Using the Process Scheduling Simulation

This document was last modified on January 1, 2007 to reflect changes made to version 1.087.

Overview

The simulator allows you to experiment with various process scheduling algorithms on a collection of processes and to compare such statistics as throughput and waiting time.

Supported algorithms include First-Come/First-Served, Shortest Job First (actually shortest remaining cpu burst time first), and Round Robin. The CPU and I/O bursts of the processes are described by probability distributions, but it is possible to choose the seed so that the same processes can be used to test different algorithms.

The simulator is a Java application that can be run on any platform supporting a modern Java runtime system. The simulator can produce log files in HTML format that include tables of data as well as graphs and Gantt charts.

System and User Requirements

System requirements:

- Computer with a Java runtime environment, version 1.1 or later.
  Java 1.4.2 or later is recommended.
- The tutorial assumes that the following files have been appropriately installed:
  - `PS.jar`
  - `Jeli.jar`
  - `psconfig`
  - `myrun.run`
  - `myexp.exp`
  - `runps.bat` (for Windows) or `runps` for UNIX

User requirements:

- Familiarity with running a program in a command line environment
- Some knowledge of CPU scheduling and the algorithms FCFS, SJF, and RR

Starting the Simulator

Windows:

- Download the psWindows.zip under the Windows label
- Extract the zip – this will make a psWindows directory.
- Go into the ps directory and double click on the runps.bat.

Unix/Linux/Mac:

- Download the psUnix.zip under the Linux label
- Extract the zip – this will make a psUnix directory.
- Go into the ps directory and at the command line type ./runps &

If the simulator does not start, make sure you have the Java runtime executables in your path. In a command window, execute: `java -version` and make sure this displays a version later than 1.0.

Basic Operation

The basic steps in doing a simulation are

- Create a configuration file describing the basic properties of the simulator
- Create an experiment which consists of a number of experimental runs
- Run the simulator on this experiment
- Log the results

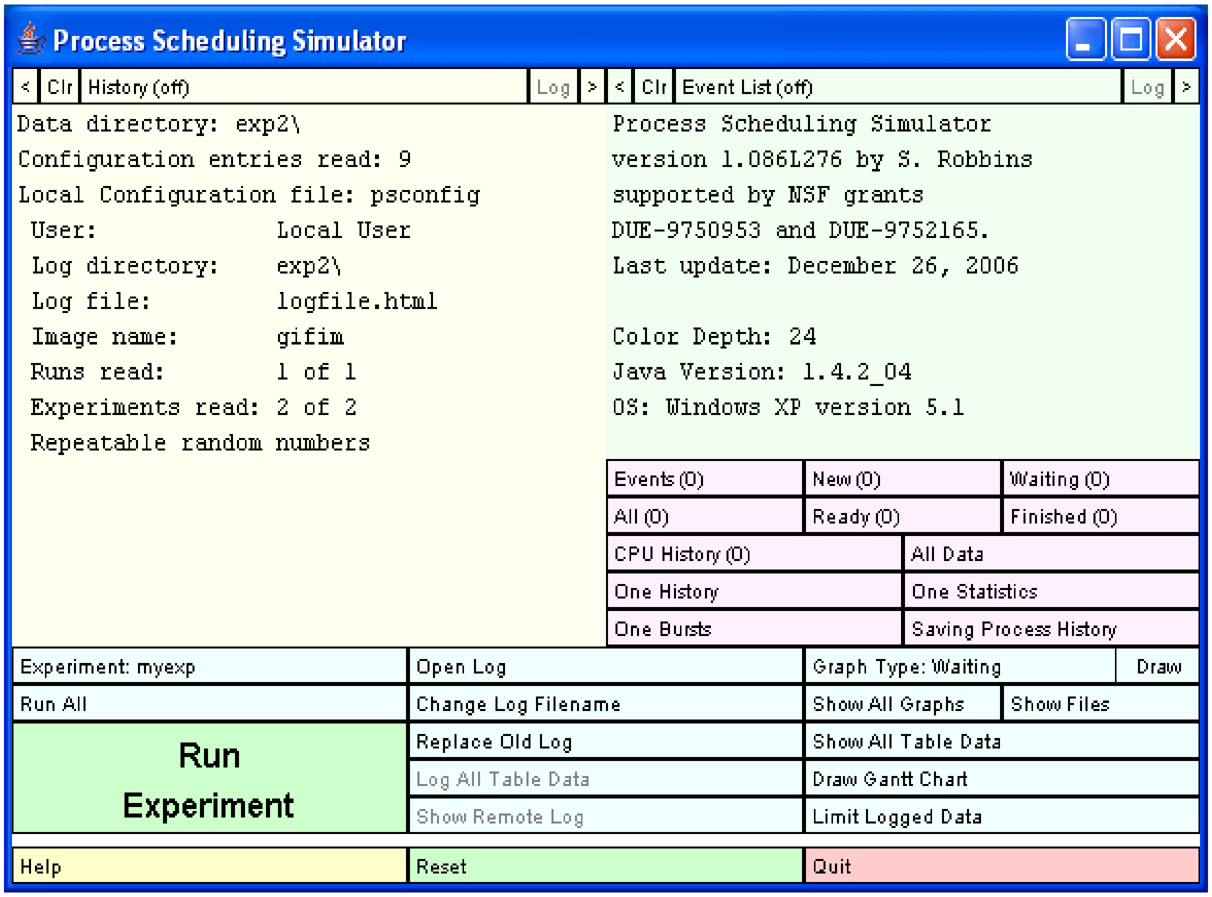
The main configuration file contains information about the location of the experiments and where to put the output files. See the Configuration Section

An experimental run consists of a scheduling algorithm and a collection of processes to be run under that algorithm. An experiment consists of a number of experimental runs which are to be compared and analyzed. The simulator organizes the input information in order to make it simple to do experiments in which one or more parameters can be varied.

To perform an experiment, you must create two files. One file (the run file) contains information about the parameters for the first run. The other file (the experiment file) contains a list of runs to be made and the parameters that vary between runs. The files contain ASCII text so that you can create them directly with a text editor. The standard distribution contains sample files.

