Final Year Project Report

Scheduling Disciplines Learning Tool

Dominic Carr

A thesis submitted in part fulfilment of the degree of

BSc (hons) in Computer Science

Supervisor: Dr Felix Balado

UCD School of Computer Science and Informatics
College of Engineering Mathematical and Physical Sciences
University College Dublin

May 6, 2010

Table of Contents

Abstract

Every operating system employs an algorithm of varying complexity in order to schedule the execution order of processes. This project is concerned with the development of a software tool which demonstrates the operation of several of the fundamental scheduling disciplines in an intuitive, interactive manner. Throughout this report the development of this tool will be explained from background reading through to a final evaluation from which conclusions are drawn and future work enumerated.

Acknowledgments

I would like to thank my supervisor Dr. Felix Balado for his guidance and feedback throughout the development of this project. His help was invaluable particularly during the report writing stages.

Project Specification

Subject: Learning tool for visualizing scheduling disciplines
Prerequisites: Operating Systems I
Project Type: Design and Implementation
Software Requirements: Java, C/C++, or any other suitable language
Hardware Requirements: None
Preassigned: No

Description

General Information: The objective of this project is to develop a visual learning tool to demonstrate process scheduling in an operating system. The tool should be interactive, so that the user can follow the evolution in time of the queue in detail.

Mandatory: Design and implementation of a GUI that will allow the user to choose between several scheduling disciplines. The program underlying this GUI will then generate an interactive animation showing how processes arriving at the queue are scheduled and processed. The evolution over time of relevant parameters will be displayed: average waiting time, average turnaround, average queue size etc. The system must implement at least two scheduling disciplines: First Come First Served (FCFS) and Shortest Job First (SJF). The processes arriving at the queue will be a tunable random mixture of CPU-bound and I/O-bound processes.

Discretionary: Implement the Shortest Remaining Time First (SRTF) and Round Robin (RR) scheduling disciplines.

Exceptional: Implement a module able to create priority queues.

Reading: Lecture notes of Operating Systems I, and references therein.

Chapter 1: **Introduction**

1.1 Project Overview

This project is concerned with the implementation and refinement of an interactive graphical learning tool to educate a student as regards the operation and performance of several, fundamental operating system (OS) scheduling disciplines. This tool implements a simulation of the First Come First Served, Shortest Job First, Shortest Remaining Time First, Round Robin scheduling disciplines and both the non-preemptive and preemptive variants of priority scheduling.

Operating system process scheduling is a key issue in operating system design. As such this is a key concept which computer science students must understand. Some students have difficulty understanding how these scheduling systems function from simply reading about them. It is the goal of this project to help in this area by providing a visual stimulus and by enabling the student to interact with the scheduling policy during the simulation. This interaction is achieved by allowing the user set up the simulation, adjust process properties and discipline settings during runtime and control the flow of the simulation.

The tool is based on a Monte Carlo simulation of process scheduling in which the processes properties are random and the process arrival times follow a Poisson distribution. The tool's graphical component visualizes the arrival, dispatch, execution and various state transitions of processes. Additionally the CPU, the various process queues and the process entry and exit points are visualized and explained.

A statistical breakdown of the disciplines performance is also displayed graphically. The tool provides at all times textual explanation of the events which are occurring. The graphical control interface allows the user to set a wide array of operational parameters such as the number of processes in the simulation. Some parameters may be tweaked during execution e.g., process priority. The user may also access additional information by clicking on the components of the simulation.

1.2 Report Overview

The structure of the remainder of this report is as follows.

Chapter 2 details the background research conducted throughout the duration of this project. The topics of operating system scheduling, scheduling disciplines, scheduling policy goals, process behavior and queuing theory will be detailed. Additionally a description of the implementation technology used and the reasons behind those choices are discussed.

Chapter 3 provides an overview of the design approach taken to developing the tool. This will detail the design stage of the project development. The most important and crucial software design decisions of the project are enumerated and discussed.

Chapter 4 provides a detailed breakdown of the main components of the project implementation. Detailed descriptions of the implementation of each component is given. These components include the scheduling disciplines, the scheduler, the statistical collection and the Graphical User Interface.

Chapter 5 discusses the testing, refinement and evaluation criteria used in the assessment of the implemented software tool. A discussion of the teaching merit of the tool and a comparison with other simulations is provided.

Chapter 6 provides a discussion of the project outcome, stating what has been achieved and to what degree the project has been successful. Finally future work is discussed involving the extension of the software tool, through the addition of functionality, the simulation of other scheduling systems and the application of queuing theory.

Chapter 2: **Background Research**

This chapter reviews the background research conducted throughout the course of this project. The project's key theoretical underpinnings are discussed such as operating system scheduling and performance analysis. Additionally the specifics of each discipline will be enumerated.

The primary source for the theoretical background research necessary for this project was [1] by William Stallings primarily Chapter 9. Therein Operating System (OS) process scheduling and the various scheduling disciplines under consideration, and the criteria for their assessment, are explained in detail. The knowledge acquired therein was supplemented by [2] by Andrew S. Tanenbaum and the lecture notes of the UCD module Operating Systems 1 (COMP30090) by Dr. Felix Balado and Dr. Tahar Kechadi [3].

2.1 Scheduling Disciplines

2.1.1 What Is A Scheduling Discipline?

In an operating system there are frequently many processes vying for use of the Central Processing Unit (CPU). Due to this a choice must be made as to which process executes next. This is handled by the short-term scheduler which implements a scheduling discipline to choose the next process to execute [1] [2]. A scheduling discipline is essentially an algorithm which implements a scheduling policy in accordance with a set of performance goals. These performance goals are measured by a statistical breakdown discussed in Section 2.2. Typically there are two other levels of scheduling: long-term scheduling and medium-term scheduling. The long-term scheduler handles the admission of processes to the ready queue. The medium-term scheduler decides which processes are to be held in main memory. Suspended processes are removed from main memory and placed in a suspended state. Though beyond the scope of the project specification the completed tool implements medium-term scheduling in addition to short-term scheduling.

2.1.2 Preemptive and Non-Preemptive Scheduling

There are two distinct categories of scheduling discipline, preemptive and non-preemptive. In non-preemptive scheduling disciplines, scheduling decisions only occur when a process voluntarily hands over control of the CPU [1]. For example a scheduling decision is made when the running process completes its CPU burst or exits the system.

In a preemptive discipline in addition to scheduling decisions being made when a process relinquishes the CPU, decisions are also made based on external factors such as the arrival of a new process at the ready queue or if the current process is running for too long. When these events occur the current process may be preempted. Preemption is performed in accordance with the performance goals of the scheduling discipline. It is an important mechanism in allowing these goals to be met, particularly in the case of interactivity. An example of this

would be the preemption used in Round Robin scheduling which is designed to minimize the average response time, these issues are discussed in Section 2.2.

2.1.3 Processes

A process may be defined in many different ways. The definition most befitting of this project is given in [1], a process is an "entity that can be assigned to and executed on a processor". It is very important to consider the statistical behavior of the processes in a system, since this greatly influences system performance for the same scheduling discipline. A convenient simplification of process behaviour for educational purposes is to consider that there are only two different kinds as regards their CPU-burst pattern: CPU bound and I/O bound. These abstractions of process behaviour are discussed next. Further discussion of process types as related to scheduling disciplines can be found in Section 2.3.

CPU Bound Process
CPU bound processes perform a great amount of computation, in the CPU, and tend to perform only a small amount of I/O. Typically the execution of a CPU bound process consists of several long CPU bursts, separated by I/O bursts. This can be seen in Figure 2.1.

Figure 2.1: typical burst pattern of a CPU bound process

I/O Bound Process
The burst pattern of an I/O bound process has many I/O bursts, these are periods when a process is waiting for I/O e.g., user input, printer. These I/O bursts separated by short CPU bursts to process the input [2]. This behavior can be seen in Figure 2.2.

