



Simple Shell

Objectives:

1. Familiarity with system calls in Unix environment.
2. Introducing processes and their nesting.

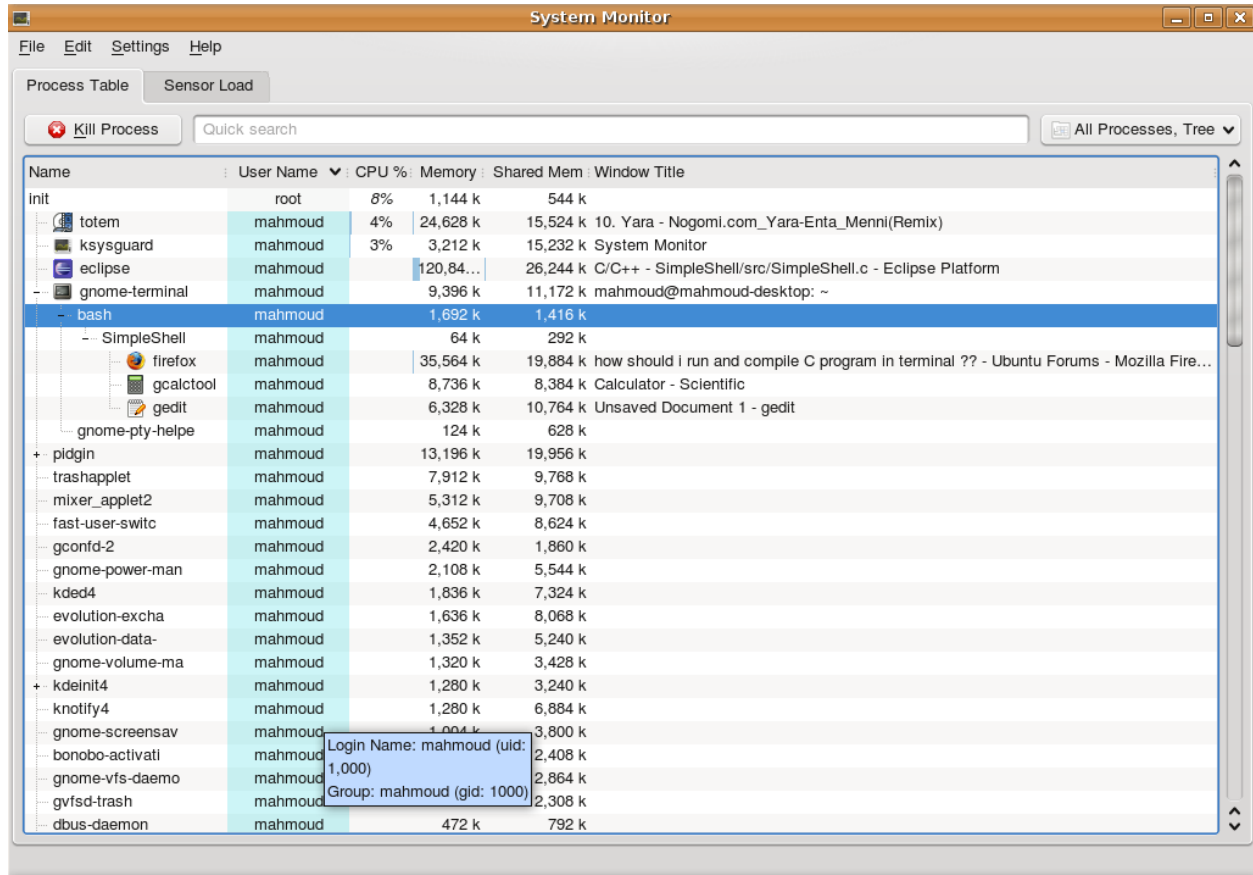
Problem Statement:

It is required to implement a Unix shell program. A shell is simply a program that conveniently allows you to run other programs. Read up on your favorite shell to see what it does.

Your shell must support the following:

1. **The internal shell command "exit" which terminates the shell**
 - **Concepts:** shell commands, exiting the shell.
 - **System calls:** exit()
2. **A command with no arguments**
 - **Example:** ls, cp, rm ...etc
 - **Details:** Your shell must block until the command completes and, if the return code is abnormal, print out a message to that effect.
 - **Concepts:** Forking a child process, waiting for it to complete and synchronous execution.
 - **System calls:** fork(), execvp(), exit(), wait()
3. **A command with arguments**
 - **Example:** ls -l
 - **Details:** Argument 0 is the name of the command.
 - **Concepts:** Command-line parameters.
4. **A command, with or without arguments, executed in the background using &.**
 - **Example:** firefox &
 - **Details:** In this case, your shell must execute the command and return immediately, not blocking until the command finishes.
 - **Concepts:** Background execution, signals, signal handlers, processes and asynchronous execution.
 - **Requirements:** You have to show that the opened process will be nested as a child process to the shell program via opening the task manager found in the operating system like in figure 1. Additionally you have to write in a log file when a child process is terminated (main application will be interrupted by a SIGCHLD signal). So you have to

implement an interrupt handler to handle this interrupt and do the corresponding action to it.



