CIS 415 Operating Systems

Project 1 Report Collection

Submitted to:
Prof. Allen Malony

Author:
Antonio Jesus Silva Paucar

Report

Introduction

The project is first one for the class CIS415. Consist in writing a basic shell terminal using system calls to perform different tasks in the system(move, copy, change directory, cat, etc). The pseudo shell I have written can perform all the operations specified in the project's instructions and compile without any error or memory leak.

Background

During the development of the project, I found some problems with the format of the strings. Sometimes, the strings inside the array will end in "/0" or in "/n", so I got to set my main to catch both those cases when calling a function. I saved my tokens in a 2-dimensions arrays, which create a problem when passing the contents since, depending of the location of the token. If the location is the last one, the token would be saved with an ending "/n/o", which would create a problem when passing the name of a file, since that extra "/n" would generate a inconsistence when I tried to use the strcmp() function, for instance, whenever using "cd folder", even if the folder exists, it wouldn't recognize it. I had to write a little corrector for this case.

I have added a c file named "addons.c" that contains some functions to help me with memory allocation, memory release(when is a 2D array),a function (sunir()) that join and process the path of some files so it can be processed for the correspondent system call, a calls() function that it is in charge to check if there is more than 1 <cmd> in the line being processed to throw a control codefound error if needed, and a extract() function, which extract the name of a file of the format "../input.txt" whenever we select as the destination "." or current folder when using copyfile().

Implementation

Most of the problems I have encountered is how the text is saved in the array. Sometimes I needed to eliminate the last character of the string to be able to pass it to the function and have the correct result. To delete the last char, I used:

```
if (i == number_commands-1 && number_tokens > 2
   token[2][strlen(token[2])-1] = 0;
}
else if (i == number_commands-1 && number_tokens > 1
   token[1][strlen(token[1])-1] = 0;
}
```

The code will check if the line is the last one entered and if there are more than 1 or two tokens in the line. This is done to be able to delete that last '\n' that would interfere with the correct function when passing the string to the function that is in charge of doing copy or moving. When I tried to do a function to check for more than one command per line, I did:

```
int calls(char** list, int size){
   char** list_init = array_2d(MAX_PATH, BUFFER_LENGHT);
   strcpy(list_init[0],"ls\0");
   strcpy(list_init[1],"pwd\0");
```

```
    strcpy(list_init[2],"mkdir\0");
    strcpy(list_init[3],"cd\0");
    strcpy(list_init[4],"cp\0");
    strcpy(list_init[5],"mv\0");
    strcpy(list_init[6],"rm\0");
    strcpy(list_init[7],"cat\0");
    strcpy(list_init[8],"exit\0");
    char** list_rest = array_2d(MAX_PATH, BUFFER_LENGHT);
    strcpy(list_rest[0],"ls\n");
    strcpy(list_rest[1],"pwd\n");
    strcpy(list_rest[2],"mkdir\n");
    strcpy(list_rest[3],"cd\n");
    strcpy(list_rest[4],"cp\n");
    strcpy(list_rest[5],"mv\n");
    strcpy(list_rest[6],"rm\n");
    strcpy(list_rest[7],"cat\n");
    strcpy(list_rest[8],"exit\n");
    int resultados[LISTA_COMANDOS] = {0,0,0,0,0,0,0,0,0};
    for (int i = 0; i < size; i++){
      for (int j = 0; j < LISTA_COMANDOS; j++) {
        if(strcmp(list[i],list_init[j]) == 0 || strcmp(list[i],list_rest[j]) == 0){
          resultados[j]++;
        }
      }
    }
    int suma = 0;
    for (size_t i = 0; i < LISTA_COMANDOS; i++) {
      suma+=resultados[i];
    }
    free_double(list_init, MAX_PATH);
    free_double(list_rest, MAX_PATH);
    return suma;
}
```

Which return 0 if the line is empty or 1 if the line has just one command. If there is more than one command, the function will return more than 1 and this value is used to indicate that a control codefound error is in the line. To check if a "-f" flag is entered or no I used:

```
int lineas_file=0;
if(argv[1]!=NULL){
   strcpy(modo, argv[1]);
}else{
   strcpy(modo,"X");
}
```

If there is no flag and we tried to use the argv[1], we will get an error since argv[1] would be NULL. Modo is a variable that holds the "-f" flag. If there is no flag, a random character is saved, so the program can keep being executed.

