National University of Sciences and Technology

Operating system

BSCS-11-A

Submitted to: Dr Farzana Jabeen

Date of Submission: 23 May 2023

Submitted by:

Name	CMS ID
Muhammad Ahmad Raza KHAN	371502
Wasam Khan	370080

Report

Github: https://github.com/wasam-khan/OS-project

Introduction:

We have implemented a simple command line interpreter called "Pathan Shell." It allows users to execute various commands, navigate directories, manipulate files, and perform system-related operations. The shell provides a command history feature and records each command executed.

Repository Used:

We used the second repository for our semester project. The purpose for using the second repository was that most of functions were already implemented but there were some that were not implemented so we implemented those functions.

Command Execution:

The myExecvp function is responsible for executing external commands. It utilizes the fork and execvp system calls to create a child process and execute the specified command. The parent process waits for the child process to complete using waitpid.

The StrTokenizer function is responsible for dividing a given input string into separate tokens based on a specified delimiter. It achieves this by utilizing the strtok function, which splits the string into tokens and returns a pointer to the first token. The function then iterates through the tokens using a while loop and stores each token in an array called argv. By incrementing the pointer *argv++, each token is stored in a consecutive position within the array. To mark the end of the token list, the last element of argv is set to NULL.

Tthe FreeMemory function is designed to release the memory allocated for the tokenized arguments (argv array). It accomplishes this by iterating through each element of argv until it encounters a NULL element. For each non-NULL element, the function uses the delete[] operator to deallocate the memory. Finally, the function sets each element of argv to NULL to invalidate the pointers. These two functions work together to ensure proper memory management in the code, allowing for efficient tokenization and freeing of allocated memory to avoid memory leaks.

Commands Implemented:

This is list of built in commands that are usually present in shell but were not present in repository 2 so we implemented them:

1. cd: Changes the current working directory.
2. mkdir: Creates a new directory.
3. grep: Searches for a pattern in a file.
4. alarm: Sets an alarm to interrupt the program after a specified time.
5. sleep: Suspends the program execution for a specified time.
6. cat: Displays the contents of a file.
7. echo: Prints the provided arguments.
8. touch: Creates an empty file.
9. copy: Appends the content of one file to another.
10. GetSystemInfo: Prints system information using the "uname" command.

This is list of special commands that we have implemented that use keywords defined by us to execute them:

1. Append: it appends a string at end of file
2. DeleteEmptyFiles: Deletes all empty files in the current directory and its subdirectories.

3. SortedListWithSize: It provides the list of all files in descending order of current directory.
4. History: It shows you your history, all commands you executed.

Parts Performed by each member:

Ahmad Raza Khan:

1. Delete empty files
2. Sorted lost with size
3. Echo
4. Append string to file
5. Cat command
6. Mkdir
7. History
8. Kill

Wasam Khan:

1. Change directory
2. Copy command
3. Sleep
4. Touch
5. Alarm
6. Grep
7. Get sysyem info
8. Swap files

Code:

Function:

1. void ChangeDirectory(char **argv);

```
void ChangeDirectory(char **argv)
{
   if (argv[1] == NULL)
   {
      cout << "Error: cd <directory> is missing" << endl;
      return;
   }

   if (chdir(argv[1]) != 0)
   {
      cout << "Error changing directory to " << argv[1] << endl;
   }

}
```

2. void RunCatCommand(char **argv);

```
void RunCatCommand(char **argv)
{
   if (argv[1] == nullptr)
   {
      cout << "Error: cat <filename> is missing" << endl;
      return;
   }

   ifstream file(argv[1]);
```

```
    if (!file)
    {
        cout << "Error: File " << argv[1] << " does not exist." << endl;
        return;
    }

    string line;
    while (getline(file, line))
    {
        cout << line << endl;
    }
}
```

3. void RunCopyCommand(char **argv);

```
void RunCopyCommand(char **argv)
{
    if (argv[1] == nullptr || argv[2] == nullptr)
    {
        cout << "Error: copy <source_file> <destination_file> is missing" << endl;
        return;
    }

    ifstream inputFile(argv[1]);
    if (!inputFile)
    {
        cout << "Error: Source file " << argv[1] << " does not exist." << endl;
        return;
    }

    ofstream outputFile(argv[2], ios::app);
    if (!outputFile)
    {
        cout << "Error: Failed to open destination file " << argv[2] << " for appending." << endl;
        return;
    }

    string line;
    while (getline(inputFile, line))
    {
        outputFile << line << endl;
    }

    cout << "File appended successfully from " << argv[1] << " to " << argv[2] << endl;
}
```

4. void AppendStringToFile(const string& filename, const string& content);

```
void AppendStringToFile(const string& filename, const string& content)
{
   ofstream file(filename, ios::app);
   if (file.is_open())
   {
      file << content << endl;
      file.close();
      cout << "String appended to file successfully." << endl;
   }
   else
   {
      cout << "Error: Failed to open the file for appending." << endl;
   }
}
```