After the experiment has been run by the simulator, you can produce tables or graphs of various statistics such as average waiting time and throughput. Information about the experiment including the specification of the processes, the statistics and graphs of the experimental results are stored in a log file in HTML format suitable for viewing from a browser.

The main simulator window is shown in Figure 1.



```
Data directory: exp2\
Configuration entries read: 9
Local Configuration file: psconfig
 User:              Local User
 Log directory:     exp2\
 Log file:          logfile.html
 Image name:        gifim
 Runs read:         1 of 1
 Experiments read:  2 of 2
 Repeatable random numbers
```

```
Process Scheduling Simulator
version 1.086L276 by S. Robbins
supported by NSF grants
DUE-9750953 and DUE-9752165.
Last update: December 26, 2006

Color Depth: 24
Java Version: 1.4.2_04
OS: Windows XP version 5.1
```

Figure 1: The main simulator window.

The yellow subwindow in the upper left corner labeled **History** shows the initial configuration read in from the configuration files. There are three columns of buttons at the bottom of the main window. The leftmost column of buttons allows you to choose the experiment to run and to run it. Pushing the top button in this column advances through the available experiments. Pushing the next button allows you to select which runs of the experiment to do. Pushing the large **Run Experiment**, runs the chosen experiment.

The second column of buttons controls the log file. The top button allows you to open the log file. When the log file is opened, the button is changed to a **Close Log** button. As runs are made they are logged in the log file and statistics about the run are saved. Pushing the **Log All Table Data** button puts two tables of statistics in the log file. The tables contain entries for all selected runs. All runs are selected by default. A sample set of tables is shown in Figure 2.

							Entries		Average Time	
Description	Algorithm	Time	Processes	Finished	CPU Utilization	Throughput	CPU	I/O	CPU	I/O
myrun_1	FCFS	257.05	30	30	.96104	.116711	90	60	2.74	15.20
myrun_2	SJF	256.90	30	30	.96158	.116777	90	60	2.74	15.20

		Turnaround Time				Waiting Time			
Description	Algorithm	Average	Minimum	Maximum	SD	Average	Minimum	Maximum	SD
myrun_1	FCFS	215.05	169.60	257.05	31.45	176.41	138.82	202.24	.66
myrun_2	SJF	124.87	62.00	256.90	62.17	86.23	16.69	229.82	2.37

Figure 2: Two tables of data created by the simulator when the **Log All Table Data** button is pushed.

The first table contains information about average CPU utilization, throughput, turnaround time, and waiting time for each run. The second table contains more detailed information about the turnaround time and waiting time, including the average, the minimum, maximum, and standard deviation for each.

This second column of buttons also has a button for drawing Gantt Charts. See Gantt Charts.

The third column of buttons controls the creation of graphs and tables. At the top are two buttons. The one on the left labeled **Graph Type: Waiting** will cycle through the types of graphs that can be drawn. Draw the selected type of graph by pushing the **Draw** button to the right. The entry **Graph: Waiting** means that a histogram containing information on the waiting time for the processes of each run is to be produced. Each run is displayed in a different color, and the distribution of waiting times is shown as a bar graph. The quantity graphed is the waiting time which is the time spent in the ready queue, clicking the **Draw** button pops up a window containing the graph. This window contains a **Log** button that causes the graph to be inserted in the log file. The sample graph shown in Figure 3 contains waiting time results for an experiment with two runs, one for FCFS and one for SJF.

A more complete explanation can be found in the section Detailed Operation of the Simulator.

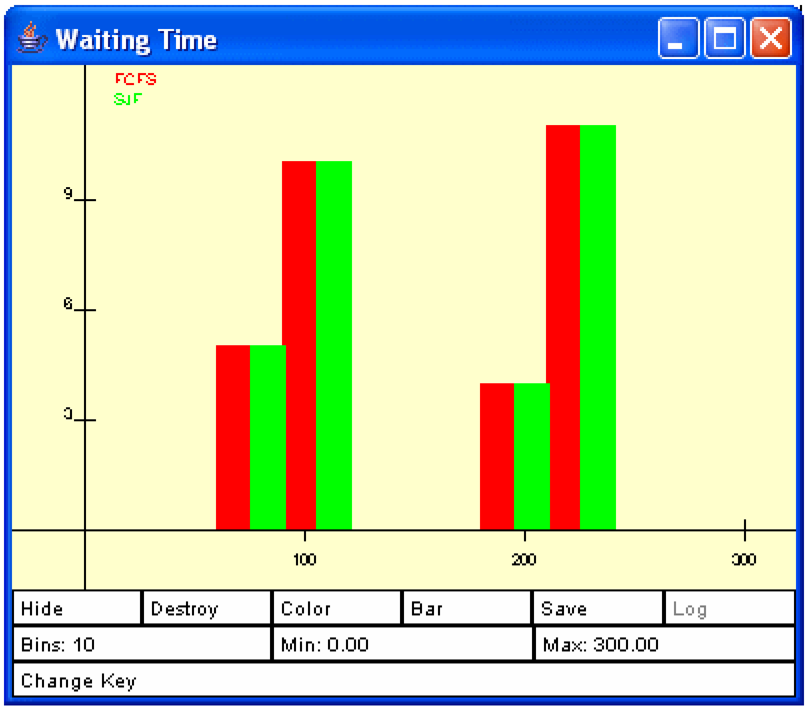


Figure 3: A graph created by the simulator when **Graph: Waiting** is selected and the **Draw Graph** button is pushed.

Specifying an Experiment: Overview

An experiment is specified by two files. Each file has a name consisting of a base name and an extension. The files are:

- The experiment file with extension **.exp** specifies a number of experimental runs. Each run is specified by a base name and a possible list of modifications.
- The experimental run file, which is also called the run file for short, has the extension **.run** and contains information about the scheduling algorithm to use, the processes to run, and when the processes arrive.

Time

The simulator uses a virtual time which is represented by a floating point value of unspecified units. When the simulator is started, the time is set to 0.0. The simulator is event driven, and events that occur at the same time may occur in any order.