For give just an example, my copy algorithm is as following:

```
void copyFile(char *sourcePath, char *destinationPath)
{
  char* filename = extract_name(sourcePath);
  if(strcmp(destinationPath, ".\0")==0){
     strcpy(destinationPath, filename);
  }
  int read_code = -1;
  int override = 0;//ask if override the file
  int end_loop = 0;//loop ending signal
  size_t lenght = 3;
  char* answer = char_string(MAX_PATH);
  char* line = char_string(MAX_PATH);
  while(end_loop == 0){//checks if the file exists or no
     if (stat(sourcePath, &st) == -1){//checking for the file to copy if exists
        break;
     }
     else if(stat(destinationPath, &st) == -1){// needs to check that exists.
        int input = open(sourcePath, O_RDONLY);//source
        if (input < 0){//chec
           break;
        }
```

```
            int output = open(destinationPath, O_CREAT | O_WRONLY, 0644);//destination
            if (output < 0){//checker
              break;
            }
            while(read_code!=0){//if 0, then end of the file.
              read_code = read(input, line, MAX_PATH);
              line[read_code] = '\0';//to signal the end.
              write(output, line, strlen(line));//<--------------system call to write files.
            }
            override = 1;
            close(input);
            close(output);
            if (override == 1){
                break;
            }
          }else{
            printf("%s", "Overwrite?(Y/N): ");//checks if it is overwritten
            getline(&answer,&lenght,stdin);
            if(strcmp(answer, "Y\n") == 0 || strcmp(answer, "y\n") == 0){
                unlink(destinationPath);//delete file first
                break;
            }else{
                break;
            }
          }
        }
        free(answer);
        free(line);
        free(filename);
      }
```

It is pretty straightforward. The code checks if the file exists and also checks if the destinationPath exists. I added an overwrite option. The logic of moveFile() is the same. Checks if the source exists and checks if the destination exists or offer the option to overwrite it. To tokenize the files, I used the following:

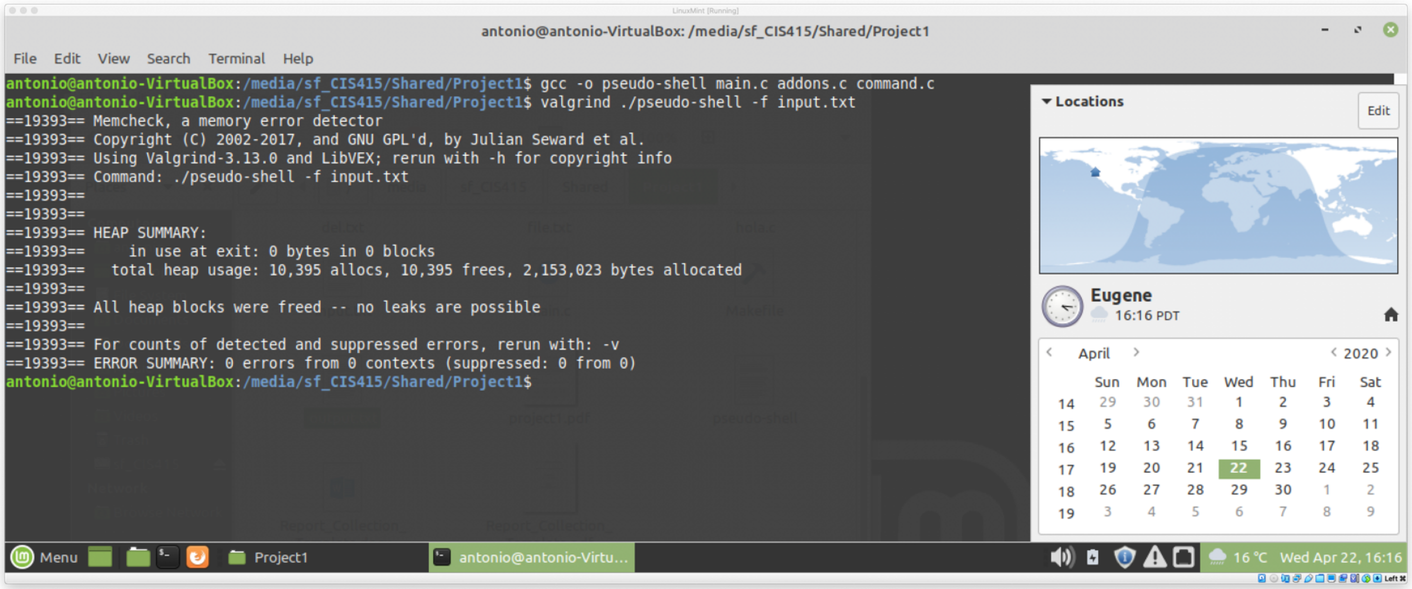
```
while ((parte = strtok_r(line, ";", &line))){//extract tokens and put into the array of strings.
  strcpy(list[number_commands],parte);// copy to an array of strings,  I used it this way for reuse the code in future projects.
  separador_si++;
```

```
    number_commands+=1;
  }
```

