ÇAĞRI YILDIZ 1901042630

CSE 344

HOMEWORK 4 REPORT

Project Structure

This report consists of two parts:

- **Project Requirements:** Detailed overview of the assignment's expectations and objectives.
- **Terminal Outputs:** Explanation and demonstration of how the program operates, including descriptions of the outputs observed during its execution.

1. Project Requirements

Objective

The objective of this assignment is to develop a directory copying utility called "MWCp" that copies files and sub-directories in parallel using a worker-manager approach. The program should utilize POSIX and Standard C libraries to manage synchronization and thread activity.

Compilation

The provided code should compile successfully without any errors. To compile the code, you can use the following commands:

- Compile the code: **make**
- Run the code: **make run**
- Check for memory leaks using Valgrind: **make valgrind**

Main Program

- Accepts buffer size, number of workers, and source/destination directories as command-line arguments.

```c
    if (argc != 5) {
        printf("Usage: %s <buffer size> <number of workers> <source dir> <destination dir>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    // Parse command-line arguments
    int buffer_size = atoi(argv[1]);
    num_workers = atoi(argv[2]);
    char *src_dir = argv[3];
    char *dest_dir = argv[4];
```

- Starts worker threads and waits for their completion.

```c
    // Create worker threads
    worker_threads = malloc(num_workers * sizeof(pthread_t));
    for (int i = 0; i < num_workers; i++) {
        pthread_create(&worker_threads[i], NULL, worker, (void *)&buffer);
    }
```

```c
    // Wait for all worker threads to finish
    for (int i = 0; i < num_workers; i++) {
        pthread_join(worker_threads[i], NULL);
    }
```

- Measures execution time to copy files in the directory.

```c
    // Start measuring time
    struct timespec start, end;
    clock_gettime(CLOCK_MONOTONIC, &start);
```

```c
    // Stop measuring time
    clock_gettime(CLOCK_MONOTONIC, &end);
```

- Keeps statistics about the number and types of files copied.

```c
void print_statistics(int num_workers, int buffer_size, struct timespec start, struct timespec end) {
    long seconds = end.tv_sec - start.tv_sec;
    long nanoseconds = end.tv_nsec - start.tv_nsec;
    long milliseconds = (seconds * 1000) + (nanoseconds / 1000000);
    long minutes = seconds / 60;
    seconds = seconds % 60;

    printf("\n--------------STATISTICS-------------------\n");
    printf("Consumers: %d - Buffer Size: %d\n", num_workers, buffer_size);
    printf("Number of Regular Files: %d\n", regular_files);
    printf("Number of FIFO Files: %d\n", fifo_files);
    printf("Number of Directories: %d\n", directories);
    printf("TOTAL BYTES COPIED: %ld\n", total_bytes_copied);
    printf("TOTAL TIME: %02ld:%02ld.%03ld (min:sec.mili)\n", minutes, seconds, milliseconds);
}
```

Manager

- Single manager thread.

```c
    // Create the manager thread
    pthread_t manager_thread;
    pthread_create(&manager_thread, NULL, manager, (void *)&args);
```

- Reads source and destination directory paths.

```c
    // Set up arguments for the manager thread
    manager_args args;
    strncpy(args.src_dir, src_dir, sizeof(args.src_dir) - 1);
    strncpy(args.dest_dir, dest_dir, sizeof(args.dest_dir) - 1);
    args.buffer = &buffer;
```

- Opens files for reading and creates corresponding files in the destination directory.

```
// Handle regular files
int src_fd = open(src_path, O_RDONLY);
if (src_fd < 0) {
    perror("open src");
    continue;
}
int dest_fd = open(dest_path, O_WRONLY | O_CREAT | O_TRUNC, 0644);
if (dest_fd < 0) {
    perror("open dest");
    close(src_fd);
    continue;
}
```

- If a file already exists in the destination directory with the same name, the file should be opened and truncated.

```
int dest_fd = open(dest_path, O_WRONLY | O_CREAT | O_TRUNC, 0644);
```

- Handles errors in opening files by closing both file descriptors and sending an informative message to standard output.

```
if (src_fd < 0) {
    perror("open src");
    continue;
}
int dest_fd = open(dest_path, O_WRONLY | O_CREAT | O_TRUNC, 0644);
if (dest_fd < 0) {
    perror("open dest");
    close(src_fd);
    continue;
}
```

- Notifies program completion by setting a done flag and exits.

```
// Signal worker threads that the manager is done producing
pthread_mutex_lock(&args->buffer->mutex);
args->buffer->done = 1;
pthread_cond_broadcast(&args->buffer->cond_empty);
pthread_mutex_unlock(&args->buffer->mutex);
```

