CS 5352
Project 2
Due – Oct. 31, 2014 at 11:59 pm

Problem: *find* is a powerful UNIX utility. Besides locating files in the given source directory and its descendent directories, it can execute shell commands on the located files. See its man page and try the following commands to gain some experience.

```
find /dev –name null –type l            //finds a link file named ‘null’ in /dev
find ~cs5352 –mtime +2 –mtime -5 –ls         //lists files modified 2 or more days ago but no before 5 days
find $HOME –name core –ok mv {} /tmp \;
```

(The last command is cryptic and requires some explanation. 'find’ would search for files named ‘core’ in the HOME and its descendent directories and execute the command ‘mv’ on all such files to move them to /tmp. ‘ok’ means that shell would prompt the user for a response before executing the given command on each file. {} (no spaces within parentheses) denotes the collection of located files. ‘\;’ is needed with ‘ok’; its purpose is not fully clear to me.)

<u>The purpose of this project is to study UNIX file system by developing a similar but much simpler utility</u>. In brief, the summary syntax of our utility is as follows.

Find source selection cmd

Like ‘find’, *Find* should traverse the <u>source</u> directory recursively, examine each entry of every descendent directory to verify if it meets the given <u>selection</u> criterion, and finally arrange for the execution of the command <u>cmd</u> on the selected entries. The source argument can be any pathname such as *dir, ./dir1, ~/dir1, $HOME/dir1,* etc. Unlike, ‘find’, *Find* uses just one selection criterion and executes a single command. The selection criterion has two parts; the first part is a file attribute, which is *name, mtime,* or *user,* and the second part is a single corresponding argument. cmd is any single shell command compatible with the selection criterion, and it is discussed later. The following examples should demonstrate use of *Find*.

Find /tmp –user lakhani –ls //find all files of user ‘lakhani’ in /tmp folder or its descendents and apply “ls” to list them.

Find . –mtime 5 –ls //find regular files in the current directory (.) and its descendents that were modified in the last 5 days and list them.

Find DIR –name core –mv -i /tmp //find files/directories named ‘core’ in DIR (and descendents) and move them to /tmp interactively.

The source is a pathname and it may contain shell meta-characters such as $ or ~ (which you do not have to process, because shell would resolve all meta-characters, if any, before executing Find). You may assume that there is no syntax error in the ‘Find’ command line, i.e., all its components are given in the right order and each command is complete.

Create a new folder, “proj2” in your ‘cs5352’ folder for this project and implement each stage in a separate file. The purpose of each stage is summarized below.

Stage1: Familiarize with file attributes and relevant system calls.
Stage2: Familiarize with common system calls for UNIX directories. It also develops a traverse procedure to list entries of the source. For simplicity, ‘Find’ is called with ‘source’ alone; its descendents are not treated (ignored) in this stage.
Stage3: Expand stage2 to traverse ‘source’ directory <u>recursively</u>; it lists entries of all descendant sub-directories.
Stage4: Implement the ‘selection’ portion of Find; the syntax of Find in this stage is as follows:
Find source selection argument

Stage5: Apply also the given command to the selected files.

Stage1: Read the man page of system calls “open”, and “fstat”. Note down the header files. Visit /usr/include/sys/stat.h and make a note of commonly used file attributes in UNIX; See lines 75-90 and lines 413 and up of stat.h.

Solaris provides three system calls for accessing file attributes: stat, fstat, and lstat. stat ( ) is applied to a file without opening it, i.e., you can read the file attributes even if you do not have permission to open it. To use fstat( ), the file must be opened first and its file descriptor should be used as the first argument. lstat ( ) is used to differentiate between link and its original file.

Copy “~cs5352/tutorials/test_stat.c” to your “proj2”. It demonstrates use of above system calls. Compile and run it by passing name of a regular file as the argument. Next edit this file and print the file size; it should help you to familiarize with attribute retrieval. Also note the use of perror( ); it shows how you can print actual error messages; see its man page.

Testing: Use test_stat.c to determine mtime of test_stat.c. Note that mtime is stored in terms of the number of seconds since the epoch day of January 1, 1970. (Use “%ld” if you need to print unsigned long decimal and “%lld” for unsigned long long decimal).

Stage2: You should first read the man page of ‘opendir’, and ‘readdir’. Visit /usr/include/sys/dirent.h and note down definition of the directory structure.

Develop stage2.c. Its main() function should first test the file type of the source argument. If it is a directory, it calls a procedure, say visitDir(source). Use the code given in the man page of ‘readdir’ as the basis of this procedure. visitDir() fetches each entry of ‘source’ directory, prints a line giving the entry name and the time of last modification. Use stat() to determine the modification time. If stat( ) fails for an entry, do not exit the procedure but loop back to process the remaining entries of the directory (use ‘continue’ statement). visitDir() should close the directory before returning control to the main(). In this stage, you may assume that “source” is a directory without any sub-directory.

