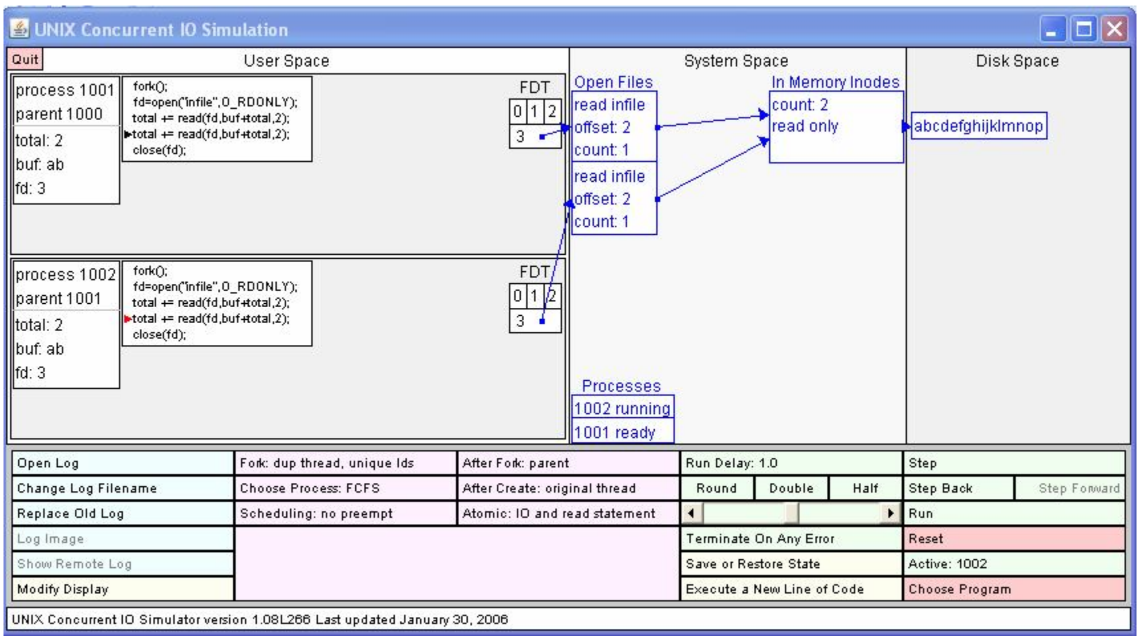
Experto A

Basic Operation

Start the simulator. Figure 1 shows the simulator with two processes after it has been executing for a while.



process 1001
parent 1000
total: 2
buf: ab
fd: 3

```
fork();
fd=open("infile",O_RDONLY);
total += read(fd,buf+total,2);
total += read(fd,buf+total,2);
close(fd);
```

FDT: 0 1 2 / 3

process 1002
parent 1001
total: 2
buf: ab
fd: 3

```
fork();
fd=open("infile",O_RDONLY);
total += read(fd,buf+total,2);
total += read(fd,buf+total,2);
close(fd);
```

FDT: 0 1 2 / 3

Open Files:
read infile
offset: 2
count: 1

read infile
offset: 2
count: 1

In Memory Inodes:
count: 2
read only

abcdefghijklmnop

Processes:
1002 running
1001 ready

Fork: dup thread, unique Ids
Choose Process: FCFS
Scheduling: no preempt
After Fork: parent
After Create: original thread
Atomic: IO and read statement
Run Delay: 1.0
Active: 1002

UNIX Concurrent IO Simulator version 1.08L266 Last updated January 30, 2006

Figure 1: The main simulator window.

The screen is divided into sections. The top part of the screen shows a pictorial representation of the computer system. The bottom part has a collection of clickable buttons and other controls that modify the simulator properties and run the simulation.

