CS23710 – Practical Sheet 7

The demonstrators are allowed to give you as much help as you need to complete sections 1 through 4 of this sheet.

However, they are only allowed to help by clarifying the questions in section 5. You must write your answers by yourself.

1 Introduction

This practical sheet contains the first part of Assignment 4. You should spend a total of about 12 hours reading and working to complete this Practical Sheet.

You will need to read a number of man pages, and you may also find it useful to look up the section on the Unix File System from the online notes:

http://www.aber.ac.uk/~dcswww/Dept/Teaching/Courses/CS23710/UNIX/notes/filesystem/filesystem.html

2 The Fantasy System

To get started, download the fantasy tar file from the web, and unpack using `tar xvf fantasy.tar`. Read `man tar` to find out about `tar`. It will build the directory structure illustrated in Figure 1

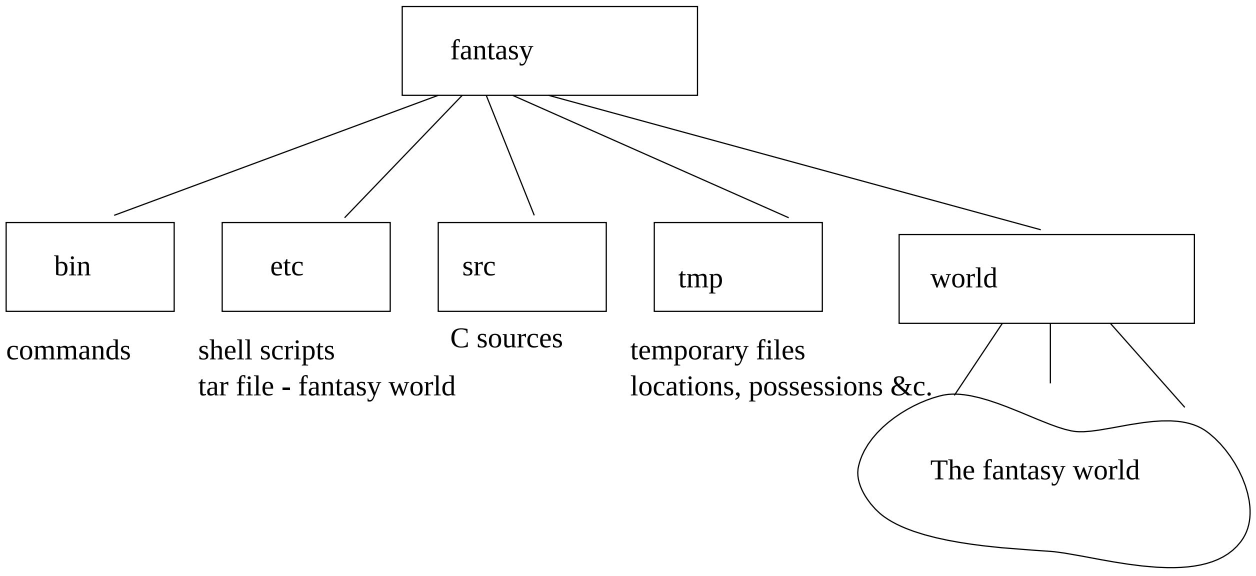

Figure 1: Fantastic Files

The `.../fantasy/world` directory will form the root and also the starting point for your adventures. Instructions for unpacking the world are given below.

The other subdirectories hold commands, shellscripts, C sources and temporary files to store information about the location and possessions of the player and other denizens of the fantasy world. This is illustrated in Figure 1.

Adapt the fantasy_startup and shell script in `.../fantasy/etc` so that they set the environmental variable `$FANTASY` to your `.../fantasy` directory instead of `/dcs/eds/fantasy`. Both bash and cshell versions of the fantasy_startup script are provided.

You may find it convenient to define aliases in your shell initialisation file (.cshrc or .bashrc) to run the fantasy_startup and fantasy_finish scripts. Mine (cshell) are

```
alias fan_start "source /dcs/eds/fantasy/etc/fantasy_startup.cshell_version"
alias fan_finish "source /dcs/eds/fantasy/etc/fantasy_finish.cshell_version"
```

The bash equivalents are

```
alias fan_start="source /dcs/eds/fantasy/etc/fantasy_startup.bash_version"
alias fan_finish="source /dcs/eds/fantasy/etc/fantasy_finish.bash_version"
```

3 The Fantasy World

A little world has been supplied. You can extract this world from the tarfile in `$FANTASY/etc` using the command `tar xvf initial_world.tar`

This world is sufficient to demonstrate all your fantasy commands, but, just for fun, you may wish to extend it, or to replace it completely. You can do this using `mkdir` and `ln -s` to create a directory tree rooted in `$FANTASY/world` with a node for each location in your adventure. Create symbolic links wherever you want to establish a path between two nodes that are not already connected by a directory/subdirectory link in the directory tree.