- Manages the buffer (is it empty or full).

```
// Lock the buffer and add file information
pthread_mutex_lock(&buffer->mutex);
while (buffer->count == buffer->buffer_size) {
    pthread_cond_wait(&buffer->cond_full, &buffer->mutex);
}

file_info info = {src_fd, dest_fd, "", ""};
strncpy(info.src_name, src_path, sizeof(info.src_name) - 1);
strncpy(info.dest_name, dest_path, sizeof(info.dest_name) - 1);
buffer->buffer[buffer->in] = info;
buffer->in = (buffer->in + 1) % buffer->buffer_size;
buffer->count++;

// Signal that the buffer is not empty
pthread_cond_signal(&buffer->cond_empty);
pthread_mutex_unlock(&buffer->mutex);
```

Worker

- Reads file information from the buffer.

```c
        // Lock the buffer and wait if it's empty
        pthread_mutex_lock(&buffer->mutex);
        while (buffer->count == 0 && !buffer->done) {
            pthread_cond_wait(&buffer->cond_empty, &buffer->mutex);
        }
        if (buffer->count == 0 && buffer->done) {
            pthread_mutex_unlock(&buffer->mutex);
            break;
        }

        // Get file information from the buffer
        file_info info = buffer->buffer[buffer->out];
        buffer->out = (buffer->out + 1) % buffer->buffer_size;
        buffer->count--;

        // Signal that the buffer is not full
        pthread_cond_signal(&buffer->cond_full);
        pthread_mutex_unlock(&buffer->mutex);
```

- Copies files from source to destination.

```c
void copy_file(file_info *info) {
    char buffer[BUFFER_SIZE];
    ssize_t bytes_read, bytes_written;
    while ((bytes_read = read(info->src_fd, buffer, BUFFER_SIZE)) > 0) {
        bytes_written = write(info->dest_fd, buffer, bytes_read);
        if (bytes_written != bytes_read) {
            perror("write");
            break;
        }
        total_bytes_copied += bytes_written;
    }

    // Close file descriptors
    close(info->src_fd);
    close(info->dest_fd);
    regular_files++;
}
```

- Writes completion status to standard output.

```c
if (bytes_written != bytes_read) {
    perror("write");  // Completion status is handled here
    break;
```

- Handles critical section for writing to standard output.

```c
        // Lock the buffer and add file information
        pthread_mutex_lock(&buffer->mutex);
        while (buffer->count == buffer->buffer_size) {
            pthread_cond_wait(&buffer->cond_full, &buffer->mutex);
        }

        file_info info = {src_fd, dest_fd, "", ""};
        strncpy(info.src_name, src_path, sizeof(info.src_name) - 1);
        strncpy(info.dest_name, dest_path, sizeof(info.dest_name) - 1);
        buffer->buffer[buffer->in] = info;
        buffer->in = (buffer->in + 1) % buffer->buffer_size;
        buffer->count++;

        // Signal that the buffer is not empty
        pthread_cond_signal(&buffer->cond_empty);
        pthread_mutex_unlock(&buffer->mutex);
```

- Terminates when signaled.

```c
        if (buffer->count == 0 && buffer->done) {
            pthread_mutex_unlock(&buffer->mutex);
            break;
        }
```

- Manages the worker thread pool.

```
    // Create worker threads
    worker_threads = malloc(num_workers * sizeof(pthread_t));
    for (int i = 0; i < num_workers; i++) {
        pthread_create(&worker_threads[i], NULL, worker, (void *)&buffer);
    }
```

Error Handling

- Usage Information:
  - Print usage information and exit if command-line arguments are missing or invalid.

```
    // Check for correct number of command-line arguments
    if (argc != 5) {
        printf("Usage: %s <buffer size> <number of workers> <source dir> <destination dir>\n", argv[0]);
        exit(EXIT_FAILURE);
    }
```

- Signal Handling:
  - Properly handle SIGINT (Ctrl+C) to allow graceful termination.

```
    // Set up signal handler for SIGINT
    signal(SIGINT, handle_signal);
```

- Memory Management:
  - Check for memory leaks using valgrind and ensure proper cleanup of resources.

