

Practical - 2

Aim [A] : Implement copy command using open, create, read, write, access and close system call. Be sure to include all necessary error checking including, ensuring the source file exists.

Test your program with following specifications.

- a. File extension with .txt , .c , .zip , .exe , .tar
- b. Copy whole directory.

Code :

```
#include<bits/stdc++.h>
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<stdarg.h>
#include<string.h>
#include<sys/types.h>
#include <sys/stat.h>
#include<fcntl.h>
#include <dirent.h>
#include<time.h>

#define BUF_SIZE 1024

using namespace std;

/// check for regular file
int is_regular_file(const char *path)
{
    struct stat path_stat;
    stat(path, &path_stat);
    return S_ISREG(path_stat.st_mode);
}

/// check for directory
int isDirectory(const char *path) {
    struct stat statbuf;
    if (stat(path, &statbuf) != 0)
        return 0;
    return S_ISDIR(statbuf.st_mode);
}

/// concat function for generate complete path!!!
char* concat(int count, ...)
{
    va_list ap;

    // Find required length to store merged string
    int len = 1; // room for NULL
    va_start(ap, count);
    for(int i=0 ; i<count ; i++)
        len += strlen(va_arg(ap, char*));
    va_end(ap);
```

```
// Allocate memory to concat strings
char *merged = (char*)calloc(sizeof(char),len);
int null_pos = 0;

// Actually concatenate strings
va_start(ap, count);
for(int i=0 ; i<count ; i++)
{
    char *s = va_arg(ap, char*);
    strcpy(merged+null_pos, s);
    null_pos += strlen(s);
}
va_end(ap);

return merged;
}

int copy_file(char *src, char *dest)
{
    int fdold, fdnew;
    fdold = open(src, O_RDONLY);

    if (fdold == -1){
        perror("Cannot Open File!");
        return -1;
    }

    fdnew = creat(dest, 0666);

    if (fdnew == -1){
        printf("Cannot Create File %s!", dest);
        return -1;
    }

    char buffer[BUF_SIZE];
    int count;
    while ((count = read(fdold, buffer, sizeof(buffer) ))> 0)
    {
        write(fdnew, buffer, count);
    }
    printf("Copied Successfull\n");

    return 0;
}

int copy_dir(char const *src, char const *dest)
{
    struct dirent *dr;
```

```

DIR *source_dir = opendir(src);
DIR *destination_dir = opendir(dest);

if (source_dir == NULL)
{
    perror("Could't open source directory");
    return -1;
}
if (destination_dir == NULL)
{
    printf("Creating destination directory\n");
    mkdir(dest, 0777);
}

while ((dr = readdir(source_dir)) != NULL)
{
    if ((( strcmp(dr->d_name, ".") && ( strcmp(dr->d_name, "..")))))
    {
        printf("\n%s\n", dr->d_name);
        if (isDirectory(concat(3, src, "/", dr->d_name))){
            if (mkdir(concat(3, dest, "/", dr->d_name), 0777) == -1){
                perror("Directory Creation Failed!");
            }
            else{
                printf("Directory Created!\n");
                copy_dir(concat(3, src, "/", dr->d_name),concat(3, dest, "/", dr->d_name));
            }
        }
        else{
            copy_file(concat(3, src, "/", dr->d_name), concat(3, dest, "/", dr->d_name));
        }
    }
}

closedir(source_dir);
closedir(destination_dir);

return 0;
}

int main(int argc, char* argv[]){

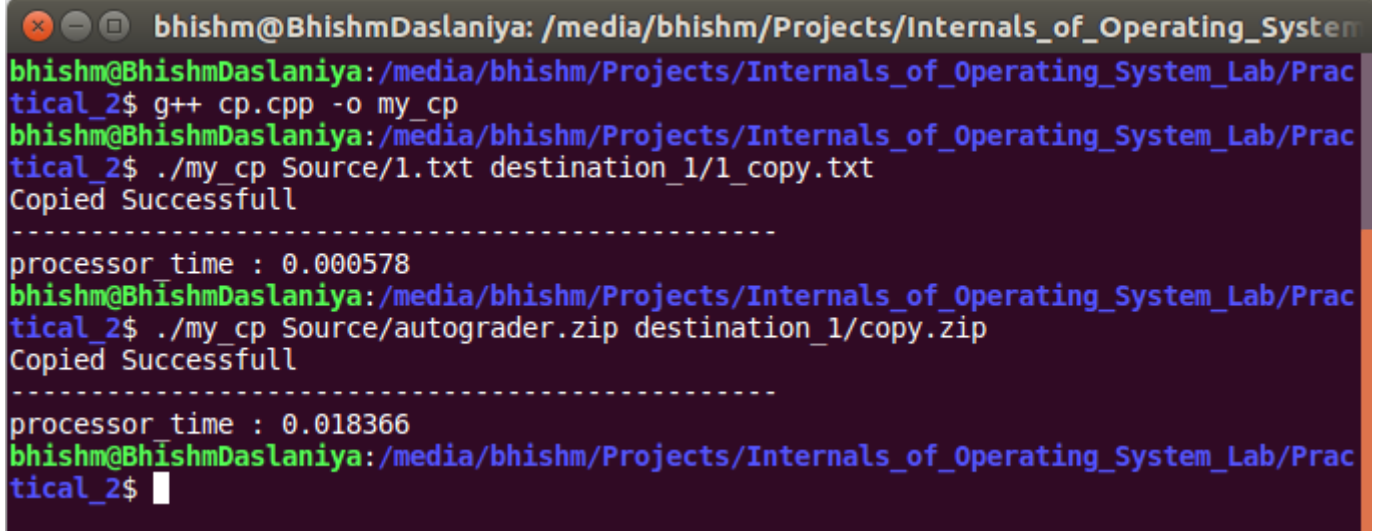
    if(argc < 3){
        perror("Needs 2 Arguments for copy commands\nEx: exeFile source_file/source_dir dest_file/dest_dir");
        return -1;
    }
    clock_t start = clock();
    if(is_regular_file(argv[1]) && (isDirectory(argv[2]) || is_regular_file(argv[2]))){
        copy_file(argv[1],argv[2]);
    }else if(isDirectory(argv[1])){

