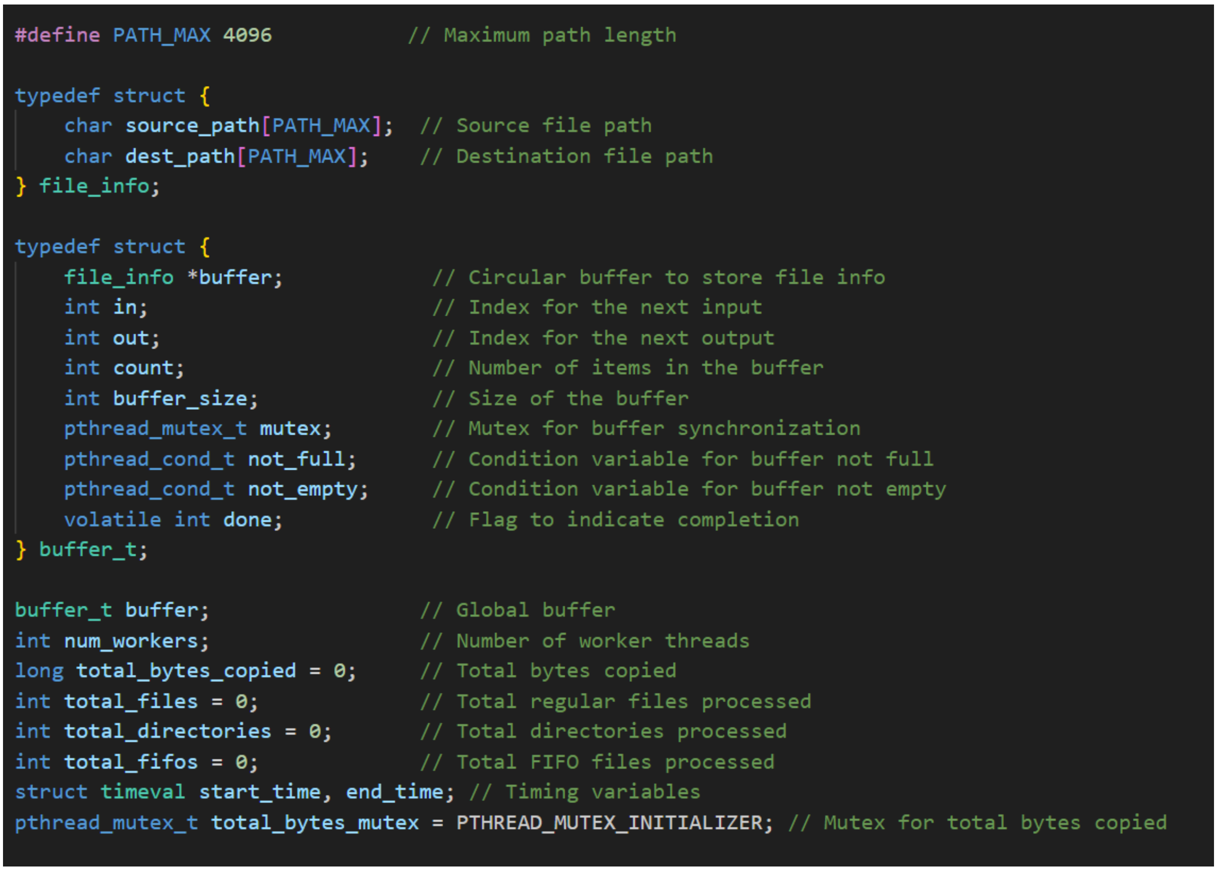
Eren Torlak 210104004090 REPORT HW4 CSE344

I have used condition variables. This hw4 is same as hw5 except barrier doesn’t exist in hw4.

I have created two struct :

Buffer struct stores mutex for critical region activities. And I hold two condition variables.

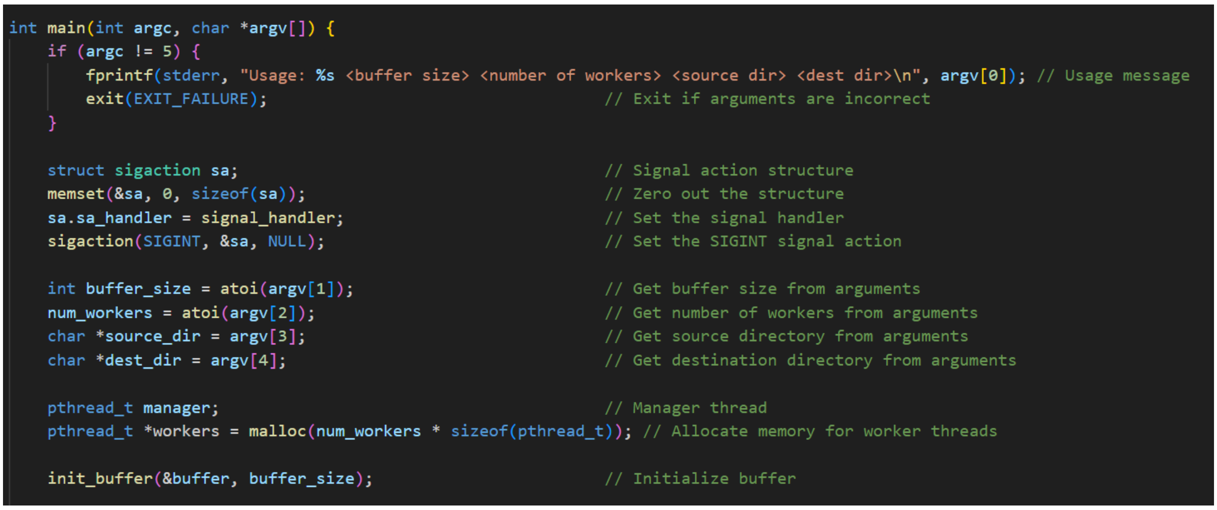


```
#define PATH_MAX 4096           // Maximum path length

typedef struct {
    char source_path[PATH_MAX];  // Source file path
    char dest_path[PATH_MAX];    // Destination file path
} file_info;

typedef struct {
    file_info *buffer;              // Circular buffer to store file info
    int in;                         // Index for the next input
    int out;                        // Index for the next output
    int count;                      // Number of items in the buffer
    int buffer_size;                // Size of the buffer
    pthread_mutex_t mutex;          // Mutex for buffer synchronization
    pthread_cond_t not_full;        // Condition variable for buffer not full
    pthread_cond_t not_empty;       // Condition variable for buffer not empty
    volatile int done;              // Flag to indicate completion
} buffer_t;

buffer_t buffer;                  // Global buffer
int num_workers;                  // Number of worker threads
long total_bytes_copied = 0;      // Total bytes copied
int total_files = 0;              // Total regular files processed
int total_directories = 0;        // Total directories processed
int total_fifos = 0;              // Total FIFO files processed
struct timeval start_time, end_time; // Timing variables
pthread_mutex_t total_bytes_mutex = PTHREAD_MUTEX_INITIALIZER; // Mutex for total bytes copied
```

I found out mutex is needed for total bytes counting.

