GEBZE TECHNİCAL UNIVERSITY

CSE344

System Programming

Homework 4 Report

1. How to Run?

Open the terminal and navigate to the source directory. Then, compile the program by typing "**make**" and execute it using "**./MWCp**". Once executed, the program will produce output as specified by its functionality. To clean up generated files, use "**make clean**", which will remove the executable file.

2. Components Used in the Code

1. Buffer Struct

A circular buffer is used to store file descriptor pairs. It includes mutex and condition variables to synchronize access.

```
typedef struct {
    int *buffer;                // Circular buffer to hold file descriptor pairs (source and destination)
    int head;                   // Head index for the circular buffer
    int tail;                   // Tail index for the circular buffer
    int max_size;               // Maximum number of file descriptor pairs the buffer can hold
    int count;                  // Current number of file descriptor pairs in the buffer
    int num_regular_files;      // Number of regular files processed
    int num_directories;        // Number of directories processed
    int num_fifo;               // Number of FIFOs processed
    size_t total_bytes_copied;  // Total number of bytes copied
    pthread_mutex_t mutex;      // Mutex for synchronizing access to the buffer
    pthread_cond_t not_full;    // Condition variable to signal when the buffer is not full
    pthread_cond_t not_empty;   // Condition variable to signal when the buffer is not empty
    int done;                   // Flag to indicate when the manager is done processing directories
} buffer_t;
```

2. Manager Thread

Recursively processes the source directory, adding file descriptors to the buffer for the worker threads to copy, and uses condition variables to manage buffer access.

```
process_directory(src_dir, dest_dir)

mark buffer as done

signal all worker threads

exit
```

3. Manager Arguments Struct

Defines the arguments passed to the manager thread, including the buffer, source directory, and destination directory.

```
typedef struct
{
    buffer_t *buffer;
    char *src_dir;
    char *dest_dir;
} manager_args_t;
```

4. Worker Thread

Continuously retrieves file descriptors from the buffer to copy data from source to destination files, using condition variables to synchronize with the manager thread.

```
while true:
        lock buffer mutex

        wait until buffer is not empty or done

        if buffer is empty and done, exit

        get file descriptors from buffer

        signal buffer not full condition

        unlock buffer mutex

        copy data from source to destination file descriptor

        close file descriptors

        increment total bytes copied

        counter

exit.
```

5. Process Directory (Called in Manager Thread)

Traverses the source directory recursively, creating corresponding directories, files, and FIFOs in the destination, and updates the buffer with file descriptors for regular files.

```
open source directory
    for each entry in source directory:
        if entry is a directory:
            create corresponding directory in destination
            increment directory counter
            recursively call process_directory
        else if entry is a regular file:
            open source and destination file descriptors
            lock buffer mutex
            wait until buffer is not full
            add file descriptors to buffer
            signal buffer not empty condition
            unlock buffer mutex
            increment regular file counter
        else if entry is a FIFO:
            create corresponding FIFO in destination
            increment FIFO counter
close source directory.
```