The top part of the screen is divided into three sections. These sections, from left to right are the User Space which shows the processes, the System Space which shows the system open file table, the in-memory inodes and the list of processes, and the Disk Space which shows the disk blocks. The User Space is divided into two regions, one for each process. For each process, the box on the far left shows the ID of the process and its parent, as well as the values of variables. To the right of the variables is a box showing the program being executed. If the program has not yet completed, a triangular arrow indicates the next instruction to be executed (the program counter). The arrow is red if the process is in the CPU (process 1002). It is hollow if the process is suspended (blocked I/O). On the right side of the User Space of each process is the file descriptor table (FDT) for that process. The first three entries corresponding to standard input, standard output, and standard error are shown on the first line of the FDT. Since the simulator is concerned with file I/O, the contents of these entries are not shown. Each of the other open file descriptors points to an entry in the system file table (SFT) in the System Space. Each SFT entry shows the file being accessed, the current file offset and the count of the number of open file descriptors using this entry. To the right of the open files is the in-memory inodes. Under this is the list of active processes, indicated by its ID (process ID). The state of each process is also given, Running, Ready, Waiting, or Zombie. Lastly, the Disk Space shows the contents of the files. You can easily try the simulator with the included programs. Figure 1 corresponds to Program 1.

Use the Choose Program button in the lower right corner to pick one of these program and either step through it with the Step button or run it with the Run button. When the program is running, the Run button changes to Pause and the Reset button changes to Abort Run. When paused, the Pause button changes to Resume. To exit the simulator, push the pink Quit button that is in the upper left corner of the the frame.

The buttons and controls are arranged in 5 columns. The first column controls the logging features and the display. The second and third columns control the scheduling of processes. The Fourth column controls the rate at which the program runs, save or restore the state of the simulator, and allows you to execute an instruction on the fly. The last column is for running the program. In the last column, the step button will execute the next instruction. The Step Back and Step Forward buttons allow you to move through already executed parts of the program. The Run button will run the program until it completes or an error occurs. It will run at a rate controlled by the slider to the left of the run button. The delay between instructions is in seconds. The Reset button will reset and start the program from the beginning. Below this is the Active button which shows which process has the CPU. Clicking on this button does a context switch. Lastly, the Choose Program button allows you to choose which program to run.

Experto B

Simple Programs

This section describes the contents of a simple program file. Each program has access to a number of variables that are assumed to be correctly declared. No variables are initialized, and using the value of an uninitialized variable is considered a fatal error. The simulator terminates a run when a fatal error occurs. The available variables are:

```
int totaln;
char bufn[LARGE_SIZE];
int childn;
int fdn;
```

The `bufn` are assumed to be arrays of arbitrarily large size. The *n* can be any non-negative integer, or can be omitted completely. These allow such variables as `child0`, `child5`, `child123`, `child` or `fd7`. Each program line has one of the following forms:

```
fdn = open("string",O_RDONLY);
totaln += read(fdm,bufn+totaln,p);
fdn = open("string",wrflags,0777);
fdn = open("string",wrflagst,0777);
fdn = open("string",wrflagsa,0777);
fdn = open("string",wrflagsta,0777);
write(fdn,"string",p);
close(fdn);
fork();
childn = fork();
if (childn) fork();
if (childn) childm = fork();
if (!childn) fork();
if (!childn) childm = fork();
childn = wait(NULL);
if (childn) {
if (!childn) {
else {
}
```

Here, *string* represents an arbitrary sequence of printing characters, *n* and *m* are non-negative integers (0, 1, 2, etc.) and *p* is a positive integer (1, 2, 3, etc.). The last 4 instruction types allow different blocks of code to be executed by parent and child processes. The open for write instructions use the following write flags that allow opening use of the append and truncate flags:

```
int wrflags = O_WRONLY | O_CREAT;
int wrflagst = O_WRONLY | O_CREAT | O_TRUNC;
int wrflagsa = O_WRONLY | O_CREAT | O_APPEND;
int wrflagsta = O_WRONLY | O_CREAT | O_TRUNC | O_APPEND;
```