Eren Torlak 210104004090 REPORT HW4 CSE344

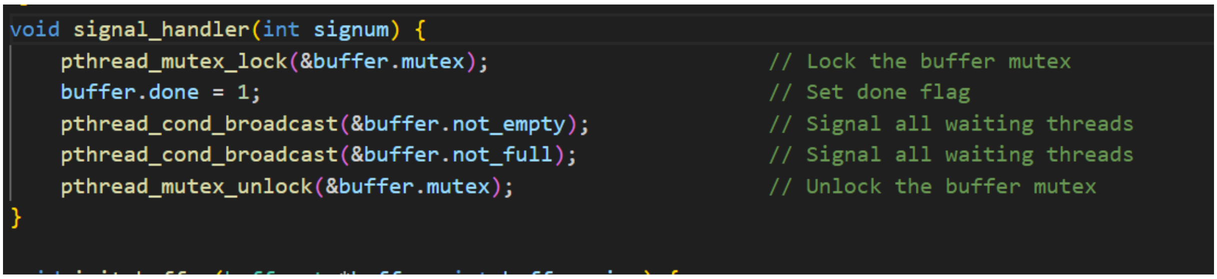
INIT PART :



```c
int main(int argc, char *argv[]) {
    if (argc != 5) {
        fprintf(stderr, "Usage: %s <buffer size> <number of workers> <source dir> <dest dir>\n", argv[0]); // Usage message
        exit(EXIT_FAILURE);                                       // Exit if arguments are incorrect
    }

    struct sigaction sa;                                          // Signal action structure
    memset(&sa, 0, sizeof(sa));                                   // Zero out the structure
    sa.sa_handler = signal_handler;                               // Set the signal handler
    sigaction(SIGINT, &sa, NULL);                                 // Set the SIGINT signal action

    int buffer_size = atoi(argv[1]);                              // Get buffer size from arguments
    num_workers = atoi(argv[2]);                                  // Get number of workers from arguments
    char *source_dir = argv[3];                                   // Get source directory from arguments
    char *dest_dir = argv[4];                                     // Get destination directory from arguments

    pthread_t manager;                                            // Manager thread
    pthread_t *workers = malloc(num_workers * sizeof(pthread_t)); // Allocate memory for worker threads

    init_buffer(&buffer, buffer_size);                            // Initialize buffer
```

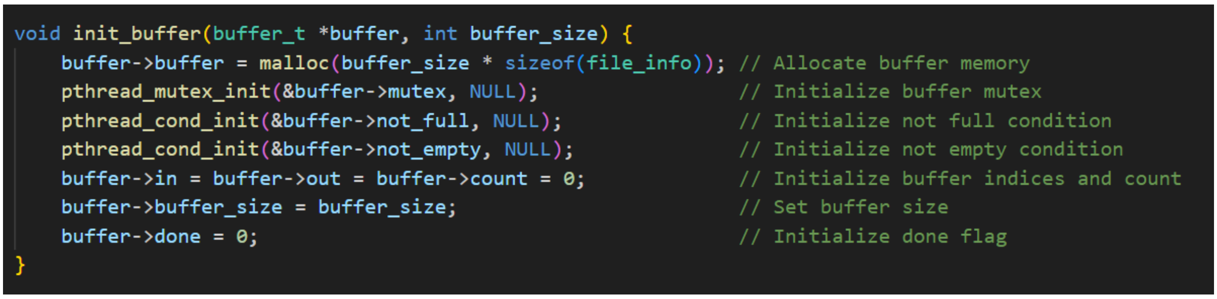
Memory is allocated according to user input for workers.

CTRL + C signal is controlled.



```c
void signal_handler(int signum) {
    pthread_mutex_lock(&buffer.mutex);                      // Lock the buffer mutex
    buffer.done = 1;                                        // Set done flag
    pthread_cond_broadcast(&buffer.not_empty);              // Signal all waiting threads
    pthread_cond_broadcast(&buffer.not_full);               // Signal all waiting threads
    pthread_mutex_unlock(&buffer.mutex);                    // Unlock the buffer mutex
}
```

Buffer initiliazed like this:

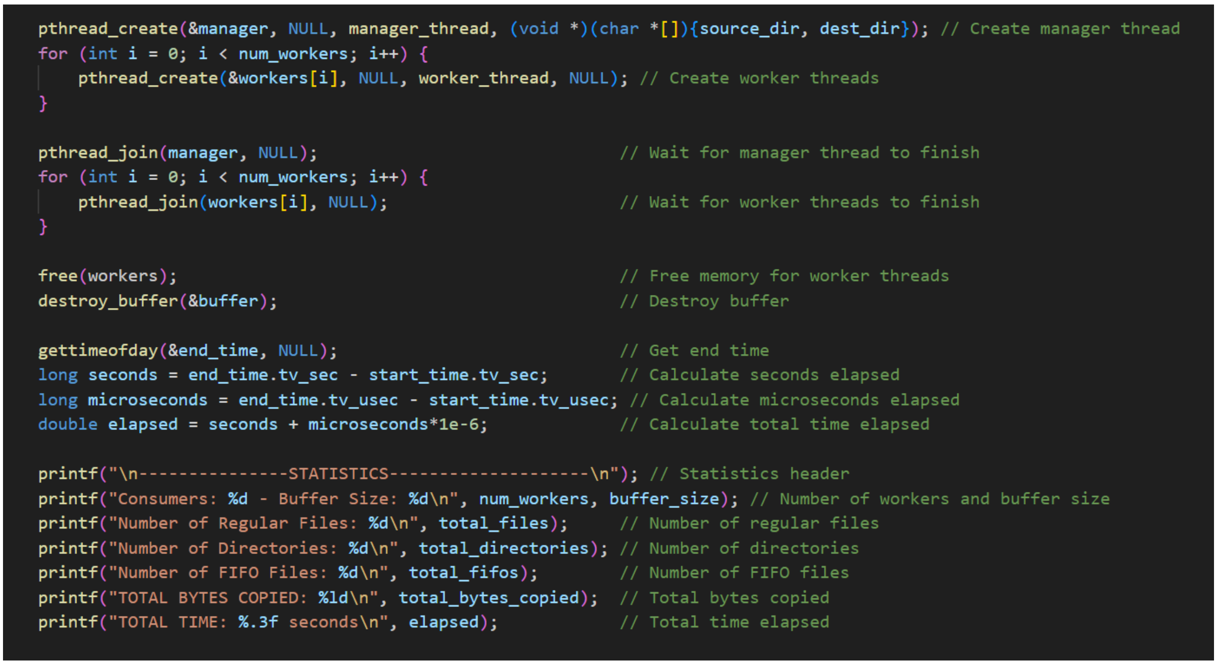


```c
void init_buffer(buffer_t *buffer, int buffer_size) {
    buffer->buffer = malloc(buffer_size * sizeof(file_info)); // Allocate buffer memory
    pthread_mutex_init(&buffer->mutex, NULL);                 // Initialize buffer mutex
    pthread_cond_init(&buffer->not_full, NULL);               // Initialize not full condition
    pthread_cond_init(&buffer->not_empty, NULL);              // Initialize not empty condition
    buffer->in = buffer->out = buffer->count = 0;             // Initialize buffer indices and count
    buffer->buffer_size = buffer_size;                        // Set buffer size
    buffer->done = 0;                                         // Initialize done flag
}
```

Eren Torlak 210104004090 REPORT HW4 CSE344

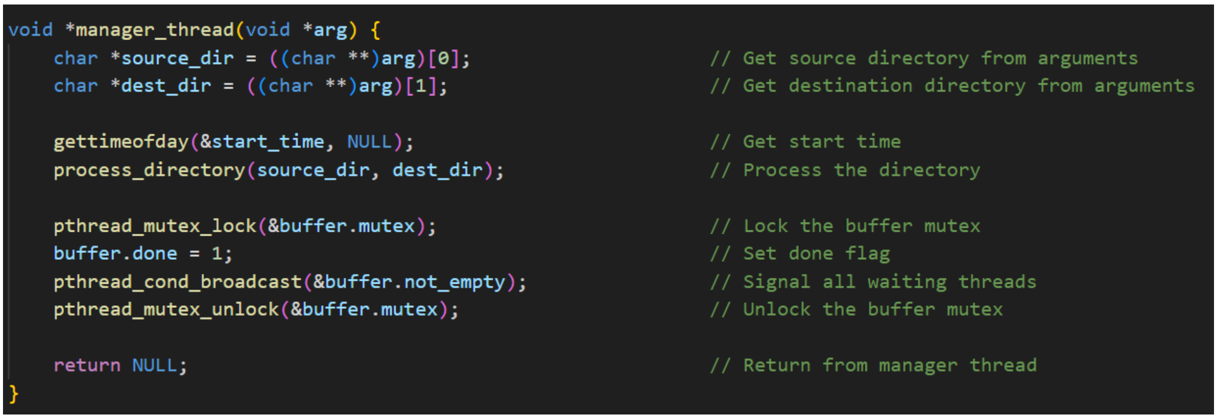
After initilization :

- Manager thread is created then each worker thread is created.



```c
pthread_create(&manager, NULL, manager_thread, (void *)(char *[]){source_dir, dest_dir}); // Create manager thread
for (int i = 0; i < num_workers; i++) {
    pthread_create(&workers[i], NULL, worker_thread, NULL); // Create worker threads
}

pthread_join(manager, NULL);                              // Wait for manager thread to finish
for (int i = 0; i < num_workers; i++) {
    pthread_join(workers[i], NULL);                       // Wait for worker threads to finish
}

free(workers);                                            // Free memory for worker threads
destroy_buffer(&buffer);                                  // Destroy buffer

gettimeofday(&end_time, NULL);                            // Get end time
long seconds = end_time.tv_sec - start_time.tv_sec;       // Calculate seconds elapsed
long microseconds = end_time.tv_usec - start_time.tv_usec; // Calculate microseconds elapsed
double elapsed = seconds + microseconds*1e-6;             // Calculate total time elapsed

printf("\n--------------STATISTICS-------------------\n"); // Statistics header
printf("Consumers: %d - Buffer Size: %d\n", num_workers, buffer_size); // Number of workers and buffer size
printf("Number of Regular Files: %d\n", total_files);     // Number of regular files
printf("Number of Directories: %d\n", total_directories); // Number of directories
printf("Number of FIFO Files: %d\n", total_fifos);        // Number of FIFO files
printf("TOTAL BYTES COPIED: %ld\n", total_bytes_copied);  // Total bytes copied
printf("TOTAL TIME: %.3f seconds\n", elapsed);            // Total time elapsed
```

Manager thread :