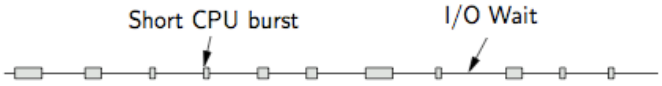

Figure 2.2: typical burst pattern of a I/O bound process

Poisson Distribution

It is not only the burst pattern of the processes which affects performance, but also the pattern of the process arrivals. In order to achieve a somewhat realistic model in the simulation process arrivals follow a Poisson distribution. The Poisson distribution is a discrete probability distribution, used in queuing theory to model random independent arrivals over some time period [1]. This distribution is used for arrival times as it is the most natural method for the generation of random events that appear at a uniform rate in continuous time. The key parameter in this distribution is λ which represents the average process arrivals/second. As defined in [16] the Poisson distribution typifies the "number of occurrences, per unit time, of an event that can occur at any instant of time". This gives us exponentially distributed process inter-arrival times with $1/\lambda$ seconds between arrivals on average. The calculation of Poisson probabilities is given in Formula (2.1), where T is some time interval.

$$Pr[k \text{ arrivals in } T \text{ seconds}] = \frac{(\lambda T)^k}{k!} \exp(-\lambda T) \tag{2.1}$$

2.2 Scheduling Parameters

There are many evaluation metrics employed to assess the performance of a scheduling discipline. Each discipline has as its performance goals a subset of these criteria, these goals are further discussed in Section 2.3. For the purposes of this project the most commonly used criteria are calculated, displayed and updated in the application as time passes.

These criteria show the general performance strengths and weaknesses of a given discipline. Though it is worth noting that a discipline's performance is greatly affected by the pattern of process arrival times and the makeup of the specific set of processes, see Section 2.1.3. For example Shortest Job First is optimal with regard to average process waiting time [1], whereas Round Robin performs very well with regard to average response time but may perform poorly in terms of turnaround time due to the overhead of process switching. These disciplines are discussed in Section 2.3.

The evaluation metrics which are considered in this project are as follows:

Response Time
This is the measure of the time interval from a process' arrival at the ready queue and the start of its execution [1]. This metric is very important from the users perspective because when a process starts running they will receive some feedback. In essence the shorter the average response time the better the user's experience [1].

Waiting Time
This is the measure of the time a process spends waiting in the ready queue over the course of its its entire lifetime [2].

Turnaround Time
This is the measure of the time taken between process arrival at the ready queue and the completion of its execution. This is considered as the main criteria for assessment of performance in a batch processing system [2].

Normalized Turnaround Time
This is defined in [1] as the "ratio of turnaround time to service time" where the service time is the total amount of CPU time the process required. Its calculation is given in Formula (2.2). This metric characterizes the process' waiting time relative to the length of its execution.

$$\text{normalized turnaround} = \frac{\text{turnaround}}{\text{service time}} \tag{2.2}$$

Ready Queue Size
This is the measure of the average size of the ready queue. It is important to dimension a queueing system, as it quantifies the average amount of processes waiting to execute. A discipline such as Shortest Job First attempts to minimize the number of waiting processes, whereas using First Come First Served may result in a large queue size.

Throughput
As defined in [1] throughput is the "number of processes completed per unit of time". The primary aim of the Shortest Job First and Shortest Remaining Time First disciplines is to maximize process throughput.

Processor Utilization
This is the percentage of the total system time for which the CPU is busy running processes i.e., performing useful work [1]. Utilization calculation is given in Formula (2.3). This criteria,

while still an important consideration, is less important than those detailed above in regards to the standard single-user system [1].

$$\text{utilization} = \frac{\text{busy time}}{\text{total time}} \times 100 \tag{2.3}$$

Percentage of Processor Idle Time
This is the percentage of total system time during which the CPU is not performing any useful work. The processor is deemed idle when a process is being timed out or when the processor is waiting for the arrival of a new process to run. The time taken to switch one process to another is called a context switch. The length of this switch is very important in a preemptive discipline as it adds much additional overhead. This can be calculated as simply the complement of the percentage of time for which the CPU is busy i.e., 100 − utilization.

2.3 Disciplines Under Consideration

In this section the principles of the scheduling systems implemented in this tool are described.

First Come First Served

This is the simplest of the non-preemptive scheduling disciplines. Under this discipline, processes are assigned to the CPU in the order of arrival at the ready queue [2]. Upon completion of the current process' CPU burst it becomes blocked performing I/O operations and the process at the head of the queue is dispatched. All processes arriving at the ready queue are placed at the end. This discipline performs poorly in the case where an I/O bound process must wait while a CPU bound process completes a lengthy CPU burst, this can result in poor throughput, high average waiting and response time [1] etc.

Shortest Job First

In this discipline, the waiting process with the shortest expected CPU burst time is chosen to run [2]. This is a non-premptive discipline, the preemptive variant is discussed below. As illustrated in [2] the Shortest Job First algorithm maximizes the throughput of the system. The average waiting time for short processes is also minimized. This discipline either assumes the burst time of a process is known in advance or each process' burst time must be estimated based on prior burst behavior [1]. In a real world operating system Shortest Job First is very difficult to implement, due to the problems associated with accurate prediction. For this project the next burst time of all processes are estimated based on their prior burst times, this is done through exponential averaging. This estimation method is detailed below.

Next Burst Time Calculation
In the case of interactive processes and many batch processes it is not feasible to know the next burst time of a process, as such it must be estimated [1] [3]. This is done by using the past burst behavior of the process as a guideline. The next burst time is typically estimated by way of an exponential averaging of all the previous burst times of the process [1]. This estimation is calculated in Formula (2.4), adapted from [3].

$$T_{n+1} = \sum_{k=0}^{(n-1)} (1-\alpha)^k \alpha t_{n-k} + (1-\alpha)^n T_1 \tag{2.4}$$

Where t_n is the length of the process' nth CPU burst and T_{n+1} is the estimated length of the process' n + 1th CPU burst. α is bounded $0 \leq \alpha \leq 1$. Assigning different values to α gives a varying degree of weight to the previous burst times. For example if $\alpha = 1$, only the most recent burst time is taken into account [3]. T_1 is an initialization value for the recursive estimation formula.

Shortest Remaining Time First

This is the preemptive variant of the Shortest Job First discipline. The scheduler always chooses to run the process with the least amount of CPU time remaining [2].

This discipline functions equivalently to Shortest Job First, except when a process enters the ready queue. In this case the predicted CPU burst time of the new process is compared with the predicted remaining burst time of the current process. Should the CPU burst of the new process be shorter the current process is preempted and the new process dispatched. This preemption lowers the level of CPU utilization, due to the overheads discussed in Section 2.2.

Round Robin

In this preemptive discipline a fixed, equal slice of CPU time, a quantum, is given to each process in turn [1]. A process is allowed to execute for one quantum, if its CPU burst is not finished when the quantum has elapsed then the process is preempted and the next process is run for its allocated quantum. If a process finishes execution, or completes its CPU burst, before the time quantum elapses the next process will be run immediately [1]. The algorithm cycles through the processes in the ready queue in a FIFO manner [1]. Round Robin typically produces the shortest average response time, of course this is dependent on the use of an appropriate time quantum.

As discussed at length in [2] the selection of an appropriate quantum is a key issue, as for a large enough quantum the discipline essentially becomes First Come First Served. As discussed in Section 2.2 a short quantum results in a lot of context switches and as a result, poor CPU utilization.

Priority Queue

In this discipline a priority, a number denoting relative importance, is given to each process. The scheduler dispatches the process of highest priority to the CPU. A priority based discipline provides good performance for high priority processes e.g., low waiting time, short response time etc. Priority based scheduling implementations may be preemptive or non-preemptive.

- In the non-preemptive case, when the current process relinquishes the CPU all the processes on the ready queue are compared and the highest priority process is chosen.
- In the preemptive case when a process joins the ready queue its priority is checked against that of the running process and if the new process is of greater priority it is dispatched.

Priority based scheduling disciplines are prone to process starvation. This means that processes of low priority never receive CPU time [1]. To combat this problem a process which has been waiting for CPU time in excess of some predefined threshold will have its priority increased.

2.4 Implementation Research

As this was primarily an implementation project it was necessary to conduct a degree of research into the best programming language and associated tools to be used for development of the learning tool. Several alternative languages were considered such as C++ and Python, however it was decided that Java would be the most appropriate, given its cross-platform nature and the extensive, well-documented and mature built-in graphics libraries (Swing and AWT [4]). C++ and Python have similar capabilities however C++ lacks a platform independent or well-documented graphics library and Python suffers from compatibility issues, related to different releases of the language. Development was done on the Mac OSX platform using Java 1.6 within the Eclipse IDE.

In the scheduling simulation we need a method of modeling the passage of time. This functionality is provided via the `Swing Timer` class [5]. Further to this it allows for variation in the time between updates, so the speed of the simulation can be tuned. This can be done in response to user demands which aids interactivity. This class also provides an easily implemented means of pausing and resuming the application upon user request.

Animation

Modules were identified in the `Swing` [4] and `Java2D` [10] libraries, which contained the necessary tools to facilitate the creation of a professional quality Graphical User Interface (GUI). It was necessary that research be conducted into the tools available for the creation of animations within Java. This involved the creation of some small rudimentary animations and the analysis of the `Java2D` library documentation accessed at [10].