5. void RunEchoCommand(char **argv);

```
void RunEchoCommand(char **argv)
{
   int i = 1;
   while (argv[i] != nullptr)
   {
      cout << argv[i] << " ";
      i++;
   }
   cout << endl;
}
```

6. void RunMkdirCommand(char **argv);

```
void RunMkdirCommand(char **argv)
{
   if (argv[1] == nullptr)
   {
```

```
        cout << "Error: mkdir <directory_name> is missing" << endl;
        return;
    }

    if (mkdir(argv[1], 0777) == 0)
    {
        cout << "Directory created successfully: " << argv[1] << endl;
    }
    else
    {
        cout << "Error creating directory: " << argv[1] << endl;
    }
}
```

7. void RunTouchCommand(char **argv);

```
void RunTouchCommand(char **argv)
{
    if (argv[1] == nullptr)
    {
        std::cout << "Error: touch <filename> is missing" << std::endl;
        return;
    }

    std::ofstream file(argv[1]);
    if (!file)
    {
        std::cout << "Error creating file: " << argv[1] << std::endl;
        return;
    }

    file.close();

    std::cout << "File created successfully: " << argv[1] << std::endl;
}
```

8. void RunSleepCommand(char **argv);

```
void RunSleepCommand(char **argv)
{
   if (argv[1] == nullptr)
   {
      cout << "Error: sleep <seconds> is missing" << endl;
      return;
   }

   unsigned int seconds = atoi(argv[1]);
   cout << "Sleeping for " << seconds << " seconds." << endl;
   sleep(seconds);


}
```

9. void RunAlarmCommand(char **argv);

```
void RunAlarmCommand(char **argv)

{

   if (argv[1] == nullptr)

   {

      cout << "Error: alarm <seconds> is missing" << endl;

      return;

   }


   unsigned int seconds = atoi(argv[1]);

   cout << "Alarm set for " << seconds << " seconds." << endl;

   alarm(seconds);




}
```

10. void myGrep(const string &pattern, const string &filename);

```
void myGrep(const string& pattern, const string& filename) {
    ifstream file(filename);
    if (!file) {
        cout << "Error: File " << filename << " does not exist." << endl;
        return;
    }

    string line;
    while (getline(file, line)) {
        if (line.find(pattern) != string::npos) {
            cout << line << std::endl;
        }
    }

    file.close();
}
```

11.void GetSystemInfo();

```
void GetSystemInfo(){
        system("uname -a");
}
```

12.void DeleteEmptyFiles();

```
void DeleteEmptyFiles()
{
    char* argv[] = { (char*)"find", (char*)".", (char*)"-type", (char*)"f", (char*)"-empty", (char*)"-delete",
nullptr };
    myExecvp(argv);
}
```

13.void SortedListWithSize();

```
void SortedListWithSize()

{

    int pipefd[2];
```

```
    pipe(pipefd);

    pid_t pid = fork();

    if (pid < 0)
    {
        cout << "Error forking process." << endl;
        return;
    }
    else if (pid == 0)
    {
        close(pipefd[0]);
        dup2(pipefd[1], STDOUT_FILENO);
        close(pipefd[1]);

        char* argv1[] = { (char*)"du", (char*)"-h", (char*)"--max-depth=1", (char*)".", NULL };
        execvp(argv1[0], argv1);

    }
    else
    {
        wait(NULL);

        close(pipefd[1]);
        dup2(pipefd[0], STDIN_FILENO);
        close(pipefd[0]);

        char* argv2[] = { (char*)"sort", (char*)"-hr", NULL };
        execvp(argv2[0], argv2);
        exit(EXIT_FAILURE);
    }
}
```

