Gebze Technical University

Computer Engineering

System Programing(CSE 344)

Homework #5

Berkan AKIN

171044073

Problem

In this project, you are expected to implement a directory copying tool called "pCp". This tool performs the overall task in parallel by creating a new thread for each file and subdirectory. Similar tools can quickly consume system resources when invoked under a large directory tree. Therefore, you are expected to implement a worker thread pool to regulate the number of active threads at any given time. Workers are blocked at a synchronization point (which can be an empty buffer in our case) and a worker is unblocked when a request arrives (when an item is placed in the buffer). The overall implementation of the tool should use a producer-consumer-based synchronization with the details described below.

Producer: Start by creating a producer thread function where you take at least 2 inputs (path names of two directories) as an array. For each file in the first directory, the producer opens the file for reading and creates a file with the same name in the second directory for writing. If a file with the same name already exists in the target directory, the file is opened and truncated. If any file opening error occurs, both files are closed, and an informative message is sent to the standard output. The two open file descriptors and the names of the files are then passed to a buffer. Additionally, you are expected to manage the buffer appropriately (whether it is empty or full, whether accessing the buffer is appropriate or execution should wait) to gracefully terminate the threads. This is a limited buffer issue on the producer side. When the producer finishes populating the buffer with file names for the given directories, it informs the rest of the program (by setting a completion flag) and exits.

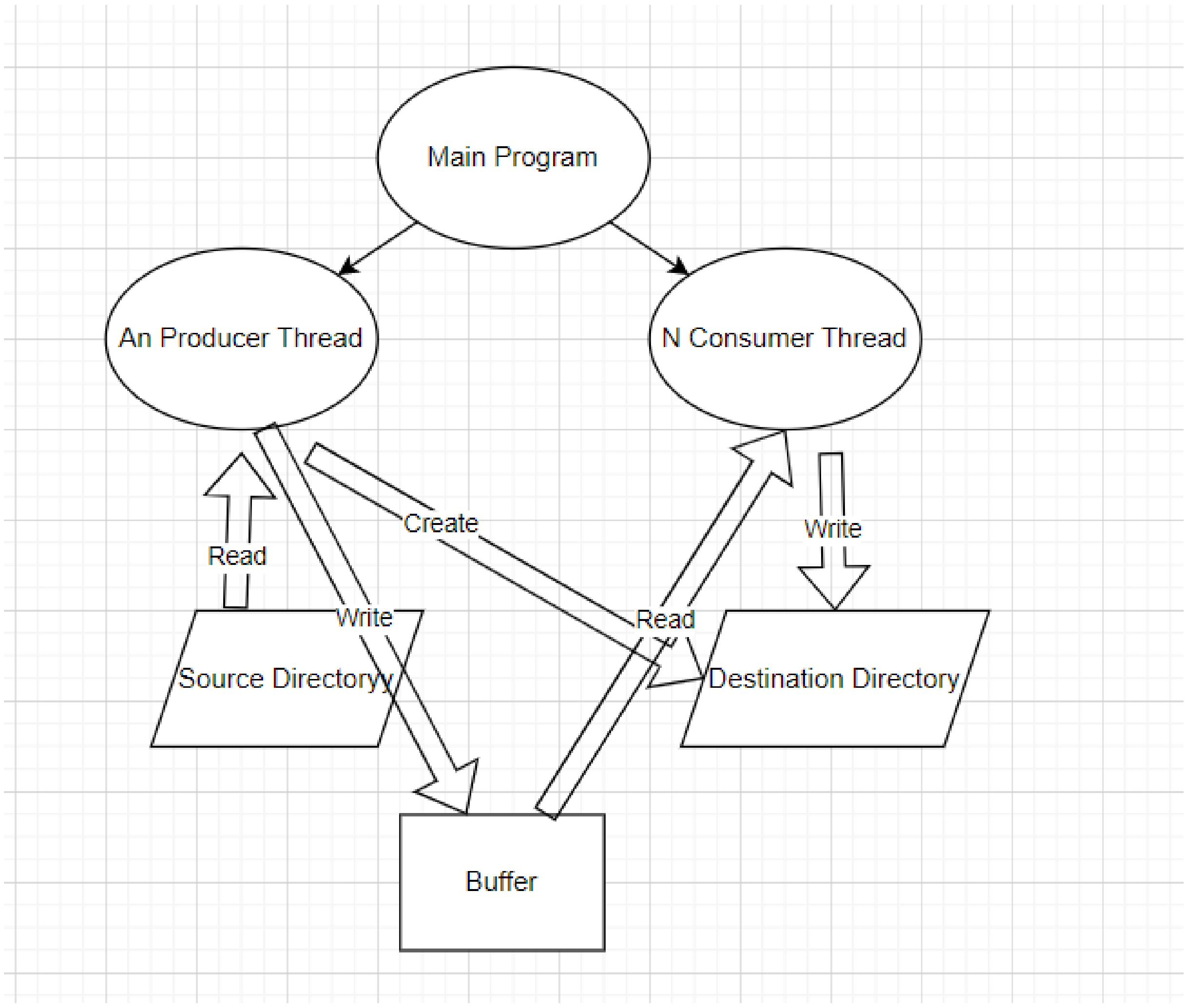
Consumer: Each consumer thread reads an item from the buffer, copies the file from the source file descriptor to the destination file descriptor, closes the files, and writes a message to the standard output indicating the completion status. Since producers and multiple consumers write to the standard output, this is a critical section that needs protection. Consumers should be terminated when they detect that a completion flag is set and there is no more input in the buffer.