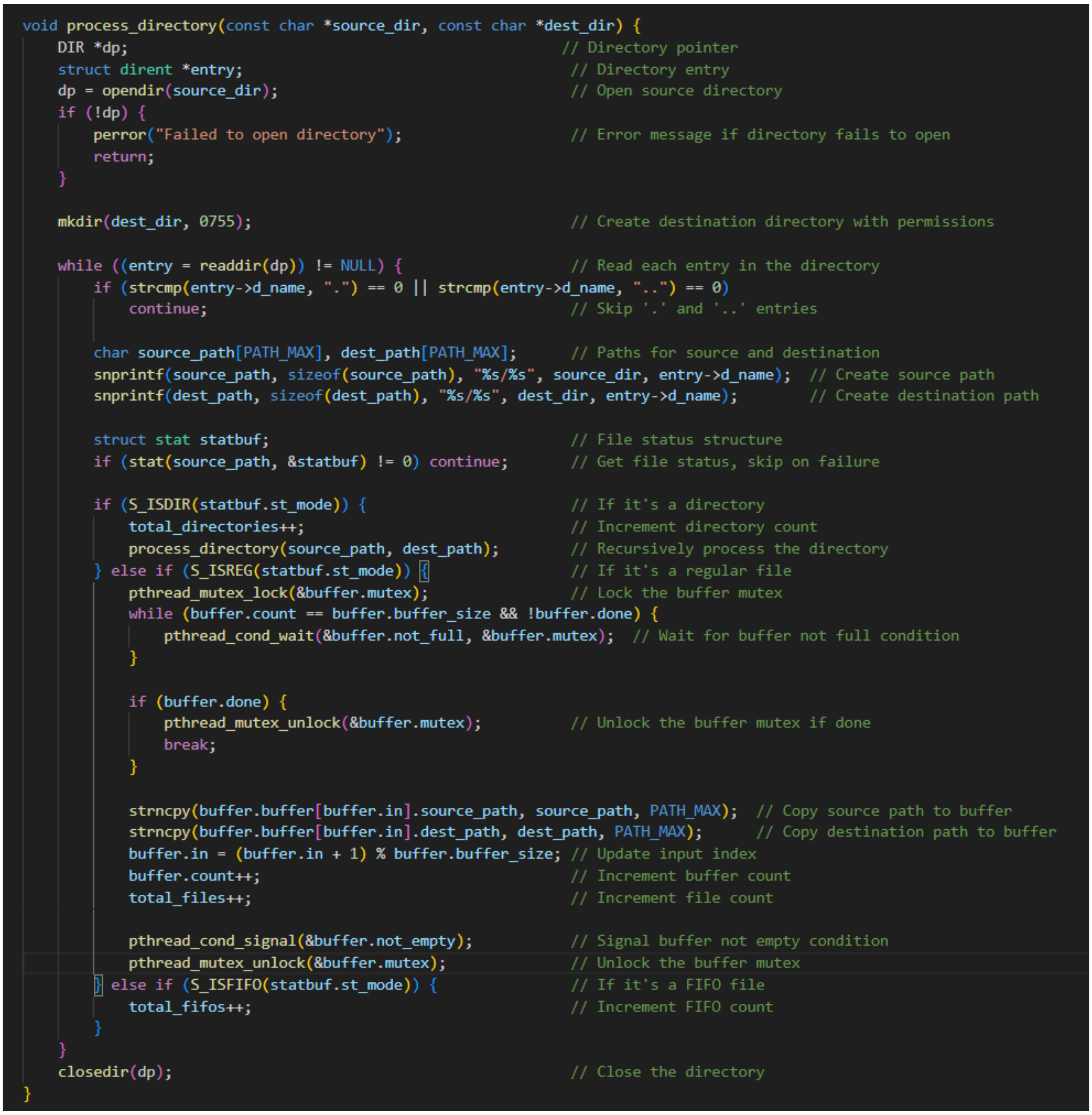


```c
void *manager_thread(void *arg) {
    char *source_dir = ((char **)arg)[0];                    // Get source directory from arguments
    char *dest_dir = ((char **)arg)[1];                      // Get destination directory from arguments

    gettimeofday(&start_time, NULL);                         // Get start time
    process_directory(source_dir, dest_dir);                 // Process the directory

    pthread_mutex_lock(&buffer.mutex);                       // Lock the buffer mutex
    buffer.done = 1;                                         // Set done flag
    pthread_cond_broadcast(&buffer.not_empty);               // Signal all waiting threads
    pthread_mutex_unlock(&buffer.mutex);                     // Unlock the buffer mutex

    return NULL;                                             // Return from manager thread
}
```

It gets arguments from user input.

Process directory recursively works coordinated with worker threads.