Similarly research was conducted into image creation software. This was necessary as the project has a significant graphical component attached to it and it was important that this interface be of a high quality while still maintaining clarity. The conclusion was reached that Photoshop would be the software package most suitable for the construction of the custom graphical elements desired for the project.

2.5 Learning Tool

As the primary goal of this project is the creation of a successful learning tool it was identified early on that there should be a period of user testing. During this period a number of individuals used the application and assessed its ease of use and teaching merit, these issues are further discussed in Chapter 5. It was hoped that the feedback received from this would enable the improvement of the tool.

2.5.1 Visualization Techniques

The single most important element enabling learning in a simulation of this nature is the primary visualization, as such research was done in order to find an appropriate visualization technique and inform the design of the GUI in general.

In [8] and [9] some useful user interface design guidelines were provided which informed the layout of components in the initial GUI prototype. The primary lesson learned from these materials was that user interfaces should be developed such that they mirror somewhat other interfaces with which the user's are comfortable with.

In trying to identify an appropriate visualization for all the components of operating system scheduling, the various ways in which seminal texts in the field had visualized these components in still imagery were analysed. In this way it was hoped a clear visualization could be developed which would be easily comprehensible to students. Many of the texts such as [1] and [2] employed similar imagery, which is simultaneously simple, clear and effective. Typically these texts used either a Gantt chart or a queuing diagram. Various simulations of process scheduling were analyzed such as those available at [12] [15]. These other scheduling simulations are discussed at length in Section 5.2.2. Particularly helpful was a clear and simple animation accessed at [6], notable also as it is targeted towards individuals unfamiliar with the subject matter.

Upon completion of background research into the various scheduling disciplines the conclusion was reached that while each discipline operates differently they can be visualized in much the same way. Alterations should be made in each case to emphasise their unique manner of operation, through a combination of textual explanation and graphical pointers. Figure 2.3 shows the prototype GUI which was initially developed. Processes waiting in the ready queue can be seen as well as a process in execution on the CPU. Also visible are the lines of transition which processes can follow and the various other queues.

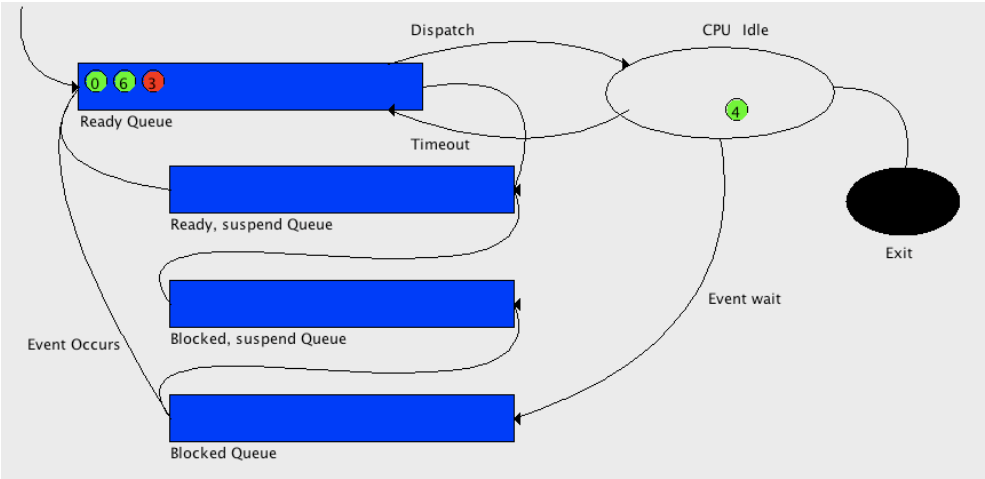

Figure 2.3: Learning tool prototype Graphical User Interface

2.6 Queuing Theory

Though it was not a part of the project specification some consideration was given to applying queuing theory to the project in order to enrich the learning tool. Queuing theory is a means of mathematically modeling real-world queuing systems [1] [3]. In the scope of this project

this means the queue of waiting processes. Queuing theory and system simulation, which is what this project implements, are the two typical means of accurately modeling system behavior [1].

2.6.1 M/M/1 Model

The simplest queuing model which we may consider is the M/M/1 model. M/M/1 follows from Kendalls three-part notation [1] for queuing systems, which specifies the inter arrival time distribution, service time distribution and the number of servers respectively. M denotes an exponential distribution, and therefore in this queueing model process arrivals follow a Poisson distribution, see Section 2.1. Additionally process service times follow an exponential distribution and there is one server. The server provides a service to the waiting items e.g., a processor which runs waiting processes. Models such as this one conform to the assumption that the scheduling discipline employed is First Come First Served. Some of the difficulties associated with linking the performance of the implemented simulation to these theoretical models is discussed in Section 6.2.3.

Chapter 3: **General Approach and Architecture**

Upon completion of the necessary background research and having gained from this knowledge and insight into the problem domain, development must move toward to solidifying an approach to the problem and conceiving of an overarching design for the project. Throughout this process the primary goal of the project as a undergraduate level learning tool was the foremost consideration.

3.1 Design Decisions

3.1.1 Monte Carlo Simulation

In designing the overall operation of the learning tool it was decided the scheduling simulation would function as a Monte Carlo simulation [17]. This was perhaps the most important design decision. A Monte Carlo simulation repeats pseudorandom experiments and computes statistics from them, it is a widely used tool for the mathematical modeling of real-world systems. This is achieved as the process arrivals are Poisson (discussed in Section 2.1), the type and properties of each process is random, and statistics are computed from these processes when appropriate i.e., a process is only used in computing response time statistics after its initial CPU burst.

Consideration was given to an alternate design in which the user would specify the properties of each individual process e.g., arrival time, priority, length. This design would have resulted in a more deterministic tool, that is to say the user would be able to predict the outcome of each choice made in scheduling. However the conclusion was reached that this design would be of lesser educational value and further removed from the operation of a real operating system. The tool would have been completely different had it been implemented in this way. The statistics collected would be skewed by the user's selection. This would have also added additional complexity to any application of queuing theory.

3.1.2 Statistical Display

As a core part of the project specification was the display of relevant performance statistics, which are discussed in Section 2.2, it was important to plan the manner of their collection and presentation. The statistical collection simply performs comparison and averaging of process properties and simulation statistics. It was decided that, in addition to a textual display, there should be an intuitive graphical display of each of the statistics showing the overall average value and the value for each relevant process. Furthermore it was suggested the average value per type of process be shown, as this would be a powerful tool to show the performance of the two varieties of process under different disciplines and conditions. The graphical display chosen was a line graph. Graphs such as these are a proven tool for displaying and interpreting data. The statistical display allows the user to track the simulation in a quantitative manner.

In addition to the average values of the metrics discussed in Section 2.2 often taken into consideration are the maximum and minimum observed values as well as the standard deviation. These parameters help to quantify the variability in performance. As such these values are also calculated and displayed as appropriate.

The evolution of the above parameters over time is displayed on a line graph, where the x-axis shows each process and the y-axis is the range between the parameters maximum and minimum values. Dashed lines are used to display the overall averages and the average per process type. It is important that students be able to understand these parameters and their importance, as such some textual explanation is given.

As to provide some context to the above parameters, particularly the CPU performance metrics, the elapsed simulation time will also be displayed.

3.1.3 Control Components

When planning key interactive components such as the control panel decisions had to be made as to which elements of the simulation the user would be able to control. It was decided that the simulation be finite in terms of the number of processes, as this would allow the user to compare the performance of the various disciplines. As such the user needed to control the number of processes in the simulation. Following from this as per the design specification it was important that the user be able to define the ratio of I/O bound and CPU bound processes present in the simulation. Figure 3.1 shows in statistical terms what happens when the user chooses a distribution with 75%($p = 0.75$) CPU bound processes. In this case, the burst time of a generated process is more likely to lie in the second uniform range.

