to AWT. The required action should be short enough and perform rapidly all its task. If some action has to be repeated then a different thread should be created and executed. User will always initiate interaction with applet, which are called events and applets will respond to these events. Some example of events is user clicks a mouse inside the applet’s window mouse click event is generated, user presses a key, key press event is generated.

Figure 3. Event Driven Applet
3.4 Solaris Time Sharing (TS) Scheduling

3.4.1 Solaris TS Algorithm

Solaris TS scheduling is based on dynamic priority, which is assigned to a process. Each process starts with an initial priority and an associated time quantum to execute. The scheduling policy for time-sharing dynamically alters priority and assigns time slices of different lengths using a multilevel feedback queue. There is an inverse relation between priority and time quantum. The higher the priority the smaller the time quantum will be and lower the priority higher will be the time quantum. If the process uses up its entire time quantum, then it is re-assigned to a lower priority, as it is considered CPU intensive, but with a larger time quantum. While interactive processes will have a lower time quantum with a higher priority. When a process returns from sleep, its priority will be escalated.

Table 1 shows dispatch table for scheduling of processes. There are 60 such priority levels, but for brevity, only a few levels are shown here.

It contains the following fields:

- **Priority**: This priority is assigned to a process. The higher number indicates a higher priority.
- **Time quantum**: This is the time quantum for a process. It shows an inverse relationship between priority and time quantum. The lowest priority (priority 0) has the highest time quantum (200 milliseconds) and highest priority (priority 59) has the lowest time quantum (20 milliseconds).
Table 1. 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>
<tr>
<td>55</td>
<td>40</td>
<td>45</td>
<td>58</td>
</tr>
<tr>
<td>59</td>
<td>20</td>
<td>49</td>
<td>59</td>
</tr>
</tbody>
</table>

- Time quantum expired: This field indicates the new priority of a process when it has used up its entire time quantum without blocking. These threads are called CPU intensive and their priorities are lowered.

- Return from sleep: The new priority of a process returning from sleep. When a process returns from sleep, its priority is escalated between 50 and 59.

Scheduler selects the highest priority process to run. The selected process runs until:

- It is blocked

- Uses all its time quantum
• Preempted by another higher priority process

3.4.2 Applet Design

This applet is designed using Java Swing components and controls. JFrame acts as a container and all other controls like textbox, label, button, spinner and table are all contained in this container. JFrame is the main component and all the applet animation will be shown in this frame. A JCanvas is contained in JPanel, which again is contained in JFrame. Graph is displayed in the JCanvas. They all are Java Components.

![Diagram showing the applet design](image)

**Figure 4. Applet Design**

There are five text boxes namely Process name, Priority, Total units, Arrival time and System call, they allow user to enter values. Status and Result are two text areas showing status of which process is executing at a specific time and statistics of the algorithm after completion of applet, respectively. User cannot edit them. Speed bar is used to increase/decrease the speed of applet. The minimum value is 10 and maximum is 50, they can be set by up and down arrow keys. The scheduling class has 60 priority levels, but
the dispatch table used in this project has 10 priority levels for brevity. It has similar fields as described in section 3.4.1 with one extra field called as Process name. When the applet starts running, based on the priority of a process this field will be updated with the name of process, which is executing at that time. Again, user cannot edit this table. As the applet progresses, a graph is drawn with it.

Graph is a visual presentation of when each process is executing. At the end when algorithm completes running all processes, Result shows the statistics of all processes, i.e. the changes in priority it goes throughout its life cycle. Step In button allows user to run applet in steps one at a time. User has to keep on clicking the button to make progress. For example, once when button is clicked, status area will show which process is ready to run or process is running, graph will draw one rectangle bar and dispatch table will be updated accordingly.

Screen shot is shown as below:
3.4.3 Applet Animation

There are five text boxes, which allows user to enter values for maximum five processes. All these five values are explained as below:

- **Process name**: It is the name of a process, which runs when applet starts.
- **Priority**: The priority of any process; its value should be between 0 and 9. If the user enters value, less than 0 or above 9 then appropriate message will be displayed.
• Total units: This value indicates the total amount of time required by a process to complete its job.

• Arrival Time: The time when a process is ready to start running. There should be at least one process that starts at value 0.