Processes

A process is specified by the following information:

- Arrival time
- Total CPU time
- CPU burst time distribution
- I/O burst time distribution

Probability Distributions

Three probability distributions are supported at this time. When a probability distribution is specified in a file, it is represented by a single line of ASCII characters. The line starts with a word indicating the type of distribution and is followed by either a single floating point number representing the mean of the distribution, or in the case of the uniform distribution two floating point numbers representing the left and right endpoints of the interval.

The following distributions are supported:

- The constant distribution. Example:

  ```
  constant 23.45
  ```

  represents a constant distribution with constant value 23.45.

- The exponential distribution. Example:

  ```
  exponential 23.45
  ```

  represents the exponential distribution with mean 23.45.

- The uniform distribution. Example:

  ```
  uniform 23.45 47.89
  ```

  represents the uniform distribution in the interval [23.45,47.89]

Experimental Runs: An Introduction

An experimental run, or just run for short, contains all of the information needed to run the simulator on one collection of processes. An experimental run specifies a scheduling algorithm, a collection of processes, and when the processes arrive. A simple format for an experimental run file is described here.

The file format is as follows:

name *experimental_run_name*
comment *experimental_run_description*
algorithm *algorithm_description*
numprocs *number_of_processes*
firstarrival *first_arrival_time*
interarrival *interarrival_distribution*
duration *duration_distribution*
cpuburst *cpu_burst_distribution*
ioburst *io_burst_distribution*
basepriority *base_priority*

Here is a sample experimental run file that must be stored in the file **myrun.run**.

```
name myrun
comment This is a sample experimental run file
algorithm SJF
numprocs 20
firstarrival 0.0
interarrival constant 0.0
duration uniform 500.0 1000.0
cpuburst constant 50.0
ioburst constant 1.0
basepriority 1.0
```

This file specifies a run of 20 processes using the shortest job first algorithm. The first process arrives at time 0.0. The interarrival times are all 0.0, so all processes arrive at the same time. Each process has a duration (total CPU time) chosen from a uniform distribution on the interval from 500 to 1000. All processes have a constant CPU burst time of 50 and a small constant I/O burst time of 1.0. The base priority must be present in the file, but it is currently not used by the simulator.

Experiments

An experiment consists of a number of experimental runs that are to be compared and analyzed. An experiment file specifies a number of experimental runs to be made. Aside from a name and comment as in the above files, there are an arbitrary number of lines specifying experimental runs to be made.

In the simplest case, an experimental run can be specified by the name of the run. In many cases, you will want to make several experimental runs in which almost everything is the same, except that one or more parameters are changed. For example, you might want to vary the quantum used for the round robin scheduling algorithm but use the same set of processes.

The simulator allows you to do this by specifying the same experimental run in each case and giving a new value to one or more parameters. The general format for a run line in the experiment file consists of the word **run** followed by the name of an experimental run, followed by a list of modifications to that experimental run.

The experiment file format is as follows:

name *experiment_name*
comment *experiment_description*
run *run_name*1 *optional_modification_list*1
...
run *run_name*n *optional_modification_list*n

Here is an example experiment file that must be stored in the file **myexp.exp**

```
name myexp
comment This experiment contains 3 runs
run myrun
run myrun cpuburst uniform 10 90
run myrun cpuburst exponential 50
```

This experiment file makes three runs. All of the runs are based on the run file **myrun.run** above. In the second run the CPU burst distribution is changed to be uniform in the interval from 10 to 90. In the third run the CPU burst is an exponential distribution with mean 50.

Experimental Run Modifications

The run lines in an experiment file reference an experimental run. Often you would like to do several runs with almost the same parameters.

Only the simplest case is described here in which the modification applies to all processes in the run.

A modification specification is a string of tokens separated by one of a small number of key words indicating which parameter is to be modified. The key word may be followed by parameters specific to the value to be modified. For example, if the value to be modified is a distribution, then a distribution is specified in the format described under Probability Distributions. The modification specifications can be in any order.

Here is a list of the key words and the appropriate parameters:

- **numprocs**: The parameter is the number of processes to create.
- **firstarrival**: The parameter is the arrival time of the first process, a floating point number.
- **basepriority**: The parameter is the base priority of the processes created. This base priority is not used unless the simulator is using priorities.
- **interarrival**: The parameter is a string representing the distribution of the interarrival times of the processes.
- **duration**: The parameter is a string representing the distribution of the total CPU time used by the processes.
- **cpuburst**: The parameter is a string representing the distribution of the CPU burst times of the processes.
- **ioburst**: The parameter is a string representing the distribution of the I/O burst times of the processes.
- **algorithm**: The parameter is a string representing the scheduling algorithm to use. See Scheduling Algorithms Supported.

- **seed**: The parameter is an integer representing a seed to be used to initialize the random number generator. See Random and Not So Random Numbers .
- **cstin**: The parameter is a non-negative floating point number and represents the time during a context switch to move a process into the CPU.
- **cstout**: The parameter is a non-negative floating point number and represents the time during a context switch to move a process out of the CPU to another state. See Context Switches.
- **priorityoff**: This takes no parameter and sets the simulator to not use priorities for this run. This is the default. See Context Switches.
- **priorityon**: This takes no parameter and sets the simulator to use non-preemptive priorities for this run.
- **prioritypreempt**: This takes no parameter and sets the simulator to use preemptive priorities for this run.
- **key**: This takes a parameter which is the key to be used when a graph of waiting time is produced. If the key contains more than one word it needs to be contained in double quotes.

Scheduling Algorithms Supported

The following discussion assumes that priorities are off, which is the default. See Priority Scheduling for a discussion of using priorities.

- **Round Robin (RR):** The user can set the quantum. Processes are removed from the ready queue in the order that they arrive.
- **First-Come/First-Served (FCFS):** Processes are taken from the ready queue in the order that they arrived. Once a process is using the CPU, it stays there until its CPU burst expires. It then goes into the I/O waiting queue until its I/O burst expires.
- **Shortest Job First (SJF):** The next process to be removed from the ready queue is the one with the shortest CPU burst time. If there is more than one process with the smallest time, the one that arrived first is taken.
- **Preemptive Shortest Job First (PSJF):** The next process to be removed from the ready queue is the one with the shortest CPU burst time. If there is more than one process with the smallest time, the one that arrived first is taken. When a process becomes ready and its burst time is shorter than the remaining burst time of the running process, the running process is preempted.
- **Shortest Job First Approximation (SJFA):** An estimate is made for the next CPU burst time based on the past behavior of the process and the process with the shortest estimated burst time is used. The initial estimate is 0 so new processes have priority. A user-settable parameter, alpha, has a value between 0 and 1. If the last estimate was b, and the last burst time was actually B, the next estimate is alpha*B + (1-alpha)*b.

