South China University of Technology

《Operating System》Experiment Report

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

Name： 向天翼 Student ID： 201836020389

Class： 18计算机联合班 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：Ubuntu 18.04.4 LTS |
| **Content** |
| 【Procedure】  The command format we implement in this session is:  *myfind* [*-p PATH*][*-n “file”*] [*-m n\_days\_before\_today*]  The three parameters are all option. If the path is not specified, the program will list files in its working directory. If filename is not specified, the program will list all the files without consider the name of files. If modified time is not specified, the program will list all the files without consider the modified time of files.  To parsing the option parameters, we use ‘getopt\_long’ function. This function will analysis the input parameters *argv* and *argc* of main function. If a pre-setting parameters identifier has been detected in *argv*, we can use *switch* structure to do the corresponding task. For example, if the parameter ‘-p’ is detected, the program will copy the string following by ‘-p’ to a global variable ‘root\_path’ as the target path.    Fig.1 Parsing option parameters  The core function of ‘myfind’ command is implement by the *find* function, which is showed in Figure 2. This function will first try to open the given path using *opendir* function. If the path does not exist, the program will output an error message. Otherwise, the program will traversal the filesystem recursively. In the *while* loop, the program will read all the directories and files. If current *dir* is ‘.’ (represent this directory) or ‘..’ (represent parent directory), then the program will skip them. If current *dir* is a directory, it will start recursive searching. If current *dir* is a file, then the program will check whether this file satisfies the name or modified time restrictions. If the conditions are met, the program will output the last modified time and complete path of this file.    Fig.2 Implement detail of the *find* function  To check whether a file is satisfied the restriction, we define a *checkFile* function. The logic flow of it is quite simple. If the name has no restriction, then the function will only check the modified time. Otherwise, it will check the name first, which is done by *match* function. After name checking, the function will compute the date difference from the last modified time to now. If it is larger than the specified modified days, it will return false.  Because the name restriction my contain wildcard character, we use recursive algorithm to implement the *match* function. Before this function calls the recursive check, it will return the value directly for some special cases. If the two input strings are both empty strings, the match is successful. If the name restriction is empty but the file name is not empty, the match fails. If the name restriction is an asterisk plus other characters but the file name is not empty, the matching fails.  After checking the above situation, this function will make a recursive checking based on wildcards. If the first character of the name restriction is a question mark or the same as the first character of the file name, the first character of both will be removed, and the removed character string will be passed into the recursive checking. If the first character of name restriction is an asterisk, there are two situations. In the first case, the asterisk is deduced as the first character of the file name, and then the name restriction and the file name with the first character removed will enter recursion. In the second case, the asterisk is deduced as a null character, and then the file name and the name restriction with the first character removed are passed in to enter recursion. The program takes the OR operating result of the two cases and returns it.    Fig.3 Implement detail of the *checkFile* and *match* function  The running result is showed in the figure below. The three input parameters of the program can be used in combination as a restriction for searching files. It can be seen from the running results that the program can excellently fulfill the requirements of the experiment.    Fig.4 Running result of *myfind* command  **Appendix: Source code for this session.**  // myfind.cpp  #include<stdio.h>  #include<iostream>  #include<getopt.h>  #include<string>  #include<string.h>  #include<unistd.h>  #include <dirent.h>  #include <time.h>  #include <stdlib.h>  #include <sys/types.h>  #include <sys/stat.h>  using namespace std;  #define MAX\_PATH\_LENGTH 65535  char root\_path[MAX\_PATH\_LENGTH];  string pattern = "";  int modifiedTime = -1;  bool match(string pattern, string fileName){  if(pattern.length() == 0 && fileName.length() == 0) return true;  if(pattern.length() == 0 && fileName.length() > 0) return false;  if(pattern[0] == '\*' && pattern.length() > 1 && fileName.length() == 0)  return false;  if(pattern[0] == '?' || pattern[0] == fileName[0])  return match(pattern.substr(1),fileName.substr(1));  if(pattern[0] == '\*')  return match(pattern.substr(1),fileName) || match(pattern,fileName.substr(1));  return false;  }  bool checkFile(string fileName,struct stat buf){  if(pattern.length() == 0 || (pattern.length() != 0 && match(pattern,fileName))){  if(modifiedTime!=-1 && (time(0) - buf.st\_mtim.tv\_sec)/(24\*3600) > modifiedTime) return false;  return true;  }  return false;  }  void find(string path){  DIR \*d;  struct dirent \*dir;  d = opendir(path.c\_str());  if(d){  if(strcmp(root\_path,path.c\_str())==0){  printf("Target path: %s\n",root\_path);  }  while((dir = readdir(d))!=NULL){  string newpath;  // memset(newpath,0,sizeof(newpath));  newpath = path + ((path[path.length() - 1]=='/')?"":"/") + dir->d\_name;  if(dir->d\_type == DT\_DIR){  if(strcmp(dir->d\_name,"..") == 0 || strcmp(dir->d\_name,".") == 0)  continue;  find(newpath);  }  else{  struct stat buf;  if(stat(newpath.c\_str(),&buf) == 0){  if(checkFile(dir->d\_name,buf)){  struct tm\* timeSet = gmtime(&(buf.st\_mtim.tv\_sec));  char timeString[50];  strftime(timeString ,50, "%Y-%m-%d %H:%M:%S::%Z", timeSet);  printf("%s %s%s%s\n",timeString,path.c\_str(),(path[path.length() - 1]=='/'?"":"/"),dir->d\_name);  }  }  else{  fprintf(stderr,"Failed to get file information: %s.\nError no:%d\n",newpath.c\_str(),errno);  }  }    }  closedir(d);  }  else{  fprintf(stderr,"Failed to read directory: %s\n",path.c\_str());  }  }  int main(int argc, char\* argv[]){  static struct option opts[]={  {"path",required\_argument,NULL,'p'},  {"name",required\_argument,NULL,'n'},  {"mtime",required\_argument,NULL,'m'},  {0,0,0,0}  };  int opt;  while((opt = getopt\_long(argc,argv,"p:n:m:",opts,NULL))!=-1){  switch (opt)  {  case 'p':  strcpy(root\_path,optarg);  break;  case 'n':  pattern = optarg;  printf("You are looking for files whose name is/like: %s\n",pattern.c\_str());  break;  case 'm':  modifiedTime = atoi(optarg);  printf("You are looking for files modified %d days before today.\n",modifiedTime);  break;  case '?':  //printf("%c", optopt);  exit(1);  default:  break;  }  }  if(strlen(root\_path) == 0){  getcwd(root\_path,MAX\_PATH\_LENGTH);  }  find(root\_path);  }    } |
| **Conclusion** |
| From this lab session I ‘ve learned the how to implement a simple ‘find’ command using system call by C++ in Linux operating system. I also learned how to parse the parameters input by the user in the development of the console program in Linux system. This process 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. |
| **Teacher’s Comments and Score** |
| Comment：  Score：           Signature：                                                 Date： |