Experto A

Basic Operation

Start the simulator. If all goes well the simulator will start up and look something like Figure 1.

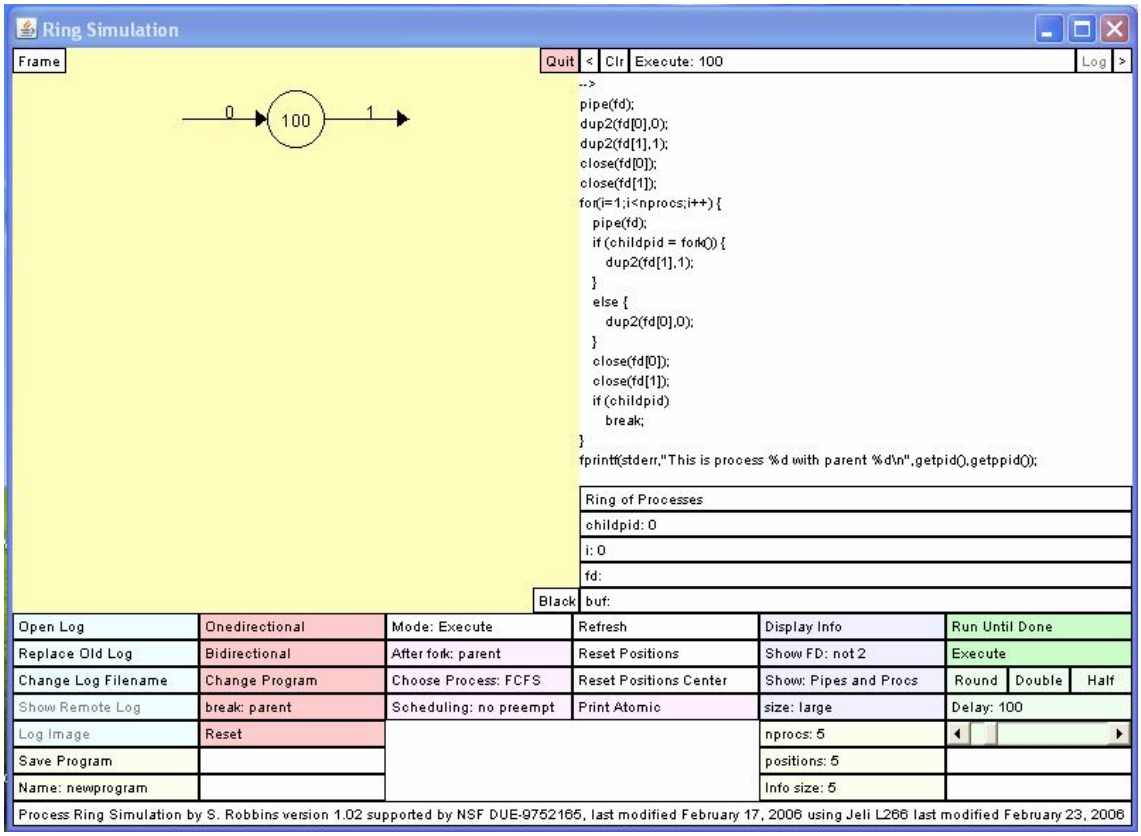

Figure 1: The main simulator window.

The figure shows one process with arrows representing standard input and standard output. Push the green **Run Until Done** button on the right side of the window and the program will step through its execution. After a few seconds the program will terminate and the screen will look like Figure 2. The rectangles represent pipes. To exit the simulator, push the pink **Quit** button that is in the upper right corner of the the yellow ring diagram. To start the program from the beginning, find the **Reset** button in the second column of buttons under the diagram.

Each process is represented by a circle with its process ID in the center. The first process ID is always 100 and as processes are created by executing **fork** calls, the process IDs are incremented, assuming no other processes are created in the system. Arrows out of the process represent file descriptors that can be used for output. Arrows into the process represent file descriptors that can be used for input. By default, standard error is not displayed since it clutters the diagram.

