CSI 402 – Systems Programming – Spring 2012

Programming Assignment V

Date given: Apr. 17, 2012 **Due date:** May 6, 2012
Weightage: 10%

The deadline for this assignment is **11 PM, Sunday, May 6, 2012**. There is no two-day grace period for this assignment. Thus, the assignment *won't* be accepted after 11 PM, Sunday, May 6, 2012.

Very important: There are two parts in this assignment. For Part (a), the C source program may be in a single file. For Part (b), the source program must be split into two or more C files in a meaningful fashion. Also, the source files for both the parts must be submitted together using the `turnin-csi402` command. Note, in particular, that your submission must have only ONE `makefile` that can create executables for both the parts. Additional specifications regarding the `makefile` will be included in the `README` file for this assignment.

The total grade for the assignment is 100 points, with 35 points for Part (a) and 65 points for Part (b).

Description of Part (a):

In Unix, file names starting with the character `'.'` are referred to as **hidden** files since the `ls` command normally does not show such files. For example, the home directory of each user may contain hidden files such as `".bash_profile"`, `".emacs"`, etc. Also, the convention in Unix is to use the file name `"."` to refer to the current directory and the file name `".."` to refer to the parent of the current directory.

The executable version of your program for Part (a) must be named `p5a`. It will be executed by a command line of the following form:

`p5a` *pathname*

Here, *pathname* gives the full path name of a directory. Your program should go through the files in the specified directory. For each hidden file in the directory, except `"."` and `".."`, your program must print to `stdout`, the file name, the (logical) size of the file in bytes and the date (month, day and year) of last modification of the file.

Your program for Part (a) should detect the following errors.

(1) The number of command line arguments is not equal to two. This is a fatal error. In this case, your program should produce a suitable error message to `stderr` and stop.

(2) The directory specified on the command line cannot be opened. This is also a fatal error. Thus, your program should produce a suitable error message to `stderr` and stop.

(3) The program is unable to obtain information about a specific file using the `stat` function. This is *not* a fatal error. In this case, your program should print a suitable message to `stdout`

giving the name of the file for which the call to `stat` failed. However, the program should *not* stop; instead, it should continue to examine the other files in the specified directory.

In writing the program for Part (a), the following material from the text by Haviland, Gray and Salama will be helpful.

(i) Pages 53 through 57 of Section 3.3 (**Obtaining file information: `stat` and `fstat`**): In particular, you must understand the specifications of the library function `stat` and the components of the `stat` structure (pages 53–54).

(ii) Pages 67 through 71 of Section 4.4 (**Programming with directories**): In particular, you must understand the specifications of the functions such as `opendir`, `closedir`, `chdir` and `readdir`. The two program examples given on pages 70 and 71 will also be helpful.

(iii) Page 318 of Section 12.4 (**Time**): This section discusses how the time values stored in the `stat` structure can be converted into conventional date/time specifications.

Description of Part (b):

You are required to write a rudimentary client-server program using **named pipes** (also called **FIFOs**) as the method for inter-process communication. You should get started on this assignment by studying Chapters 5 and 7 of the text by Haviland et al. Chapter 5 presents information about creating multiple processes using `fork()` and several variants of `exec` system calls. Chapter 7 discusses the use of FIFOs for inter-process communication. Although these chapters will be discussed in class, you are advised to get ahead and study them on your own.

The system consists of one server process and one client process. The client process must be started by the server using an explicit call to `fork()` followed by an appropriate form of `exec`. The server maintains a database of student and course information. The client sends a sequence of commands to the server, one command at a time. The server executes each command and sends back a response to the client. (The client sends a command only after receiving the response to the previous command.) The client stores the responses received from the server in a log file. The only exception to this is the `exit` command. By sending this command, the client informs the server that there are no more commands and that the client will exit right after sending the command. So, the server does not send a response to the `exit` command. The actions to be performed by the server for the `exit` command are indicated later in this handout.

Command line specification: The executable versions of your server and client programs must be named `p5b_server` and `p5b_client` respectively. The server will be executed using the following shell command

`p5b_server` *initdbfile* *finaldbfile* *cmdfile* *logfile*

where the command line arguments have the following interpretation. (Bear in mind that the client is started by the server.)

- The argument *initdbfile* gives the name of the initial database file for the server.
- The argument *finaldbfile* gives the name of the file that will contain final database after the server has executed all the commands sent by the client.
- The argument *cmdfile* gives the name of the file containing the commands. The client will read this file and send commands (one at a time) to the server.
- The argument *logfile* gives the name of the file that stores the response sent by the server for each command.

All of the above are text files. Their formats are discussed below.

Initial and final database files: Each of these files contains information about students and courses in the following format. The first line gives the number (say, N) of students in the database. This is followed by N lines, where each line gives information about one student. Each such line contains the following fields in the specified order: name of the student (a string of length at most 30 with no spaces), the number of courses (say, m) in which the student is enrolled and a sequence of m course ids (where each course id is a nonnegative integer). The fields are separated by one or more spaces. Following the N lines of student information is a line containing an integer (say, C) that gives the number of courses for which information is stored in the database. That line is followed by C lines, where each line gives information about one course. Each such line contains the following fields in the specified order: course id (a nonnegative integer), number of credits (a positive integer) and the course schedule (a string of length at most 20 characters with no spaces). The fields on each such line are also separated by one or more spaces. *You may assume that there are no errors in the initial database.*

Command file: Each line of the command file consists of a command name (a string of length 4) followed by one or more values needed by the command. On each command line, the fields are separated by one or more spaces. If a command is successfully executed, the server returns the value 1 to the client; otherwise, the server returns the value 0 to the client. In addition to this success/failure indication, some of the commands need another return value (e.g. the `tcre` command discussed below). The possible commands and their meanings are as follows.