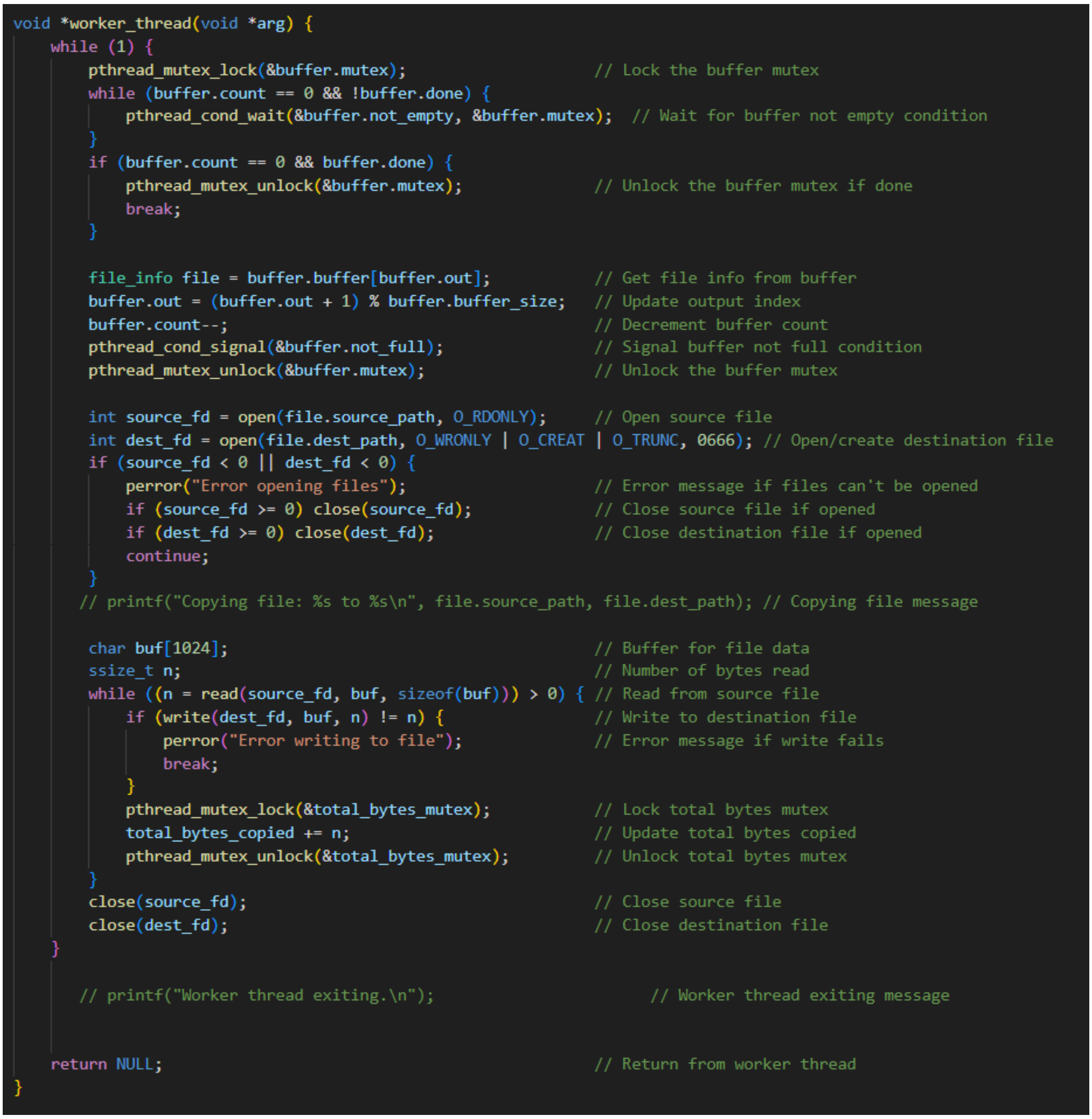
Process directory function explanation is comments (pdf says short report) :



```
void process_directory(const char *source_dir, const char *dest_dir) {
    DIR *dp;                                                 // Directory pointer
    struct dirent *entry;                                     // Directory entry
    dp = opendir(source_dir);                                 // Open source directory
    if (!dp) {
        perror("Failed to open directory");                   // Error message if directory fails to open
        return;
    }

    mkdir(dest_dir, 0755);                                    // Create destination directory with permissions

    while ((entry = readdir(dp)) != NULL) {                   // Read each entry in the directory
        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") == 0)
            continue;                                         // Skip '.' and '..' entries

        char source_path[PATH_MAX], dest_path[PATH_MAX];      // Paths for source and destination
        snprintf(source_path, sizeof(source_path), "%s/%s", source_dir, entry->d_name);  // Create source path
        snprintf(dest_path, sizeof(dest_path), "%s/%s", dest_dir, entry->d_name);        // Create destination path

        struct stat statbuf;                                  // File status structure
        if (stat(source_path, &statbuf) != 0) continue;       // Get file status, skip on failure

        if (S_ISDIR(statbuf.st_mode)) {                       // If it's a directory
            total_directories++;                              // Increment directory count
            process_directory(source_path, dest_path);        // Recursively process the directory
        } else if (S_ISREG(statbuf.st_mode)) {                // If it's a regular file
            pthread_mutex_lock(&buffer.mutex);                // Lock the buffer mutex
            while (buffer.count == buffer.buffer_size && !buffer.done) {
                pthread_cond_wait(&buffer.not_full, &buffer.mutex);  // Wait for buffer not full condition
            }

            if (buffer.done) {
                pthread_mutex_unlock(&buffer.mutex);          // Unlock the buffer mutex if done
                break;
            }

            strncpy(buffer.buffer[buffer.in].source_path, source_path, PATH_MAX);  // Copy source path to buffer
            strncpy(buffer.buffer[buffer.in].dest_path, dest_path, PATH_MAX);      // Copy destination path to buffer
            buffer.in = (buffer.in + 1) % buffer.buffer_size; // Update input index
            buffer.count++;                                   // Increment buffer count
            total_files++;                                    // Increment file count

            pthread_cond_signal(&buffer.not_empty);           // Signal buffer not empty condition
            pthread_mutex_unlock(&buffer.mutex);              // Unlock the buffer mutex
        } else if (S_ISFIFO(statbuf.st_mode)) {               // If it's a FIFO file
            total_fifos++;                                    // Increment FIFO count
        }
    }
    closedir(dp);                                             // Close the directory
}
```

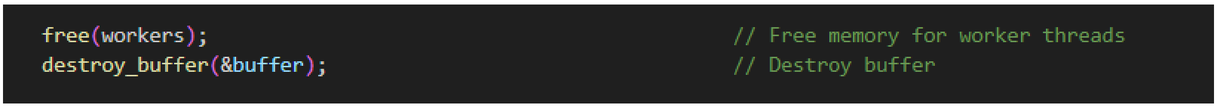
Worker threads :

It does read from source and writes to destination . it is sycned with other threads. There arent any busy waiting. Condition variables are used. Also mutexes are used for entering critical region.