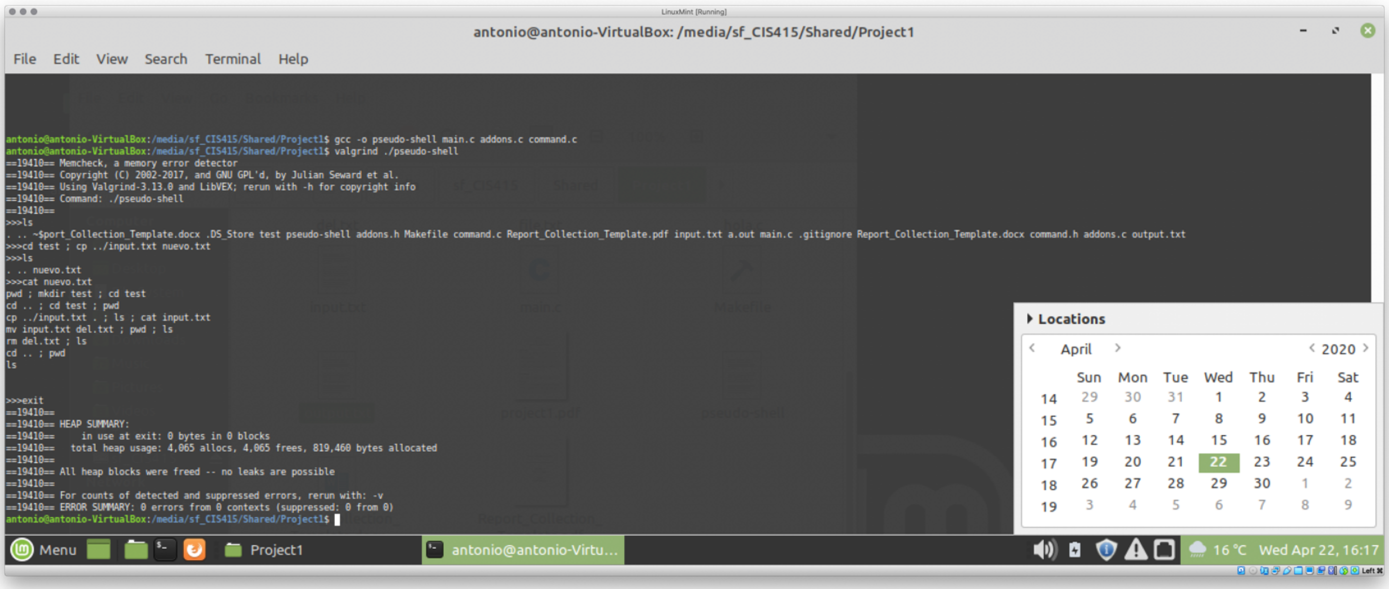
Strtok() will return a token from the line, the then copy it to a 2d array name list that be later processed by the program.