A scheduling algorithm is specified with a string and a possible optional floating point parameter depending on the scheduling algorithm. The following strings are used to specify scheduling algorithms:

- **RR** *quantum* represents the Round Robin algorithm with the given quantum.
- **FCFS** represents the First-Come/First-Served algorithm.
- **SJF** represents Shortest Job First scheduling.
- **PSJF** represents Preemptive Shortest Job First scheduling.
- **SJFA** *alpha* represents the Shortest Job First Approximation using the given value of alpha.

Priority Scheduling

The simulator supports the scheduling of processes with fixed priorities. A priority is a floating point value with larger numbers representing higher priorities. By default, priorities are ignored. The scheduler supports two modes of priority scheduling, preemptive and non-preemptive. Priority scheduling is specified by an optional line in the run file or an Experimental Run Modification in the experiment file. The key words are **prioriryoff**, **priorityon**, and **prioritypreempt**. The key word **priorityon** corresponds to non-preemptive priority scheduling and **prioritypreempt** corresponds to preemptive priority scheduling. When specified in the run file, a line containing one of these key words must appear after the algorithm specification. See Experimental Runs: Advanced.

Configuration

The initial configuration of the simulator is determined by a configuration file, **psconfig** which currently can be either in the directory from which the HTML file was loaded, or the current directory. The configuration file consists of a number of lines of ASCII text, each containing a key word followed by a key value. These lines may appear in any order.

At this time the supported key words are:

- **logdir**: key value is the directory that will contain the log file
- **logfn**: key value is the name of the log file
- **imagename**: the key value is the base name of the GIF file graphs used when graphs are inserted into the log file.
- **user**: key value is the full name of the user that is inserted into the log file.
- **portable**: key value is either the word "true" or the word "false" (without the quotes). If true, a portable random number generator is used that allows experiments to be repeated and give exactly the same results.
- **quiet**: pushing a button does not generate a sound.
- **noquiet**: pushhing a button generates a click.
- **run**: key value is the name of an experimental run. A file with the appropriate name must be present.
- **exp**: key value is the name of an experiment. A file with the appropriate name must be present.

Here is a sample **psconfig** file:
```
logdir .
logfn logfile.html
quiet .
imagename gifim
user Local User
portable true
run myrun
exp myexp
```

When the log file is opened, the simulator logs information about all of the available experimental runs and experiments. Here is what might be produced for the above configuration.

Configuration File:

```
logdir .
logfn logfile.html
quiet .
imagename gifim
user Local User
portable true
run myrun
exp myexp
```

Number of Run Files: 1

```
--------------------
name myrun
comment This contains two types of processes
algorithm SJF
seed 5000
numprocs 15
firstarrival 0.0
interarrival constant 0.0
duration uniform 10.0 15.0
cpuburst constant 10.0
ioburst uniform 10 20
basepriority 1.0

numprocs 15
firstarrival 0.0
interarrival constant 0.0
duration constant 4.0
cpuburst constant 1.0
ioburst uniform 10.0 20.0
basepriority 1.0
```

Number of Experiment Files: 1

```
--------------------
name myexp
comment This experiment contains 2 runs
run myrun algorithm FCFS key "FCFS"
run myrun algorithm SJF key "SJF"
```

Experimental Run Information for 1 Run

myrun: This contains two types of processes Seed: 0 Algorithm: SJF						
Group	Processes	First Arrival	Interarrival	Duration	CPU Burst	I/O Burst
1	15	0.0	constant 0.00	uniform 10.00 15.00	constant 10.00	uniform 10.00 20.00
2	15	0.0	constant 0.00	constant 4.00	constant 1.00	uniform 10.00 20.00

Experimental Runs For 1 Experiment			
Experiment	Commentary	Run	Modifications

myexp	This experiment contains 2 runs	myrun_1	algorithm FCFS
			key FCFS
		myrun_2	algorithm SJF
			key SJF

Figure 4: The description of an experiment as it might appear in the log file.

The first table contains information about the processes to be created such as the number of processes, the first arrival time, the duration distribution, the CPU burst distribution, and the I/O burst distribution.

The second table describes the experiment that was done which consists of two runs, one using the algorithm FCFS and the other using SJF.

Detailed Operation of the Simulator

The process scheduling simulation shows two text areas at the top of the screen. Each text area has 5 buttons at the top. The middle (and largest) button turns on or off the display of entries in the text area. The button to the left of it (**Clr**) clears the contents of the text area and the button to the right (**Log**) stores the information displayed in the log file. This button is disabled when the log file is closed. The two small buttons on the left and right of the text area labeled < and > decrease and increase the size of the font used to display text in the text area.

The leftmost text area is labeled **History**. When active, it displays various information about the progress of the simulation. It also shows the results of any of the information buttons, some of which are located just below the other text area.

The rightmost text area is labeled **Event Log**. When it is active it shows events are they occur.

Below the **Event Log** are about a dozen buttons which display information about processes and events of the latest run. There are two types of buttons. The first type appears in a 3 by 3 grid. Each one contains a label with a number if parentheses. When pushed, these buttons immediately display information in the **History** window. The number in parentheses approximates the number of lines of output generated.

The other buttons are below these in a 2 by 2 grid. These display detailed information about a particular process. When you click one of the other buttons, it will prompt you for a process number. Enter the number and push return to display information about that process. The pointer must be inside the button when data is entered.

- **Events**: displays all of the events in the event queue.
- **New**: displays the processes that have been created but not yet arrived.
- **Waiting**: displays all of the processes waiting for I/O to complete.
- **All**: displays information about all processes.
- **Ready**: displays information about processes in the ready queue.