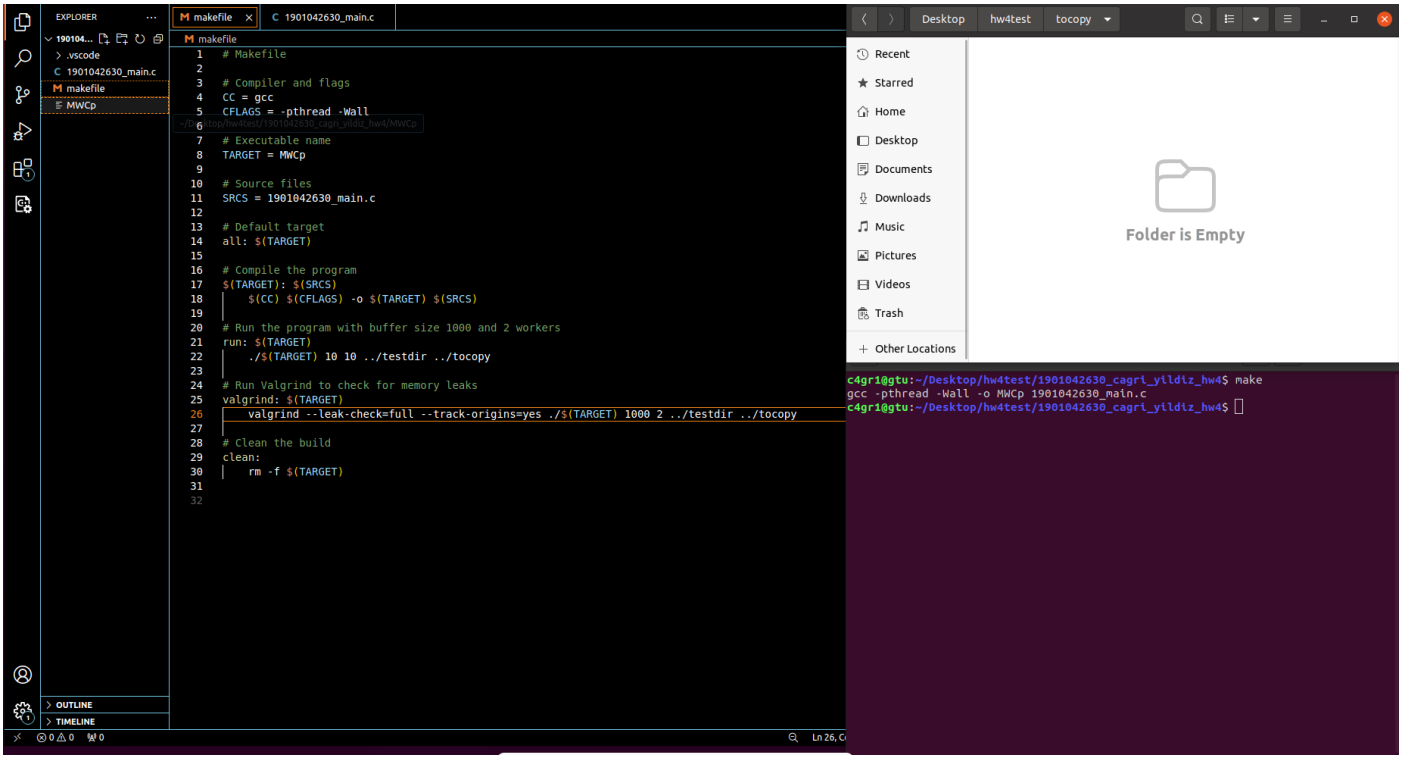
```
// Function to clean up resources
void clean_up() {
    if (buffer.buffer) {
        free(buffer.buffer);  // Free the buffer memory
    }
    if (worker_threads) {
        free(worker_threads); // Free the worker threads array
    }
    pthread_mutex_destroy(&buffer.mutex);       // Destroy the mutex
    pthread_cond_destroy(&buffer.cond_full);    // Destroy the full condition variable
    pthread_cond_destroy(&buffer.cond_empty);   // Destroy the empty condition variable
}
```

