South China University of Technology

《Operating System》Experiment Report

Experiment Title： Session 3: Implementation of “find” command

Name： 谭演锋 Student ID： 202130100456

Class： 计算机全英联合班 Group：

Collaborator：

Teacher： 钟竞辉

|  |
| --- |
| **Description** |
| 【Objective and Requirement】  **Objective:** Implement a “myfind” command using system calls.  **Requirement:**  The “myfind” command starts from the specified directory and recursively looks up the specified file. The command format is as follow:  *myfind PATH -option parameters*   * PATH: The directory for looking up. * -option parameters:   + -name “file”: Specify the name of the file to be found. Implement supporting fuzzy search using wildcard characters like asterisk, question mark and so on.   + -mtime n: Search by time, search for files modified n days before today.   Finally, output the search results to standard output.  【Environment】  Operating System：CentOS7 |
| **Content** |
| 【Procedure】  In order to search file the codes recursively search for files and directories in a given path based on specified criteria such as file name and modification time. The program accepts command-line options to filter the search.  **“findFiles”** Function:  The function is responsible for recursively searching for files in the specified directory.  First it opens the directory using “**opendir**” and .If opendir fails (returns nullptr), it prints an error message using perror and exits the function.  Then declare a structure **dirent** to hold information about a directory entry. The dirent structure is a data structure used in C programming to represent a directory entry. It is commonly used in conjunction with functions provided by the <dirent.h> header, which are used for directory manipulation in Unix-like operating systems.    Next, we create a loop to read its entries using “**readdir**” for all the file in the directory one by one.  In the loop:  Firstly it skips entries for the current directory (".") and the parent directory ("..").  Then it use funcion **snprintf(full\_path, sizeof(full\_path), "%s/%s", path, entry->d\_name)** t. The snprintf function can safely concatenate the path of the current directory o construct the full path of the file. (path) and the name of the directory entry (entry->d\_name) into the full\_path array.  Next we use the **stat** function to obtain information about a file, given its path. **stat(full\_path, &file\_stat)** is used toretrieve information about the file specified by the full\_path. The function returns 0 on success and -1 on failure. The result is checked, and if it's -1, an error occurred. If something wrong close the directory    Next we check the **name** option, the line **if (name && fnmatch(name, entry->d\_name, FNM\_PERIOD) != 0)** is used to check two conditions:  **name**: The condition checks if name is not nullptr (i.e., it has been provided).  **fnmatch(name, entry->d\_name, FNM\_PERIOD)** : This uses the fnmatch function to perform filename matching. It compares the filename pattern (name) against the actual filename (entry->d\_name) in the directory entry. FNM\_PERIOD is a flag specifies that a leading period in string must be matched explicitly by a period in pattern. If this flag is not set, a period in pattern will match any character in string. If find the file, it will return 0.  Next we check the **mtime** option. The line if (mtime > 0) means whether the **mtime** is specified. Then we use **time(nullptr)** to get current time.Then we use function  **file\_stat.st\_mtime** retrieves the modification time of the file. Finally calculate the time\_difference by divide (60 \* 60 \* 24) a day’ s seconds to get the day difference and judge whether it is match the mtime option,  Finally, we Print the matching file path.  Additionally, we recursively search directories by use function **S\_ISDIR(file\_stat.st\_mode**) to judge whether the file is a directory.    Fig.1 Detail implementation of find file  In the main function, we use the int argc, char\* argv[] to get argument. if (argc < 2)Checks if the number of command-line arguments is less than 2 (meaning only the program name is provided), which means the parameter is not enough.  Then we retrieves the path from the command-line arguments by argv[1].  We set name =nullptr and mtime = -1 to indicate the without these parameters.  **if (strcmp(argv[i], "-name") == 0 && i + 1 < argc)**: Checks if the current argument is -name and if there is a subsequent argument available(specific name of the file). If true, sets the name variable to the subsequent argument and skips processing the next argument by  incrementing i.  **if (strcmp(argv[i], "-mtime") == 0 && i + 1 < argc)**: Checks if the current argument is -mtime and if there is a subsequent argument available(detail time). If true, converts the subsequent argument to an integer using atoi and sets the mtime variable, then skips processing the next argument.  Finally, call the function.    Fig.2 Implement detail of the *main* function  The command format we implement in this session is:  *myfind* [*PATH*][*-name“file\_name”*] [*-mtime last modified n\_days\_before\_today*]  The **name** and **mtime** parameters are all option. If only path is specified, the program will list files in its working directory. It will also search the directory inside the path, here is the example of **tww**.    Fig.4 Running result of ./myfind /root/os/lab1command  If **name** is specified, we can get the file path.    Fig.4 Running result of ./myfind /root/os/lab1 -name "t\_lab3.txt" command  The myfind command can also support the **fuzzy search**, for example using the “?.txt” to find all the .txt file with 1 character, and “\*.txt” to find all the .txt file in the path.    Fig.4 Running result of fuzzy search command  If only **mtime** is specified, we can get all the files modified before n days ago in the path    Fig.4 Running result ./myfind /root/os/lab1 -mtime 7 command  If both **mtime** and **name** is specified, we can get all the files with specified name and modified before n days ago in the path. Following is the example command, where the 1.txt is modified 4 days ago.    Fig.4 Running result of both mtime and name are specified |
| **Conclusion** |
| In the experiment, I first met difficult in using main function to get parameter as a command system calls. Then I also met difficult in how to sloven the problems of options parameter and the needed parameters. I also find problem in getting the information of data and judge whether it is the directory. I also do not know how to get and calculate modify days.  From the experiment “Implement a “myfind "command using system calls”on a Linux environment. I have a deeply understanding of how to manipulate file in the OS. I learn using the function **opendir** and **readdir** to access the dir and get the file data. Then I learn using the struct **dirent** to store and access the file information like the name. I also learn to use the struct **stat** to get the file status such as the st\_mtime for last modified time, st\_mode for the type of the file. It give me a deeply understanding of how the OS and the File work. From the experiment, I also learn using the function like **fnmatch** for file name matching, strcmp for string comparing and atoi for translating the char into the int, which helps me a lot on coding technique. Finally, I first learn how to construct the **command line tool** by the C++, using the main function with parameter **argc** for number of argument and **argv**[] for details data. It is the first time for me learn the usage of these parameters.  The experiment most impressed me that this experiment meaningfully enhances my understanding of the principle and usage of system calls and recursive algorithm in solving file-system management problems in the computer operating system. I also learn the relative data structure and functions to operate the file. |
| **Teacher’s Comments and Score** |
| Comment：  Score：           Signature：                                                 Date： |