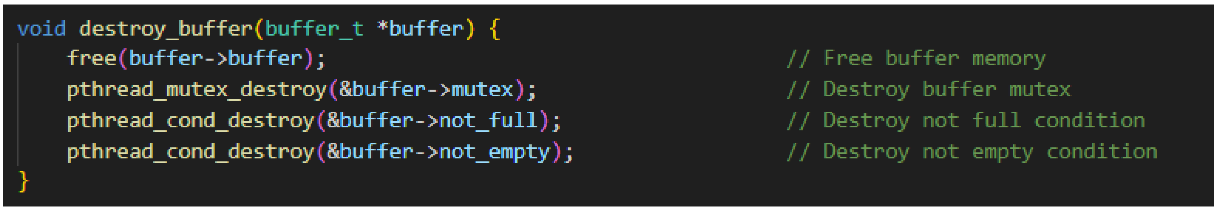


```c
void *worker_thread(void *arg) {
    while (1) {
        pthread_mutex_lock(&buffer.mutex);                    // Lock the buffer mutex
        while (buffer.count == 0 && !buffer.done) {
            pthread_cond_wait(&buffer.not_empty, &buffer.mutex);  // Wait for buffer not empty condition
        }
        if (buffer.count == 0 && buffer.done) {
            pthread_mutex_unlock(&buffer.mutex);              // Unlock the buffer mutex if done
            break;
        }

        file_info file = buffer.buffer[buffer.out];           // Get file info from buffer
        buffer.out = (buffer.out + 1) % buffer.buffer_size;   // Update output index
        buffer.count--;                                       // Decrement buffer count
        pthread_cond_signal(&buffer.not_full);                // Signal buffer not full condition
        pthread_mutex_unlock(&buffer.mutex);                  // Unlock the buffer mutex

        int source_fd = open(file.source_path, O_RDONLY);     // Open source file
        int dest_fd = open(file.dest_path, O_WRONLY | O_CREAT | O_TRUNC, 0666); // Open/create destination file
        if (source_fd < 0 || dest_fd < 0) {
            perror("Error opening files");                    // Error message if files can't be opened
            if (source_fd >= 0) close(source_fd);             // Close source file if opened
            if (dest_fd >= 0) close(dest_fd);                 // Close destination file if opened
            continue;
        }
       // printf("Copying file: %s to %s\n", file.source_path, file.dest_path); // Copying file message

        char buf[1024];                                       // Buffer for file data
        ssize_t n;                                            // Number of bytes read
        while ((n = read(source_fd, buf, sizeof(buf))) > 0) { // Read from source file
            if (write(dest_fd, buf, n) != n) {                // Write to destination file
                perror("Error writing to file");              // Error message if write fails
                break;
            }
            pthread_mutex_lock(&total_bytes_mutex);           // Lock total bytes mutex
            total_bytes_copied += n;                          // Update total bytes copied
            pthread_mutex_unlock(&total_bytes_mutex);         // Unlock total bytes mutex
        }
        close(source_fd);                                     // Close source file
        close(dest_fd);                                       // Close destination file
    }

       // printf("Worker thread exiting.\n");                       // Worker thread exiting message


    return NULL;                                              // Return from worker thread
}
```

In the end :



```c
  free(workers);                                            // Free memory for worker threads
  destroy_buffer(&buffer);                                  // Destroy buffer
```

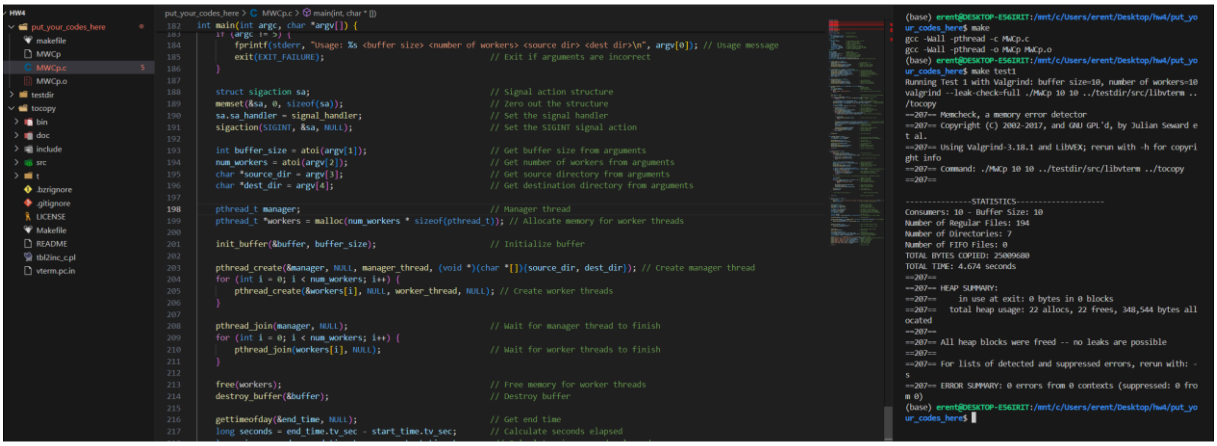


```c
void destroy_buffer(buffer_t *buffer) {
    free(buffer->buffer);                                     // Free buffer memory
    pthread_mutex_destroy(&buffer->mutex);                    // Destroy buffer mutex
    pthread_cond_destroy(&buffer->not_full);                  // Destroy not full condition
    pthread_cond_destroy(&buffer->not_empty);                 // Destroy not empty condition
}
```

Eren Torlak 210104004090 REPORT HW4 CSE344

Test 1 :