Main Function:

```
int main()
{
   char *path2;
   char *arr[250];
   char *Tokenized;
   char input[250];
   string timestamp;
   vector<pair<string,string>> commandHistory;
   char *argv[TOKENSIZE];

   while (true)
   {

            time_t now = time(0);
            tm *ltm = localtime(&now);

            // Format the timestamp
            char buffer[20];
            strftime(buffer, sizeof(buffer), "%Y-%m-%d %H:%M:%S", ltm);

            timestamp = buffer;


      cout << "Pathan Shell-> ";
      cin.getline(input, 250);
```

```
StrTokenizer(input, argv);
if (strcmp(input, "exit") == 0)
{
   break;
}
else if (strcmp(input, "\n") == 0)
{
   continue;
}
else if (strcmp(input, "history") == 0)
{
   cout << "Command History:" << endl;
   for (const auto& command : commandHistory)
   {

      // Print the timestamp and command
      cout << "[" << command.first << "] " << command.second << endl;
   }
   continue;
}



else if (strcmp(argv[0], "cd") == 0)
{
   ChangeDirectory(argv);
   commandHistory.push_back(make_pair(timestamp,argv[0])); // Add the command to history
   continue;
}
else if (strcmp(argv[0], "mkdir") == 0)
     {
     RunMkdirCommand(argv);
     commandHistory.push_back(make_pair(timestamp,argv[0])); // Add the command to history
```

```
        continue;
        }
        else if (strcmp(argv[0], "grep") == 0)
    {
       if (argv[1] == nullptr || argv[2] == nullptr)
       {
          cout << "Error: grep <pattern> <file> is missing" << endl;
          continue;
       }

       string pattern(argv[1]);
       string filename(argv[2]);
       myGrep(pattern, filename);
       commandHistory.push_back(make_pair(timestamp,argv[0])); // Add the command to history
        continue;
    }
        else if (strcmp(argv[0], "alarm") == 0)
        {
        RunAlarmCommand(argv);
        commandHistory.push_back(make_pair(timestamp,argv[0])); // Add the command to history
        continue;
        }
        else if (strcmp(argv[0], "sleep") == 0)
        {
        RunSleepCommand(argv);
        commandHistory.push_back(make_pair(timestamp,argv[0])); // Add the command to history
        continue;
        }

    else if (strcmp(argv[0], "cat") == 0)
    {
       RunCatCommand(argv);
```

```
        commandHistory.push_back(make_pair(timestamp,argv[0])); // Add the command to history
    continue;
}
else if (strcmp(argv[0], "echo") == 0)
{
    RunEchoCommand(argv);
    commandHistory.push_back(make_pair(timestamp,argv[0])); // Add the command to history
    continue;
}
else if (strcmp(argv[0], "touch") == 0)
      {
      RunTouchCommand(argv);
      commandHistory.push_back(make_pair(timestamp,argv[0])); // Add the command to history
      continue;
      }
else if (strcmp(argv[0], "copy") == 0)
{
    RunCopyCommand(argv);
        commandHistory.push_back(make_pair(timestamp,argv[0])); // Add the command to history
    continue;
}
else if (strcmp(argv[0], "sortwithsize") == 0)
{
    SortedListWithSize();
    commandHistory.push_back(make_pair(timestamp,argv[0])); // Add the command to history
    continue;
}

  else if (strcmp(argv[0], "delemptyfiles") == 0)
{
    DeleteEmptyFiles();
    commandHistory.push_back(make_pair(timestamp,argv[0])); // Add the command to history
```

```
            continue;
        }
        else if (strcmp(argv[0], "systeminfo") == 0)
        {
            GetSystemInfo();
                commandHistory.push_back(make_pair(timestamp,argv[0])); // Add the command to history
            continue;
        }
        else if (strcmp(argv[0], "append") == 0)
        {
            if (argv[1] == nullptr || argv[2] == nullptr)
            {
                cout << "Error: append <file> <content> is missing" << endl;
                continue;
            }

            AppendStringToFile(argv[1], argv[2]);
            commandHistory.push_back(make_pair(timestamp,argv[0])); // Add the command to history
            continue;
        }
        string command(input);
        commandHistory.push_back(make_pair(timestamp,command)); // Add the command to history
        myExecvp(argv);
    }
    return 0;
}
```

Execution Function:

```
void myExecvp(char **argv)
{
```

```
    pid_t pid;
    int status;
    int childStatus;
    pid = fork();
    if (pid == 0)
    {
        childStatus = execvp(*argv, argv);
        if (childStatus < 0)
        {
            cout << "ERROR: wrong input" << endl;
        }
        exit(0);
    }
    else if (pid < 0)
    {
        cout << "something went wrong!" << endl;
    }
    else
    {
        int status;
        waitpid(pid, &status, 0);
    }
}
```

Tokenizer:

```
void StrTokenizer(char *input, char **argv)
{
    char *stringTokenized;
    stringTokenized = strtok(input, " ");
    while (stringTokenized != NULL)
    {
```

```
        *argv++ = stringTokenized;
        stringTokenized = strtok(NULL, " ");
    }
    *argv = NULL;
}
```

Environment:

```
int GetEnv()
{
    char *path2;
    char *arr2[250];
    char *Tokenized;
    path2 = getenv("PATH");
    Tokenized = strtok(path2, ":");
    int k = 0;
    while (Tokenized != NULL)
    {
        arr2[k] = Tokenized;
        Tokenized = strtok(NULL, ":");
        k++;
    }
    arr2[k] = NULL;
}
```

Conclusion:

The Pathan Shell provides a basic command line interface with support for executing external commands, navigating directories, manipulating files, and performing system-related operations. It offers a command history feature and implements several built-in commands for convenience. The code demonstrates the usage of system calls like fork and execvp, as well as file and directory manipulation functions. With further enhancements, such as additional error handling

and more advanced features, this shell could serve as a starting point for building more sophisticated command line interfaces.