SSN COLLEGE OF ENGINEERING, KALAVAKKAM
(An Autonomous Institution, Affiliated to Anna University, Chennai)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1411 - OPERATING SYSTEMS LAB

LAB EXERCISE 2

Implementation of System calls

Submission Date:16-03-2022

Name: Jayannthan P T *Dept: CSE 'A'* *Roll No.: 205001049*

Implementing cp command in C using system calls

Algorithm:

1) If argc greater than 4, then print error : too many arguments
2) Else if argc is lesser than 2, then print error :more arguments required
3) Else
   i. Open file using call open() using filename as argument provided in read-only mode and store the file pointer in file_descriptor1
   ii. If file_descriptor1 is equal to -1 then print error and exit
   iii. Else then read the contents using call read() and store the return value in contents
   iv. Close the file
   v. If "i" is present in argument then
      a. Create a file using creat() and store the file pointer in file_ descriptor2
      b. If file_ descriptor2 is equal to -1 then print error and exit
      c. Write the "contents" into file_ descriptor2 using write() call
      d. Close the file
   vi. Else then
      a. Open a file using open() and store the file pointer in file_ descriptor2
      b. If file_ descriptor2 is less than 0 then prompt for overwrite
         - If answer is yes overwrite in the same file
         - Else create new file and write the contents into it
      c. Else then create new file and write the contents into it

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
int main(int argc, char *argv[])
{
```

```
if (argc > 4)
    printf("Too many arguements\n");
else if (argc < 2)
    printf("Atleast Two arguements required\n");
else
{
    printf("Opening file1:\n");
    int file_descriptor1 = open(argv[1], O_RDONLY);
    if (file_descriptor1 == -1)
        printf("Source File does not exist\n");
    else
    {
        char contents[100];
        printf("Reading file1:\n");
        int re = read(file_descriptor1, contents, 100);
        printf("Checking for i:\n");
        if (argc > 3 && strcmp(argv[3], "i") == 0)
        {
            printf("Found i:\n");
            printf("Creating file2:\n");

            int file_descriptor2 = creat(argv[2], S_IRUSR | S_IWUSR);
            if (file_descriptor2 < 0)
            {
                printf("!!!ERROR!!!\n");
            }
            else
            {

                printf("copying into file2:\n");
                int wr = write(file_descriptor2, contents, sizeof(contents));

                printf("Closing file2:\n");
                close(file_descriptor2);
                printf("\nSuccessfully Copied\n");
            }
            close(file_descriptor1);
        }
        else
        {

            printf("i not found:\n");

            printf("Checking file2:\n");
            int file_descriptor2 = open(argv[2], O_WRONLY);
            if (!(file_descriptor2 < 0))
            {
                char ch;
                printf("Overwrite %s file?(y/n) ", argv[2]);
                scanf(" %c", &ch);
                if (!(ch == 'y' || ch == 'Y'))
                {
                    close(file_descriptor2);
                    close(file_descriptor1);
```

```
                    }
                    else
                    {

                        printf("Creating file2:\n");
                        int file_descriptor2 = creat(argv[2], S_IRUSR | S_IWUSR);
                        if (file_descriptor2 < 0)
                        {
                            printf("!!!ERROR!!!\n");
                        }
                        else
                        {

                            printf("writing into file2:\n");
                            int wr = write(file_descriptor2, contents, sizeof(contents));
                            close(file_descriptor2);
                            printf("\nSuccessfully Copied\n");
                        }
                        close(file_descriptor1);
                    }
                }
                else
                {
                    printf("Creating file2:\n");

                    int file_descriptor2 = creat(argv[2], S_IRUSR | S_IWUSR);
                    if (file_descriptor2 < 0)
                    {
                        printf("!!!ERROR!!!\n");
                    }
                    else
                    {

                        printf("copying into file2:\n");
                        int wr = write(file_descriptor2, contents, sizeof(contents));

                        printf("Closing file2:\n");
                        close(file_descriptor2);
                        printf("\nSuccessfully Copied\n");
                    }
                    close(file_descriptor1);
                }
            }
        }
    }
    return 0;
}
```

Output:

```
~/OS-Lab$ ./cp main.c temp.c
Opening file1:
Reading file1:
Checking for i:
i not found:
Checking file2:
Creating file2:
copying into file2:
Closing file2:

Successfully Copied
```

```
~/OS-Lab$ ./cp main.c temp.c i
Opening file1:
Reading file1:
Checking for i:
Found i:
Creating file2:
copying into file2:
Closing file2:

Successfully Copied
```

```
Successfully Copied
~/OS-Lab$ ./cp main.c temp.c
Opening file1:
Reading file1:
Checking for i:
i not found:
Checking file2:
Overwrite temp.c file?(y/n) y
Creating file2:
writing into file2:

Successfully Copied
```

Implementing ls command in C using system calls

Algorithm:

1) If argc greater than 4, then print error: too many arguments
2) Else if argc is lesser than 1, then print error: more arguments required
3) Else if argc is equal to 2
    i. Open directory using call opendir() using directory-name as argument provided and store the file pointer in dir
    ii. If dir is null then print error and exit
    iii. Else display the names of all files and directories with name not starting with "."
    iv. Close the pointer dir
4) Else If "r" is present in argument then
    i. Open directory using call opendir() using directory-name as argument provided and store the file pointer in dir
    ii. If dir is null then print error and exit
    iii. Else display the names of all files and directories with name recursively
    iv. Close the pointer dir
5) Else If "a" is present in argument then
    i. Open directory using call opendir() using directory-name as argument provided and store the file pointer in dir
    ii. If dir is null then print error and exit

iii. Else display the names of all files and directory name
iv. Close the pointer dir

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <dirent.h>
#include <fcntl.h>

DIR *dir, *temp;
struct dirent *tmp;
void recursive(struct dirent *entry)
{
    if (entry == NULL)
    {
        return;
    }

    recursive(readdir(dir));
    printf("  %s\n", entry->d_name);
}

// void normal(struct dirent *entry, int n)
// {
//     if (entry == NULL)
//     {
//         return;
//     }
//     for (int i = 0; i < n; i++)
//     {
//         printf("\t");
//     }

//     printf("  %s\n", entry->d_name);

//     if (entry->d_type == DT_DIR && !(strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") == 0 || (entry->d_name[0] == '.')))
//     {
//         temp = opendir(entry->d_name);
//         tmp = readdir(temp);
//         normal(&tmp, n + 1);
//         closedir(temp);
//     }
//     normal(readdir(dir), n);
// }

int main(int argc, char *argv[])
{
    if (argc > 4)
        printf("Too many arguements\n");
    else if (argc < 1)
        printf("Atleast one arguement required\n");
```

```
    else
    {
        // printf("%d\n",argc);
        if (argc == 2)
        {
            struct dirent *entry;
            if ((dir = opendir(argv[1])) == NULL)
            {
                printf("CANNOT OPEN GIVEN DIRECTORY");
            }
            else
            {
                printf("Contents of the given:\n");
                while ((entry = readdir(dir)) != NULL)
                {
                    if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") ==
0 || (entry->d_name[0] == '.'))
                        continue;
                    printf("  %s\n", entry->d_name);
                }
                closedir(dir);
            }
        }
        else if (argc > 1 && strcmp(argv[2], "r") == 0)
        {
            printf("ls -R\n");
            struct dirent *entry;
            if ((dir = opendir(argv[1])) == NULL)
            {
                printf("CANNOT OPEN GIVEN DIRECTORY");
            }
            else
            {
                printf("Contents of the given:\n");
                recursive(&entry);
                closedir(dir);
            }
        }
        else if (argc > 1 && strcmp(argv[2], "a") == 0)
        {
            // DIR *dir;
            struct dirent *entry;
            if ((dir = opendir(argv[1])) == NULL)
            {
                printf("CANNOT OPEN GIVEN DIRECTORY");
            }
            else
            {
                printf("Contents of the given:\n");
                while ((entry = readdir(dir)) != NULL)
                {
                    printf("  %s\n", entry->d_name);
                }
                closedir(dir);
```

```
            }
        }
        // else if (argc > 1 && strcmp(argv[2], "R") == 0)
        // {
        //     // DIR *dir;
        //     struct dirent *entry;
        //     if ((dir = opendir(argv[1])) == NULL)
        //     {
        //         printf("CANNOT OPEN GIVEN DIRECTORY");
        //     }
        //     else
        //     {
        //         printf("Contents of the given:\n");
        //         if ((entry = readdir(dir)) != NULL)
        //         {
        //             normal(&entry, 0);
        //         }

        //         closedir(dir);
        //     }
        // }
    }
    return 0;
}
```