Main Program: The main program should take the buffer size, the number of consumer threads, and the source and destination directories as command-line arguments (the tool should have only one producer thread). The main program should initialize the threads and use pthread_join to wait for their completion. Use gettimeofday to get the time before the creation of the first thread and after the last join. Display the total time taken to copy the files in the directory. The tool should be able to copy regular files and FIFOs. It should also be able to recursively copy subdirectories. Keep track of the number and types of files copied. Track the total number of bytes copied. Experiment with different buffer sizes and different numbers of consumer threads. Report your results and comment on the buffer/consumer thread number combinations that yield the best results. Observe what happens when you exceed the per-process limits of open file descriptors. The tool should check if this causes an error. Be mindful of memory leaks in your program. Signals should be handled properly as well.

Solitions

1. First, import the necessary POSIX headers and the pthread library.
2. Take the buffer size, the number of consumer threads, and the source and destination directories as command-line arguments.
3. Create a thread pool for worker threads. This is a structure where a specific number of worker threads will be used. The thread pool should have a buffer used for synchronization.
4. Create the producer thread. This thread reads files from the source directory and assigns copying tasks to the worker threads. The producer should control the state of the buffer when adding file information to it (it can add if the buffer is empty or wait if it is full).
5. Create the worker threads. Workers retrieve file information from the buffer, copy the files, and update statistics. Workers should wait when the buffer is empty and should be awakened when there is a new task.
6. In the main program, use the gettimeofday function to get the start time.
7. Wait for the worker threads to complete (using pthread_join).
8. Use the gettimeofday function in the main program to get the end time and calculate the total copying time.
9. Report the statistics and results.
10. Pay attention to open file descriptors. Check and handle the situation when you exceed the allowed limit of open file descriptors for a process.
11. Check for memory leaks and clean up if necessary.
12. Handle signals properly. It is especially important to handle the SIGINT signal (Ctrl+C) to ensure graceful termination of the program.

System Design



1. **Main Program**

   a. Create all producer and consumer threads.

   b. Use pthread_join to wait for the completion of all threads.

   c. Get the end time using the gettimeofday function.

   d. Calculate the total copying time.

   e. Report the statistics and results.