When a **pipe** call is executed, a rectangle representing the pipe is displayed with two new file descriptors connected to the process. A **dup2** call moves the file descriptor arrows appropriately and the **fork** call creates a new process. When processes and pipes are created, they are positioned so that when the standard program is run, the processes and pipes will appear in a ring. The processes and pipes can be moved around on the

screen by dragging them with the mouse. The pipes can be rotated by dragging close to the corner of the pipe rectangle. The numbers that represent the file descriptors can also be dragged.

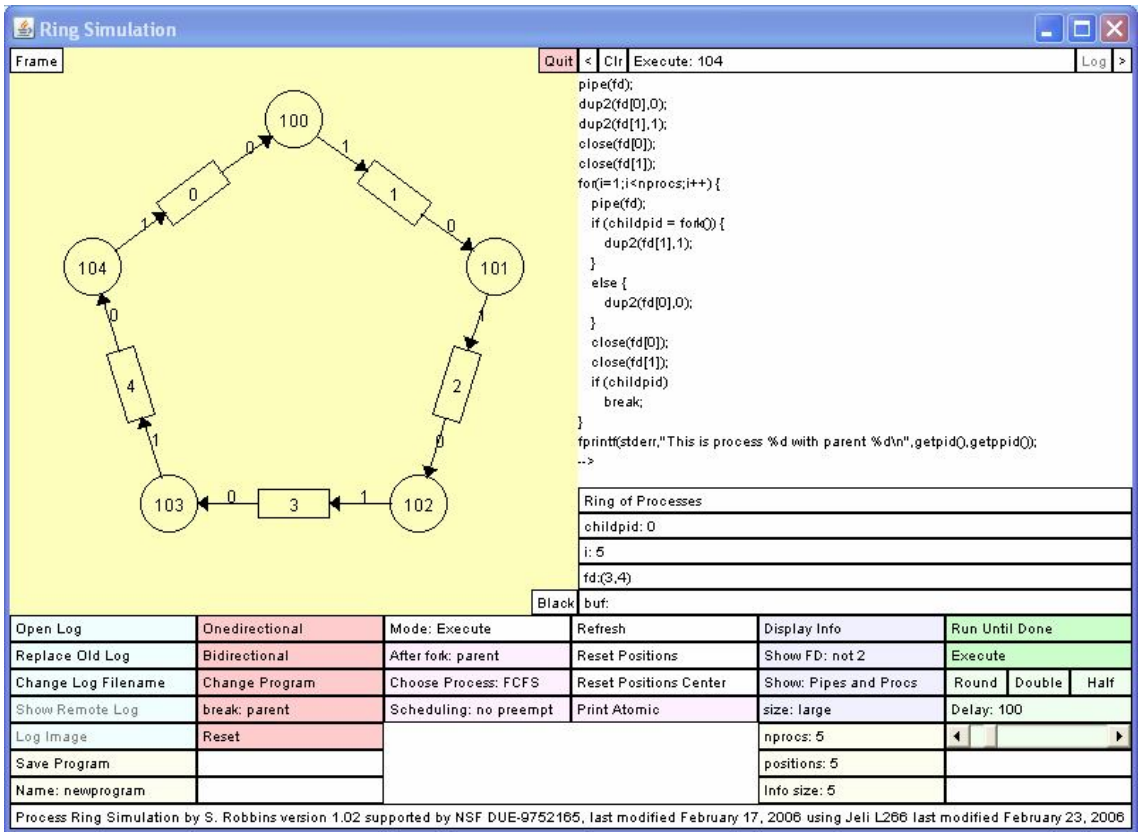

Figure 2: The main simulator window.

To the right of the diagram is the program window. The arrow is above the instruction to be executed next. The process ID of the active process is shown in the **Execute: pid** button at the top of this window. Clicking on this button will cycle through the active processes. Just below the program window are 5 labels showing the name of the program (if it has one, the default program being called **Ring of Processes**) and the values of the local program variables. These include the integers **childpid** and **i**, the **fd** array and the contents of the buffer, **buf**. Below the diagram and the program are 6 columns of buttons for controlling the simulation.