• System call: This value indicates that a process will be blocked by a higher priority. Therefore, that process will go to sleep for 3 clock cycles. After which it will return from sleep with a higher priority. A value of 0 specifies that the process is not blocked and it does not go to sleep.

When user clicks Run button, all the values are validated. The applet will start only if these values are validated within the range. Status text is updated whenever process is ready to run, is being serviced, goes to sleep, return from sleep, process completes its execution and algorithm finishes. The process that arrives at time 0, will be the first to start. Process name, its time at which it is ready to run will be updated in the status text and status text shows that it is ready to run. Based on the process initial priority, one row of dispatch table is highlighted and graph is drawn for that time. If the user uses up all its time quantum, it will be assigned a new priority again from the dispatch table. Similarly, if a process will be blocked at a particular system call then it will enter into a sleep mode. After it returns from sleep, process is assigned with a new higher priority from dispatch table. Above procedure will continue until all processes finishes executing.
A user should click Run button only once and applets starts animation as stated above whereas choosing to click “Step in” button instead will let the user run applet in steps. Applet progresses only when user clicks on Step in button. This helps user to understand and study algorithm carefully at a steady speed.

3.4.4 Applet Implementation

The tools used to implement this project are NetBeans 7.1.1 IDE, JDK 6 and JRE 7. Netbeans is Integrated Development Environment that is open source and enables developers to build Java applications for desktop, enterprise and mobile applications. There are four classes namely SolarisApplet.java, ProcessActInfo.java, ProcessInfo.java and drawGraph.java. All these classes are interrelated and values are passed between them internally to make the algorithm run successfully. User will enter values in five text boxes as mentioned in section 3.4.3, select the speed at which applet should start running.

Figure 6. Relationships Among Classes
JSpinner is used to implement the speed bar. Code snippet is:

```java
Object speed;

speed = spinner.getValue();

Thread.sleep(10000/(Integer)speed);
```

Spinner’s value can be stored in a variable with data type Object. Then it is type casted to for an integer value.

When the user clicks on Run, ActionPerformed(java.awt.event.ActionEvent e) method is called. This method is a listener method that means when any action is performed like a button click, this method is called. It also checks which button is clicked to start execution. Therefore, first all the values of five text boxes are validated. Appropriate message is displayed using JOptionPane. JOptionPane makes it easy to create a dialog box that prompts the user to do something. There are many methods of JOptionPane, I have used one method called showMessageDialog. This tells the user that something has happened and prompts to click OK. A small code snippet is as follows:

```java
JOptionPane.showMessageDialog(this, "Maximum 5 processes can be entered", "Check Number of Processes", JOptionPane.ERROR_MESSAGE);
```

This prompts a dialog box saying that number of processes can be maximum 5.

All the values for each process are passed to a different class ProcessActInfo.java which resides outside of SolarisApplet.java class. This class consists for setter() and getter() methods to set and get the values respectively. Additionally, it will decrement the time required to complete each process and increment the counter. Therefore, when the thread
is created and starts running this class will be used to get values of a process. For example, if process A is executing, using this class get() method will get the priority, total time required, arrival time and system call.

Simultaneously, processes are added to a queue using an ArrayList. Each ArrayList instance has a capacity. The capacity is the size of the array used to store the elements in the list. As elements are added to an ArrayList, its capacity grows dynamically. This is how it differs from an array, which has fixed size. ArrayList is used because user can enter any number of processes less than six. This queue keeps a track of number of processes entered by user.

A private class ExeAlgorithm() is called which contains the logic to execute algorithm using threads. This class extends a Thread class, which in turn implements Runnable interface. A class whose instances are intended to be executed by a thread should implement the Runnable interface. The class must define a method of no arguments called run(). There are two ways to create threads:

- Implementing a runnable interface
- Extending thread class

I have used extending thread class to create a thread. This class overrides run() method and defines the code to be executed by thread. The start() method is invoked to start the thread running. Two new queues are created using ArrayList which acts as ready queue
and finish queue. When a process is ready to run, it is inserted into ready queue, while a process that has completed executing enters finish queue. A clock variable is set at the beginning, which tracks the time for each iteration. Based on the value of clock, code checks each process’s arrival time to insert it into ready queue. It updates the status text which process is ready to run with process name and at what time.