2. **Producer Thread:**

   a. Create a loop for each file and subdirectory in the source directory.

   b. Perform the file opening operation.

   c. If a file with the same name exists in the destination directory, open the file and truncate it.

   d. Add the file information to the buffer.

e. If the buffer is empty, send a wake-up signal.

f. Close the file and print an informative message in case of an error..

3. **Cunsomer Thread**

a. Check the buffer, retrieve the file information if available.

b. Copy the file from the source to the destination.

c. Close the file and print a message indicating the completion status of the copy.

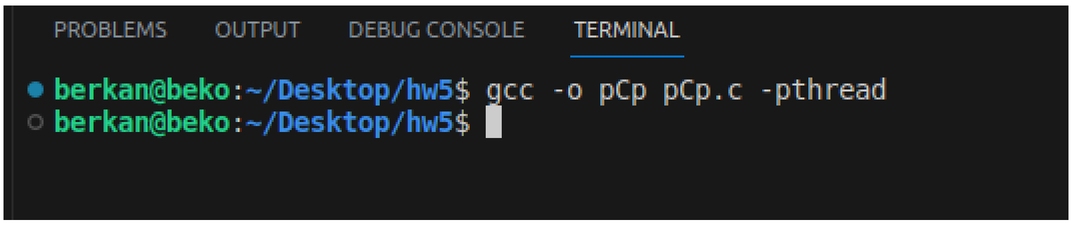
d. Update the statistics.

e. If the buffer is empty, send a wait signal.

f. Check the "done" flag, if the buffer is empty and the "done" flag is set, terminate the thread..

How to run the program

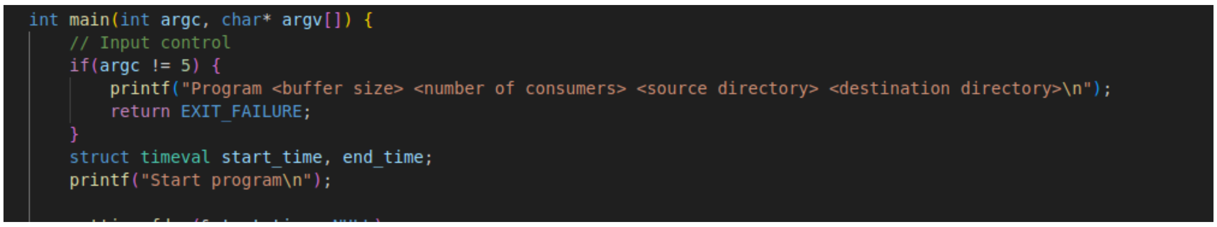
Compilation



```
berkan@beko:~/Desktop/hw5$ gcc -o pCp pCp.c -pthread
berkan@beko:~/Desktop/hw5$
```

gcc -o pCp pCp.c -pthread

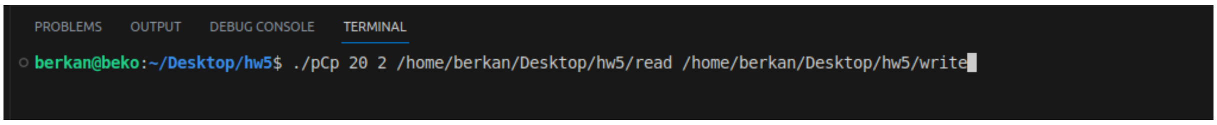
Running the program



```
int main(int argc, char* argv[]) {
    // Input control
    if(argc != 5) {
        printf("Program <buffer size> <number of consumers> <source directory> <destination directory>\n");
        return EXIT_FAILURE;
    }
    struct timeval start_time, end_time;
    printf("Start program\n");
```

Program <buffer size> <number of consumers> <source directory> <destination directory>