There are two basic ways of running the program. The green **Run Until Done** button on the rightmost column of buttons will start the program running and automatically step through the program at a given rate. The rate can be modified using the light green slider labeled **Delay** that is located below the button. The delay is in milliseconds and represent the amount of time the simulator pauses after each step is displayed. The limiting factor on the speed of execution is the time it takes to redisplay after each step. When the **Run Until Done** button is pushed, it changed to a **Stop Running** button which can be used to stop the execution. In this mode of operation, the scheduling of the processes is controlled by the third column of buttons as described later. The green **Execute** button executes one line of code. This can be used to single-stop through the program. After each instruction is executed, you may push the **Execute: pid** button at the top of the program to change which process will be executed next.

Experto B

Simple Program Modifications

The second column of buttons can be used to make simple modifications to the program. The **Onedirectional** button loads the standard default ring program. The **Bidirectional** button loads a more complicated version that creates a bidirectional ring of processes. The **Change Program** button presents a list of loadable programs. Each program listed corresponds to one line in the configuration file **ringconfig**. Adding lines to this file allows you to choose additional programs. Programs can be created using a standard text editor or by the simulator. The **break** button allows you to cycle through possibilities for breaking out of the main for loop of the program. The table below gives the relationship between the **Break** button and the code that is used:

Button	**Break: parent**	**Break: child**	**Break: parent and child**	**Break: none**
Code	`if (childpid) break;`	`if (!childpid) break;`	`break;`	none

The **Reset** button starts the program from the beginning again.

Scheduling

The simulator allows control over several aspects of the scheduling of the processes using the buttons in the third and fourth column. The **After fork:** button controls which process executes after a fork. The possibilities are parent, child, either, and random. Either means that either the child or parent will execute and the choice is made randomly with equal probability. Random means that any ready process can execute and the choice will be made among the ready processes with equal probability. The **Choose Process** button controls which process will execute next when the running process loses the CPU. A process can lose the CPU by having its quantum expire, by blocking on a wait for child call, by blocking on a read from a pipe, or by terminating. The choices are **FCFS** (the process that entered the ready queue first is chosen), **Next** (the ready process with the next highest PID is chosen) and **Random** (a random ready process is chosen). The **Scheduling** button controls the process scheduling algorithm. The default is **no preempt** in which a process runs until it blocks or terminates. When **RR** is chosen, round robin scheduling is used and a slider appears below the scheduling button allowing am integer quantum to be set. The quantum represents the number of instructions to be executed before the process loses the CPU. When **Random** is chosen, a slider appears below the scheduling button allowing a probability to be set. This represents the probability that a process will lose the CPU after executing an instruction. To the right of the **Scheduling** button is the **Print** button. The default is **Print Atomic** in which the output of **fprintf** is assumed to be atomic. If **Print Not Atomic** is chosen, a probability slider appears which gives the probability of losing the CPU after each character of the **fprintf** is output.

Display Information

There are several ways to display information about the state and history of the simulation. Most of these are controlled by the **Display Info** button at the top of the 5th column of buttons. Clicking on this lists some items that can be displayed. Some of

these items may be disabled if the corresponding information is not available. The list of display options includes:

- **Display Output** to show the output generated by the **fprintf** instructions. Each process has a color associated with it and the output of each process is shown in the color of that process. The output can be displayed entirely in black by pushing the **Color** button at the bottom of this frame. This changes the button to a **Black** button. Pushing it again will change the display back to color. The processes in the main diagram can also be displayed in color by pushing the pick **Black** button that is in the lower right corner of the diagram window.
- **Display Process Info** shows information about all processes including the process ID, the original parent process ID (the ID of the process that created it), the actual parent process ID as reported by **getpid()** (that may be the ID of the init process) and the current process state. The state will be one of: Running, Ready, Waiting for child, Read Blocked, Semaphore Blocked, Zombie, or Done.
- **Display History** will display a list of all instructions executed. Each instruction is preceded by the process ID of the process executing the instruction.
- **Display Gantt Chart** displays a Gantt chart showing the state history of each of the processes.
- **Variables** displays the values of the variables for each process.
- **Semaphores** displays each semaphore and either its value or the list of waiting processes.
- **Frames** displays all of the frames created by the **Frame** button.
- **Commentary** displays the commentary for the current programs if these is one.

