# FileView

| | |
|---|---|
| *Assignment Out:* | March 19, 2009 |
| *Code Due:* | April 3, 2009 |

## 1 Purpose

This assignment should serve as an introduction to the C language and to systems programming.

Your task is to create a viewer that shows the files that are currently being used by a given process identified by its process id. The viewer should periodically update the view to show when files are closed and new ones opened. It should also display information about the files such as their lengths and update this information as the program runs.

## 2 Getting the Information

While you might think that looking inside a process to determine what it is doing with files might be quite difficult, in reality, on Linux at least, it is rather easy.

There is a directory /proc that contains a large collection of information about what the machine is currently doing including what each process is doing. For each process, identified by its process id, there is a subdirectory /proc/process id that contains process-specific information. One of the subdirectories in this directory is *fd* which contains one entry for each file currently open by the process. The name of the entry is the file number (a small integer), and the contents of the entry are a soft link to the actual file.

Thus all you need to do, given the process id, is to find the corresponding fd directly, list all the files there, get the contents of the link and any relevant file information, and then display the result. There are some complications however.

First the easy things. You should use the opendir and readdir calls to handle scanning a directory. You should use the readlink call to get the contents of a link and either stat or lstat (depending on what information you are looking for) to get information about the file specified by the link.

Next the tricky things. Not all files are created equal. Some files are pipes. These have dummy link names of pipe:[#] where # is an identifying inode number. You can tell which end the pipe is by looking at the file permissions on the link. You might be able to find the other end of the pipe by finding another file entry for a pipe with the same inode number but the opposite permissions.

Other files are sockets. These can be UNIX domain sockets, TCP sockets, or UDP sockets. All the information the link gives you, however, is socket:[#] where # is the inode number. To find out more about the socket you need to look in the /proc/net directory. Here there are several files

that might let you gleam more information about a socket given its inode number. The particular files are unix, tcp, tcp6, udp and udp6. A unix domain socket is identified by its file name in the system (which is in the unix file). A tcp socket is identified by its source and target endpoints (host id and port number); these can be found in either file tcp or tcp6 depending on how the socket was created. A udp socket is identified by is local address (host and port).

Opening and reading from the `tcp` and `udp` file descriptors generates text in the following format (try `cat /proc/net/tcp` to see the current TCP sockets established on your machine):

```
 sl  local_address rem_address   st tx_queue rx_queue tr tm->when retrnsmt uid timeout inode
 42: 140A740A:820A 0601740A:006F 06 00000000:00000000 03:00000E1F 00000000 0   0       0 2 e98
```

The local and remote addresses consist of a host address and port number separated by a colon. You are responsible for parsing this information and storing it in a readable hexadecimal format.

The format of streams for UNIX domain sockets is:

```
Num RefCount Protocol Flags    Type St  Inode  Path
23: 00000003 00000000 00000000 0001 03  7849   @/var/run/hald/dbus-w3CCLgNblA
```

The file names in the `/proc` directory correspond to this inode number. You should iterate over the socket's contents and compare inode numbers with the file descriptor names to find the appropriate file (`fgets`, `atoi`, and `sscanf` should come in handy here).

## 3 Display and Interaction

You should create a textual display of the current open files within a window. To do this you should use the ncurses package. The display should update periodically (say every 3 seconds).

See `http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/` for a good introduction to curses programming, or browse the manpage (`man ncurses`). You will need to look up and use the following functions to get started:

- initscr() and endscr(): (de) initializes the curses library
- cbreak() or raw(): disables input buffering (your program will be able to get a key as soon as the user types it – without a newline)
- noecho() : suppresses echoing of user input
- nonl() : disables terminal driver newline handling
- intrflush(stdscr, 0) : disables flushing of the input buffer when an interrupt key is pressed
- keypad(stdscr, 1) : enables arrows and F1, F2, ...

Before you output text to the display, use `move(row, col)` to place the cursor at the correct position on the screen. To output text to the display, you need to use ncurses' output functions. Feel free to choose between the `addch()`, `printw()`, or `addstr()` classes of functions.

You might want to do some highlighting in the display, for example emphasizing new files when they are opened, deemphasizing closed files (as opposed to simply removing them), or highlighting changed information such as file lengths.

Your program should take keystroke input during the display (again using curses). The minimum here is to accept 'q' as a command for the application to quit. You can take other input, for example, to determine the sort order of the display, to show additional screens of files, or to limit the types of files that are being displayed. In addition you should catch SIGTERM and write out a text file describing all the currently opened files into a file 'files.dmp' in the current directory.

For extra credit, try implementing scrolling on a process that has a larger number of file descriptors than the screen can hold (eg Firefox). Any non-trivial uses of the `ncurses` will also be rewarded with extra-credit.

To find the pids of the processes you are currently running, type `ps -u [your username here]`. Firefox or your window manager usually has a lot of different types of files open that you can use to test your program.

## 4 Support Code

We have provided a Makefile and stencil code for this project. Find it at /course/cs032/pub/fileview. After copying the files over to your personal directory, you can compile, run, or debug your program using the provided Makefile.

- make compile : compiles your code
- make run : runs your code
- make debug : starts up a new terminal, and runs your program in that window while displaying debug output on the bottom

The entry point for all your code should be in `init()` if you wish to use the `make debug` target we have provided you.

## 5 Handin

Your grade will be based on your design check and your handin. Your handin should be very easy to compile and run. Your code should be easy to read.

Make a `README` file which explains any quirks in your code, as well as any special instructions about running your program. You should also mention any design decisions you made which you found troubling.

Handin with the following command, which you should run from the directory which contains `build.xml` so we'll get everything.

```
/course/cs032/bin/cs032_handin fileview
```