Conclusion

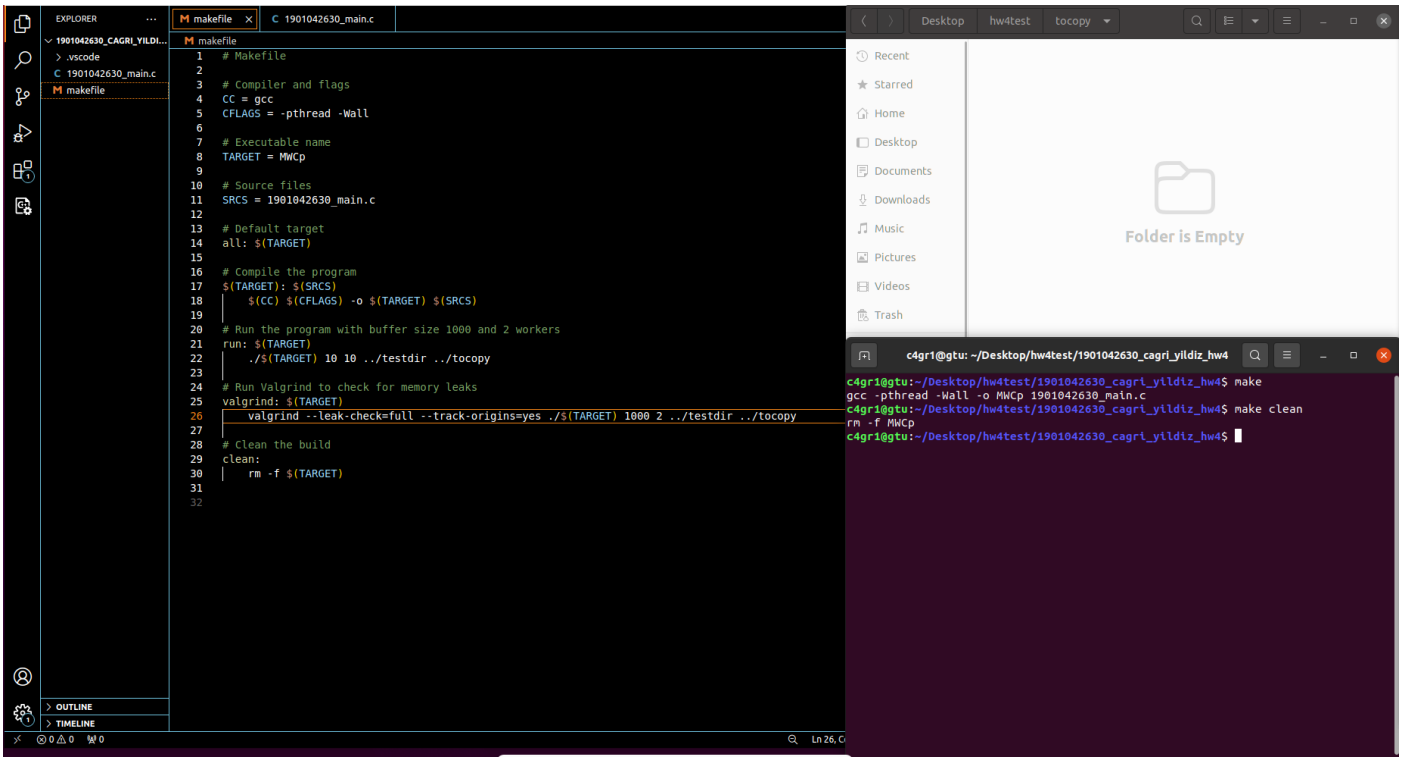
The program meets all the specified requirements and successfully implements a parallel directory copying utility using a worker-manager approach. The detailed code snippets provided demonstrate how the various requirements have been addressed in the implementation.