(a) Command `addc`: This command is used to add a new course to a student's course list or to add a new student to the database. The parameters specified in this command are the student's name and a course id. The server must verify that information about the course is available in the current database, and if so, add the specified course to the list of courses for which the student is registered. If the student's name does not appear in the database, the server should treat the command as specifying registration information for a new student and add the information to the database.

(b) Command `drpc`: This command is used to drop a course from a student's list of courses. The parameters of this command are the student's name and a course id. The server must verify that the student is registered for the course. If so, the server must drop the specified course from the list of courses for which the student is registered.

(c) Command `wdrw`: This command is used to withdraw a student from all the courses for which the student is registered. This command specifies only one parameter, namely the student's name. The server should verify that information about the student is available in the current database. If so, the command asks the server to withdraw the specified student from all the courses for which the student is registered. (However, the student's name is not removed from the database.)

(d) Command `tcre`: The purpose of this command is to obtain the total number of credits for which the student is registered. This command also specifies only one parameter, that is, the student's name. The server should verify that information about the student is available in the current database. If so, the server must return the total number of credits for which the student is registered. If the student's name does not appear in the database, the server should return the value -1 as the total number of credits.

(e) Command `newc`: The purpose of this command is to add information about a new course to the database. This command specifies three values in the following order: a course id, the number of credits and the schedule. The server must first verify that the number of credits is positive (i.e., strictly greater than zero) and that the current database does not contain information about the specified course. If these conditions are satisfied, the server must add the specified information about the course to the database.

(f) Command `csch`: The purpose of this command is to change the schedule information for an existing course. This command specifies a course id and a schedule. The server must first verify that the current database contains information about the course. If so, the server must modify the schedule for the specified course using the information included in the command.

(g) Command `ccre`: The purpose of this command is to change the number of credits for an existing course. This command specifies a course id and the number of credits. The server must first verify that the current database contains information about the course and that the number of credits specified in the command is positive. If so, the server must modify the number of credits for the specified course using the information included in the command.

(h) Command `gsch`: The purpose of this command is to get the schedule information for an existing course. This command specifies a course id. The server must verify that the current database contains information about the course. If so, the server must return the schedule for the specified course. If the course doesn't exist, the server must return the string `"Error"` as the schedule.

(i) Command `gcre`: The purpose of this command is to get the number of credits for an existing course. This command also specifies only a course id. The server must first verify that the current database contains information about the course. If so, the server must return the number of credits for the specified course. If the course doesn't exist, the server must return the value `-1` as the number of credits.

(j) Command `exit`: This command is *not* in the command file. The client sends this command to the server to indicate that all the commands from the command file have been executed and that the client will exit. The server does not send a response to this command. Instead, the

server should save the current database into the file specified by the command line argument *finaldbfile* and exit. (If the file already exists, it should be overwritten.)

Log file: Each line of the log file contains the following fields separated by one or more spaces: the sequence number of a command, the command name, whether the command was unsuccessful or successful (indicated by integer values 0 and 1 respectively), and when the command requires a return value (i.e., for the commands `tcre`, `gsch` and `gcre`), the integer or string value returned by the server. Note that the sequence number is not specified in the command file. The client must keep track of the sequence number as it reads and sends commands to the server. The sequence number starts at 0 and is incremented by 1 for each command.

Outlines for the server and client programs are provided on page 6 of this handout. You may assume the following.

1. There will be information about at most 100 students and at most 100 courses in the database at any time. Each student will be registered for at most 10 courses.
2. There are no errors in the initial database file.
3. Each line in the command file will contain one command. Each command will have all the information needed for the command and will not contain any extraneous information. The fields for each command will be separated by one or more spaces.

Errors to be detected: Your program for Part (b) must detect and report the following errors.

(i) The number of command line arguments is incorrect.

(ii) The initial database or the final database or the command file or the log file cannot be opened.

(iii) The specified command is not one of `addc`, `drpc`, `wdrw`, `tcre`, `newc`, `csch`, `ccre`, `gcre` and `gsch`. (The command names are *case sensitive*; thus, `ADDC` is not a valid command.)

In the case of fatal errors (i) and (ii) above, your server program should print a suitable error message to `stderr` and stop. (In these cases, the server should *not* start the client process.) In the case of (iii), which is *not* a fatal error, the client should send the command to the server and let the server produce a response indicating an error. This response must be written to the log file. Assume that erroneous commands don't have any return value (other than the failure indicator).

Information about `README` file: The `README` file for this assignment will will be available by 10 PM on Tuesday, April 24, 2012. The name of the file will be `prog5.README` and it will be in the directory `~csi402/public/prog5` on `itsunix.albany.edu`.

(over)

Part (b) of Program V – Outlines for Server and Client

Outline for server:

```
1.  Read initial database into a suitable data structure (in memory).

2.  Create and appropriately open the command and reply FIFOs.

3.  Fork the client process.

4.  do {
      (a)  Receive command from client.
      (b)  if (command != exit) {
             (i)   Execute command. (This may modify the data structure
                   in memory.)
             (ii)  Send reply to client.
           }
    } while (command != exit)

5.  Write modified database to the specified file and stop.
```

Outline for client:

```
1.  Open the command file and the log file.

2.  Open the command and reply FIFOs appropriately.

3.  while (there are commands in the command file) {
      (a) Get the next command.
      (b) Send the command to the server.
      (c) Receive reply from server.
      (d) Write an appropriate entry into the log file.
    }

4.  Send exit command to server and stop.
```