You can show information about a particular process by clicking on the circle representing that process in the yellow display window. This pops up two windows, one showing the history (instructions executed) of that process and one showing the current values of the variables for that process. In addition, a frame containing information about a given pipe can be displayed by clicking on the pipe in the diagram. Two Text Displays appear, the upper one containing the current contents of the pipe and the lower containing a history of everything that has been read from the pipe.

Experto C

Program Format

The simulator allows a fairly general program that creates a collection of processes and pipes. The original version of the program was intended to simulator a process ring and has the format given below. The ring format is as follows:

```
<instructions>
for (i=1;i<nprocs;i++) {
  <instructions>
  if (childpid = fork()) {
     <instructions>
  }
  else {
     <instructions>
  }
  <instructions>
  <optional conditional break instruction>
}
<instructions>
```

The program assumes the following data:

```
int i;
int childpid;
int fd[2];
int fd1[2];
char buf[BUFSIZE];
```

where `BUFSIZE` is assumed to be large enough to handle anything generated by the program. In addition, a constant called **nprocs** is used in the **for** loop.

Assumptions:

- The buffer, `buf` is of unlimited size and the instructions guarantee that it will always contain a string.
- Pipes are of unlimited size and writes to a pipe are atomic.
- Reads from a pipe are atomic and a read will read everything in the pipe.
- A read from a pipe will block only if the pipe is empty and there are processes that have the pipe open for writing.
- A write to a pipe will always write the full amount requested unless there are no processes that can read from the pipe. In this case a SIGPIPE signal is simulated and the process terminates.
- The program does not check return values of system calls and so no error checking is done. If an error occurs such as a `dup2` with improper argument or a write to a closed or never opened file descriptor, the operation is ignored.
- No operations are performed on standard error except with the `fprintf` instructions. A write, close, or dup2 on standard error will produce undetermined results.

Changing The Program

The program can be modified from inside the simulator and saved for later use. The first button in the third column is the **Mode** button. It starts with **Mode: Execute** which allows for the running of the program. Clicking on this button changes it to **Mode: Program** and 6 new buttons appear below it. The first two buttons can be used to move the pointer up and down. The pointer is shown as an arrow in the program window in the upper right part of the main display. The next two buttons can be used to add a new instruction to the program after the pointer or to delete the instruction after the pointer.

The **Add Instruction** pops up a menu listing the possible instructions to add. The **Delete Instruction** button is active only when there is a valid instruction after the pointer. Some instructions cannot be deleted such as the `for` loop, the `if (childpid=fork())` and the `else`. The next button allows most instructions to be modified by conditionals such as `if (i==1)` or `if (i!=1)`. These are most relevant after the for loop because at this point only the original parent has i=1. To run the modified program, click on the **Mode** button again to return to execute mode.

The current program can be saved using the yellow **Save Program** button in the first column of buttons. This saves the program using the file name given in the button below. You can change the name of the file to be used by clicking on this **Name:** button. By adding a program line to the **ringconfig** file, these programs can be accessible to later runs of the simulator.

Recordando

La instrucción dup2 (oldfile, newfile) hace que newfile sea una copia de oldfile, cerrándo primero newfile si es necesario. Así por ejemplo, la instrucción dup2 (fd[0], 0) hace que la entrada estándar provenga del extremo de lectura del pipe.

En C el valor true es cualquier entero distinto de 0. El valor false es 0.

En C la instrucción break hace que se aborte el lazo que la contiene.