```

```

        copy_dir(argv[1],argv[2]);
    }else{
        /// File format not match!!!
        perror("File format is not regular or directory!");
    }
    clock_t end = clock();
    double processor_time = ((double) (end - start)) / CLOCKS_PER_SEC;
    cout<<"-----\n";
    cout<<"processor_time : " << processor_time <<"\n";
}


```

Output:


```

bhishm@BhishmDaslaniya: /media/bhishm/Projects/Internals_of_Operating_System
bhishm@BhishmDaslaniya:/media/bhishm/Projects/Internals_of_Operating_System_Lab/Prac
tical_2$ g++ cp.cpp -o my_cp
bhishm@BhishmDaslaniya:/media/bhishm/Projects/Internals_of_Operating_System_Lab/Prac
tical_2$ ./my_cp Source/1.txt destination_1/1_copy.txt
Copied Successfull
-----
processor time : 0.000578
bhishm@BhishmDaslaniya:/media/bhishm/Projects/Internals_of_Operating_System_Lab/Prac
tical_2$ ./my_cp Source/autograder.zip destination_1/copy.zip
Copied Successfull
-----
processor time : 0.018366
bhishm@BhishmDaslaniya:/media/bhishm/Projects/Internals_of_Operating_System_Lab/Prac
tical_2$ 

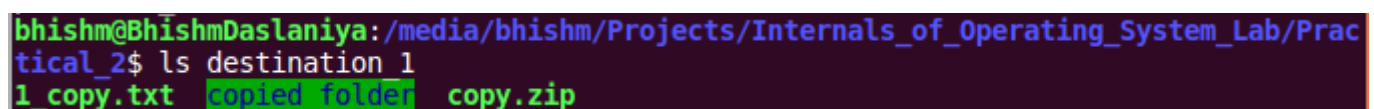
```



```

bhishm@BhishmDaslaniya:/media/bhishm/Projects/Internals_of_Operating_System_Lab/Prac
tical_2$ ./my_cp Source destination_1/copied_folder
Creating destination directory
1.txt
Copied Successfull
2.txt
Copied Successfull
3.jpg
Copied Successfull
4.pdf
Copied Successfull
autograder.zip
Copied Successfull
Silicon.Valley.S06E01.REPACK.mkv
Copied Successfull
-----
processor time : 9.64812