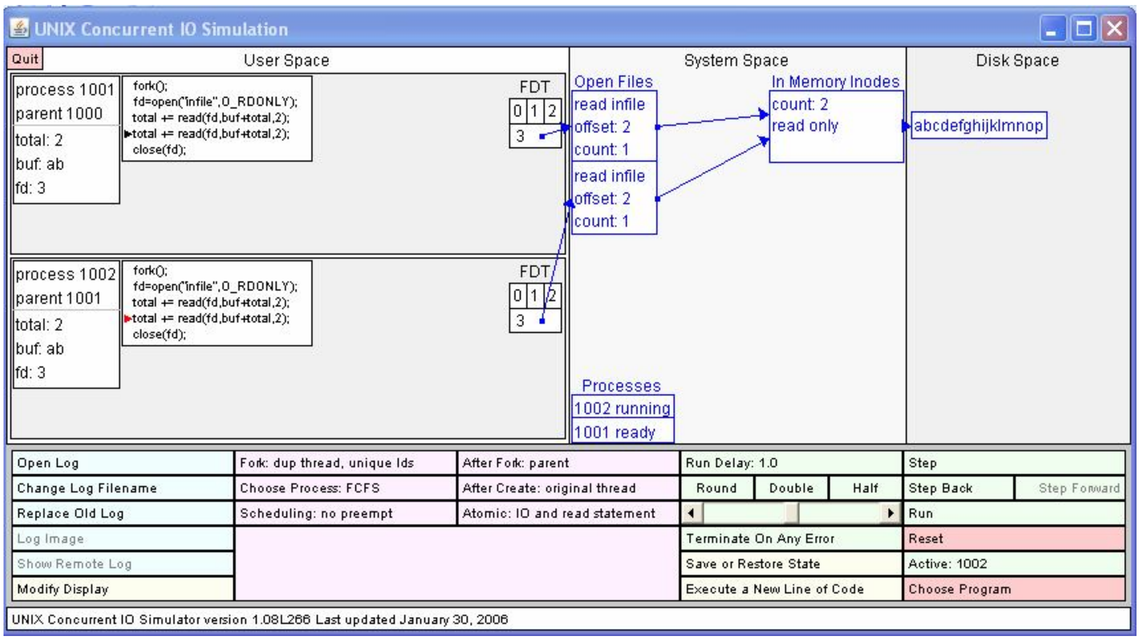
The simulator does little checking on the syntax of the program file lines. These lines are only checked so as to distinguish one type of instruction from another. For example, only three of the above lines contain an open brace, so the following are valid lines in a program file:

```
{
child2 {
{ !child4
```

and these are interpreted as:

```
else {
if (child2) {
if (!child4) {
```

Experto C



process 1001
parent 1000
total: 2
buf: ab
fd: 3

```
fork();
fd=open("infile",O_RDONLY);
total += read(fd,buf+total,2);
total += read(fd,buf+total,2);
close(fd);
```

process 1002
parent 1001
total: 2
buf: ab
fd: 3

```
fork();
fd=open("infile",O_RDONLY);
total += read(fd,buf+total,2);
total += read(fd,buf+total,2);
close(fd);
```

read infile
offset: 2
count: 1

read infile
offset: 2
count: 1

count: 2
read only

abcdefghijklmnop

1002 running
1001 ready

Fork: dup thread, unique Ids
Choose Process: FCFS
Scheduling: no preempt
After Fork: parent
After Create: original thread
Atomic: IO and read statement
Run Delay: 1.0
Active: 1002

UNIX Concurrent IO Simulator version 1.08L266 Last updated January 30, 2006

Figure 1: The main simulator window.

Scheduling

The simulator allows control over several aspects of the scheduling of the processes using the purple buttons in the second and third columns. The scheduling parameters determine when context switched take place and which process is chosen when there is a context switch.

In addition, the top button in the second column determines what is duplicated when a fork occurs. The POSIX thread library specifies that a fork only duplicates the currently running thread, not all of the thread of the process. This is the default for the simulator.

The second button in the second column determines how the next process is chosen when a context switch takes place. The possibilities are:

- FCFS: the next process in the ready queue is chosen.
- Next: the ready process with the next highest ID is chosen.
- Random: a random ready process is chosen.

The third button in the second column controls when context switches take place. The possibilities are:

- Nonpreemtive: No context switch for after most instructions.
- RR: round robin scheduling. Under the scheduling button will be a slider to control the quantum. The quantum indicates how many instructions can be executed before a process is removed from the CPU.
- Random: Under the scheduling button will be a slider to control the probability that the CPU is lost after an instruction is executed.