At each clock cycle, I have to keep checking the system call value, if any process enters into sleep mode. This check is done even before scheduling a process to start running. A process entering sleep mode will wake up from sleep after 3 clock cycles, this is implemented in the form of interval and true or false flag value is set. Interval is incremented in each loop iteration. Flow chart is as follows:
Figure 7. Flow Diagram of Applet Animation
Code snippet is as follows:

```java
for(int k =0; k<queue1.size();k++){
pSyscall = (ProcessActInfo)queue1.get(k);
    if(pSyscall.getInterv() == 3 && (pSyscall.getSysSleep()).equals("true")){
pSyscall.setSysSleep("false");
        for(int row=0;row<10;row++){
            Object valuePr=disTable.getValueAt(row, 1);
            intparsedV = Integer.valueOf((String)valuePr);
            if(pSyscall.getPriority() == parsedV){
                Object valueNewPr=disTable.getValueAt(row, 4);
                parsedNwPr = Integer.valueOf((String)valueNewPr);
                pSyscall.setPriority(parsedNwPr);  //update new Priority after returning from sleep
                Object valueTQ = disTable.getValueAt(parsedNwPr, 2);
                parsedNwTQ = Integer.valueOf((String)valueTQ);
                pSyscall.setTQleft(parsedNwTQ);  //update new time quantum
                break;
            }
        }
    }
}
```

This loop runs until the end of ready queue called queue1. This code will get all the values of a particular process. If the process is already in the sleep mode and its interval
is 3 then its flag variable will be set to true and it will return from sleep. At this time, its priority and time quantum should also be incremented. However, if process has only gone to sleep then increment the interval value by 1.

Next, the process should be serviced now. The code implemented is:

```java
pt=(ProcessActInfo)queue1.get(0);
if((pt.getSysSleep()).equals("true") && queue1.size() == 1 ){
    pr = -1;
    flag = false;
    updateGui("At Time " + clk + " No process served");
    try
        { Thread.sleep(10000L/(Integer)speed); }
    catch(InterruptedException e){}

    //break lab1;
}
else{
    if((pt.getSysSleep()).equals("true"))
        pr = -1;
    else
        pr = pt.getPriority();

    for(int j=1;j<queue1.size();j++){
```
if(!((ProcessActInfo)queue1.get(j)).getSysSleep().equals("true")){
    if(pr<((ProcessActInfo)queue1.get(j)).getPriority(){
        pt = (ProcessActInfo)queue1.get(j);
        pr = pt.getPriority();
    }
}
}

This code takes the ready queue and checks which process has the highest priority. That process is chosen to be serviced. A process in sleep mode is not considered to run. If there is only process in the ready, which is in sleep state then nothing will be served for that time. Graph is drawn using the time as x axis and process number as y axis. This is achieved by fillRect() method in java. This method takes arguments as x coordinate, y coordinate, height and width.

Dispatch table is highlighted by selecting a particular row based on priority of process. Code snippet is:

    for(int row=0;row<10;row++){
        disTable.setValueAt("", row, 0);
    }
    if(flag==true){
        disTable.setRowSelectionInterval(pt.getPriority(), pt.getPriority());
        disTable.setValueAt(pt.getName(), pt.getPriority(), 0); }

If a process uses up its entire allocated time quantum then priority of process decreases, which is selected from the dispatch table. This is made sure by checking the time quantum left for that process. If the total time left to complete the process is 0 then process is inserted into finish queue. The code snippet as below:

```java
if(pt.getTQleft() == 0 && pt.getTimeLeft() != 0) {
    // compare priority from text area and dispatch table
    // if they match, get time quantum expired. i.e new priority
    for (int row = 0; row < 10; row++) {
        Object valuePr = disTable.getValueAt(row, 1);
        int parsedV = Integer.valueOf((String) valuePr);

        if (pt.getPriority() == parsedV) {
            Object valueNewPr = disTable.getValueAt(row, 3);
            parsedNwPr = Integer.valueOf((String) valueNewPr);
            pt.setPriority(parsedNwPr);  // update new Priority

            Object valueTQ = disTable.getValueAt(parsedNwPr, 2);
            parsedNwTQ = Integer.valueOf((String) valueTQ);
            pt.setTQleft(parsedNwTQ);  // update new time quantum

            break;
        }
    }
}
```
else if(pt.getTimeLeft()==0)
{
    updateGui("At Time " + (clk + 1) + ":Process " + pt.getName() + " done.");
    queue1.remove(pt);
    queue2.add(pt);
}