```



```

bhishm@BhishmDaslaniya:/media/bhishm/Projects/Internals_of_Operating_System_Lab/Prac
tical_2$ ls destination_1
1_copy.txt  copied_folder  copy.zip

```

Assignment :

a. Find the real time, processor time, user space time and kernel space time for copy command implementation.

Output :

```
bhishm@BhishmDaslaniya:/media/bhishm/Projects/Internals_of_Operating_System_Lab/Practical_2$ time ./my_cp Source/autograder.zip destination_1/copy.zip
Copied Successfull
-----
processor_time : 0.018735s

real    0m0.061s
user    0m0.001s
sys     0m0.024s
```

b. Copy the source file in two different files parallely. Find out does it take same time to finish. (Create two threads) (Files has to be atleast 100 MB)

Code :

```
// #include<bits/stdc++.h>
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<stdarg.h>
#include<string.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<dirent.h>
#include<time.h>
#include<pthread.h>

#define BUF_SIZE 1024

using namespace std;

struct thread_args {
    char *src;
    char *dest;
};

int is_regular_file(const char *path)
{
    struct stat path_stat;
    stat(path, &path_stat);
    return S_ISREG(path_stat.st_mode);
}

/// check for directory
int isDirectory(const char *path) {
    struct stat statbuf;
    if (stat(path, &statbuf) != 0)
        return 0;
```

```

    return S_ISDIR(statbuf.st_mode);
}

/// concat function for generate complete path!!!
char* concat(int count, ...)
{
    va_list ap;

    // Find required length to store merged string
    int len = 1; // room for NULL
    va_start(ap, count);
    for(int i=0 ; i<count ; i++)
        len += strlen(va_arg(ap, char*));
    va_end(ap);

    // Allocate memory to concat strings
    char *merged = (char*)calloc(sizeof(char),len);
    int null_pos = 0;

    // Actually concatenate strings
    va_start(ap, count);
    for(int i=0 ; i<count ; i++)
    {
        char *s = va_arg(ap, char*);
        strcpy(merged+null_pos, s);
        null_pos += strlen(s);
    }
    va_end(ap);

    return merged;
}

```

```

void *copy_file(void *args)
{
    time_t start = time(0);

    struct thread_args * str = (struct thread_args *) args;

    char *src = str->src;
    char *dest = str->dest;

    int fdold, fdnew;
    fdold = open(src, O_RDONLY);

    if (fdold == -1){
        perror("Cannot Open File!");
        return 0;
    }

    fdnew = creat(dest, 0666);

```

```

    if (fdnew == -1){
        printf("Cannot Create File %s!", dest);
        return 0;
    }

    char buffer[BUF_SIZE];
    int count;
    while ((count = read(fdold, buffer, sizeof(buffer) ))> 0)
    {
        write(fdnew, buffer, count);
    }

    time_t end = time(0);

    // double calc_time = (end - start) * 1000 / CLOCKS_PER_SEC;
    printf("Copied Successfull! [Time taken to copy file = %lds]\n", end - start);

    return 0;
}

void *copy_dir(void * args)
{
    struct dirent *dr;

    time_t start = time(0);

    struct thread_args * str = (struct thread_args *) args;

    char *src = str->src;
    char *dest = str->dest;

    DIR *source_dir = opendir(src);
    DIR *destination_dir = opendir(dest);

    if (source_dir == NULL)
    {
        perror("Could't open source directory");
        return 0 ;
    }
    if (destination_dir == NULL)
    {
        printf("Creating destination directory\n");
        mkdir(dest, 0777);
    }

    while ((dr = readdir(source_dir)) != NULL)
    {
        if ((( strcmp(dr->d_name, ".") && ( strcmp(dr->d_name, "..")))))
        {
            printf("\n%s\n", dr->d_name);
        }
    }
}