Let us quickly review the permission bits of a directory by considering an example. opendir (“/X/Y/Z”) would fail if the caller does not have the search permission (i.e., the execution bit is not set) for any component of the pathname such as ‘X’ or ‘Y’. It would succeed only, if the search permission is granted for both ‘X’ and ‘Y’ and read permission for ‘Z’; the caller needs only the read permission for ‘Z’. In case of a failure during opening a directory or processing an entry, visitDir() should print a line of error message using perror( ) and return, but it should not call exit() to quit.

Note that the file type should be determined by testing its ‘ftype’ attribute, i. e., you should not run opendir (“X”) to test if ‘X” is a directory, because it can fail even if ‘X” is a directory.

Testing: ‘Find ~cs4352/tutorials/sched’.

(Copy file “monitor” from ~cs5352/projects/proj2 to your proj2 folder and leave it untouched as soon as possible.)

Stage3: Copy stage2.c to stage3.c. Expand visitDir(arg) so that it would visit the arg directory recursively in the depth-first order as shown by the pseudo-code below. You may assume that there are only three types of files: directory, regular file, and everything else, denoted as ‘other’. Let ‘Dir’, ‘Reg’ and ‘Oth’ denote their acronyms. visitDir() should print one line for each entry of ‘source’ and its descendant sub-directories by giving the acronym and the entry name. If the entry is a non-directory, no processing is needed. If the entry is a directory, it should call itself to visit that sub-directory.

```
vistDir(path)
```

```
{
Dir = opendir(path)
While ((entry = readdir(Dir)) not NULL)
{
  If entry → name is "." Or ".." continue;
  Full_path = malloc (strlen(path) + strlen(entry → name) + 2)
 Strcat path, "/", and entry → name in full_path.
  If (lstat( ) on full_path < 0)
      {perror(); free (full_path); continue;}.
  Get ftype
  If ftype is directory {print; visitDir(full_path);}
  Else ftype is regular file, print;
  Else print – other type
Free (Full_path)
}
Closedir(Dir)
}
```

Note that "." and ".." are always the first two entries in any directory. Note that visitDir() should skip only "." And ".." entries, but not those that begin with a dot such as ".XYZ". The output of stage 3 should contain one-line for the directory and then one line for each of the entries in that directory.

Testing: "~cs5352/projects/proj2" contains two test directories, Test1 and Test2. Run the sample binary program '~cs5352/projects/proj2/stage3_Find', also stored there, on 'Test1' as the source. Format output of your stage3.c accordingly. Then copy onlyTest1.tar to your proj2 directory and untar it (tar –xf Test1.tar). Your program should be able to visit all entries of Test1.

Next run the same sample program on Test2 using the command '~cs5352/projects/proj2/stage3_Find ~cs5352/projects/proj2/Test2'. Watch for the error messages. Your stage3 should produce similar error messages.

Stage4: Copy stage3.c to stage4.c. The purpose of stage4.c is to implement the 'selection' portion of the Find command given as follows;

Find source selection arg

There are two tasks to perform: identify the selection argument and apply it to entries of source. For the first task, define a global table, an array of strings, say, selTable[ ], and initialize it with three strings "-name", "-mtime", "-user", and NULL. Develop a procedure, say ProcArg( ), to process the command line. It should extract the command line arguments and print a line such as "source=…, selection=…, arg=…". Next it should use a "for-loop" to compare 'selection' with the selTable entries and save the selection attribute and the accompanied arg as two global variables, i..e, these are not passed as the arguments to vistDir( ).

We next consider arg argument by considering the three permissible selections, "-name F", "-user N", and "-mtime M". "-name F" is meant to find <u>file/directory</u> named F and print its pathname with respect to the given source directory (Follow the output given in testing section). F can be of any file type. Since it is hard for a user to remember upper/lower case letters in file names, visitDir() should use strcasecmp() instead of strcmp().

'-mtime M' is meant to detect <u>regular</u> files modified within the last M days. Since UNIX stores the modification time in terms of seconds since the epoch time (Jan. 1, 1970), M days should be first converted into number of seconds.

"-user N" is meant to find the files/directories owned by the user N; obviously N is the login name of the user. The problem is that UNIX stores user-id of the owner as file attribute and not the login-name. For this purpose, you would need to access the passwd entry of N, obtain the uid, and then use it. See the man page of the system call, getpwnam() and /usr/include/pwd.h. (This conversion may appear to be complex but it can be coded in just two lines of C.)