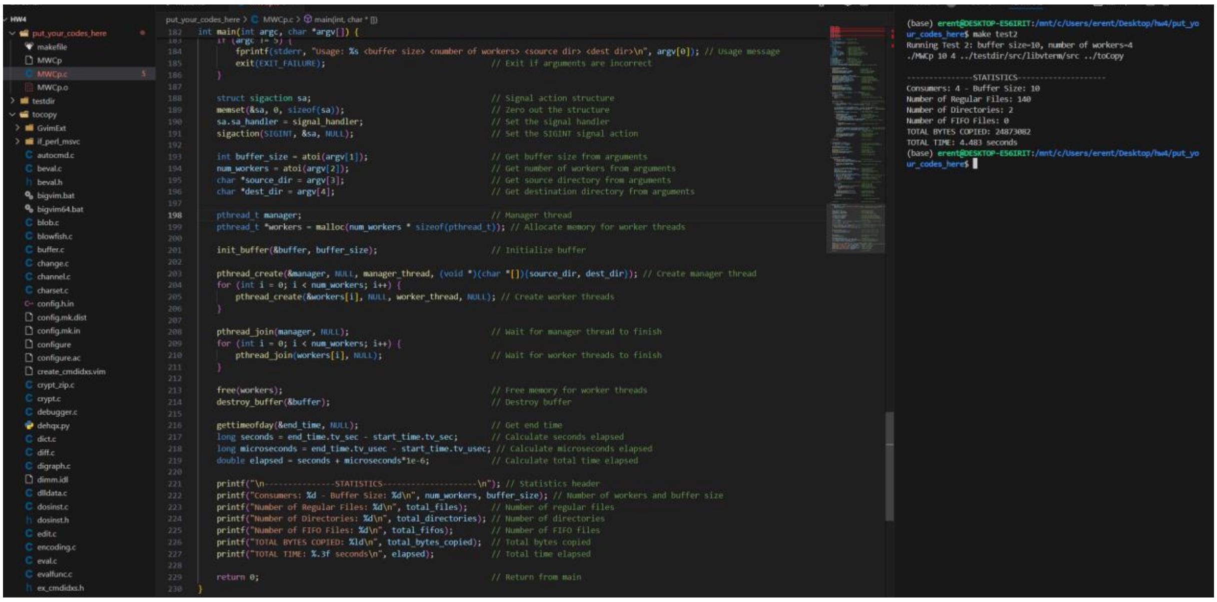
No memory leak.



```c
int main(int argc, char *argv[]) {
    if (argc != 5) {
        fprintf(stderr, "Usage: %s <buffer size> <number of workers> <source dir> <dest dir>\n", argv[0]); // Usage message
        exit(EXIT_FAILURE);                                           // Exit if arguments are incorrect
    }

    struct sigaction sa;                                              // Signal action structure
    memset(&sa, 0, sizeof(sa));                                       // Zero out the structure
    sa.sa_handler = signal_handler;                                   // Set the signal handler
    sigaction(SIGINT, &sa, NULL);                                     // Set the SIGINT signal action

    int buffer_size = atoi(argv[1]);                                  // Get buffer size from arguments
    num_workers = atoi(argv[2]);                                      // Get number of workers from arguments
    char *source_dir = argv[3];                                       // Get source directory from arguments
    char *dest_dir = argv[4];                                         // Get destination directory from arguments

    pthread_t manager;                                                // Manager thread
    pthread_t *workers = malloc(num_workers * sizeof(pthread_t)); // Allocate memory for worker threads

    init_buffer(&buffer, buffer_size);                                // Initialize buffer

    pthread_create(&manager, NULL, manager_thread, (void *)(char *[]){source_dir, dest_dir}); // Create manager thread
    for (int i = 0; i < num_workers; i++) {
        pthread_create(&workers[i], NULL, worker_thread, NULL); // Create worker threads
    }

    pthread_join(manager, NULL);                                      // Wait for manager thread to finish
    for (int i = 0; i < num_workers; i++) {
        pthread_join(workers[i], NULL);                               // Wait for worker threads to finish
    }

    free(workers);                                                    // Free memory for worker threads
    destroy_buffer(&buffer);                                          // Destroy buffer

    gettimeofday(&end_time, NULL);                                    // Get end time
    long seconds = end_time.tv_sec - start_time.tv_sec;               // Calculate seconds elapsed
```

```
(base) erent@DESKTOP-E56IRIT:/mnt/c/Users/erent/Desktop/hw4/put_yo
ur_codes_here$ make
gcc -Wall -pthread -c MWCp.c
gcc -Wall -pthread -o MWCp MWCp.o
(base) erent@DESKTOP-E56IRIT:/mnt/c/Users/erent/Desktop/hw4/put_yo
ur_codes_here$ make test1
Running Test 1 with Valgrind: buffer size=10, number of workers=10
valgrind --leak-check=full ./MWCp 10 10 ../testdir/src/libvterm ..
/tocopy
==207== Memcheck, a memory error detector
==207== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward e
t al.
==207== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyri
ght info
==207== Command: ./MWCp 10 10 ../testdir/src/libvterm ../tocopy
==207==

---------------STATISTICS--------------------
Consumers: 10 - Buffer Size: 10
Number of Regular Files: 194
Number of Directories: 7
Number of FIFO Files: 0
TOTAL BYTES COPIED: 25009680
TOTAL TIME: 4.674 seconds
==207==
==207== HEAP SUMMARY:
==207==     in use at exit: 0 bytes in 0 blocks
==207==   total heap usage: 22 allocs, 22 frees, 348,544 bytes all
ocated
==207==
==207== All heap blocks were freed -- no leaks are possible
==207==
==207== For lists of detected and suppressed errors, rerun with: -
s
==207== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 fro
m 0)
(base) erent@DESKTOP-E56IRIT:/mnt/c/Users/erent/Desktop/hw4/put_yo
ur_codes_here$
```