```

```

    if (isDirectory(concat(3, src, "/", dr->d_name))) {
        if (mkdir(concat(3, dest, "/", dr->d_name), 0777) == -1) {
            perror("Directory Creation Failed!");
        }
        else {
            printf("Directory Created!\n");
            str->src = concat(3, src, "/", dr->d_name);
            str->dest = concat(3, dest, "/", dr->d_name);
            copy_dir((void*) str);
        }
    }
    else {
        str->src = concat(3, src, "/", dr->d_name);
        str->dest = concat(3, dest, "/", dr->d_name);
        copy_file((void*) str);
    }
}

closedir(source_dir);
closedir(destination_dir);

time_t end = time(0);

// double calc_time = (end - start) * 1000 / CLOCKS_PER_SEC;
printf("Copied Successfull! [Time taken to copy file = %lds]\n", end - start);

return 0;
}

int main(int argc, char* argv[]) {
    if (argc < 4) {
        perror("Needs 3 Arguments for copy commands\nEx: exeFile source_file/source_dir\n\n dest_file_1/dest_dir_1 dest_file_2/dest_dir_2");
        return 0;
    }

    int n = 2;

    pthread_t tid[n];

    for (int i = 0; i < n; i++) {
        struct thread_args * t_arg = (struct thread_args *) malloc(sizeof (struct thread_args));
        t_arg->src = argv[1];
        t_arg->dest = argv[i+2];

        if (is_regular_file(argv[1])) {
            pthread_create(&tid[i], NULL, copy_file, (void *) t_arg);
        } else if (isDirectory(argv[1])) {
            pthread_create(&tid[i], NULL, copy_dir, (void *) t_arg);
        } else {

```



```

        perror("File format is not regular or directory!");
    }

}

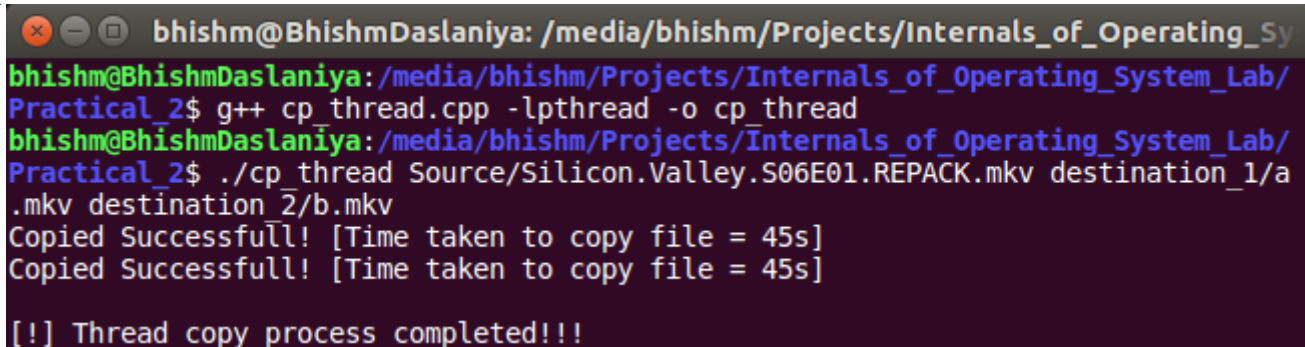
for (int i = 0; i < n; i++){
    pthread_join(tid[i], NULL);
}

printf("\n[!] Thread copy process completed!!!\n");

return 0;
}

```

Output :



```

bhishm@BhishmDaslaniya: /media/bhishm/Projects/Internals_of_Operating_Sy
bhishm@BhishmDaslaniya:/media/bhishm/Projects/Internals_of_Operating_System_Lab/
Practical_2$ g++ cp_thread.cpp -lpthread -o cp_thread
bhishm@BhishmDaslaniya:/media/bhishm/Projects/Internals_of_Operating_System_Lab/
Practical_2$ ./cp_thread Source/Silicon.Valley.S06E01.REPACK.mkv destination_1/a
.mkv destination_2/b.mkv
Copied Successfull! [Time taken to copy file = 45s]
Copied Successfull! [Time taken to copy file = 45s]

[!] Thread copy process completed!!!