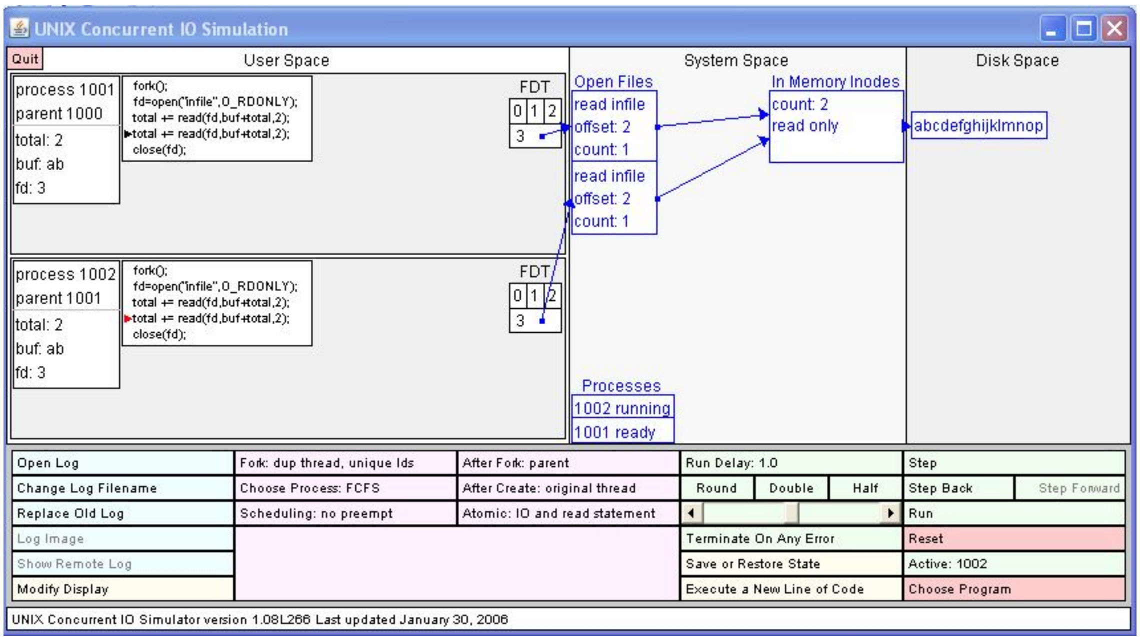
The first button of the third column controls what process executes after a fork instruction. The possibilities are:

- parent: the parent always continues execution unless a context switch is forced by random or round robin scheduling.
- child: the child process always executes next after a fork.

- either: Either the parent or child will execute next, chosen at random with equal probabilities.
- Random: A random ready process is chosen.

The third button of the third column indicates whether I/O is atomic. If I/O is atomic, then a process never loses the CPU during a read or write operation. If I/O is not atomic, under this button will be a slider giving the probability that the instruction loses the CPU after each I/O byte is processed. When stepping through a program using non-atomic I/O, each step processes one byte. When a non-atomic I/O instruction is executing, the triangular program counter arrow is preceded by an integer giving the number of bytes processed so far. For a read operation, the total number of bytes to read is given in the instruction, but is reduced if there are fewer bytes in the file. For a write operation, it is considered a fatal error to write more bytes than in the string in the second parameter of the write instruction.

Experto D



```
process 1001
parent 1000
total: 2
buf: ab
fd: 3
```

```
fork();
fd=open("infile",O_RDONLY);
total += read(fd,buf+total,2);
▶total += read(fd,buf+total,2);
close(fd);
```

```
process 1002
parent 1001
total: 2
buf: ab
fd: 3
```

```
fork();
fd=open("infile",O_RDONLY);
total += read(fd,buf+total,2);
▶total += read(fd,buf+total,2);
close(fd);
```

```
read infile
offset: 2
count: 1

read infile
offset: 2
count: 1
```

```
count: 2
read only
```

```
abcdefghijklmnop
```

```
1002 running
1001 ready
```

UNIX Concurrent IO Simulator version 1.08L266 Last updated January 30, 2006

Figure 1: The main simulator window.

Local logging

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. The second 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. While the log is closed, the third 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.

After the log is successfully opened, the Open Log button changes to Close Log, the second button changes to Stop Log and the third button changes to Log Comment. 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.

Advanced Configuration

The default configuration file is called ioconfig. The table below lists some configuration options.

Keyword	Values	Meaning
user	anything	The name of the current user. This appears in the log file.
quiet	none	Turns off all sounds generated by the simulator.