Example

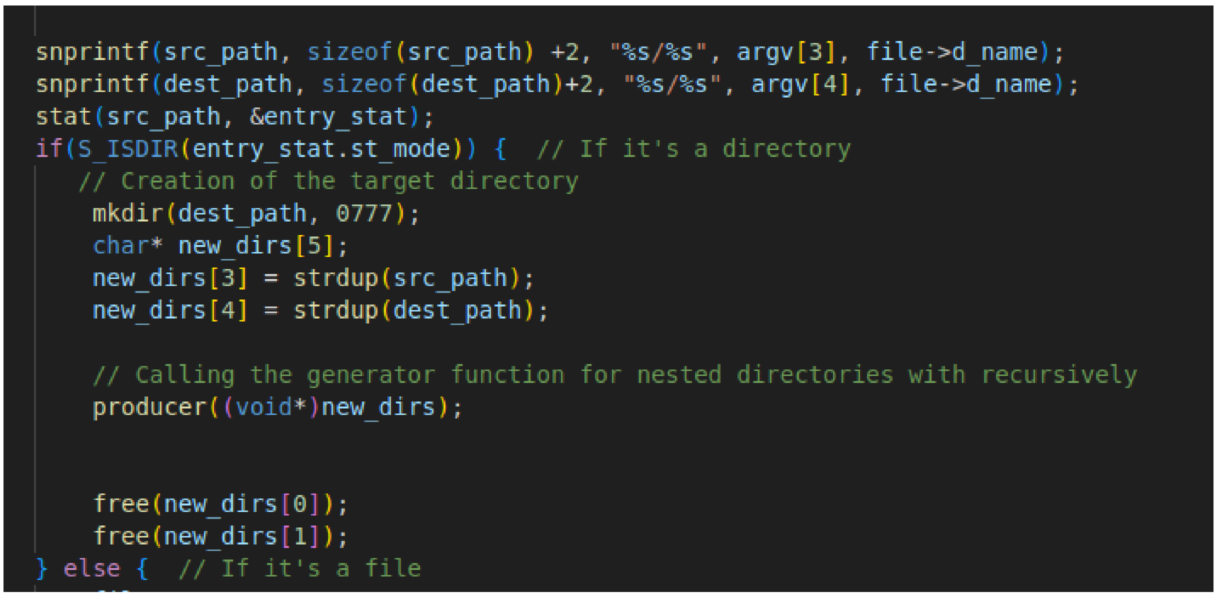


```
berkan@beko:~/Desktop/hw5$ ./pCp 20 2 /home/berkan/Desktop/hw5/read /home/berkan/Desktop/hw5/write
```