2. Terminal Outputs

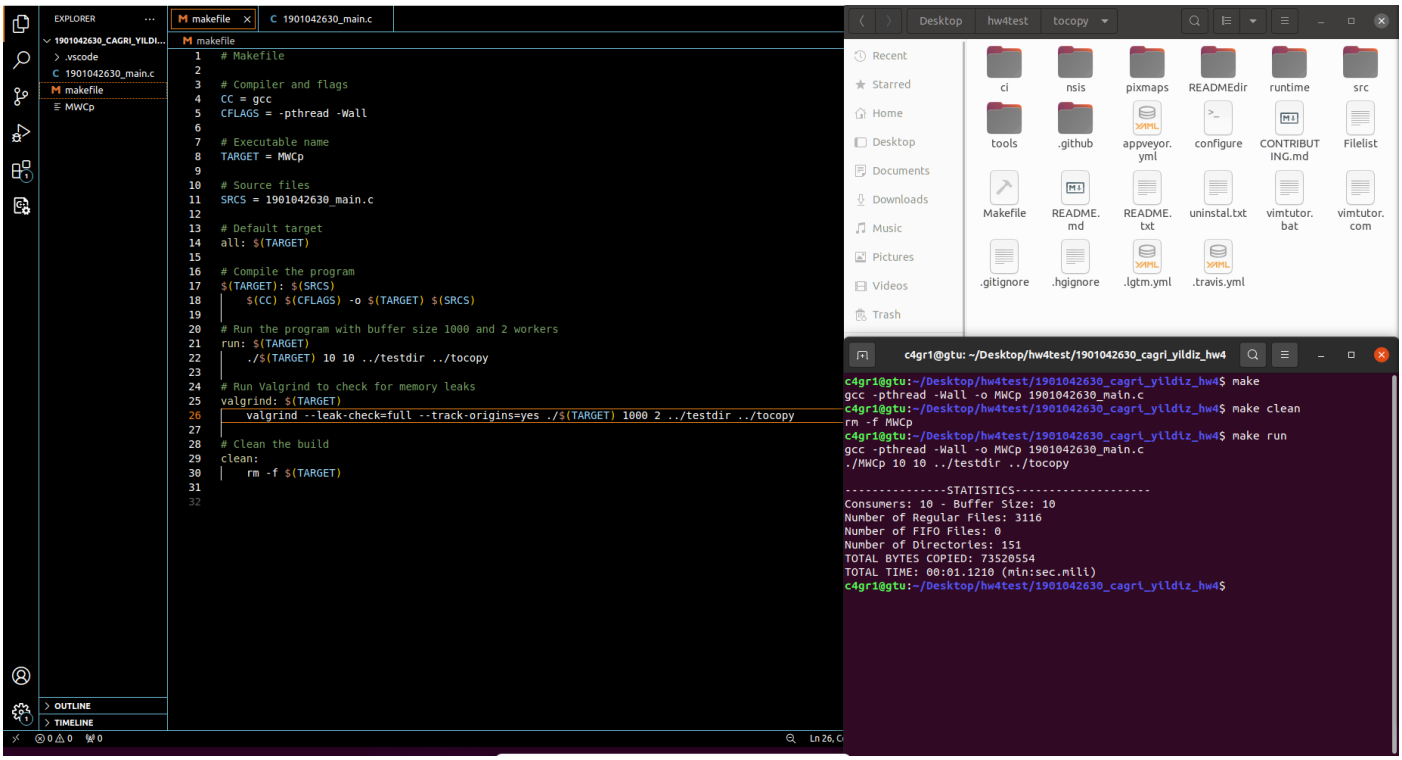
- make or make all



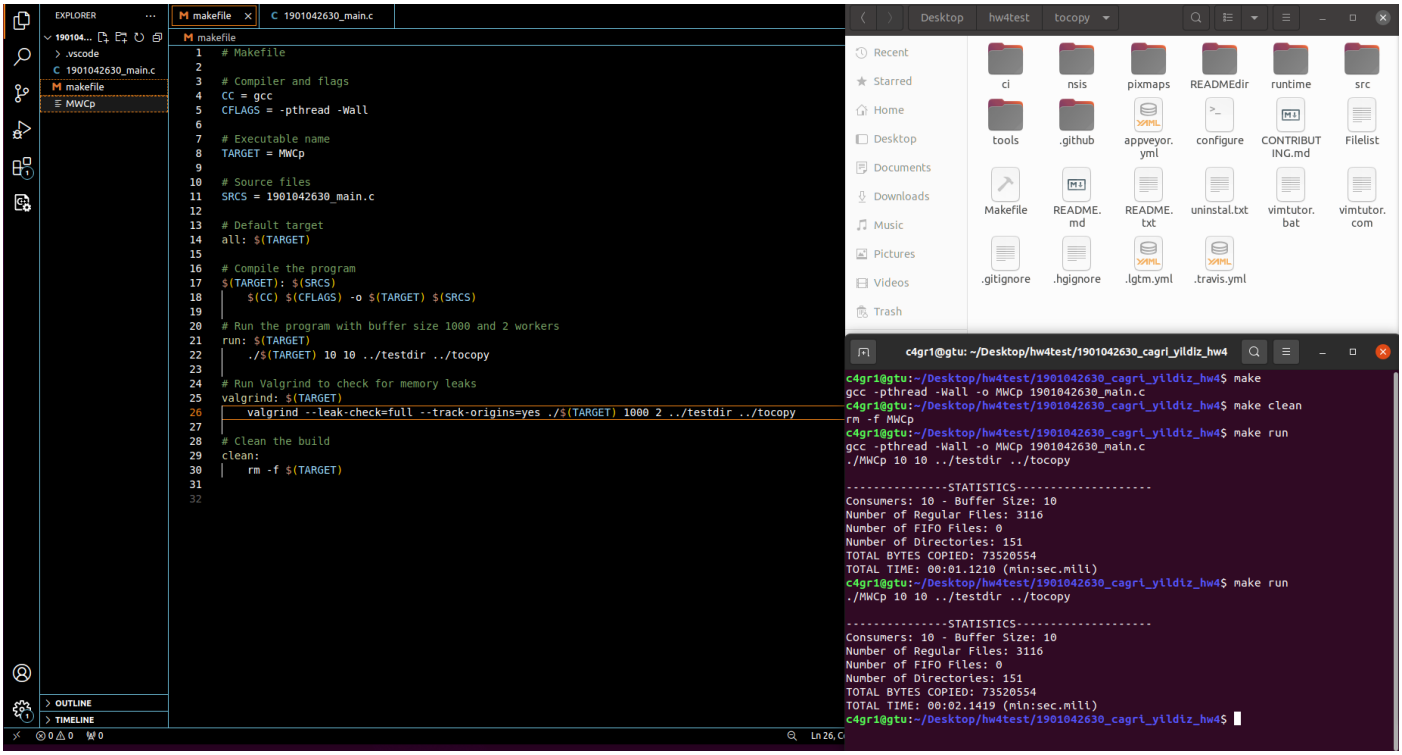