| Name | User Name | CPU % | Memory | Shared Mem | Window Title |
|------------------|-----------|-------|-----------|------------|-----------------------------------------------------------------------------------------|
| init | root | 8% | 1,144 k | 544 k | |
| totem | mahmoud | 4% | 24,628 k | 15,524 k | 10. Yara - Nogomi.com_Yara-Enta_Menni(Remix) |
| ksysguard | mahmoud | 3% | 3,212 k | 15,232 k | System Monitor |
| eclipse | mahmoud | | 120,84... | 26,244 k | C/C++ - SimpleShell/src/SimpleShell.c - Eclipse Platform |
| gnome-terminal | mahmoud | | 9,396 k | 11,172 k | mahmoud@mahmoud-desktop: ~ |
| - bash | mahmoud | | 1,692 k | 1,416 k | |
| - SimpleShell | mahmoud | | 64 k | 292 k | |
| firefox | mahmoud | | 35,564 k | 19,884 k | how should i run and compile C program in terminal ?? - Ubuntu Forums - Mozilla Fire... |
| gcalc | mahmoud | | 8,736 k | 8,384 k | Calculator - Scientific |
| gedit | mahmoud | | 6,328 k | 10,764 k | Unsaved Document 1 - gedit |
| gnome-ptty-helpe | mahmoud | | 124 k | 628 k | |
| pidgin | mahmoud | | 13,196 k | 19,956 k | |
| trashapplet | mahmoud | | 7,912 k | 9,768 k | |
| mixer_applet2 | mahmoud | | 5,312 k | 9,708 k | |
| fast-user-switc | mahmoud | | 4,652 k | 8,624 k | |
| gconfd-2 | mahmoud | | 2,420 k | 1,860 k | |
| gnome-power-man | mahmoud | | 2,108 k | 5,544 k | |
| kded4 | mahmoud | | 1,836 k | 7,324 k | |
| evolution-excha | mahmoud | | 1,636 k | 8,068 k | |
| evolution-data- | mahmoud | | 1,352 k | 5,240 k | |
| gnome-volume-ma | mahmoud | | 1,320 k | 3,428 k | |
| kdeinit4 | mahmoud | | 1,280 k | 3,240 k | |
| knotify4 | mahmoud | | 1,280 k | 6,884 k | |
| gnome-screensav | mahmoud | | 1,004 k | 3,800 k | |
| bonobo-activati | mahmoud | | 1,000 k | 2,408 k | |
| gnome-vfs-daemo | mahmoud | | 1,000 k | 2,864 k | |
| gvfsd-trash | mahmoud | | 1,000 k | 2,308 k | |
| dbus-daemon | mahmoud | | 472 k | 792 k | |

Figure 1 Firefox, Calculator and Gedit are child processes to the SimpleShell process

Description:

To process the user command, do the following:

Step 1: Your command shell should take the user command and its parameter(s), i.e., “ls” and “-l” in this example, and convert them into C strings. (Recall that a C string terminates with a null string, i.e., \0.)

Step 2: The command shell should create a child process via fork ().

Step 3: The child process passes the C strings—the command and parameter(s)—to execvp ().

Step 4: The parent process, i.e., the command shell, should wait, via wait (), for the child process to finish.

Step 5: The command shell gets the next command and repeats the above steps. The command shell terminates itself when the user types exit.

In case a user wants to execute the command in background (i.e. as a background process), he/she writes & at the end of the command. For example, a user command can be:

Shell > firefox &

In this case, your command shell should not wait for the child by skipping the Step 4.

You should keep a log file for your shell program such that whenever a child process terminates, the shell program appends the line “Child process was terminated” to the log file. To do this, you have to write a signal handler that appends the line to the log file when the SIGCHLD signal is received.

Hint:

- When a child dies, the parent process receives SIGCHLD (or SIGCLD) signal.
- To see the set of all signals supported on your system, type, kill -l
- Use a process monitor package to monitor your processes. Provide a screenshot for your shell parent process and some child processes spawned as background processes.
- **Suggested packages:** KSysguard or Gnome-System-Monitor.

Deliverables:

- Complete source code, commented thoroughly and clearly.
- Object Code.
- A report that includes:
 - A description of the overall organization of your code and the major functions.
 - Sample runs.
 - Screenshots for the processes hierarchy in KSysguard (or any similar package) during the execution of your shell program.

Notes:

- Languages used: C/C++.
- Students will work individually.
- No cheating. It will be hardly penalized for both parties.