./pCp 20 2 /home/berkan/Desktop/hw5/read /home/berkan/Desktop/hw5/write

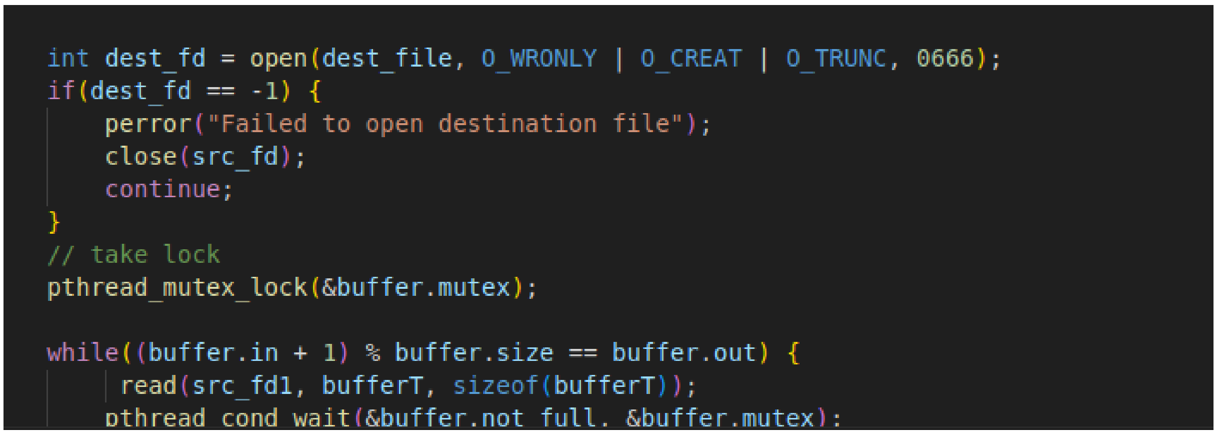
Requirements and Test Case

1) Producer creates files and directories in destination directory.
Code Snippet
Creates directory



```
snprintf(src_path, sizeof(src_path) +2, "%s/%s", argv[3], file->d_name);
snprintf(dest_path, sizeof(dest_path)+2, "%s/%s", argv[4], file->d_name);
stat(src_path, &entry_stat);
if(S_ISDIR(entry_stat.st_mode)) {  // If it's a directory
   // Creation of the target directory
    mkdir(dest_path, 0777);
    char* new_dirs[5];
    new_dirs[3] = strdup(src_path);
    new_dirs[4] = strdup(dest_path);

    // Calling the generator function for nested directories with recursively
    producer((void*)new_dirs);


    free(new_dirs[0]);
    free(new_dirs[1]);
} else {  // If it's a file
```

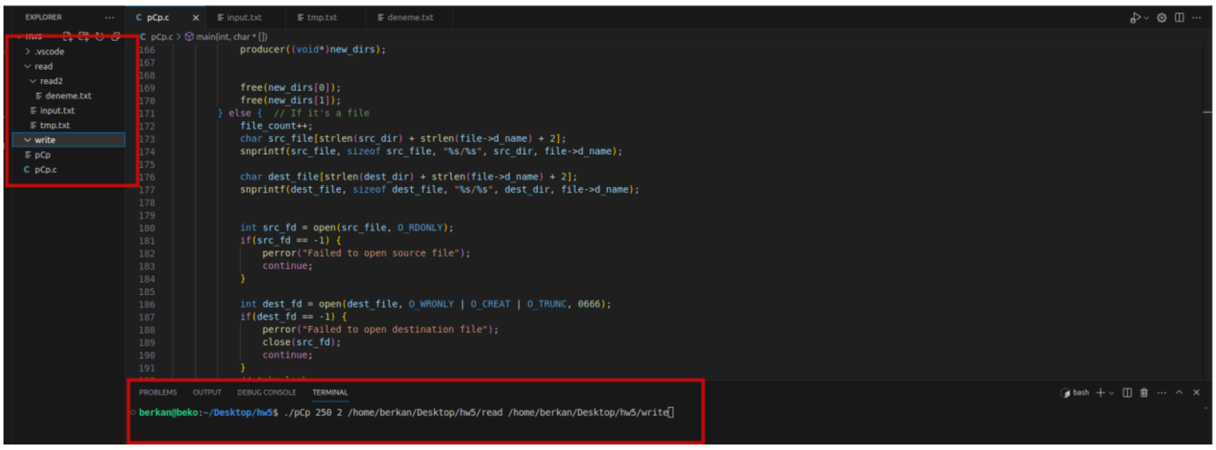
Create file



```
int dest_fd = open(dest_file, O_WRONLY | O_CREAT | O_TRUNC, 0666);
if(dest_fd == -1) {
    perror("Failed to open destination file");
    close(src_fd);
    continue;
}
// take lock
pthread_mutex_lock(&buffer.mutex);

while((buffer.in + 1) % buffer.size == buffer.out) {
     read(src_fd1, bufferT, sizeof(bufferT));
    pthread_cond_wait(&buffer.not_full, &buffer.mutex);
```

2) Copy destination path
Copy files and directories from source to destination directories.