- **Finished**: displays information about the processes that have completed.
- **CPU History**: displays a time-stamped log of processes entering and leaving the CPU.
- **One History**: displays a time-stamped log of the state changes of one process.
- **One Statistics**: displays various information about one process.
- **One Bursts**: displays the CPU and IO bursts of one process. the CPU.

The display appears in the **History** text area.

Below this is 3 columns of buttons. The first column is for running the simulator. The first button in this column shows which experiment will be run. Pushing this button cycles through the experiments if more than one is listed in the the configuration file. The next button by default is labeled **Run All** meaning that all runs of this experiment will be done. Pushing this button cycles through the runs for the current experiment, allowing you to execute only that run. The simulator is actually run by pushing the last of these buttons, labeled **Run Experiment**.

The second column is mainly for logging the results of the simulator. The last column if for displaying various tables and graphs. These are described below.

The second column of buttons is primarily related to logging of data.

- **Open Log**: opens the log file and enables the other logging buttons. This button then changes to **Close Log**.
  Note: Closing the log and reopening it erases the old log file by default (but see the third button below).
- The second button has different operations when the log is closed and when it is open.
  When the log is closed: the button is labeled **Change Log**, and can be used to change the directory and filename of the log file.
  When the log is open, the button toggles between the labels **Start Log** and **Stop Log**.
  When the log is first opened, logging starts and the button is labeled **Stop Log**. Pushing this button changes it to **Start Log** and stops the logging of data. The log file remains open.
- The third button also has different operations when the log is closed and when it is open.
  When the log is closed, the button toggles between **Replace Old Log** and **Append To Old Log**.
  When the former is active, opening a log file will replace one that already exists by the same name. When the latter is active, log information is appended to an existing log file. When this option is active, images are given longer filenames to minimize the possibility of a conflict.
  When the log is open, this button is label **Log Comment** and pops up a window allowing the user to enter a comment in the log file.
- **Log All Table Data**: creates two tables in the log file containing statistics about all saved data. Data is saved automatically when an experiment is done or when the **Save Data** button is pushed, but it does not appear in to log file until the **Log All Table Data** is pushed. The **Limit Logged Data** button in the next column can be used to limit which data goes into the log file.
- **Show Local Log**: displays the log file in a browser window. This button becomes active after the log file has been opened. It attempts to open a browser window containing the log file. This should work correctly on most Windows XP, Linux, Mac OSX, and UNIX systems.

The third column of buttons is primarily related to tables and graphs. Figure 2 show a window that appears when you push the **Draw** button. Graphs can be resized and you can modify various parameters used to draw the graph. You can set the number of bins as well as the minimum and maximum value graphed. To change these, click the mouse in the appropriate button and you will be prompted for the new value. If setting the minimum or maximum causes some processes to not be displayed, the number of missing processes is displayed in the min and max box in parentheses.

The **Change Key** button can be used to change the key in the graph. If there is more than one run displayed, pushing this button prompts for the index of the run to change. The index numbers appear with the keys on the graph when this occurs. After choosing a run, enter the new key. In a similar way, the **Color** button allows you to change the color used to graph a particular run. After choosing the index, click on the color you want to use.

- **Graph: Waiting**, **Graph: Waiting Box**, **Graph: Waiting Ratio**, **Graph: Waiting Ratio Box**, **Graph: Turnaround**, **Graph: Turnaround Box**, **Graph: CPU Bursts**, **Graph: CPU Bursts Box**, **Graph: Completed** or **Graph: Completed Box** toggles among the these types of currently supported graphs. Two types of graphs are supported, bar graphs and Box and Whiskers Graphs. **Waiting** refers to type a process is in the ready queue. **Waiting Ratio** is the ratio of CPU time to the sum of ready time plus CPU time. This ratio is useful for comparing processes with a wide range of CPU times. Larger values indicate less waiting time. **Completed** gives the distribution of fraction completed for the processes. This is only useful for the lower level interface since at the end of an experiment, all processes have completed.
- **Show All Graphs**: displays all of the graphs that have been previously hidden.
- **Show Files** puts copies of the configuration file, the run files and the experiment files in the history window.
- **Show All Table Data** displays a table containing the information that would be put in the log file when the **Log All Table Data** is pushed.
- **Draw Gantt Chart** will give a list of runs. Selecting one of these will draw a Gantt chart for that run.
- **Limit Logged Data**: Allows a selected subset of the saved data to be saved or graphed. See Selecting Data to be logged or graphed.

Below the buttons are two narrow progress bars showing the progress of the simulation. The first is red and shows the fraction of total CPU time that has been used so far. When the red bar reaches the end, all processes are complete. The second progress bar is blue and shows the fraction of processes that have completed.

Finally, at the bottom of the window are three buttons.

- **Help**: displays a help window which gives detailed descriptions of the various buttons used by the simulator. A message appears when the pointer is moved inside one of these buttons. Useful help information has only been implemented for a few of the buttons.
- **Reset**: removes all processes and initializes all variables. The table data previously saved is not erased.
- **Quit**: exits the simulator. If the log file was open it is closed.

Random and Not so Random Numbers

Java has a pseudo-random number generator which is used by default when values are selected from a distribution. One of the problems with using this generator is that unless all of the distributions are constant, different runs on the same data produce slightly different results.

There are times when you may want to repeat an experiment or run in order to look at it in more detail. The option **portable true** allows you to do this. The simulator uses a portable pseudo-random number generator with a fixed seed so that if the same run is repeated, it should produce exactly the same results. The seed is reset at the start of each run, so that a run is not affected by previous runs.

By default, the simulator uses one random number generator for calculating all distributions. CPU bursts and I/O bursts for processes are calculated on the fly for each process when they are needed. A CPU burst times is calculated when the process enters the ready queue. Since the scheduling algorithm can affect the order in which processes enter the ready queue, the same set of processes running under different scheduling algorithms will have different burst times. This makes it difficult to compare the algorithms.

If a seed value is specified in the Experimental Run file, independent random number generators are used for each instance of a distribution. This implies that if the same creator is used to create processes which are run through different algorithms, those processes will have exactly the same values for the CPU and I/O bursts for both runs.

Selecting Data to be Displayed or Graphed

By default, the **Log All Table Data** and **Draw Graph** buttons include data from all of the experimental runs made. You can use the **Limit Logged Data** button to determine which runs are included.