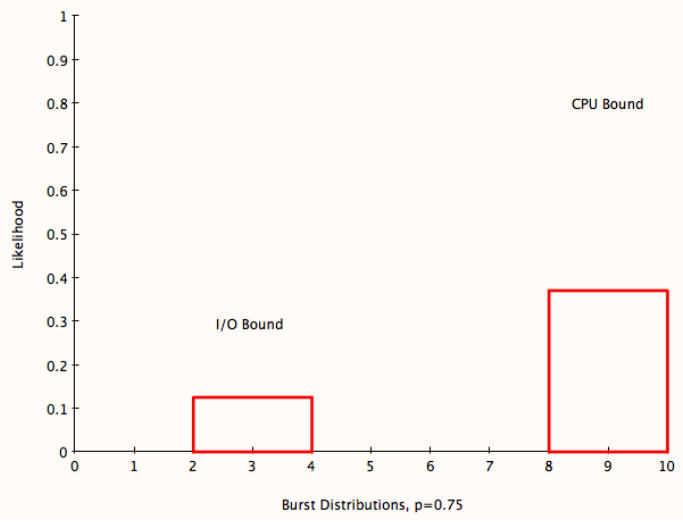

Figure 3.1: Burst time distribution, p=0.75

Looking at each scheduling discipline under consideration there are some parameters specific to each, as such it was decided that in the case of Round Robin scheduling the user be able to specify the allocated time quantum. Similarly in the case of Shortest Job First the user must be able to specify the α value, see Section 2.3. As the overhead associated with process preemption is an important performance variable the decision was made to allow the user to modify the context switch time.

In terms of providing useful runtime interaction it was decided that the user would be able to pause the simulation and step through each CPU cycle. This would allow the user to stop and examine the statistics at a given time point or examine the processes in order to ascertain

more clearly why specific decisions were made e.g., examine estimated burst times, priorities etc. In order to add an interactive component to the statistical graph display it was decided some method of accessing process information from inside the graphs be provided.

To maximize the level of user interaction it was decided that the user be able to click on the various visual components of the simulation in order to access additional explanation. In the case of processes it was decided that when a priority based scheme was in use the user be able to modify its priority.

Additionally consideration was given to allowing the user to modify expected burst time. However it was felt that this would interfere with educating the student about the exponential averaging formulas used to predict burst times. It was felt so as to not overwhelm the student with information that only process information relevant to the current discipline be shown. For example a process' priority may be hidden when it is not relevant to scheduling decisions.

3.1.4 Visualization Design

Firstly the main components which needed to be present in the simulation were identified. These were the process queues, the CPU, the individual processes and the process state transitions. Additionally some method of showing process execution would be needed. As discussed in Section 2.5.1, the layout of these components was heavily influenced by diagrams used in several operating systems texts such as [1]. The scheduling diagram from [1] can be seen in Figure 3.2.

Gantt Chart

The visualization detailed above was chosen over other common scheduling visualization schemes such as Gantt charts [1]. This was due the inability of Gantt charts to show process arrival and state transitions in a clear and easily understood manner. This means it is harder to showcase why scheduling decisions are made. In order to show these elements other disjoint components such as a queue and CPU visualization would need to be shown alongside the Gantt chart. As such there would not be a single coherent visualization. The conclusion was also reached that a Gantt chart would not provide as immersive a simulation as the implemented scheme. Techniques used in other simulations are discussed in Section 5.2.2.

3.1.5 Domain Decomposition

A broad breakdown of the main functional components of a scheduling system was performed. These were found to be the scheduler, the scheduling disciplines, the statistics collection (the computation and storage of performance metrics), the processes and their associated GUI's. The scheduler would be responsible for process generation, process suspension and maintaining the process queues. The scheduler would employ the scheduling discipline which chooses which process to execute when appropriate. The scheduler would employ a statistics collection mechanism to collect and store statistical data. It was decided each process would maintain its state internally, so the scheduler could easily check for changes in state. Alternatively consideration was given to making use of the Observer-Observable pattern in which a process would actively notify the scheduler of state changes but it was felt this would add unnecessary complexity.

The current scheduling discipline and the scheduler would update the primary GUI, the

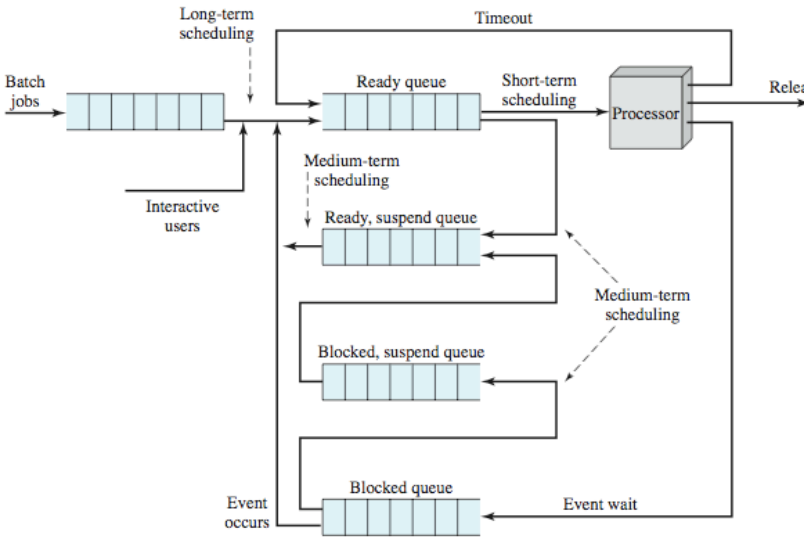

Figure 3.2: The queuing diagram used in [1]

interactive animation. The statistics class would control its own GUI, which would be hidden until accessed by the user. The decision was made to omit an explicit CPU class, as it is only necessary to maintain a reference to the currently running process. Figure 3.3 shows the breakdown of the main project components and their interaction.

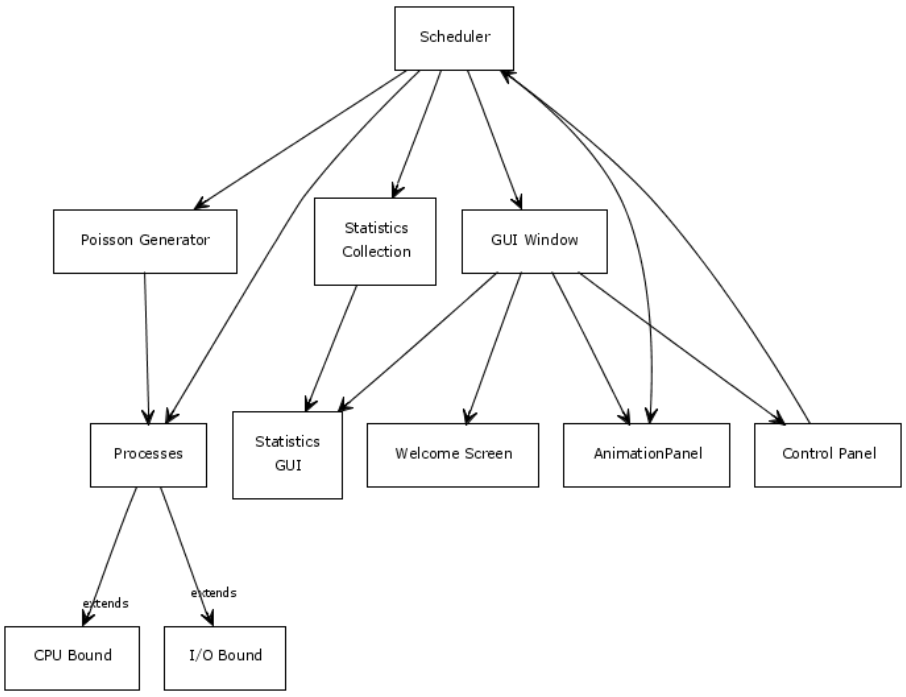

Figure 3.3: Project components and their interactions

3.2 Teaching Merit

To be successful as a learning tool the software package would need to have a sufficient degree of teaching merit. A number of key considerations were made with this in mind.

- It was clear the animation should be intuitive, this is another reason for basing the design upon common educational tools used in textbooks. The control system should also be intuitive, familiar even.
- The presented information should be tiered, this is a commonality among successful educational applications. As such the user is not overwhelmed or constantly reminded once they have learned why certain steps are performed. In this way the explanation given in the simulation should be concise while allowing a less experienced user to access additional content through tooltip text and extra panels.
- The clear visualization of statistical data and the process scheduling was also deemed to be key. This is achieved through an intuitive animation and statistical display.

Chapter 4: **Detailed Design and Implementation**

4.1 Implementation Overview

Upon completion of background reading and having identified the main design goals and overarching structure of the learning tool, the necessary components of the project where broken down into logical function units. These formed a class breakdown for the software. This breakdown was performed in an effort to aid future extensibility, for example by implementing each discipline in its own class allowing for easy addition of another discipline in future.