```

Observation : Yes, It both threads takes same time to copy file.

c. Once you have correctly designed and tested the program, run the program using a utility (ptrace) that traces system calls. (Find out which system call/calls program has made internally)

Code:

```

#include<bits/stdc++.h>
#include <dirent.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <sys/ptrace.h>
#include <sys/reg.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/user.h>
#include <sys/wait.h>
#include <unistd.h>

#define BUF_SIZE 1024

using namespace std;

/// check for regular file
int is_regular_file(const char *path)
{

```

```

    struct stat path_stat;
    stat(path, &path_stat);
    return S_ISREG(path_stat.st_mode);
}

// check for directory
int isDirectory(const char *path) {
    struct stat statbuf;
    if (stat(path, &statbuf) != 0)
        return 0;
    return S_ISDIR(statbuf.st_mode);
}

// concat function for generate complete path!!!
char* concat(int count, ...)
{
    va_list ap;

    // Find required length to store merged string
    int len = 1; // room for NULL
    va_start(ap, count);
    for(int i=0 ; i<count ; i++)
        len += strlen(va_arg(ap, char*));
    va_end(ap);

    // Allocate memory to concat strings
    char *merged = (char*)calloc(sizeof(char),len);
    int null_pos = 0;

    // Actually concatenate strings
    va_start(ap, count);
    for(int i=0 ; i<count ; i++)
    {
        char *s = va_arg(ap, char*);
        strcpy(merged+null_pos, s);
        null_pos += strlen(s);
    }
    va_end(ap);

    return merged;
}

int copy_file(char *src, char *dest)
{
    int fdold, fdnew;
    fdold = open(src, O_RDONLY);

    if (fdold == -1){
        perror("Cannot Open File!");
        return -1;
    }
}

```

```

fdnew = creat(dest, 0666);

if (fdnew == -1){
    printf("Cannot Create File %s!", dest);
    return -1;
}

char buffer[BUF_SIZE];
int count;
while ((count = read(fdold, buffer, sizeof(buffer) ))> 0)
{
    write(fdnew, buffer, count);
}
printf("Copied Successfull\n");

return 0;
}

int copy_dir(char const *src, char const *dest)
{
    struct dirent *dr;

    DIR *source_dir = opendir(src);
    DIR *destination_dir = opendir(dest);

    if (source_dir == NULL)
    {
        perror("Could't open source directory");
        return -1;
    }
    if (destination_dir == NULL)
    {
        printf("Creating destination directory\n");
        mkdir(dest, 0777);
    }

    while ((dr = readdir(source_dir)) != NULL)
    {
        if ((( strcmp(dr->d_name, ".") && ( strcmp(dr->d_name, "..")))))
        {
            printf("\n%s\n", dr->d_name);
            if (isDirectory(concat(3, src, "/", dr->d_name))){
                if (mkdir(concat(3, dest, "/", dr->d_name), 0777) == -1){
                    perror("Directory Creation Failed!");
                }
            }
            else{
                printf("Directory Created!\n");
                copy_dir(concat(3, src, "/", dr->d_name),concat(3, dest, "/", dr->d_name));
            }
        }
    }
}