- make clean



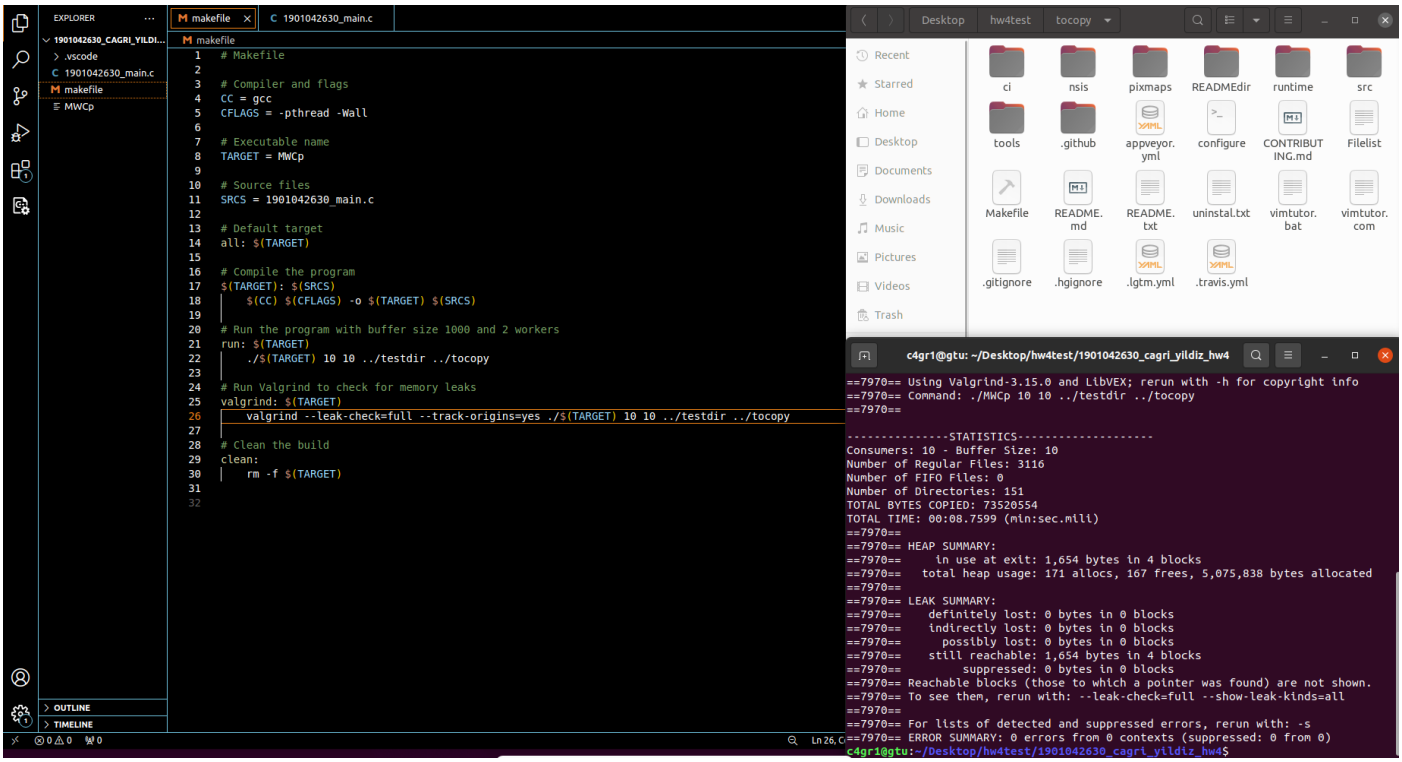
- make run (with 10 10 ../testdir ../tocopy)