Experto D

Local Loggin

The simulator can display information in a log file. Log files are in HTML format and can be displayed by or printed from a standard browser. The first column of light blue buttons controls the logging functions of the simulator. Initially, these buttons are used to control the logging functions before the log file is open or after it is closed. The first button opens the log. When the log file is successfully opened most of the buttons change their function to control the log functions appropriate when logging is in progress.

While the log is closed, the second button toggles between the modes **Replace Old Log** and **Append to Log**. In the former case, opening the log file overwrites any file with the same name that already exists. In the latter case, the new log is appended to the old one. The third button can be used to change the name and directory used for storing the log file. Pushing this button pops up a dialogue box in which the user can change the directory and log file names. The names of the files for storing the graph image files created by the simulator can also be changed. The **Show Remote Log** button can be used to pop up a browser window containing the log file generated when the log is stored remotely.

After the log is successfully opened, the **Open Log** button changes to **Close Log**, the second button changes to **Log Comment**, and the third button changes to **Stop Log**. **Close Log** terminates logging and closes the log file. **Stop Log** temporarily stops adding information to the log file but keeps the log file open. When pushed, this button changes to **Start Log** which resumes the logging. The **Log Comment** button pops up a window which allows the user to enter comments into the log file. The **Log Image** button puts a copy of the currently displayed diagram in the log file.

Most of the frames that display history information have there own **Log** buttons which becomes active when the log is open and stores the information displayed in the log file. The program window on the right side of the main simulator window also has a **Log** button which can put a copy of the program in the log file. The Gantt charts can also be logged.

Configuration

Configuration is controlled by default by the file **ringconfig**. The simulator can be started with an optional command line parameter giving the name of the configuration file. The configuration file contains lines which can be used to set values for various simulator parameters. The following configuration parameters are supported:

- **user username** sets the name displayed in the log file to **username**
- **size n** sets the size of the diagram to at least n by n pixels.
- **directory dirname** sets the directory for containing the log file to **dirname**
- **nprocs n** sets the value of the `nprocs` program variable to **n**.
- **rundelay n** sets the delay between running steps to **n** milliseconds.
- **program filename** causes the simulator to read in the program with the given filename. The name appears in the list under the **Change Program** menu. There may be several of these lines in the configuration file.
- **infosize n** sets the initial number of lines for processes in the **Process Info** frame to **n**.

- **positions n** sets the number of positions for processes around the circle to be **n**.
- **small** sets the diagram to use small processes and pipes.
- **large** sets the diagram to use large processes and pipes.
- **showFD** true turns on the display of all file descriptor arrows.
- **showFD false** turns off the display of all file descriptor arrows.
- **showFD2 true** turns on the display of the standard error file descriptor arrow.
- **showFD2 false** turns off the display of the standard error file descriptor arrow.
- **parent-child true** turns on the display of the arrow from parent to child.
- **parent-child false** turns off the display of the arrow from parent to child.
- **nopreempt** sets the scheduling to non-preemptive.
- **quantum n** sets the scheduling so that a process loses the CPU after executing n instructions.
- **random p** sets the scheduling so that with probability p a process loses the CPU after executing an instruction.
- **FCFS** sets the scheduling so that the process chosen from the ready queue is the one that arrived first.
- **inorder** sets the scheduling so that the process chosen from the ready queue is the one with the next larger process ID after the current process.
- **outoforder** sets the scheduling so that a random process is chosen from the ready queue.
- **fork parent** allows the parent to continue executing after a fork.
- **fork child** causes the child process to execute after a fork.
- **fork either** causes either the parent or child process to execute with equal probability after a fork.
- **fork random** causes a random process from the ready queue to execute after a fork.
- **remoteport n** sets the port for remote logging to n. This would not normally be used.
- **prompt pmt** sets the default prompt output when the original process
- **comfont n** sets the default size for the commentary font to n.
- **comrows n** sets the default number of rows for the commentary frame to n.
- **comcolumns n** sets the default number of columns for the commentary frame to n.