For the second task, revise visitDir(). It should print only those entries, which meet the selection criterion. Print complete pathname and the file type of the selected entries (complete pathname is necessary, because a file with the same name may exist in multiple directories).

Testing:

```
Find $HOME/cs5352 –name proj2
Output: Dir       /home/lakhani/cs5352/proj2

Find ~cs5352/projects/proj2/proj2/Test1 –name  makeFile
Output: Reg       /home/courses1/cs5352/projects/proj2/proj2/Test1/Makefile
        Reg       /home/coursses1/cs5352/projects/proj2/proj2/Test1/lib/Makefile

Find $HOME/cs5352 –mtime 2
Output (nothing – no file modified)

Find ~cs5352/students/lakhani –user lakhani
Output: Reg       /home/courses1/cs4552/students/lakhani/proj2/Test1

Find ~cs5352/students/lakhani –user cs5352
```

Stage5: Copy stage4.c to stage5.c. Expand ProcArg( ) to process the rest of the command line of 'Find'. This part of the command line consists of a command name, possible options, and arguments and its syntax is as follows: 'Find source selection arg cmd-name cmd-options cmd_arg'. For example,

```
Find .  –mtime 5 –mv –i  /tmp
```

Cmd-name is any regular file processing command such as cat or cp. To implement this part, declare a global string array, say cmd_list [ ], allocate storage, and save the command name in cmd_list [0], options in cmd_list[1],…, cmd_list[k-1], and arguments of the command in cmd_list[k+1], ….
Declare char **cmd_list; and cmd_list = (char **) malloc (sizeof(char *)*N); Note that exact value of N is known to ProcArg(). To illustrate use of cmd_list, consider the above Find command. For this example, copy argv[4]+1 to cmd_list[0], argv[5] to cmd_list[1], argv[6] to cmd_list[3]. Notice that cmd_list[2] is left unassigned and it is filled later by visitDir(). Note that command options begin with '-', but the cmd-arg do not begin with '-', and therefore, your program should test the first character of each argument after fetching argv[4] until it detects one without '-'. For simplicity, you may assume that the command line is of the form: cmd option <source> <dest>, where <dest> is optional. Example: -cat –n; -rm –i; -cp –i /tmp/gopal; -ls –l ; <source> is the entry captured by the visitDir(). Command list ends with null.

After visitDir() has determined that an entry, say myData, meets the selection criterion, it would save its pathname in cmd_list[2], because k=2 for this example. The cmd_list[ ] would then contain the complete command; for this example, "mv –i X/Y/myData /tmp", where X/Y is the directory containing myData.

visitDir() should arrange execution of the command for the selected entries. There are two ways to implement this part:

(1) Batch execution: main ( ) creates a shell script file, visitDir() dumps cmd_list[] for the selected file, when visitDir( ) returns, main( ) closes this file and calls system( ) to execute the shell command; "sh –f File". visitDir() should prepare cmd_list[] a test buffer using sprintf ( ) and dump the buffer in this file.

(2) Dynamic execution: visitDir( ) prepares cmd_list[ ] and calls a procedure, say cmdExec( ). It runs fork()/execv( ) and executes the command the way you did in the At project. Since commands should be executed in the search order, visitDir( ) should wait for the child to finish execv() before proceeding to search the next entry. If you decide to use this approach, it is best that main ( ) computes pathfind ( ) to locate the path of the executable file of the command and saves it as another global variable and that visitDir( ) uses it in execv( ). Follow test_environ.c in ~cs5352/tutorials for code to use fork/execv.

As before, visitdir() should output one line for each selected entry. Verify that commands are performed correctly for selected entries.

Testing:
Find $HOME/cs5352 –name proj2 –ls -l
Find ~cs5352/projects/proj2/proj2Test1 –name   README –cp –i /tmp
Find $HOME/cs5352 –mtime 2 –ls -i
Find ~cs5352/students/lakhani –user lakhani –cat –n
Find ~cs5352/students/lakhani –user cs5352 –ls -i

Final Testing: As stated before, two test directories are stored in the folder "~cs4552/projects/proj2" for you to test your project. Copy Test1.tar to your 'proj2' folder, untar it and use as the source. Note that you may have to untar Test1.tar every time before you use it, because some of files may be moved out of directory if 'mv' is the command. Do not copy Test2 and use "~cs5352/projects/proj2/Test2" as the source. You have access to all entries of Test1, but not to all entries of Test2. If visitDir() fails to fetch attributes of an entry of Test2 because of permission problems, it should run perror(), print an appropriate error message, skip that entry and move to the next.

Remove all except stage3.c, stage4.c and stage5.c and their executables and my monitor file before submitting proj2 for grading.

Grading: Stage3: 40 points; stage4 and stage5 30 points each. Follow proj2.grad file for grading details.