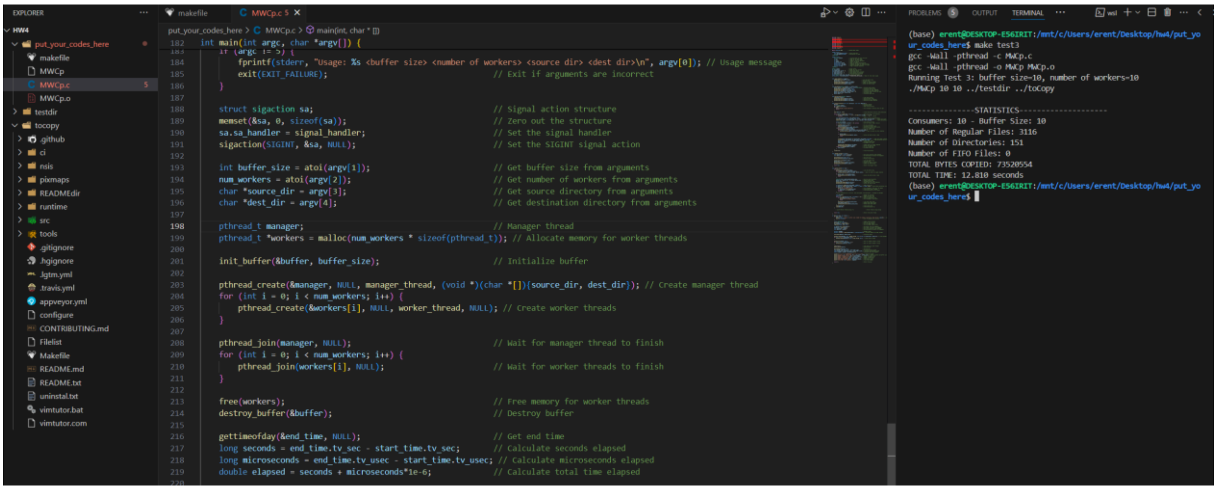
Test 2 :



```c
int main(int argc, char *argv[]) {
    if (argc != 5) {
        fprintf(stderr, "Usage: %s <buffer size> <number of workers> <source dir> <dest dir>\n", argv[0]); // Usage message
        exit(EXIT_FAILURE);                                           // Exit if arguments are incorrect
    }

    struct sigaction sa;                                              // Signal action structure
    memset(&sa, 0, sizeof(sa));                                       // Zero out the structure
    sa.sa_handler = signal_handler;                                   // Set the signal handler
    sigaction(SIGINT, &sa, NULL);                                     // Set the SIGINT signal action

    int buffer_size = atoi(argv[1]);                                  // Get buffer size from arguments
    num_workers = atoi(argv[2]);                                      // Get number of workers from arguments
    char *source_dir = argv[3];                                       // Get source directory from arguments
    char *dest_dir = argv[4];                                         // Get destination directory from arguments

    pthread_t manager;                                                // Manager thread
    pthread_t *workers = malloc(num_workers * sizeof(pthread_t)); // Allocate memory for worker threads

    init_buffer(&buffer, buffer_size);                                // Initialize buffer

    pthread_create(&manager, NULL, manager_thread, (void *)(char *[]){source_dir, dest_dir}); // Create manager thread
    for (int i = 0; i < num_workers; i++) {
        pthread_create(&workers[i], NULL, worker_thread, NULL); // Create worker threads
    }

    pthread_join(manager, NULL);                                      // Wait for manager thread to finish
    for (int i = 0; i < num_workers; i++) {
        pthread_join(workers[i], NULL);                               // Wait for worker threads to finish
    }

    free(workers);                                                    // Free memory for worker threads
    destroy_buffer(&buffer);                                          // Destroy buffer

    gettimeofday(&end_time, NULL);                                    // Get end time
    long seconds = end_time.tv_sec - start_time.tv_sec;               // Calculate seconds elapsed
    long microseconds = end_time.tv_usec - start_time.tv_usec; // Calculate microseconds elapsed
    double elapsed = seconds + microseconds*1e-6;                     // Calculate total time elapsed

    printf("\n---------------STATISTICS--------------------\n"); // Statistics header
    printf("Consumers: %d - Buffer Size: %d\n", num_workers, buffer_size); // Number of workers and buffer size
    printf("Number of Regular Files: %d\n", total_files);        // Number of regular files
    printf("Number of Directories: %d\n", total_directories); // Number of directories
    printf("Number of FIFO Files: %d\n", total_fifos);           // Number of FIFO files
    printf("TOTAL BYTES COPIED: %ld\n", total_bytes_copied);  // Total bytes copied
    printf("TOTAL TIME: %.3f seconds\n", elapsed);               // Total time elapsed

    return 0;                                                         // Return from main
}
```

```
(base) erent@DESKTOP-E56IRIT:/mnt/c/Users/erent/Desktop/hw4/put_yo
ur_codes_here$ make test2
Running Test 2: buffer size=10, number of workers=4
./MWCp 10 4 ../testdir/src/libvterm/src ../toCopy

---------------STATISTICS--------------------
Consumers: 4 - Buffer Size: 10
Number of Regular Files: 140
Number of Directories: 2
Number of FIFO Files: 0
TOTAL BYTES COPIED: 24873082
TOTAL TIME: 4.483 seconds
(base) erent@DESKTOP-E56IRIT:/mnt/c/Users/erent/Desktop/hw4/put_yo
ur_codes_here$
```

Test 3 :



```c
int main(int argc, char *argv[]) {
    if (argc != 5) {
        fprintf(stderr, "Usage: %s <buffer size> <number of workers> <source dir> <dest dir>\n", argv[0]); // Usage message
        exit(EXIT_FAILURE);                                           // Exit if arguments are incorrect
    }

    struct sigaction sa;                                              // Signal action structure
    memset(&sa, 0, sizeof(sa));                                       // Zero out the structure
    sa.sa_handler = signal_handler;                                   // Set the signal handler
    sigaction(SIGINT, &sa, NULL);                                     // Set the SIGINT signal action

    int buffer_size = atoi(argv[1]);                                  // Get buffer size from arguments
    num_workers = atoi(argv[2]);                                      // Get number of workers from arguments
    char *source_dir = argv[3];                                       // Get source directory from arguments
    char *dest_dir = argv[4];                                         // Get destination directory from arguments

    pthread_t manager;                                                // Manager thread
    pthread_t *workers = malloc(num_workers * sizeof(pthread_t)); // Allocate memory for worker threads

    init_buffer(&buffer, buffer_size);                                // Initialize buffer

    pthread_create(&manager, NULL, manager_thread, (void *)(char *[]){source_dir, dest_dir}); // Create manager thread
    for (int i = 0; i < num_workers; i++) {
        pthread_create(&workers[i], NULL, worker_thread, NULL); // Create worker threads
    }

    pthread_join(manager, NULL);                                      // Wait for manager thread to finish
    for (int i = 0; i < num_workers; i++) {
        pthread_join(workers[i], NULL);                               // Wait for worker threads to finish
    }

    free(workers);                                                    // Free memory for worker threads
    destroy_buffer(&buffer);                                          // Destroy buffer

    gettimeofday(&end_time, NULL);                                    // Get end time
    long seconds = end_time.tv_sec - start_time.tv_sec;               // Calculate seconds elapsed
    long microseconds = end_time.tv_usec - start_time.tv_usec; // Calculate microseconds elapsed
    double elapsed = seconds + microseconds*1e-6;                     // Calculate total time elapsed
```

```
(base) erent@DESKTOP-E56IRIT:/mnt/c/Users/erent/Desktop/hw4/put_yo
ur_codes_here$ make test3
gcc -Wall -pthread -c MWCp.c
gcc -Wall -pthread -o MWCp MWCp.o
Running Test 3: buffer size=10, number of workers=10
./MWCp 10 10 ../testdir ../toCopy

---------------STATISTICS--------------------
Consumers: 10 - Buffer Size: 10
Number of Regular Files: 3116
Number of Directories: 151
Number of FIFO Files: 0
TOTAL BYTES COPIED: 73520554
TOTAL TIME: 12.810 seconds
(base) erent@DESKTOP-E56IRIT:/mnt/c/Users/erent/Desktop/hw4/put_yo
ur_codes_here$
```