6. Checking File Descriptor Limit Function

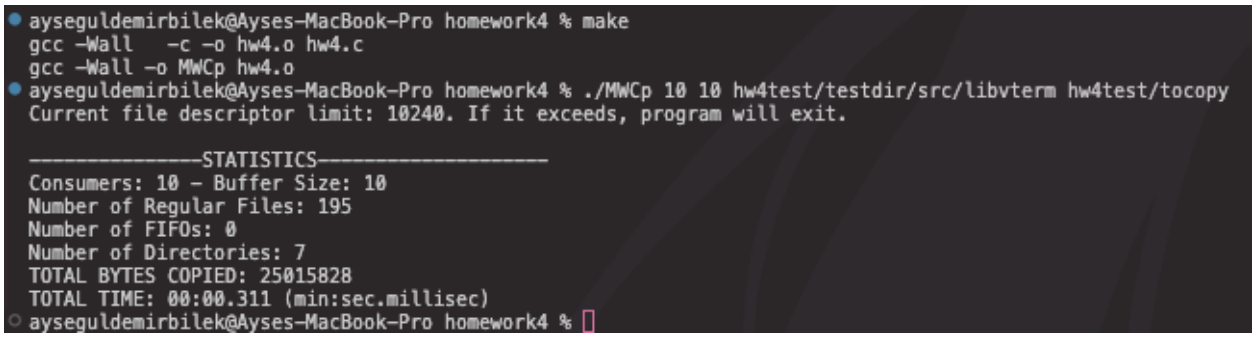
Retrieves and displays the current file descriptor limit and if this limit is exceeded program simply informs and exits after cleaning up resources.

```c
void check_fd_limit() {
    struct rlimit rl;
    if (getrlimit(RLIMIT_NOFILE, &rl) == -1) {
        perror("getrlimit");
        exit(EXIT_FAILURE); }
    char msg[100];
    snprintf(msg, sizeof(msg), "Current file descriptor limit: %llu. If it exceeds, program will exit.\n", (unsigned long long)rl.rlim_cur);
    safe_print(msg);
}
```

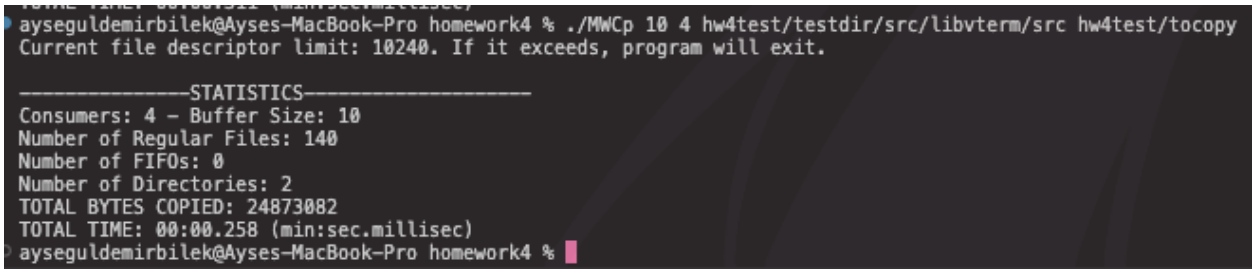
3. Output

Overall look, the main function initializes by parsing command-line arguments, checking the file descriptor limit, and creating the buffer. It sets up a signal handler for SIGINT, starts a timer, and creates both manager and worker threads. After waiting for all threads to complete, the timer stops, statistics are printed, and resources are cleaned up before the program exits. Here's the mentioned tests on the homework pdf:

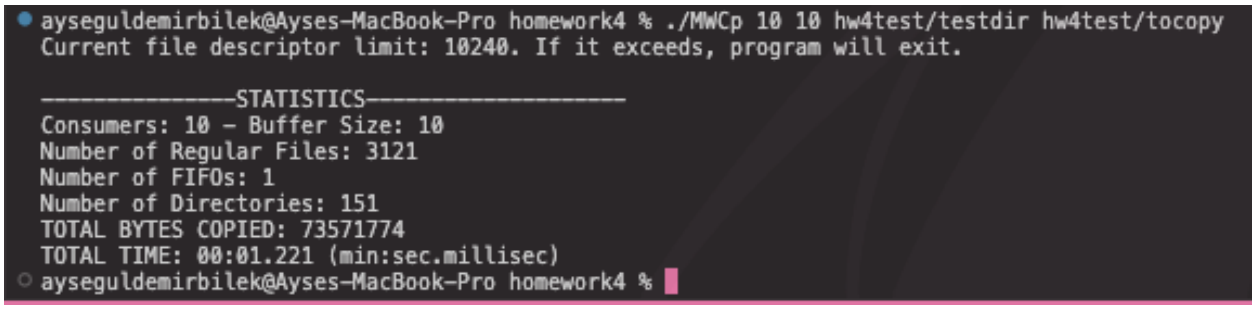
1.



2.



3.



Note: In the last part, I've created a fifo myself to test the testdir directory with the fifo scenario. It isn't an error.