```

```

    }
    else{
        copy_file(concat(3, src, "/", dr->d_name), concat(3, dest, "/", dr->d_name));
    }
}
}

closedir(source_dir);
closedir(destination_dir);

return 0;
}

int main(int argc, char* argv[]){

    if(argc < 3){
        perror("Needs 2 Arguments for copy commands\nEx: exeFile source_file/source_dir dest_file/dest_dir");
        return -1;
    }

    pid_t child;
    int insyscall = 0;
    long orig_eax, eax;

    child = fork();

    if(child == 0){
        ptrace(PTRACE_TRACEME, 0, NULL, NULL);
        raise(SIGCONT);
        if(is_regular_file(argv[1]) && (isDirectory(argv[2]) || is_regular_file(argv[2]])){
            copy_file(argv[1],argv[2]);
        }else if(isDirectory(argv[1])){
            copy_dir(argv[1],argv[2]);
        }else{
            /// File format not match!!!
            perror("File format is not regular or directory!");
        }
    }else{
        int status;
        printf("-----");
        while (waitpid(child, &status, 0) && !WIFEXITED(status)) {
            struct user_regs_struct regs;
            ptrace(PTRACE_GETREGS, child, NULL, &regs);
            fprintf(stderr, "[!] Intercepted system call %lld.\n", (regs.orig_rax));
            ptrace(PTRACE_SYSCALL, child, NULL, NULL);
        }
    }
}

```

Output :

```

bhishm@BhishmDaslaniya:/media/bhishm/Projects/Internals_of_Operating_System_Lab/
Practical_2$ g++ my_ptrace.cpp -o ptrace
bhishm@BhishmDaslaniya:/media/bhishm/Projects/Internals_of_Operating_System_Lab/
Practical_2$ ./ptrace Source/1.txt destination_1/new.txt
[!] Intercepted system call 234.
[!] Intercepted system call 4.
[!] Intercepted system call 4.
[!] Intercepted system call 4.
[!] Intercepted system call 4.
[!] Intercepted system call 4.
[!] Intercepted system call 4.
[!] Intercepted system call 4.
[!] Intercepted system call 2.
[!] Intercepted system call 2.
[!] Intercepted system call 85.
[!] Intercepted system call 85.
[!] Intercepted system call 0.
[!] Intercepted system call 0.
[!] Intercepted system call 1.
[!] Intercepted system call 1.
[!] Intercepted system call 0.
[!] Intercepted system call 0.
[!] Intercepted system call 5.
[!] Intercepted system call 5.
[!] Intercepted system call 1.
Copied Successfull
[!] Intercepted system call 1.
[!] Intercepted system call 231.
-----bhishm@BhishmDaslaniya:/media/bhishm/P
Practical_2$ Mals of Operating System Lab/P

```

Aim [B] : Write a program for 'ls' command using 'opendir()' and 'readdir()' system call.

Code :

```

#include<bits/stdc++.h>
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<stdarg.h>
#include<string.h>
#include<sys/types.h>
#include <sys/stat.h>
#include<fcntl.h>
#include <dirent.h>

```

```
using namespace std;
```

```

void list_dir(char *path) {
    struct dirent *entry;
    DIR *dir = opendir(path);

    if(dir!=NULL){
        while ((entry = readdir(dir)) != NULL) {
            cout<<"-> "<< entry->d_name << endl;

```

```
    }
    closedir(dir);
} else {
    perror("could not open directory");
    return;
}
}
```

```
int main(int argc, char *argv[]) {
    list_dir(argv[1]);
}
```

Output :

```
bhishm@BhishmDaslaniya:/media/bhishm/Projects/Internals_of_Operating_System_Lab/
Practical_2$ g++ ls.cpp -o my_ls
bhishm@BhishmDaslaniya:/media/bhishm/Projects/Internals_of_Operating_System_Lab/
Practical_2$ ./my_ls Source/
-> .
-> ..
-> 1.txt
-> 2.txt
-> 3.jpg
-> 4.pdf
-> autograder.zip
-> Silicon.Valley.S06E01.REPACK.mkv
```

```
bhishm@BhishmDaslaniya:/media/bhishm/Projects/Internals_of_Operating_System_Lab/
Practical_2$ ./my_ls .
-> .
-> ..
-> ~/.lock.Practical_2.odt#
-> check_thread.cpp
-> cp.cpp
-> cp_thread
-> cp_thread.cpp
-> destination_1
-> destination_2
-> ls.cpp
-> my_cp
-> my_ls
-> Practical_2.odt
-> Source
```

Conclusion: From this practical I learnt about how to implement some useful linux commnad in C/C++.