The implementation timeline was as follows. Firstly a rudimentary scheduler was implemented which created a list of processes to be run, methods were added to manage the process arrival, dispatch, blocking etc. Next the First Come First Served discipline was implemented. This discipline was implemented first to provide a solid basis for the testing of the GUI and the development of the other disciplines.

Following successful integration, an initial GUI and statistical collection class were implemented. When these components were tested and an initial working system was realized the other disciplines were implemented. This completed prototype was then tested by users and refinements and bug fixes were applied, testing is discussed in Chapter 5. In the following sections a detailed description of the implementation of the core components of the project will be given.

4.2 Process Implementation

Two different concrete process classes `CPUBoundProcess` and `IOBoundProcess` were implemented, see Section 2.1.3. These classes extend from an abstract base class `CPUProcess`. These two classes have varying bounds on the percentage of their generated total length for which they must run on the CPU, this is achieved by generating individual CPU bursts using a pseudorandom number generator, Java's built-in `Math.random()`, with an upper and lower bound. In this way a varying behavior is observed in which CPU bound processes have much longer burst times and shorter I/O waits. These classes also define separate icons and display colors, so the user may easily differentiate. By making use of a common base class and interface methods the scheduler and the GUI may treat and represent these processes similarly.

As previously discussed process interarrival times follow an Exponential distribution. These arrival times are set by the scheduler when each process is generated. Processes estimate their own next burst time when a new burst is generated, this is done as per Formula 2.4. Processes maintain their state internally by way of an enumerated type. Public methods of the `CPUProcess` class are used to update process state, these are called primarily by the `Scheduler` class. The most important of these are the `waitToRun()` and `execute()` methods. The `execute()` method is invoked on the currently running process. This is used to decrement

burst length and perform a check to see if the burst is complete. The `waitToRun()` is invoked on the waiting processes. This method performs different actions depending on the processes state e.g., if the process is blocked a check is performed to see if the I/O is complete.

Several of the GUI classes are tied to the process implementation. The `ProcessPanel` class provides the standard visualization of a process, this can be seen in Figure 4.1. This panel may be clicked to in order to display the `ProcessControlPanel` which allows the user to view additional information about a process. When a priority based discipline is employed the user may modify a process' priority using a `JSlider` in the `ProcessControlPanel`. The information displayed in the `ProcessControlPanel` is specific to the scheduling discipline in use. In the case of Shortest Job First information relating to burst time predication is shown. This can be seen in Figure 4.3.

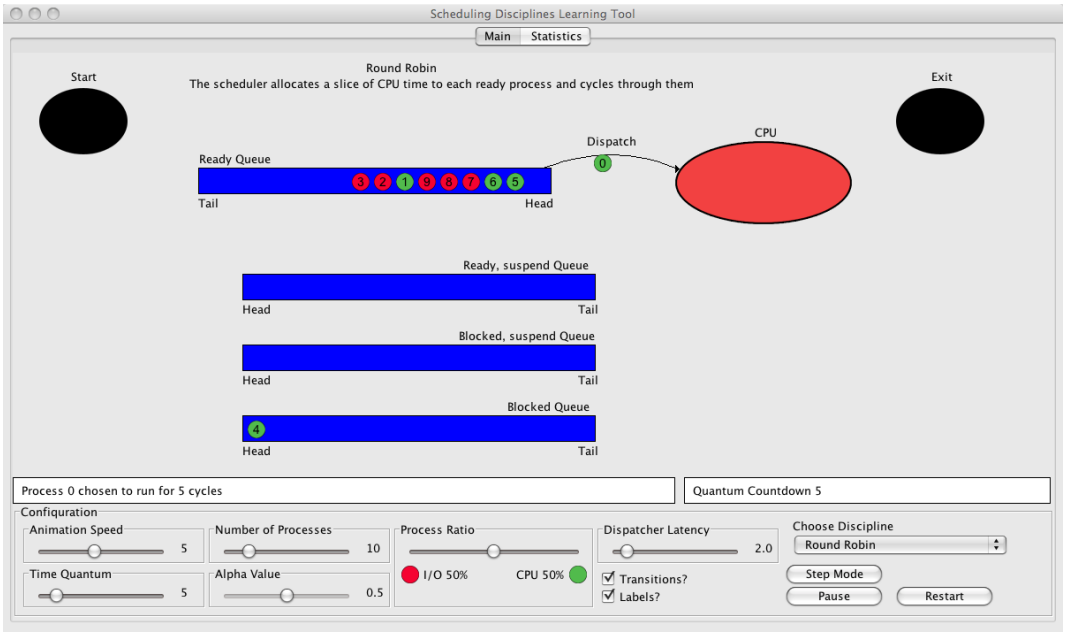

Figure 4.1: The main visualization

4.3 Scheduler

The `Scheduler` class performs the functions of the medium and short-term schedulers, see Section 2.1.1. The `Scheduler` class contains a member variable of the abstract class `SchedulingDiscipline` class which can be instantiated as any of the implemented scheduling disciplines as per the choice of the user. As all scheduling disciplines provide the same interface there is no need for any other code modification. Methods are provided in the `Scheduler` for use by the scheduling discipline in order to timeout or dispatch (`setCurrent()`) a process.

When the simulation is started the `Scheduler` class generates the processes according to the amount and type ratio set by the user (`createQueue()` method). The processes arrival times are then set, this is done using the `ExponentialRandom` class which generates a pseudorandom sequence following a Poisson distribution, this implementation is discussed below.

The `Scheduler` performs operations based on a `Timer` object which executes the `performSchedule()` method as per the users selected speed. Each time the `Timer` executes the scheduler performs a variety of operations. Methods are invoked to manage the arrival of processes, check

for changes in process state, execute the current process if applicable, suspend processes (as discussed below) etc. Processes are stored in separate `ArrayLists` as per their current state and corresponding queue, this convenience allows for the appropriate list of processes to be sent to a suitable `Statistics` class method. Additionally the `schedule()` method of the current discipline is invoked. The discipline then performs an action if it is in accordance with its own goals. The operation of each discipline is discussed in Section 4.4.

In association with each action taken the `Scheduler` invokes a animation and/or textual update to the GUI. Animation updates are invoked on the `AnimationPanel` through one of provided animation methods. Textual updates are made to one of two text display panels. These panels contain a component allowing the user to access additional expanded information regarding a given event (through buttons marked by a question mark graphic), These panels are shown in Figure 4.1. The second of these text display panels is used to show discipline specific information such as the quantum time in the Round Robin discipline.

Exponential Random Variable Generation

This class provides one public method, `nextExponential()` which makes use of Java's built-in uniform random number generator (`Math.random()`) to provide an implementation of simple method for the generation of Exponential random variables, this method was detailed by Dr. Donald Knuth [16]. This method is shown in Formula 4.1, where X is exponentially distributed and U is a uniform random variable, taking values in the range $0 - 1$.

$$X = -\frac{1}{\lambda} \times \ln U \tag{4.1}$$

Process Suspension

Process suspension occurs when there are more than a fixed maximum number of processes in the system i.e., in the ready and blocked queues. Processes are resumed when the total number of process in the system is lower than another predefined threshold. For the purposes of clarity the suspension threshold corresponds to the number of process which can be shown in the `QueuePanel` class, which visualizes the ready queue.

4.4 Scheduling Disciplines

Each of the scheduling disciplines is implemented as either a direct or indirect subclass of the abstract `SchedulingDiscipline` class. This base class provides a common interface so the scheduler may simply invoke the `schedule()` method of any discipline without needing to know which discipline is in use, this is in keeping with the principles of object-oriented design. The `SchedulingDiscipline` class also provides all of the necessary accessor methods for use by the GUI e.g., `getName()`. Each subclass provides it's own implementation of the `schedule()` method. Several disciplines implement helper methods to assist with scheduling e.g., to ascertain highest priority or shortest process. All disciplines perform GUI updates via the `Scheduler` to provide explanation regarding scheduling choices. Each discipline maintains a reference to the `Scheduler`, and maintains its own short textual information for use in the GUI. In the following subsections an overview of the implementation of each discipline will be given.

First Come First Served

The state of the `Scheduler` is checked to ascertain if a process is running. If this is not the case, and provided that the ready queue is not empty the discipline selects the process at the head of the queue, performs a textual update and invokes the `setCurrent()` method of the `Scheduler`, which assigns a selected process to run.

Round Robin

Processes are chosen from the head of the queue as per the conditions of First Come First Served. Once a process is dispatched the quantum counter is decremented, once it has elapsed the `timeout()` method of the `Scheduler` is invoked and the next process selected from the head of the queue. If the process completes its burst before the quantum elapses, the quantum is reset and the next process dispatched.