During the execution of any process, it goes through different priorities. Sometimes its priority is decremented, sometime it increments and other times it remains same. A statistics is shown after algorithm finishes executing all the processes. A 2D array is used to store values whenever priority changes.

pt.getCounter();

arr[pt.getIdx()][pt.getCt1()]=pt.getPriority();

The following code displays the results on the user interface:

for(int l=0;l<statArr.size();l++){
    String res;
    pt = (ProcessActInfo)statArr.get(l);
    intupto = pt.getCt1();
String prname = pt.getName();
res = prname + " ------>
for(int m=0;m<=upto;m++){  
   res+= " " + arr[1][m];  
}
resultText.append(res + "\n");
}

The following screen shot shows an applet, which has completed execution.

Figure 8. Run Button Example
As you can see, dispatch table has one highlighted row, which is for the process that executed the last. Status text shows the history of complete algorithm. Result text shows the information about the changes in process priorities. Graph shows a visual representation of when each process executes.

Step in Method:
This method does not implement threads. It is executed whenever the user clicks on Step in button. I have used two flags that will keep a track of user click. One of the flag is set when the user clicks Step in for the first time and initializes some of the variables which are to be set only once during execution. After then second variables and finish queue is checked to keep the applet running. If the finish queue equals to the number of processes entered then applet stops executing.

The following screenshot shows a partial execution of Step in method. I have clicked step in button 3-4 times and so this is the state of applet. Run button is disabled at this point. Result text will display information only when algorithm completes its execution.
3.5 Principle of Locality

3.5.1 Concept of Principle of Locality

Virtual memory is an extension to main memory, which removes the burden of small, limited amount of memory and allows safe sharing of memory among multiple programs [3]. At any particular time rather than whole program, only a chunk of the program is present in the main memory. As required, the next chunk of program will be moved to main memory. If again the previous chunk is needed, it will be moved back into main memory. This leads to increase in disk access and accessing a disk is an expensive and time-consuming operation as compared to main memory access. Principle of locality was born to make virtual memory systems efficient [4].
There are two kinds of Locality:

- **Temporal locality**: Temporal locality exhibits that if a particular memory location is referenced, then it is likely that the same location will be referenced again in the near future.

- **Spatial locality**: It indicates that if a particular memory location is accessed, the likelihood that its nearby locations will be referenced in the near future is higher.

In the Figure 10 shown below, small squares in the circle exhibits spatial locality whereas the squares in different colors, which are scattered exhibits temporal locality.

**Figure 10.** Graph Representation of Temporal and Spatial Locality
3.5.2 Applet Design

Similar to Solaris TS algorithm, this applet is also designed by using Java Swing components. There are different java components like textbox, label, button, spinner and checkbox. A JPanel is placed on JApplet, which acts a container for above-mentioned components. There are 3 JPanels for temporal locality, spatial locality and image for graph. Graph is plotted on this image, which is shown below:

Figure 11. Graph Image
Temporal Locality: The concept taken here is based on the execution of ‘for loop’. In a ‘for loop’ we give start and end value, similarly this applet lets the user enter values of how long the loop should execute. There is a text box for Page address, where the pages that will be accessed during the execution should be entered. Another text box is of String of Page address, consisting of a set of pages that will accessed continuously during loop execution. This is done to exhibit the concept of temporal locality that a page will tend to be accessed in near future. There is a loop2 with the same above user inputs.

Spatial Locality: The approach used here is execution of an array. An array is declared by assigning a fixed size to it. This applet asks user to enter the size of array, the starting point where array should start executing. Once the user enters these two values, Page address text box is populated with page addresses that user can select to enter in the String of page address text box. Page address outside of this range will give an error message to enter appropriate values. This applet will exhibit that nearby address locations are accessed frequently.