Pushing this button brings up a window containing one line for each run that has been finished. Each line contains the name of a run and two buttons. Pushing the **Log** button changes it to **NoLog** and indicates that the corresponding run should not be included. Pushing the same button again resets it. The other button is marked **Show**. When it is pushed the line disappears from the data editor window. You can show the missing lines by pushing the **Show Some** button which changes it to **Show All**. Pushing it again resets it. When finished, you can push the **Done** button.

Experimental Runs: Advanced

The first three lines of an experimental run file specify the name, comment, and algorithm as described previously. The next lines are optional, and any combination can occur, but they must be in the order specified below.

The first optional line contains one of three words, **priorityoff**, **priorityon**, or **prioritypreempt** corresponding to no priority, non-preemptive priority, and preemptive priority, respectively.

The second optional line starts with the key word **key** followed by a parameter which is the key to be used in the waiting time graphs. It could be any string.

The third optional line starts with the key word **seed** and has a parameter which is an integer. This is used to initialize the random number generator. See Random and Not So Random Numbers.

The fourth and fifth optional lines specify the context switch time. See Context Switch.

The file contains any number of sets of 7 lines giving the number of processes, first arrival time, interarrival distribution, duration distribution, CPU burst distribution, IO burst distribution and base priority as discussed earlier.

The file format is as follows with at most one of the blue lines and either or both of the red lines and the block of seven magenta lines repeated any number of times.

name *experimental_run_name*
comment *experimental_run_description*
algorithm *algorithm_description*
priorityoff
priorityon
prioritypreempt
key *key_string*
seed *seed_value*
cstin *context_switch_time_in*
cstout *context_switch_time_out*
numprocs *number_of_processes*
firstarrival *first_arrival_time*
interarrival *interarrival_distribution*
duration *duration_distribution*
cpuburst *cpu_burst_distribution*
ioburst *io_burst_distribution*
basepriority *base_priority*

Here is a sample run file which uses the second format which would be stored in the file **thisrun.run**:

```
name thisrun
comment This specifies two types of processes
algorithm FCFS
priorityon
key Non-preemptive FCFS with processes having two priorities
numprocs 10
firstarrival 0.0
interarrival constant 1.0
duration uniform 500.0 1000.0
cpuburst constant 50.0
ioburst constant 1.0
basepriority 1.0
numprocs 30
firstarrival 0.0
interarrival constant 1.0
duration uniform 50.0 100.0
cpuburst exponential 20
ioburst exponential 500
basepriority 2.0
```

This defines two types of processes, 10 of which are longer running use little I/O, and are of low priority and 30 of which are relatively short, use more I/O and have higher priority. The scheduling algorithm is non-preemptive First-Come, First-Served using fixed process priorities.

Context Switches

Be default, the simulator assumes that context switches occur instantaneously. A context switch consists of two operations, moving the current process out of the CPU and moving a new process into the CPU. By default, the simulator assumes that both of these times are zero. You can set either of these time in the run file or in the exp file using a run modification.

During a context switch in, the process remains int he ready queue so this time contributes to the waiting time of the process. When a context switch out occurs, the process changes state immediately to a switching state. In the Gantt charts, this is shown as a block space. If I/O is requested the I/O is delayed until the context switch out is done. If the process was preempted, it does not arrive in the ready queue until the context switch out is done.

Files Used by the Simulator

The simulator is very picky about the format of the files which describe experiments. If a word is spelled incorrectly, the wrong case is used in a key word, or lines are out of order the simulator will reject the entire file. When starting up you should always check the yellow **History** window for information that was read in. The following are sample lines that might be displayed in the window:

```
  Runs read:          5 of 5
  Experiments read:   3 of 3
```

Gantt Charts

The **Draw Gantt Chart** button in the last column of buttons is for producing Gantt Charts. If more than one run has been made, a menu pops up allowing you to choose which run to use. Select one of the runs and a Gantt Chart will be displayed. If only one run has been done, this step is skipped. A sample Gantt Chart is shown in Figure 5.

