

GENERAL INSTRUCTIONS

This homework has two parts: one part with a set of theory questions and another part, which is programming. Programming can be done in either C or Java. Submissions should include the **annotated** source code. Programs that do not compile will get a low grade. Make sure your programs do not crash when given bad input, but rather produce a useful warning or error message and take the appropriate action (recover or quit).

SUBMISSION INSTRUCTIONS

The compressed files (either tar or zip) will be submitted using email to the TA. Please make sure that you send me a **single compressed file with all the documents and code. The compressed file should not exceed 1MB.**

THEORY/Written QUESTIONS (30 points)

- [Windows Operating System]** Download and run the *AutoRuns* utility from <http://www.microsoft.com/technet/sysinternals/utilitiesindex.mspx>. Find out what programs are configured to run during system bootup or login. Do you recognize these programs? Identify the programs not provided by Microsoft that automatically start when you bootup or login.
- [Windows Operating System]** Download and run the *Process Explorer* utility from <http://www.microsoft.com/technet/sysinternals/utilitiesindex.mspx>. Find out what processes are currently running on your system. Choose a process, e.g. winword.exe and find out which DLLs it has loaded. Now click on the *System Information* button (Ctrl+I) to see details of CPU activity in your system. Now start a new program, e.g. Microsoft Excel. Describe how CPU usage changes when you start this program, a minute after you have started this program, and when you terminate this program. Click on System Information button (Ctrl+I) to see this information in detail. Provide an explanation for this CPU usage pattern. Click on the *System Information* button (Ctrl+I) to see details of memory usage activity in your system. Provide information about the usage of physical memory, kernel memory and paging in your system. Now start a new program, e.g. Microsoft Excel. Describe how this usage pattern changes when you start this program, a minute after you have started this program, and when you terminate this program. Provide an explanation for this usage pattern.
- [Linux Operating System]** A shell script is a sequence of shell commands written in an executable script file. Executing this file instructs the shell to execute all commands in

the order of their appearance in the script file. There exist several shell scripting tutorials available on the web, e.g. search by entering the keywords *Linux shell script tutorials*. Go through one of these tutorials and then write a shell script that displays various system parameters by using shell commands like *who*, *whoami*, *date*, *hostname*, etc.

4. [Linux Operating System] *ps* is a command that displays information about all processes currently running in your system. Read man page of *ps* command. Enter the following commands: (1) *ps -ef | more* and (2) *ps -aux | more*. Both of these will result in displaying a long list of processes. Identify what processes are started when the system is booted, and what processes are started later on. For each process, find out who owns it, what code it is running, and how much CPU/memory it has used.

Now, store the details of all processes owned by root in a file called *root-processes-1*, and all processes owned by you in a file called *my-processes-1*. Next, restart your system, and create similar files, *root-processes-2* and *my-processes-2*. Compare *root-processes-1* with *root-processes-2*, and *my-processes-1* with *my-processes-2*. Explain the differences between the two.

PROGRAMMING (70 points)

The Shell or Command Line Interpreter is the fundamental User interface to an Operating System. Your first project is to write a simple shell - picoshell – that has the following functionality:

The shell must support the following internal commands:

1. *cd <directory>* - change the current default directory to *<directory>*. If the *<directory>* argument is not present, report the current directory. If the directory does not exist an appropriate error should be reported. This command should also change the PWD environment variable.
2. *clear* - clear the screen.
3. *ls <directory>* - list the contents of directory *<directory>*
4. *set* - list all the environment strings
5. *echo <comment>* - display *<comment>* on the display followed by a new line (multiple spaces/tabs may be reduced to a single space)
6. *help* - display the user manual using the more filter
7. *pause* - pause operation of the shell until 'Enter' is pressed
8. *batch <file>* - execute the commands inside the file and return the results
9. *exec <binary file/image path>* - spawn another process (or thread) to execute that binary on the background and return to the parent process for more commands. After the binary is done, its execution output and its code is printed out in the command line.

10. The shell must support background execution of programs. An ampersand (&) at the end of the command line indicates that the shell should return to the command line prompt immediately after launching that program (similar to 9 but using "&")
11. history- print a listed history for commands with the ability to run another command from the list by selecting it
12. exit - quit the shell

All other command line input is interpreted as program execution done by the shell running the programs as its own child processes (i.e. using fork or clone or thread for java).

Design a simple command line shell that satisfies the above criteria and implement it on the specified UNIX platform using either C or JAVA. You are encouraged to use the online code and resources provided through the class website. **However, you will have to cite ALL your sources to avoid plagiarism!**

Write a simple manual describing how to use the shell. The manual should contain enough detail for a beginner to UNIX to use it. For example, you should explain the concepts of I/O redirection, the program environment, and background program execution. The manual **MUST** be named readme and must be a simple text document capable of being read by a standard Text Editor.

For an example of the sort of depth and type of description required, you should have a look at the on-line manuals for csh and tcsh (man csh, man tcsh). These shells have much more functionality than yours and thus, your manuals don't have to be quite so large.

The submission should contain only source code file(s), include file(s), a Makefile and the readme file (all lower case please). No executable program should be included. Make sure that you read how to create Makefiles for C and JAVA (yes, you can create them for Java as shell scripts).

EXTRA CREDIT (20 points)

The shell must be able to take its command line input from a file. i.e. if the shell is invoked with a command line argument:

```
picoshell shellfile
```

shellfile is assumed to contain a set of command lines for the shell to process. When the end-of-file is reached, the shell should exit. If the shell is invoked without a command line argument it solicits input from the user via a prompt on the display.

The shell must support i/o-redirection on either or both STDIN and STDOUT:

```
program arg1 arg2 <input > output
```

executes the program with arguments `arg1` and `arg2`, the `STDIN` `FILE` stream replaced by input and the `stdout` `FILE` stream replaced by output.

`stdout` redirection should also be possible for the internal commands: `dir`, `environ`, `echo`, & `help`.

With output redirection, if the redirection character is `>` then the outputfile is created if it does not exist and truncated if it does. If the redirection token is `>>` then outputfile is created if it does not exist and appended to if it does.

Grading Points (70 points total)



- Submission of required files only, with name, student number etc on all submitted files (2 marks)
- Warning free compilation and linking of executable with proper name (1 marks)
- Support for both keyboard and batchfile inputs (2 marks)
- Performance of internal commands and aliases (10 marks)
- External command functionality (10 marks)
- Background execution (20 marks)
- Readability, suitability & maintainability of source code and makefile (15 marks)
- User manual (10 marks)
 - Description of operation and commands (2 marks)
 - Description of environment concepts (2 marks)
 - Description of i/o redirection (2 marks)
 - Description of background execution (2 marks)
 - Overall layout and display of understanding (2 marks)

Please make sure that your name is inside the submitted source code and the readme files.