Output:

```
~/OS-Lab$ ./ls .
Contents of the given:
  replit.nix
  Makefile
  main.c
  Assignment1
  Assignment2
  duplicate_main.c
  full
  ls
  grep
  cp
  Assignment3
  a.out
```

```
~/OS-Lab$ ./ls . a
Contents of the given:
  .
  ..
  .cache
  .ccls-cache
  replit.nix
  .breakpoints
  Makefile
  .replit
  main.c
  Assignment1
  Assignment2
  duplicate_main.c
  full
  ls
  grep
  cp
  Assignment3
  a.out
```

```
~/OS-Lab$ ./ls . r
Contents of the given:
  ls
  a.out
  Assignment3
  cp
  grep
  full
  duplicate_main.c
  Assignment2
  Assignment1
  main.c
  .replit
  Makefile
  .breakpoints
  replit.nix
  .ccls-cache
  .cache
  ..
  .
```

Implementing grep command in C using system calls

Algorithm:

1) If argc greater than 4, then print error: too many arguments
2) Else if argc is lesser than 2, then print error: more arguments required
3) Else if
    i. Open file using call open() using filename as argument provided in read-only mode and store the file pointer in file_descriptor
    ii. If file_descriptor is equal to -1 then print error and exit
    iii. Else then read the contents using call read() and store the return value in buf
    iv. Close the file
    v. If argc is equal to 3 then
        a. Iterate through the buf and store each line in line
        b. Check for the input expression in line
        c. If it is present then display
    vi. Else If “c” is present in argument then
        a. Iterate through the buf and store each line in line
        b. Check for the input expression in line
        c. If it is present then increment the count value
        d. Display count

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <dirent.h>
#include <fcntl.h>
int main(int argc, char *argv[])
{
    if (argc > 4)
        printf("Too many arguements\n");
    else if (argc < 2)
        printf("Two arguements required\n");
    else
    {
        int file_descriptor = open(argv[2], O_RDONLY);
        if (file_descriptor == -1)
            printf("File does not exist\n");
        else
        {
            if (argc == 3)
            {
                // printf("Normal Grep\n");
                char line[100], buf[1024];
                int l = 0, i = 0, nr, count = 0;
                nr = read(file_descriptor, buf, 1024);
                close(file_descriptor);
                while (l < nr)
```

```
                {
                    for (i = 0; buf[l] != '\n' && l < nr; i++, l++)
                    {
                        line[i] = buf[l];
                    }
                    line[i] = '\0';
                    l++;
                    if (strstr(line, argv[1]))
                        printf("%s\n", line);
                }
            }
            else if (argc > 3 && strcmp(argv[3], "c") == 0)
            {
                // printf("Grep -c\n");
                char line[100], buf[1024];
                int l = 0, i = 0, nr, count = 0;
                nr = read(file_descriptor, buf, 1024);
                close(file_descriptor);
                while (l < nr)
                {
                    for (i = 0; buf[l] != '\n' && l < nr; i++, l++)
                    {
                        line[i] = buf[l]; // extracting lines
                    }
                    line[i] = '\0';
                    l++;
                    if (strstr(line, argv[1]))
                    {
                        count++;
                    }
                }
                printf("%d\n", count);
            }
        }
    }
    return 0;
}
```

Output:

```
~/OS-Lab$ ./grep main main.c
int main(void) {
~/OS-Lab$ ./grep main main.c c
1
```

Learning Outcome:

- Implemented various system commands in C using system calls
- Learned to handle system calls in C program