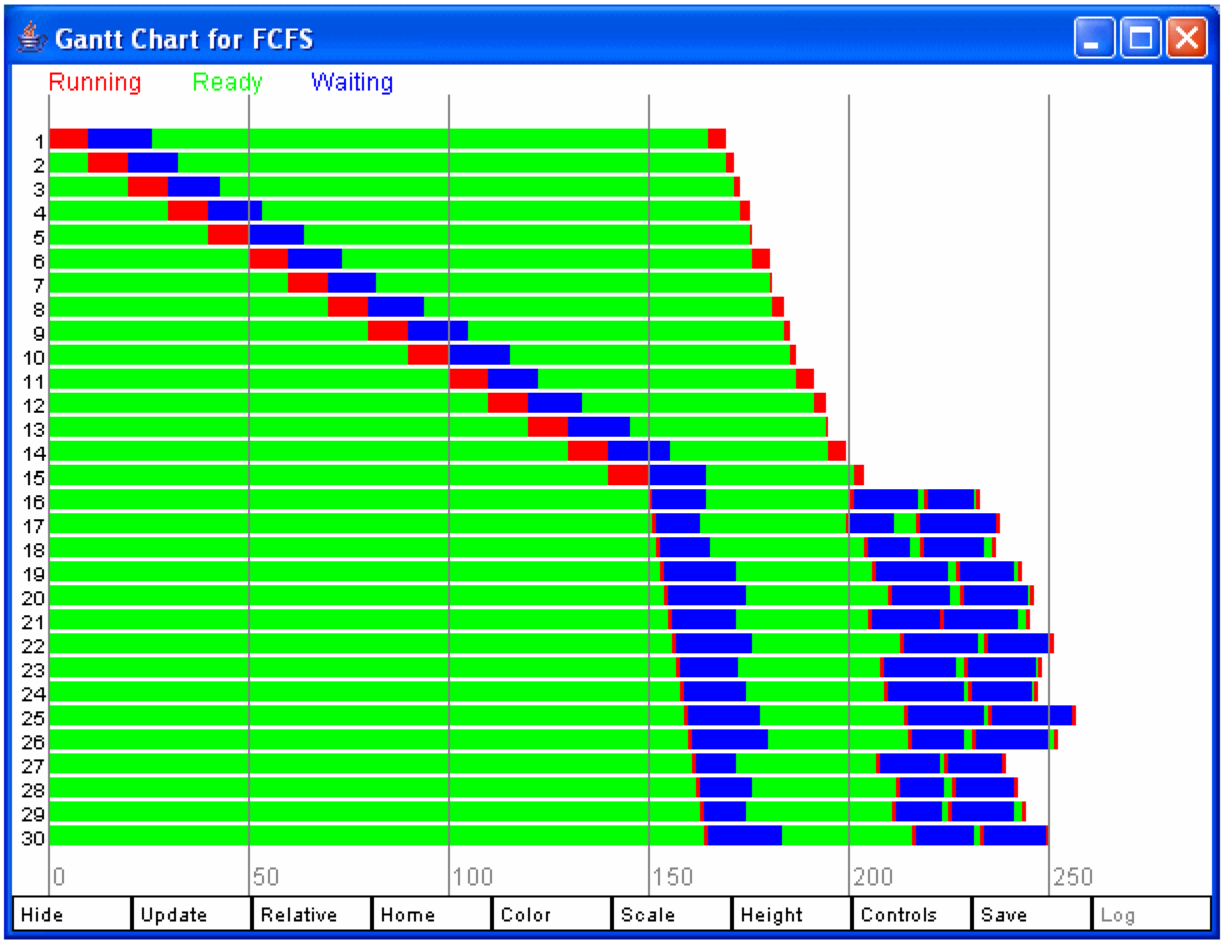
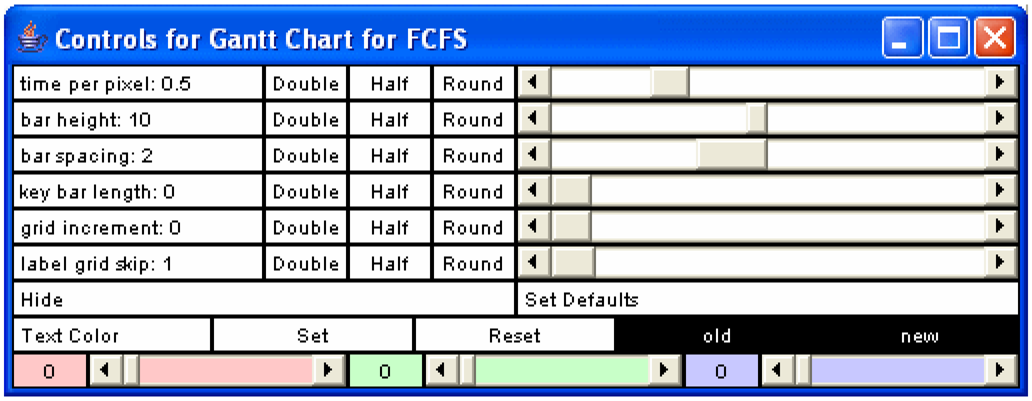


Figure 5: A Gantt Chart

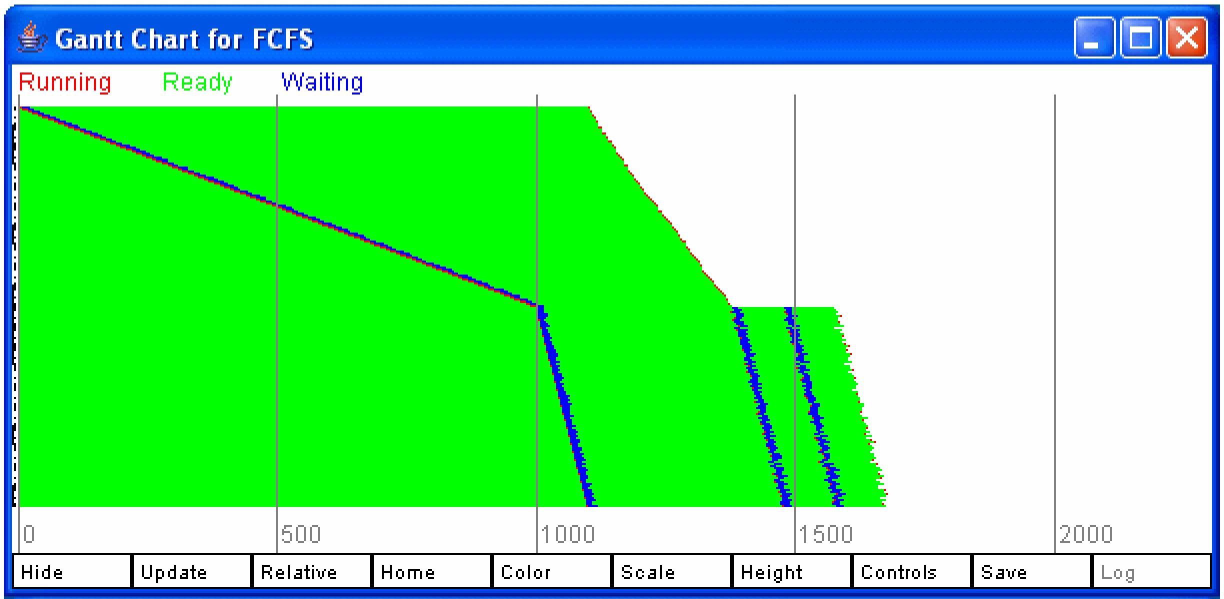
The `Log` button can be used to put a copy of the Gantt chart in the log file. This might take a while on a slow machine as each pixel needs to be grabbed from the image and then converted to a GIF file. The `Controls` displays additional sliders for controlling the height and spacing of the bars. It is possible to get a useful Gantt Chart of hundreds of processes by making the bars 1 pixel high. Figure 6 shows the controls window. Figure 7 shows two Gantt chars, each with 200 processes, one for FCFS and one for SJF. Figure 8 shows a zoomed in view of one of these.