Priority Queue

In the non-preemptive implementation, provided no process is currently executing, a helper method is invoked. This method iterates over the ready queue and returns the process of highest priority which is then dispatched. To avoid process starvation every 20 cycles the priority of processes which have not received CPU time is increased. The preemptive variety is implemented as a subclass of the nonpreemptive scheme. In its `schedule()` method a call is first made to the superclass `schedule()` method. Following this the priority of all the ready processes are compared to the currently running process and if one is of higher priority it is dispatched. This is done by first invoking `timeout()` and then `setCurrent()`.

Shortest Job First

When there is no process currently running and the ready queue is non-empty a helper method is invoked to iterate over the ready processes and select the process with the lowest expected burst time, as earlier discussed the estimation calculation is performed internally by each process. As such the helper method performs a simple comparison between all processes.

Shortest Remaining Time First

Shortest Remaining Time First is implemented as a subclass of the `ShortestJobFirst` class implementing all of its behavior. A call is made to the super class `schedule()` method and following this the remaining estimated burst time of the running process is checked against that of all the processes on the ready queue. If a shorter process is found it is dispatched. No distinction need be made between newly arriving processes and those which were present when the current process was chosen, as they would of been chosen if their estimated burst times were shorter.

4.5 Statistics

All statistical calculation is performed in the `Statistics` class, which is invoked by the `Scheduler` on each CPU cycle. The appropriate data e.g., process queues are passed from the

`Scheduler` to the `Statistics` class. This class is decomposed into a number of methods each responsible for calculating statistics for a different evaluation criteria. For each appropriate criteria the average, maximum, minimum and the standard deviation values are calculated. In the case of the CPU performance statistics the utilization, idle time and throughput are calculated. For educational purposes, the average value is further subdivided into an average per type of process. This allows the user to see differences in performance between I/O and CPU bound processes under different disciplines and arrival conditions.

4.6 Graphical User Interface

The GUI is composed of four main elements and their corresponding classes; `AnimationPanel`, `ScheduleGUI`, `StatisticsPanel`, and `ControlPanel`. Each of these main GUI classes makes use of a number of helper classes which encapsulate display entities such as the `QueuePanel` class.

In the planning and design stages of the project the process scheduling visualization was modeled upon techniques used in noted operating system texts [2] [1], fora discussion of this see Section 2.5.1. Additionally visualization techniques used in other simulations were looked at. All of this influenced the initital prototype animation view design as shown in Figure 2.3.

Several additions and augmentations were made for clarity and intuitiveness based upon feedback received from "beta" testers and the project supervisor, Dr. Felix Balado. These included the addition of a CPU burst bar showing a process' progress in the CPU, visualizations of the entry and exit points (see Figure 4.1), hiding of all but current state transition and additional textual explanation. The CPU burst bar can be seen in Figure 4.2. The burst bar is primarily useful in showing the burst length variance between I/O and CPU bound processes.

Additionally functionality was added allowing users to click on queues, processes and other GUI components for additonal information. In the case of a process this allows the user to modify the priority, this is shown in Figure 4.3.

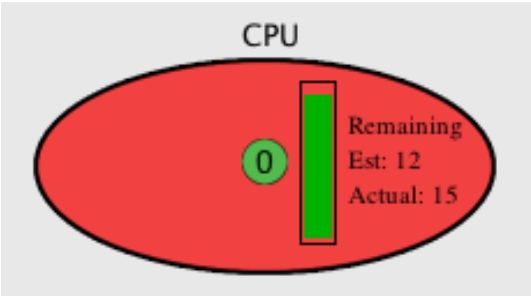

Figure 4.2: CPU bound process running on the CPU

4.6.1 Animation Panel

The `AnimationPanel` encapsulates the main visualization of process scheduling. It displays the process queues, the CPU, the start and end states and the process state transitions. Each of these components is represented by its own class e.g., `QueuePanel`, `CPUPanel`, `ExitPanel`. This was done in order to encapsulate behavior and better organize the GUI code. The

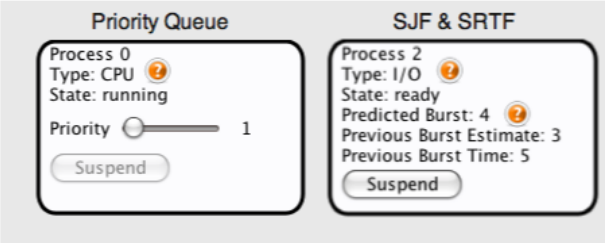

Figure 4.3: `ProcessControlPanel` under Priority queue and SJF

`AnimationPanel` is shown in Figure 4.1.

The `AnimationPanel` provides methods to be invoked to visualize each state transition, all of these methods follow a similar template in which they set several components to be visible and invoke the `translate()` method, which takes the transition path as an argument. The transition methods are invoked by the scheduler corresponding to changes in the state of the simulation i.e., process dispatch.

The `translate()` method operates by using a `FlatteningPathIterator` object to divide the predefined path (e.g., from one queue to another) into a series of points and quickly transitioning the `ProcessPanel` across this path. The rate of this transition is controlled by the speed selected by the user.

Each of the elements visible in the `AnimationPanel` can be clicked in order to display further information about them e.g., the user may click on a queue or the CPU etc. Initially some additional pointers are visible instructing the user that they may click these components. Figure 4.4 shows the panel displayed when the ready queue is clicked.

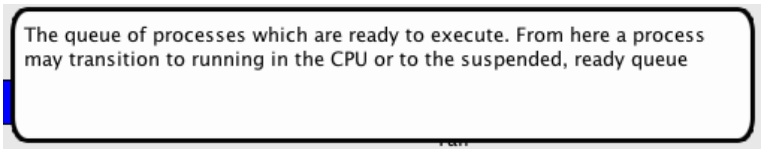

Figure 4.4: The explanation panel describing the ready queue

4.6.2 SchedulerGUI

This is the main windowing frame of the GUI. The other graphical components such as the `StatisticsPanel`, the `ControlPanel` and the `AnimationPanel` are all contained within it. Before the simulation begins a welcome screen is displayed to educate the user about the basic operation of each discipline. Figure 4.5 shows the main GUI when the application is first launched, from this initial screen the user may configure the simulation using the `ControlPanel`.

4.6.3 Control Panel

The `ControlPanel` class is the primary user interactivity component. The `ControlPanel` as shown in Figure 4.6 allows the user to specify several parameters for the simulation. When the simulation begins these parameters are passed to the `Scheduler`. The `JSlider` controllers were used as they provide an intuitive interface with which the standard user would be familiar. Consideration was given to alternative interfaces such as text entry boxes,

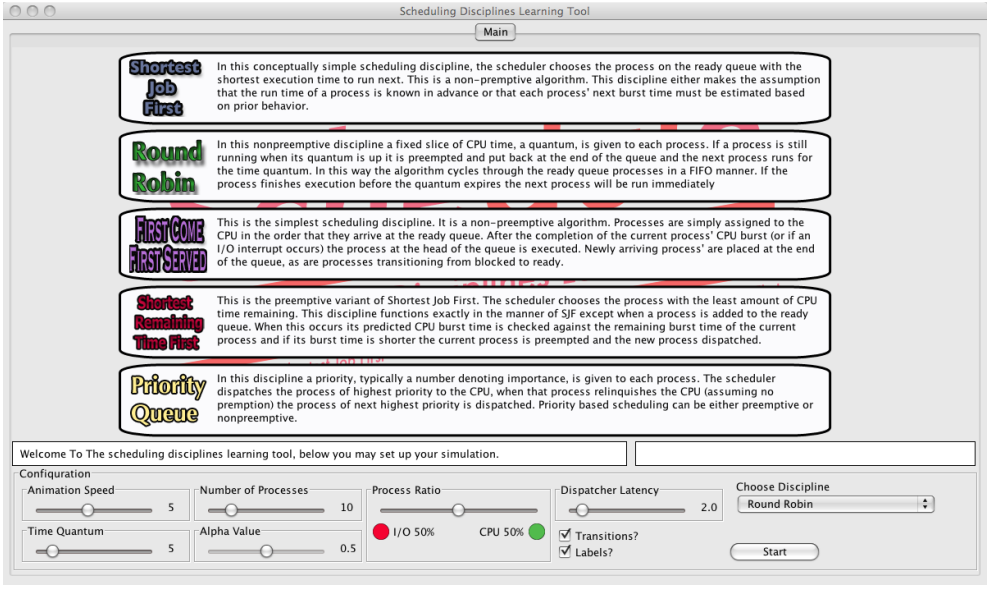

Figure 4.5: The application welcome screen

but this was thought to be cumbersome. During the simulation the `ControlPanel` allows the user to pause the simulation, vary the speed and enter and exit the step-by-step mode. Additionally the user may restart the simulation. The elements which the user may control and the reasons for their selection are discussed in Section 3.1.3. Each component of the control panel is discussed in the supplied user guide in Section 7.1.