Initial state write directory is empty

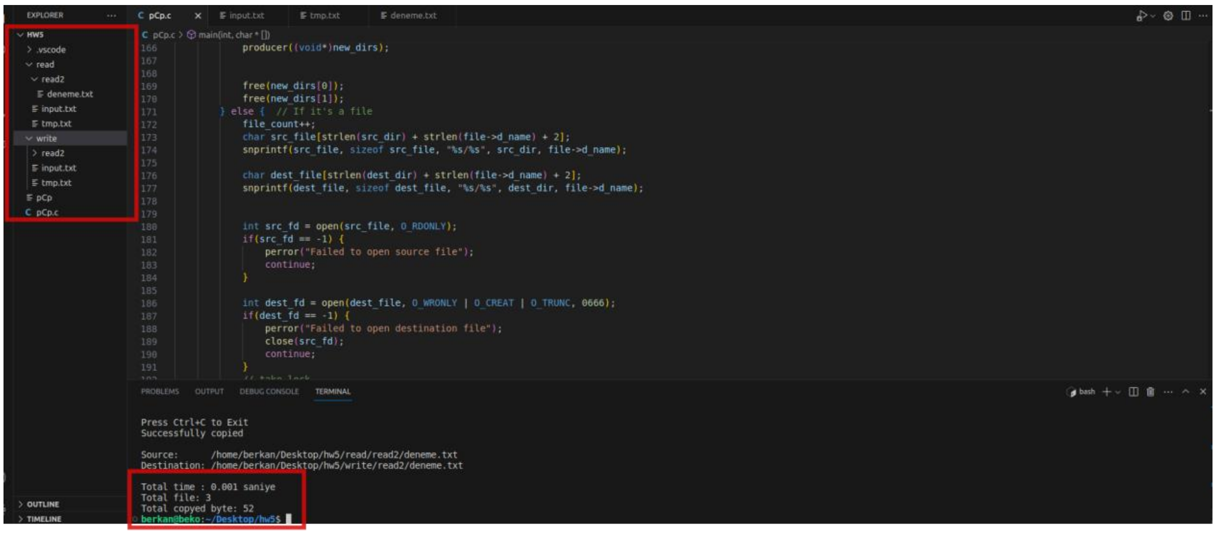


```c
            producer((void*)new_dirs);


            free(new_dirs[0]);
            free(new_dirs[1]);
        } else {  // If it's a file
            file_count++;
            char src_file[strlen(src_dir) + strlen(file->d_name) + 2];
            snprintf(src_file, sizeof src_file, "%s/%s", src_dir, file->d_name);

            char dest_file[strlen(dest_dir) + strlen(file->d_name) + 2];
            snprintf(dest_file, sizeof dest_file, "%s/%s", dest_dir, file->d_name);


            int src_fd = open(src_file, O_RDONLY);
            if(src_fd == -1) {
                perror("Failed to open source file");
                continue;
            }

            int dest_fd = open(dest_file, O_WRONLY | O_CREAT | O_TRUNC, 0666);
            if(dest_fd == -1) {
                perror("Failed to open destination file");
                close(src_fd);
                continue;
            }
```

```
berkan@beko:~/Desktop/hw5$ ./pCp 250 2 /home/berkan/Desktop/hw5/read /home/berkan/Desktop/hw5/write
```