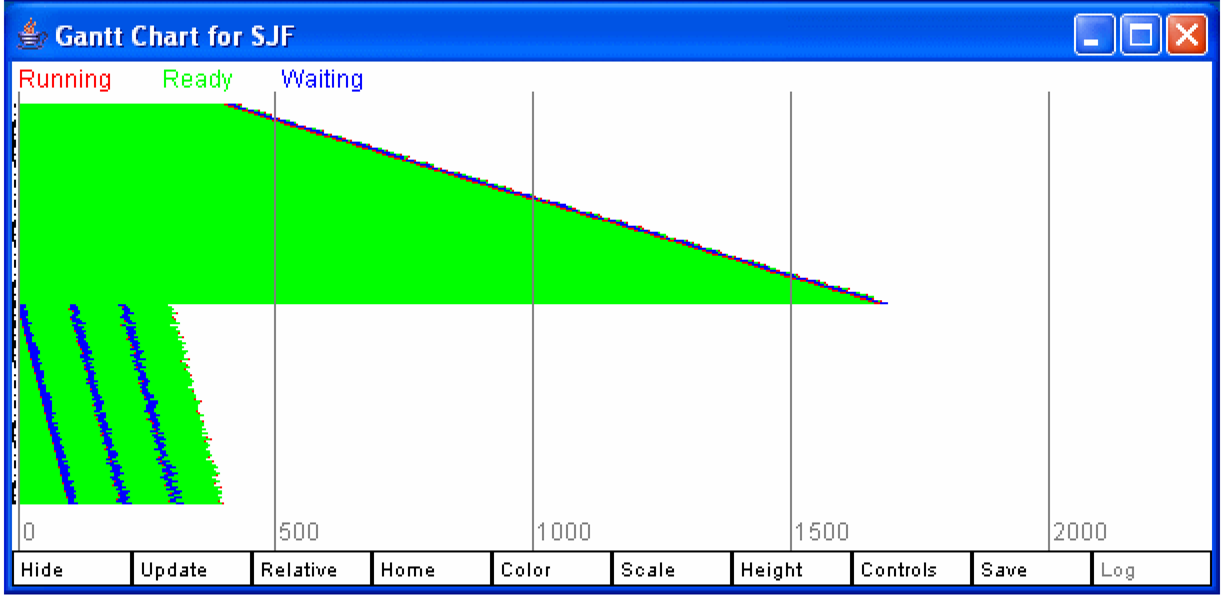
Controls for Gantt Chart for FCFS

time per pixel: 0.5	Double	Half	Round	
bar height: 10	Double	Half	Round	
bar spacing: 2	Double	Half	Round	
key bar length: 0	Double	Half	Round	
grid increment: 0	Double	Half	Round	
label grid skip: 1	Double	Half	Round	
Hide			Set Defaults	
Text Color	Set	Reset	old	new
0		0		0

Figure 6: Gantt Chart Controls.



Gantt Chart for FCFS



Gantt Chart for SJF

Figure 7: Two Gantt Charts, each with 200 processes.

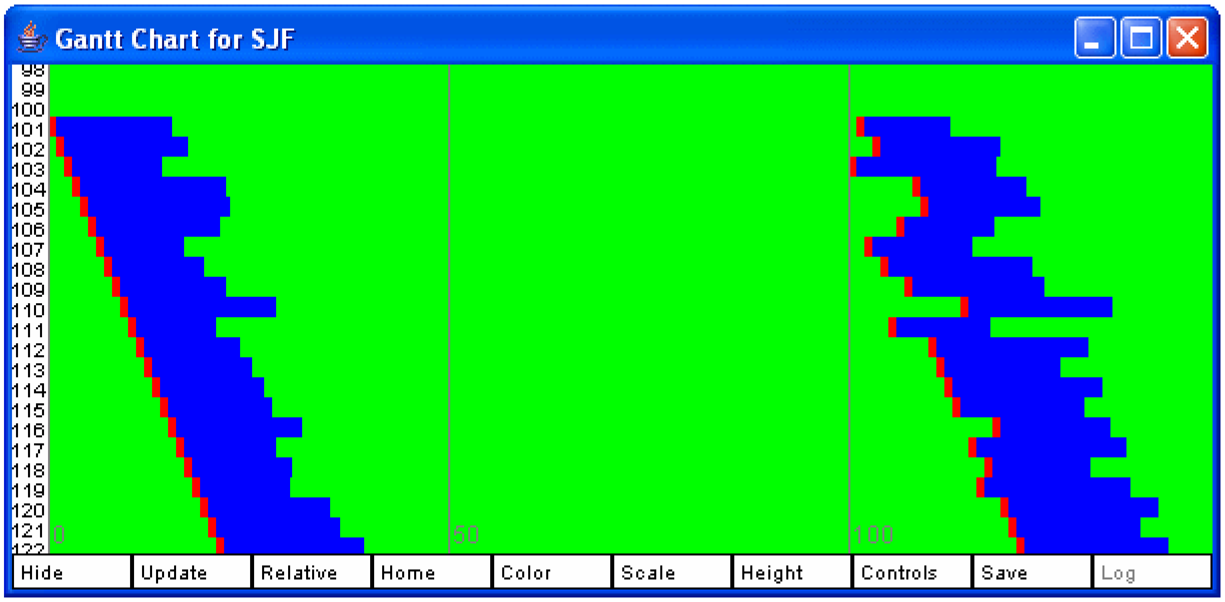


Figure 8: A zoomed in view of a Gantt Chart from Figure 7.

Additional Features

The **History** and **Event** windows of the simulator are scrollable text boxes that can contain useful information. Clicking on the name of the box turns on automatic logging to that box. For example, clicking on **Event Log (off)** changes this to **Event Log *time*** and causes all events to be logged in this window.

Each of these windows has a **Clr** button at the top to empty it of text and a **Log** button which is active whenever the log is open. Pushing this button inserts a copy of the contents of the box into the log file.

There are two small buttons on either end of the top of these boxes. Pushing the leftmost one reduces the size of the font displayed while pushing the rightmost one increases the size of the font. These do not affect what gets put into the log file.

I/O bursts of 0 are treated as very small I/O bursts. This means that when a process finishes its CPU burst the scheduler picks a different process from the ready queue and then puts the process back in the ready queue. Only if the ready queue was empty will that same process be given the CPU again.