Project Overview

Creating a Networked Application

Description

This semester the class will create a suite of networked applications. This suite will culminate in the creation of a server, and also a client to use the server.

In order to accomplish this task, we will construct these applications one iteration at a time. Each iteration will be an individual effort with the final iteration being a group effort (See "Grading" below).

By the end of the sixth or seventh iteration, you should have a program similiar to a UNIX shell. You can then use this shell as the front-end to your networkable client.

The description for each iteration, along with scoring guidelines will be given to you at the beginning of that iteration.

Also, model iterations will be provided so that anyone who has problems with one iteration will not be penalized for the next iteration. These models will be posted on the fourth day after the due date of the iteration that the model is for. Programs for an iteration cannot be turned in once a model solution is posted.

Specifications

1. Each program MUST be completed in either C or C++. It is advisable that you NOT switch between the two langauges.
2. You will be expected to supply proper comments and headers FOR EACH FILE. Failure to do so will be reflected in the grade awarded. See "Comments" for some guidelines.
3. You will be expected to use defensive programing. Failure to do so will be reflected in the grade awarded. See "Defensive Coding" for some guidelines.
4. It is advised that you use proper object-oriented design (and yes, C can be object-oriented). Failure to do so will interfere will you being able to easily modify your program for future iterations.
5. These programs should be tested on the Dell machines (Linux). If you wish you may code them on the Blades, but verify that the program will work on the Linux machines.

Iterations

```
Iteration I (50 points) -

    * Handling command-line arguments
    * Handling environmental variables correctly

Iteration II (50 points) -

    * Parsing strings into usable tokens
    * Constructing a better prompt through parsing

Iteration III (50 points) -

    * Using the exec*() family to run outside executables.
    * Using dynamic memory to create argument tables.
```

```
Iteration IV (50 points) -

    * Using fork() to create child processes
    * Using GNU make - creating a makefile for your project

Iteration V (50 points) -

    * Using signals to control process activity

Iteration VI (50 points) -

    * Using pipes to create a flow of execution

Iteration VII (50 points) -

    * Using POSIX threads
    * Using thread-level signals

Iteration VIII (Group Project; 100 Points ) -

    * Using sockets to create a server-client application
```

Grading

Each individual project will be worth 50 points. These points are awarded based on the scoring guide which will be supplied to you at the beginning of each iteration.

The group project will be worth 100 points. Each group must present their project outside of class. At that time, each member of the group will complete a review of their fellow group members. Each group member will be awarded their final project grade based on:

- 70% Overall score on the group project
- 30% Average score of peer reviews (including yourself)

Comments and Headers

1. There should a descriptive header for each file.
2. Comments should be gramatically correct with minimal spelling errors.
3. Comments should be written in such a manner that a person can follow the execution of the code without actually reading the code.
4. There is such a thing as supplying too many comments. Do not comment every line of code. Merely, supply comments for code that is hard to follow.

Defensive Coding

1. Check all meaningful return values for error conditions.
2. Do NOT use any unbounded function calls (i.e. strcpy(), strcmp(), sprintf()). Instead use their bounded counterparts (i.e. strncpy(), strncmp(), snprintf()).
3. Do NOT assume valid input unless you have verified it elsewhere.
4. If you use malloc() or "new", verify that you have been given the memory. Also, for every malloc() there should be a corrosponding free(), and for every "new" there should be a "delete."
5. Make libral use of the assert() function or create a function that will assist you in debugging your program.

Submission

1. Copy all code necessary to make your code compile into your directory located under the ~cs284/ directory. Do NOT copy object files or binaries - only source files (and the Makefile)!
   The directory name will look like:
   ~cs284/TEACHER/AFSLOGIN/prog#
   where:
   TEACHER = your teacher's AFS (ercal or matt)
   AFSLOGIN = your AFS username
   # = the iteration we are on (prog1 would be for Iteration I).
2. You will receive your grade through e-mail. If you have a problem with this policy, please notify the instructor.