The control panel provides an option for the user to set up a simulation and run it in its entirety in one click. This functionality is aimed at a time when the user may be interested in viewing statistical information quickly.

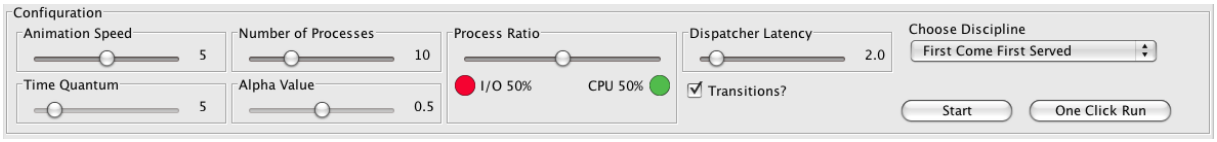

Figure 4.6: The control panel in which the user may specify a number of parameters

4.6.4 Statistics Panel

The `StatisticsPanel` is responsible for the display of the computed statistical data. The `StatisticsPanel` is updated by the `Statistics` class after each computation cycle. Each criteria has an associated GUI component (`StatPanel`) which contains a line graph (`LineGraph`). The `StatisticsPanel` facilitates the display of information regarding the six evaluation criteria, the associated graphs and explanation detail regarding each. Figure 4.7 shows the statistics display, showing the response time line graph under the priority queue scheduling discipline, along the top the textual breakdown for each criteria is shown. Originally the statistics and the scheduling animation were displayed in the same panel however a decision was reached to split the statistics view off into its own panel so there would be adequate screen space for the statistical graphs.

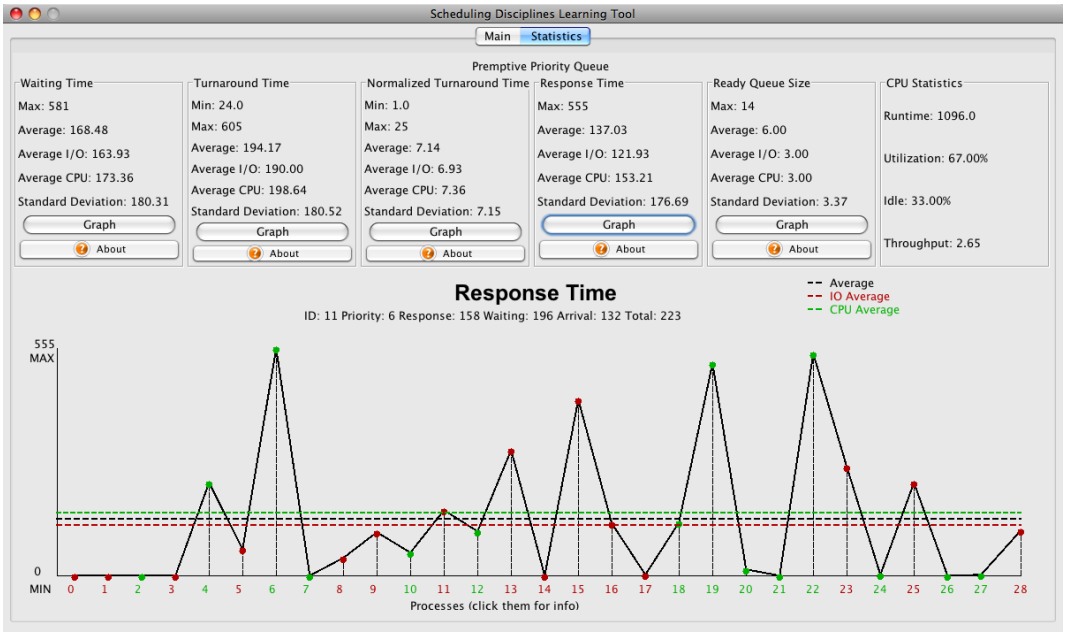

Figure 4.7: The display of the collected statistical data under the priority queue scheduling system

4.6.5 Learning Aids

The primary learning aids present to the user are the statistics display and the primary animation. As discussed previously in Section 3.2 additional explanation is provided when the user wishes to access it. For example if the user wished to know more about the exponential averaging formula they may access this information through the `ProcessControlPanel` which will display an explanation panel in the animation. Additionally a brief explanation of each scheduling decision is provided as they are made. The user may access more detailed information about scheduling decisions as shown in Figure 4.8.

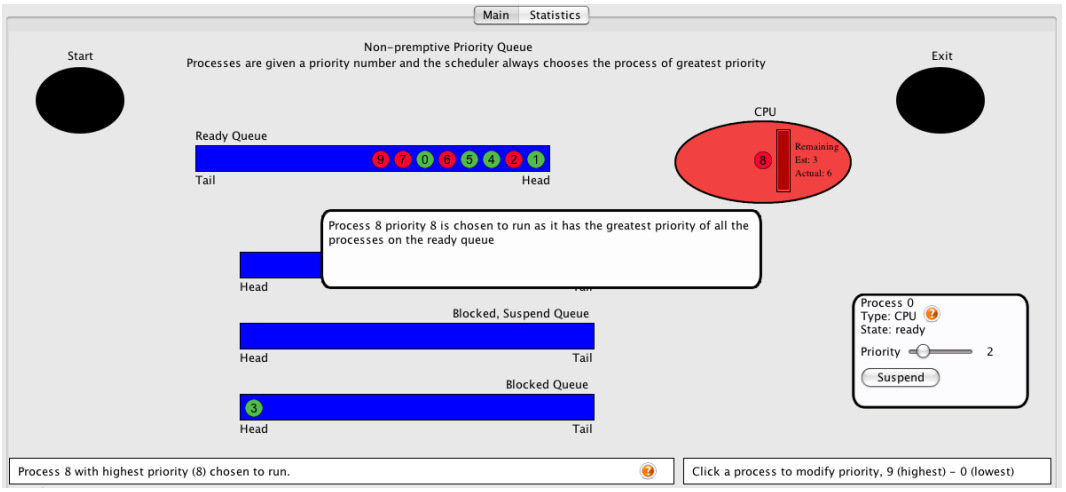

Figure 4.8: Additional scheduling information provided

4.7 Deployment

The software tool can be run directly as a runnable Jar file on any Java 1.6 enabled system. A Mac OSX application bundle has also been created allowing the tool to run as an essentially native Mac OSX application, as Java is bundled with OSX. Some consideration was given to deployment to add an ease of use for student users.

Chapter 5: **Testing and Evaluation**

This chapter details the testing and evaluation of the implemented learning tool which followed on from the initial implementation. This testing served to ensure both the correctness of the software and to augment its teaching merit and usability. Additionally some comparison between the implemented software and other scheduling simulations is given.

5.1 Testing

Rigorous testing was conducted to ascertain that the simulation and the underlying code was behaving as expected. Log files were generated and scrutinized for each discipline to check for correct behavior. The behavior of the GUI was monitored to check for consistency between itself and the simulation state. This method was effective in identifying logical bugs and removing them. Additionally the code itself was repeatedly scrutinized, following the addition of each new layer of functionality. Testing was also conducted by the "beta" testers and reported bugs were corrected.

5.2 Evaluation

In this section some evaluation of the learning tool will be given focusing on the feedback received during user testing. Finally a comparison will be made between the implemented tool and some other operating system scheduling simulations.

5.2.1 Feedback

Feedback on the prototype tool was received from the project supervisor and two "beta" testers. This feedback resulted in modification to several areas of the simulation in the furtherance of the design goals. Notably the default speed was reduced and greater speed variability added, additional explanation was provided through new popup panels and a CPU burst counter was added to track the progress of the running process. Other changes included the omission of all but the current state transition in order to emphasize the current operation being performed.

This feedback was a key phase of the learning tools development and helped to strengthen the outcome of the project by augmenting its teaching merit. Most of the feedback received was positive particularly as regards the statistical display. None of the users had any particular problems understanding how to set up and run the simulation. As a result of this testing period the completed tool has a more intuitive animation and interface.

5.2.2 Comparison With Other Tools

In this section a brief comparison between the implemented learning tool and several other scheduling simulations is given.