At a time, either temporal locality applet can execute or spatial locality and graph will be plotted for the same. Speed bar is use to control the speed of plotting of graph, its maximum value is 50 and minimum is 10. Start button is provided to start the execution of applet. The last panel consists of a graph, where x axis represents time and y axis represents page address. If the loop should run from 1 to 4, the graph plotting will start from 1 on a axis and corresponding y axis. The point at which they intersect, a small dot
like rectangle is plotted. Status text will show which page address is accessed at a particular time.

Screen shot is as below:

![Screen shot of a principle of locality](image)

**Figure 12. Principle of Locality Screen Shot**

### 3.5.3 Applet Implementation

![Class Relationships diagram](image)

**Figure 13. Class Relationships**
The tools used here are same as Solaris TS applet mentioned in section 3.4.4. The implementation is done using two classes PrinciplesOfLocality.java and DrawObject.java. DrawObject class is used to get the x coordinates, y coordinates, height and width of points to be plotted. It is used to create object and draw graph using points.

An ideal programming practice is to have a different class to draw an applet that is done here. User has to choose which locality should be displayed, this can be done by clicking the checkbox. Selection of locality will enable text boxes for that particular locality. Values are required to be entered in the five text boxes, two values for duration of i, page address and string of page addresses.

Speed bar value can be set, it can be changed when applet is in execution because of the use of multithreading. How multithreading is used to implement speed bar and plotting graph will be explained in the following section using code snippets how multithreading works. To start running the applet, click on Start button. ActionPerformed method is called to run the applet. First, if no Locality check box is selected, user is prompted by a message to select one.

Both the start and end values of ‘i’ are validated if they are not blank, start value should not be greater than end value as this is an example of incremental loop.

Page addresses are splitted by comma and stored in string or integer ArrayList. Following code snippet shows that:

```java
Collections.addAll(list1Str, textFieldPageAddress1.getText().split("","));
```
Collections.addAll(list1NewStr, textFieldPageStringAddress1.getText().split(","));

I have given the maximum value of 100 to be entered as page number and minimum is 0. This condition is validated to check the value. I have assumed that in each iteration of loop 3 pages will be accessed. Therefore, if length of loop is 3, pages in string of page address text box should be exactly 9. This condition is validated and suitable message is displayed. String of page addresses will contain only those addresses that are mentioned in Page address text box. An address which is not among the page address is not allowed be entered, validation is done for the same.

The image graph has static starting point, which forms the basis of plotting. This value is added to x axis to plot a point for each iteration. Code snippet is:

DrawObjectdobj;

    for (int i = loop1from; i <= loop1to; i++) {
        for (int j = 0; j < 3; j++) {
            // temp = 0;
            dobj = new DrawObject(axisX + (i * 50) + (j * 20), axisY - (list1NewInt.get(x) * 5), heightOfGraph, 7, true, list1NewInt.get(x));
            doal.add(dobj);
            x++;
        }
    }
An object called doal is defined for DrawObject class. This object will only create a list of points to be plotted and stores them in ArrayList. The plotting of graph is not yet started. If all the conditions are successfully validated then the applet will start running by overriding the run() method to implement thread functionality. It is needed to handle speed bar simultaneously while graph is drawn.