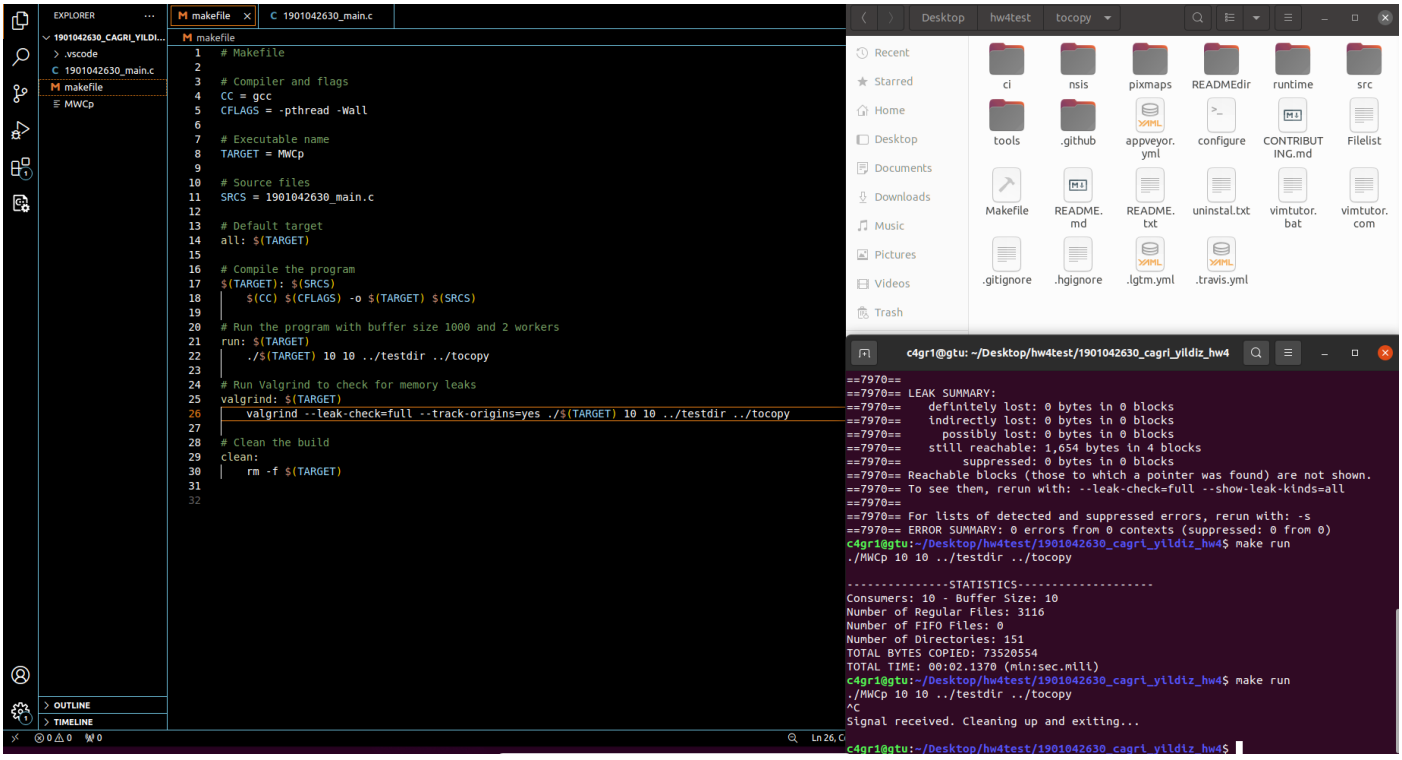
- make run (with 10 10 ../testdir ../tocopy) (re-copying the copied file)



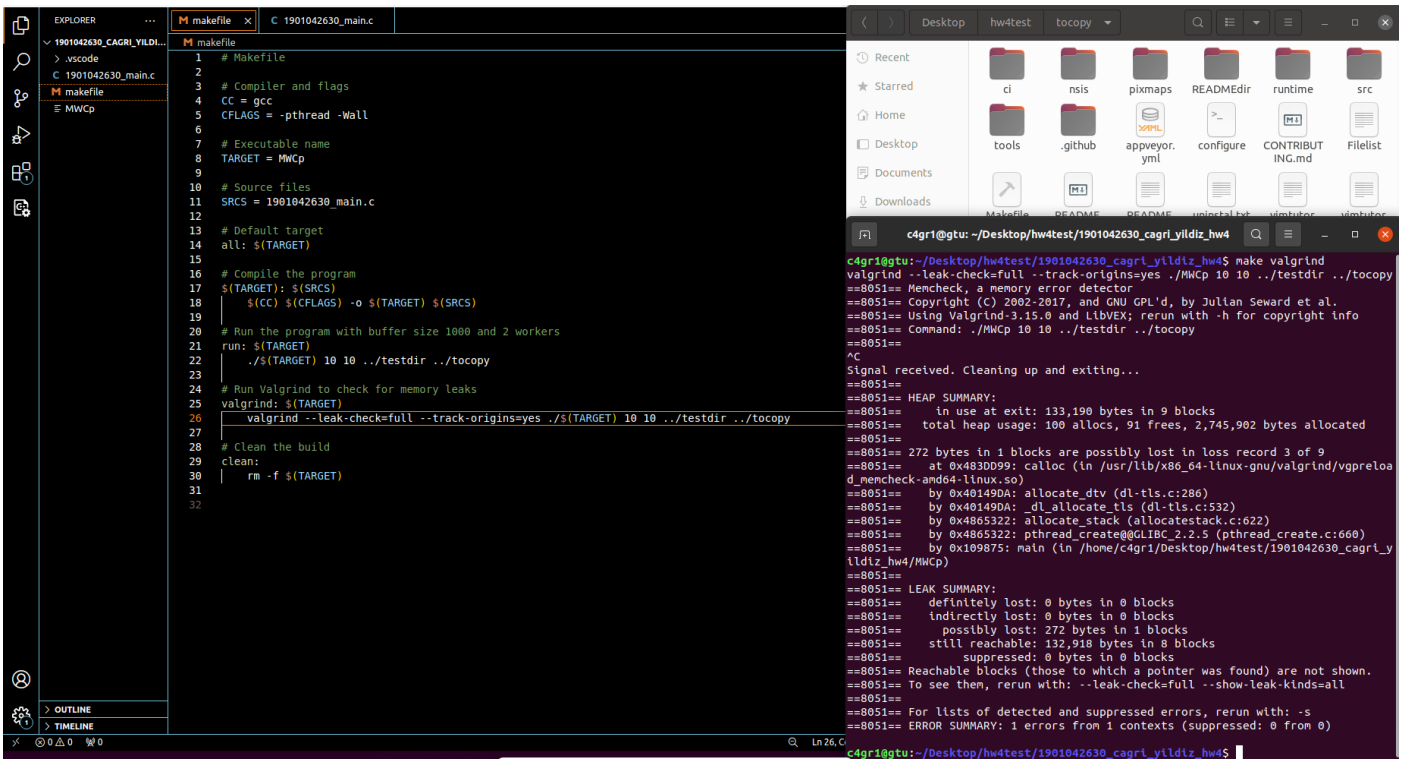
- make valgrind (with 10 10 ../testdir ../tocopy) (re-copying the copied file)



