

A Tiny Unix Shell (tsh) – Part I Revisited

Introduction

The focus of this assignment is on writing quality modular software. Learning the basic syntax of C is easy, but large-scale software is much more. Modularity is the key to managing complexity.

Larger programs are usually broken up into meaningful modules. Each C (source) module is compiled into an object module, and the object modules are linked together to form an executable.

A *module* is a C source file that defines groups of functions and data that belong together. For example, all functions that you wrote in Part I of this project belong together, since they all define primitives for manipulating the job list.

Functions defined in a different module can be accessed by means of *header* files. A header file is an interface to a module that defines what the module does. It contains function declarations (prototypes), #defines, structs, and external variable declarations.

The best way to learn modularity is through programming. In this assignment you will reorganize the first part of the project in a modular fashion. This assignment must be completed *individually*.

What to do

In the first part of this project, you have written a collection of functions for manipulating jobs. Separate the code into three files:

1. A *header* file called `joblib.h` that contains all definitions and function prototypes. Surround the entire file content by

```
#ifndef _JOBLIB
#define _JOBLIB

    ... /* variable declarations and
         function prototypes here */

#endif
```

This is to guarantee that these definitions are not included multiple times into the executable.

2. A main *source* file called `tsh.c` that contains the main function only.
3. A *source* file called `joblib.c` (job library) that contains all functions for manipulating the list of jobs. The file `joblib.c` should also define the global variables related to jobs (in particular, `nextjid` and the array `jobs`).

The first line in each source file should include the header file:

```
#include "joblib.h"
```

Test for compiling errors by compiling each module individually, for example:

```
gcc -c joblib.c
```

The option `-c` tells the compiler to compile only (and not link the objects together). To compile the entire project, you could simply use the command line

```
gcc tsh.c joblib.c -o tsh
```

The compiler will produce two object modules `tsh.o` and `joblib.o`, and link them together into the executable `tsh`. With larger projects that may need to link other libraries at compile time (eg., programs using threads must link the `pthread`s library at compile time), this procedure may become a bit tedious. We will, instead, use a makefile for easy compilation.

To promote the importance of self-learning skills, we will let you learn how to create a makefile on your own. Search the Internet for information on writing a Unix makefile, and write a simple makefile for this project. **Hint:** Type in “simple makefile” (within double-quotes) in your preferred browser and consult a few of the links the browser brings up.

Note: The path for the program `make` on the Unix cluster is

```
/usr/ccs/bin/make
```

Modify the profile for your shell to include the path `/usr/ccs/bin/` in the list of search paths. To do so, edit the file `~/.profile` and add the line

```
PATH=/usr/ccs/bin/:$PATH
```

after the last `PATH` line in the file.

Evaluation

Your score will be computed out of a maximum of 40 points based on the following distribution:

30 Correctness.

10 Comments and style.

Hand In Instructions

Hand in a printed copy of the following:

1. Source code of each of your files (`tsh.c`, `joblib.c`, `joblib.h` and `Makefile`).
2. A sample output of your code.
3. A short README file explaining any bugs and deviations from the assignment specifications.

Make sure you have included your name and your Unix login name in the README file. Leave your source code in your Unix account. Good luck!