Code snippet is:

```java
try {
    java.awt.EventQueue.invokeLater(new Runnable() {

        @Override
        public void run() {

            initComponents();

            }

    });

} catch (Exception ex) {

    ex.printStackTrace();

}

This is an auto generated code by java for a run() method. This is called as UI thread.

I have created below thread to plot the graph.
@Override

public void run() {
    try {
        for (DrawObject a : doal) {
            doalNew.add(a);
            this.repaint();
            if (checkBoxLoop3.isSelected()) {
                if (list3CheckInt.contains(a.getActualValueY())) {
                    String temp = textAreaStatus3.getText();
                    temp += a.getActualValueY() + " Page Accessed Before\n";
                    textAreaStatus3.setText(temp);
                } else {
                    String temp = textAreaStatus3.getText();
                    temp += a.getActualValueY() + " Page Accessed First Time\n";
                    textAreaStatus3.setText(temp);
                }
                list3CheckInt.add(a.getActualValueY());
            } else {
                String temp = textAreaStatus3.getText();
                temp += a.getActualValueY() + " Page Accessed First Time\n";
                textAreaStatus3.setText(temp);
            }
        }
        Thread.sleep(5000 / speedValue);
    } catch (InterruptedException ex) {
        Logger.getLogger(PrinciplesOfLocality.class.getName()).log(Level.SEVERE, null, ex);
    }
}
When the Array list is ready, the UI thread will create another thread and the list will be provided to that thread. Now there will be two thread running in to the application while the plotting the graph points, one created by me which takes list of array objects and start plotting graph and another is UI thread which responds to the user when any action is done by to change the value of speed bar.

Description about paint(final Graphics g) method:

This method is over ridden to draw the graph on the plotted image. When the data is ready to be plotted on the graph, repaint() method is called by the applet which indirectly call the paint() method. When user clicks the start button it starts calculating the X axis, Y axis, Height of graph, Width of graph and a Boolean variable to represent the graph to be drawn. It checks which X and Y axis points needs to be shown with their specified height and width. To determine the initial position, it uses a static position of the panel after the increasing the value of Y to show it on Y axis and decrease the X axis value to plot the graph. The X and Y position is decided based on the value of Loop (value of I) and height and width of the every point. It will be plotted on panel using the java applet function “fillRect” method which takes X coordinate, Y coordinate, height and width of the point. Code snippet as follows:
@Override

    public void paint(final Graphics g) {

super.paint(g);

g.setColor(new Color(0, 148, 255));

    if (doalNew != null) {

for (DrawObject a : doalNew) {

    if (a.isShouldShow()) {

    g.fillRect(a.getAxisX(), a.getAxisY(), a.getHeight(), a.getWidth());

    } } }

As mentioned above, in each iteration 3 addresses will be accessed. First address will always be on the left of the column, second in the middle and third on the right. This makes it clear which address is accessed first.
4.1 Solaris TS Applet

I have entered three processes here. At least one process should have arrival time of zero.

The speed bar value is set to 50. Result shows the priority changes of all the processes.

Figure 14. Result 1 of Solaris TS Applet
Here, I have entered four processes. At least one process should have arrival time of zero.

The speed bar value is set to 30. Result shows the priority changes of all the processes.

Figure 15. Result 2 of Solaris TS Applet
4.2 Locality of Reference

4.2.1 Temporal Locality

Select the checkbox for which the graph is to be plotted. Below screenshot shows the plotting of Temporal Locality for two loops.

Figure 16. Result 1 of Temporal Locality
Below screenshot shows the plotting of Temporal Locality only for one loop:

Figure 17. Result 2 of Temporal Locality
4.2.2 Spatial Locality

Starting point of array is 36 here; graph will start plotting from that point. This shows that the concept of spatial locality is preserved.

Figure 18. Result 1 of Spatial Locality
Starting point of array is 20 here; graph will start plotting from that point.

Figure 19. Result 2 of Spatial Locality
Chapter 5

FUTURE WORK

The current implementation of Solaris TS applet allows user to enter one system call. This applet can be extended by implementing more than one system call. A process in its life cycle will encounter more number of system calls. There are 60 priority queues, and current implementation has 10 priority queues. More number of priority queues can be implemented to show a large variety of priority levels.

In Locality of reference applet, user has to enter page address, which is directly used. An extension to this implementation can be that this page address will act as virtual page number and then mapped to physical page address. Graph can be plotted based on that. Another approach can be to use a small example, which consists of ‘for loop’ and array. The applet can be shown to iterate at each line of execution.
BIBLIOGRAPHY

[1] OSCAL
http://gaia.ecs.csus.edu/~zhangd/oscal/oscal.htm

[2] CPU scheduling
http://www.ayomaonline.com/academic/cpu-scheduling/

Computer Organization and Design by David A. Patterson and John L. Hennessy


[6] Paper by Adel M. Agina,
http://projects.edte.utwente.nl/pi/papers/AnimationPaper.html

[7] Animation website
http://williamstallings.com/OS/Animation/Animations.html

[8] Animation website
http://courses.cs.vt.edu/~csonline/OS/Lessons/index.html

[9] Java applets
http://docs.oracle.com/javase/tutorial/deployment/applet/

[10] Schildt, Herbert Java: The complete reference

http://www.agentgroup.unimo.it/didattica/curriculum/marco/MAIN/didattica/TecnInterne

tWeb/java/J4c_AppletWebstart.html

[12] Java Applets

http://support.novell.com/techcenter/articles/ana19970701.html