- The tool available at [12] implemented at the University of Macedonia Thessaloniki, provides a good simulation of process scheduling. This visualization is similar in design to the implemented learning tool animation. The simulation runs a Round Robin discipline but this is not made clear and there are no tunable parameters available or additional disciplines to select. No guideline is provided as to any variation between processes. This simulation serves to show the basic scheduling principles well. Notably this simulation allows the user to step back in the simulation a feature absent from the tool developed in this project.
- The second tool looked at [15] was developed at the University Of Dallas. It provides a Gantt chart view of process execution and a statistical readout after the run is over. The process model used here has only a single CPU burst. Processes are set up manually by the user. This was considered as a possible operational mode in the tool but was relegated in favor of a Monte Carlo simulation, which calls for random generation. In many ways this tool was implemented with an entirely different design philosophy from the tool implemented in this project. The Gantt chart view was considered for use in the implemented tool however it was thought to be a limited visualization which doesn't show clearly process transitions and process behavior.

Typically many of the other operating system simulations looked at such as [14] [13] are in some sense incomplete, they do not include event waiting, implementations of I/O and CPU bound processes, do not simulate all the fundamental disciplines etc. Many do not perform graphical simulation. However, [14] and [13] both provide an interesting feature in that two or more disciplines can be run concurrently, this idea is discussed as a possible extension to the projects developed tool in Section 6.2.2.

It was interesting to compare the implemented software to similar simulation tools. This shows the vast array of different visualization techniques and optional parameters which could be used in such a tool. Additionally it can be seen that the completed project contains functionality not found in any one other available simulation tool. This was a key reason for the submission of the project specification.

Chapter 6: **Conclusions and Future Work**

6.1 Conclusions

A software tool has been developed which visually simulates process scheduling using several well known scheduling disciplines. The implemented tool achieves its design goals allowing the user to interact with the fully animated simulation both in set up and during its execution. Additionally the user may view appropriate statistical information in order to assess performance.

In terms of meeting the specification goals the finished project has done this completely. All of the disciplines have been implemented, the relevant statistics are calculated and displayed and the simulation is interactive and well animated.

This software has been tested and refined to better serve as an educational tool for undergraduate students. It is hoped that this software package can be used to guide and augment a students understanding of these core ideas. Certainly it can be seen that the tool fills a gap left by other scheduling simulations, as discussed in Section 5.2.2.

Ultimately the design decisions made in Chapter 3 have proven well founded. In particular the implementation of a Monte Carlo simulation has resulted in a high quality learning tool. This has greatly increased the educational value of the tool over a deterministic approach. Additionally the chosen visualization and statistical display methods provide an excellent visual stimulus for learning.

6.1.1 Concluding Remarks

If this project were to be re-implemented it is this author's opinion that another language more suited to animation development be used, perhaps Flash. However I feel the approach taken was well founded and little would be changed in that regard. I feel I learned a lot about managing working within a large code-base. The importance of the initial encapsulation offered by the system design was shown many times throughout development.

6.2 Future Additions

As with any project, over the course of development a number of possible additions and interesting extensions become evident. In this final section a discussion of the most interesting of these possible extensions is given.

6.2.1 Simulation of Additional Disciplines

The developed software tool could be extended to simulate the operation of other scheduling disciplines such as the Highest Response Ratio Next and Multilevel Feedback Queue disciplines. In the case of Highest Response Ratio Next this should require only minor modifications to the existing code to support the selection of the new discipline and the implementation of a new discipline class. The Multilevel Feedback Queue scheme would be implemented similarly but would require some modification to the GUI and the scheduler, due to the division of the ready queue into several queues.

6.2.2 Statistical Comparison

To augment the tool and allow for easier comparison between the scheduling disciplines the software tool could be extended to run all of the scheduling disciplines simultaneously on the same set of processes. It is envisaged that this would allow the user to switch between the visualization and statistical display of each discipline. There is little to be gained perhaps from switching between visualizations however the ability to compare the performance of each discipline statistically at each stage in the simulation would be quite powerful and illuminating.

6.2.3 Application Of Queuing Theory

The complexities of the implemented scheduling simulation and diverse scheduling disciplines used make it difficult to apply queuing theory. For instance as per the project specification the processes burst times do not follow a exponential distribution in the implemented tool. Rather their service times vary around two general mean service times, one per each type of process. This would suggest the application of the M/G/1 model, which assumes a First Come First Served discipline in which services time follow a general distribution [1]. There exist other models in the vast literature of queuing theory which could be applied to deal with the issues of finite process populations and queue re-entry, an in depth investigation of these would be the first step towards applying queuing theory to this project.

6.2.4 Miscellaneous

An additional mode could be added to the simulation in which the burst times of each process follow an Exponential distribution. This would allow for easier application of queuing theory and a simulation closer to real-world operating systems.

Processes could be enabled to spawn child processes. This would be useful in showing students other fundamental OS topics.

There are several internal parameters used in the simulation which could be be tunable through an extended `ControlPanel` class. These parameters include the process suspension thresholds and the process burst length bounds. The `ControlPanel` functionality should be extended to allow the user to step back through previous simulation cycles.

The implemented tool simulates scheduling as related to a singular processor. It would be interesting and pertinent to recent trends in computing to simulate multiple processors. This would require some modification to the `AnimationPanel` and the `Scheduler` classes.

Chapter 7: **Appendix**

7.1 User Guide

7.1.1 Control Panel

- Process Ratio - This controls the percentage of CPU and I/O bound processes which are created when the simulation is run.
- Number of Processes - The number of process which will be used in the simulation.
- Alpha Value - Linked to SJF and SRTF, gives varying weights to previous burst times.
- Time Quantum - For use in the Round Robin discipline, this is the slice of CPU time which each process is given before it is preempted.
- Animation Speed - Controls speed of the animation.
- Dispatcher Latency - The time taken to switch from one process to another, particularly important for Round Robin.
- Choose Discipline - This combobox allows the user to choose which discipline they wish to simulate. The disciplines operation are summarized in the welcome screen.
- Step Mode - This mode allows the user to step through the simulation one CPU cycle at a time
- Step - Step forward by one cycle.
- Start - Runs the animated simulation.
- One Click Run - runs the entire simulation, useful if you want to quickly view and compare statistics.
- Transitions - Turns GUI process transitions on and off.

The above information may be accessed in the simulator by mousing a component, this displays an associated tooltip.

7.1.2 Interaction

During the simulation you may click on any of the components of the simulation for additional explanation, for example you may click on the ready queue, or the CPU. If you click on a process you can view some additional information regarding it and in certain circumstances modify certain attributes. Similarly in the Statistical display you can access additional explanation and move between the various graphs using the graph display buttons. By clicking on an associated green or red dot in the graph display you can view the properties of that process, which may enlighten you as to why, for instance, that process has a poor response time.

Bibliography

[1] William Stallings. *Operating Systems: Internals and Design Principles*. 781 pages. ISBN: 0138874077. Prentice Hall, 1997.

[2] Andrew S. Tanenbaum. *Modern Operating Systems*. 1104 pages. ISBN: 0136006639. Prentice Hall, 2007.

[3] Dr. Felix Balado and Dr. Tahar Kechadi. *Operating Systems 1 Lecture Notes*.

[4] James Elliott et al. *Java Swing, Second Edition*. 1280 pages. ISBN:0596004087. O'Reilly Media, 2002.

[5] http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/Timer.html

[6] http://computer.howstuffworks.com/operating-system7.htm

[7] http://developer.apple.com/mac/library/documentation/Java/Conceptual/Java14Development/03-JavaDeployment/JavaDeployment.html

[8] http://developer.apple.com/Mac/library/documentation/UserExperience/Conceptual/AppleHIGuideli

[9] Joel Spolsky. *User Interface Design for Programmers* 159 pages. ISBN: 1893115941. Apress, 2001.

[10] Jonathan Knudsen. *Java 2D Graphics* 339 pages. ISBN: 1565924843. O'Reilly Media, 1999.

[11] John Zukowski. *Java Collections* 420 pages. ISBN: 1893115925. Apress, 2001.

[12] http://www.it.uom.gr/teaching/opsysanimation/animations/PROCESS.SWF

[13] http://fac-staff.seattleu.edu/quinnm/web/education/JavaApplets/applets/CpuScheduling.html

[14] http://www.acxe12.net/diplom/

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

[16] Donald E. Knuth (1969). Seminumerical Algorithms. The Art of Computer Programming, Volume 2. Addison Wesley.

[17] Metropolis, N and Ulam, S. (1949). "The Monte Carlo Method". Journal of the American Statistical Association 44, pages 335341