I have added a Makefile in the source for testing to program. It can be started with "make main" for interactive mode, "make mainv" for checking interactive mode in valgrind, "make mainf" for file mode and "make mainvf" for file mode in valgrind. Both files can be tested and no leaks.

Performance Results and Discussion



```
antonio@antonio-VirtualBox:/media/sf_CIS415/Shared/Project1$ gcc -o pseudo-shell main.c addons.c command.c
antonio@antonio-VirtualBox:/media/sf_CIS415/Shared/Project1$ valgrind ./pseudo-shell -f input.txt
==19393== Memcheck, a memory error detector
==19393== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==19393== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==19393== Command: ./pseudo-shell -f input.txt
==19393==
==19393==
==19393== HEAP SUMMARY:
==19393==     in use at exit: 0 bytes in 0 blocks
==19393==   total heap usage: 10,395 allocs, 10,395 frees, 2,153,023 bytes allocated
==19393==
==19393== All heap blocks were freed -- no leaks are possible
==19393==
==19393== For counts of detected and suppressed errors, rerun with: -v
==19393== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
antonio@antonio-VirtualBox:/media/sf_CIS415/Shared/Project1$
```



```
antonio@antonio-VirtualBox:/media/sf_CIS415/Shared/Project1$ gcc -o pseudo-shell main.c addons.c command.c
antonio@antonio-VirtualBox:/media/sf_CIS415/Shared/Project1$ valgrind ./pseudo-shell
==19410== Memcheck, a memory error detector
==19410== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==19410== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==19410== Command: ./pseudo-shell
==19410==
>>>ls
. .. ~$port_Collection_Template.docx .DS_Store test pseudo-shell addons.h Makefile command.c Report_Collection_Template.pdf input.txt a.out main.c .gitignore Report_Collection_Template.docx command.h addons.c output.txt
>>>cd test ; cp ../input.txt nuevo.txt
>>>ls
. .. nuevo.txt
>>>cat nuevo.txt
pwd ; mkdir test ; cd test
cd .. ; cd test ; pwd
cp ../input.txt . ; ls ; cat input.txt
mv input.txt del.txt ; pwd ; ls
rm del.txt ; ls
cd .. ; pwd
ls


>>>exit
==19410==
==19410== HEAP SUMMARY:
==19410==     in use at exit: 0 bytes in 0 blocks
==19410==   total heap usage: 4,065 allocs, 4,065 frees, 819,460 bytes allocated
==19410==
==19410== All heap blocks were freed -- no leaks are possible
==19410==
==19410== For counts of detected and suppressed errors, rerun with: -v
==19410== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
antonio@antonio-VirtualBox:/media/sf_CIS415/Shared/Project1$
```

The two screenshots above show the program running smoothly, showing all the requirements, and terminating correctly, no errors or memory leaks. The output is in the source folder.

Conclusion

My main problem during the development of this project was the string formats. It took me a bit of try and fail to figure out one way to find a solution for this case. I found that most of the system call functions would return a value in case of success or fail. I initially implement some checkers, like if the file exists, or the folder couldn't be created, etc but I removed since the project asked no extra print statements. I did add a print to check if the user is ok with overwriting a file. The implementation of the calls was pretty straightforward. It didn't present so much problem, the understanding how to pass the inputs to the programs was the most complicated part for me.

Sources

http://man7.org/linux/man-pages/man2/unlink.2.html
https://stackoverflow.com/questions/1726298/strip-first-and-last-character-from-c-string
https://www.geeksforgeeks.org/strtok-strtok_r-functions-c-examples/
https://www.geeksforgeeks.org/strncat-function-in-c-cpp/