Every node in your world has a description. This is represented by a text file called `description` which contains text describing the location.

Every node in your world includes a file called `light`, which contains a single 1, to indicate that the location is light, so you can see, or a 0, to indicate that the location is dark, so you cannot see.

Movable objects are represented by files. The name of the file is the name of the object. The file contains a single line of text indicating the kind of object it is – treasure, weapon, denizen or light source – further text terminated by a % sign describing the object, a number indicating its weight in kilograms, and another number whose meaning depends on the kind of object that is represented. This last number represents the value of a treasure, whether a light source is on (1) or off (0), the process identifier of a denizen, or the signal produced by a weapon. (Denizens and weapons will be addressed in the final practical sheet).

When you are satisfied with your fantasy world, change directory to `$FANTASY/etc` and use the command `tar cvf initial_world.tar $FANTASY/world` to back the world up as a tar file in `$FANTASY/etc`. This tar file will be used to initialise the fantasy world after it has been modified during the course of a game[1].

4 Commands

C sources for the fantasy commands `start`, `look` and `quit` have been supplied in `$FANTASY/src`. For the other fantasy commands, you have been supplied with C programs that write a message like "This is the command *command* with arguments *arguments* ... not yet implemented ...".

Your job is to replace these with proper implementations, using the Unix application program interface (API). As far as possible, you should use the Posix interfaces. Later, (Practical Sheet 9) you will also implement an independent dragon for your fantasy world.

You can compile and link your fantasy commands using `make` from the directory `$FANTASY/src`. To see how this works, look up `man make` and look at the file `$FANTASY/src/makefile`

[1]If you move objects around in the world, and then restart the game, you may find several copies of the same object at different locations in the fantasy world. One way to fix this is to modify the 'start' function so that it deletes the old world before extracting the new one. This is risky, so please take care if you decide to do it.

This simple makefile allows you to compile and link each fantasy command using `make` *command.*

If you look at the C sources for the commands that have been written already, you will notice that they #include `location.h` and `movable_object.h`.

`location.h` and location.c provide the functions `get_current_location` and `set_current_location`, which read and set the player's location in the fantasy world. You will need to modify the definition of FANTASY_DIR in `location.h` so that it refers to your fantasy directory instead of `/dcs/eds/fantasy`

`moveable_object.h` and `moveable_object.o` allow you to manipulate objects in your fantasy world.

4.1 Basic Commands

The basic commands to explore in the fantasy world are:

- `start` is supplied; it initialises the fantasy world, sets the player's current location to `.../fantasy/world`, and executes `look`;
- `quit` is also supplied; it closes down the process representing a dragon, writes a message indicating that the game is over and recommends that the user restore the environment variable PATH to its original value;
- `look` is supplied; with no arguments, `look` displays the description of the current location, lists any movable objects, and also lists all exits; if *thing* is a movable object, `look` *thing* displays its description, weight and value, if *thing* is a direction `look` *thing* says so; this is also supplied;
- `go` *path* is not supplied; `go` makes *path* the current location, and executes `look` by calling the `system()` function; you are expected to implement this command, with help from the demonstrators if you need it.

Your implementation will consist of a C program that makes system calls Although a shellscript could provide the required functionality, your current aim is to learn how to write C programs that make system calls.

Therefore, in this and in later worksheets, shellscripts will not be acceptable as implementations of the fantasy commands. Similarly, programs which depend heavily on particular shells will result in reduced marks.

Your implementations should include error checking and should handle inappropriate arguments gracefully. You may need to extend `fan_err.h` to achieve this. Try to make sure that your 'go' command doesn't allow you to move out of the fantasy world. (You'll need to check for things like `..` in the `path` argument of 'go').

You may find the library function `realpath` and the system calls `chdir` and `stat` useful.

Use `man` to look up the system calls that appear in the sample programs. Ask the demonstrators for help if you see something you don't understand in the man pages.

5 Assessed work associated with Practical Sheet 7

The demonstrators are only allowed to clarify the questions in this section; they are not allowed to help you with your answers.

1. (a) Briefly describe the Unix section 2 function 'write'.

(b) Explain how ‘write’ is used in the ‘showfile’ function which is part of the `look` command in the fantasy world.

(c) Indicate how ‘write’ differs from the section 3S function ‘fprintf’, and assess the relative merits of each of these functions.

2. (a) Briefly describe the section 3S function ‘system’.

   (b) Explain how ‘system’ is used in the fantasy command `start`.

   (c) How might excessive use of ‘system’ compromise portability?

3. List the section 2 functions that appear in your implementation of ‘go’, and briefly describe the role of each of these in your program.

Your answers to the above questions will be required as part of your submission for Assignment 4. **15/50 marks**