Second state

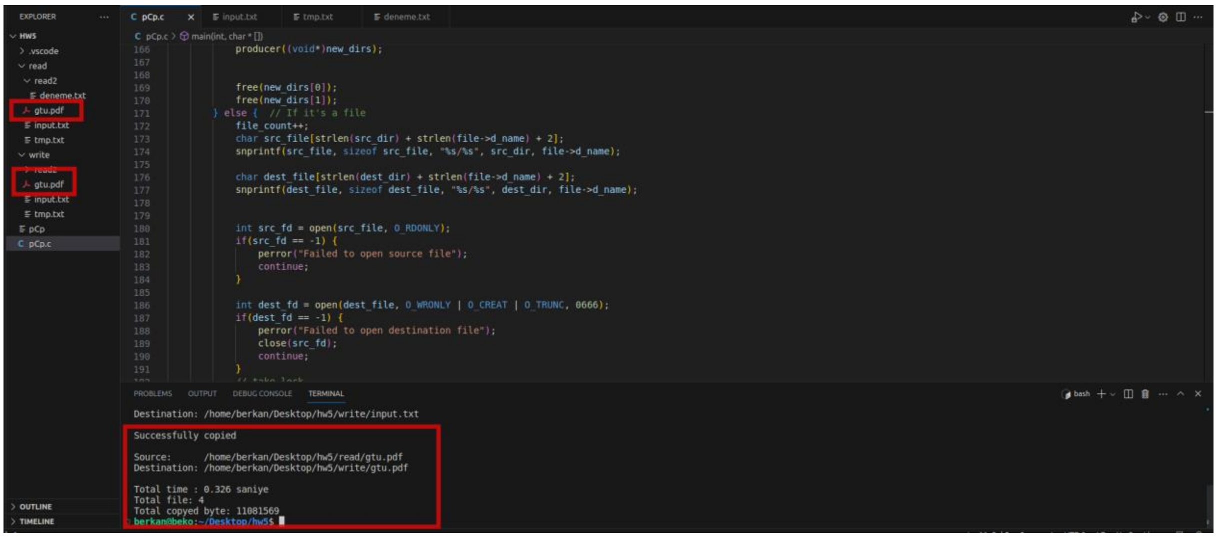
File and directory copyed



```c
            producer((void*)new_dirs);


            free(new_dirs[0]);
            free(new_dirs[1]);
        } else {  // If it's a file
            file_count++;
            char src_file[strlen(src_dir) + strlen(file->d_name) + 2];
            snprintf(src_file, sizeof src_file, "%s/%s", src_dir, file->d_name);

            char dest_file[strlen(dest_dir) + strlen(file->d_name) + 2];
            snprintf(dest_file, sizeof dest_file, "%s/%s", dest_dir, file->d_name);


            int src_fd = open(src_file, O_RDONLY);
            if(src_fd == -1) {
                perror("Failed to open source file");
                continue;
            }

            int dest_fd = open(dest_file, O_WRONLY | O_CREAT | O_TRUNC, 0666);
            if(dest_fd == -1) {
                perror("Failed to open destination file");
                close(src_fd);
                continue;
            }
```

```
Press Ctrl+C to Exit
Successfully copied

Source:      /home/berkan/Desktop/hw5/read/read2/deneme.txt
Destination: /home/berkan/Desktop/hw5/write/read2/deneme.txt

Total time : 0.001 saniye
Total file: 3
Total copyed byte: 52
berkan@beko:~/Desktop/hw5$
```

Copying a file of at least 10 megabytes



```c
                producer((void*)new_dirs);


                free(new_dirs[0]);
                free(new_dirs[1]);
            } else {  // If it's a file
                file_count++;
                char src_file[strlen(src_dir) + strlen(file->d_name) + 2];
                snprintf(src_file, sizeof src_file, "%s/%s", src_dir, file->d_name);

                char dest_file[strlen(dest_dir) + strlen(file->d_name) + 2];
                snprintf(dest_file, sizeof dest_file, "%s/%s", dest_dir, file->d_name);


                int src_fd = open(src_file, O_RDONLY);
                if(src_fd == -1) {
                    perror("Failed to open source file");
                    continue;
                }

                int dest_fd = open(dest_file, O_WRONLY | O_CREAT | O_TRUNC, 0666);
                if(dest_fd == -1) {
                    perror("Failed to open destination file");
                    close(src_fd);
                    continue;
                }
```

```
Destination: /home/berkan/Desktop/hw5/write/input.txt

Successfully copied

Source:      /home/berkan/Desktop/hw5/read/gtu.pdf
Destination: /home/berkan/Desktop/hw5/write/gtu.pdf

Total time : 0.326 saniye
Total file: 4
Total copyed byte: 11081569
berkan@beko:~/Desktop/hw5$
```