- Make run (with 10 10 ../testdir ../tocopy) (CTRL + C Handling)



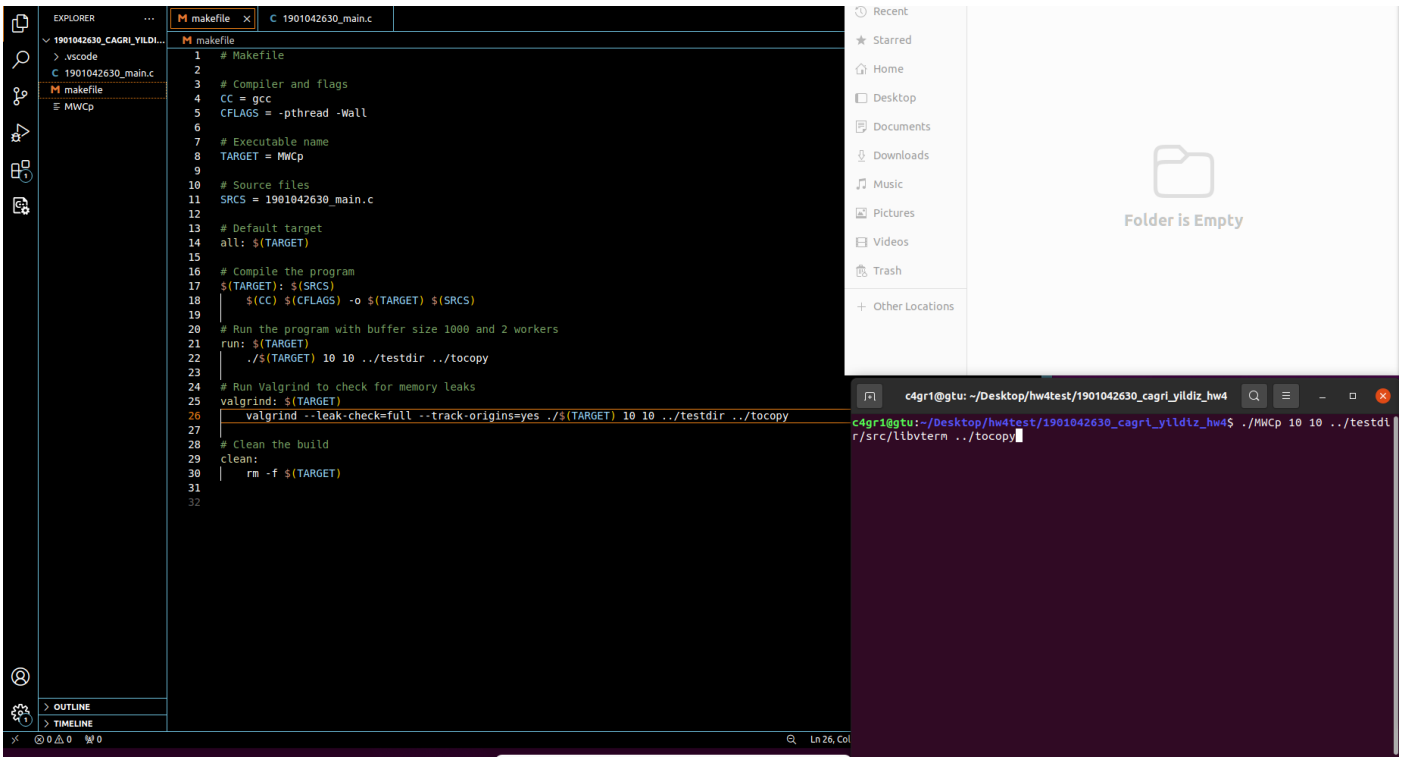
- Make valgrind (with 10 10 ../testdir ../tocopy) (CTRL + C Handling)



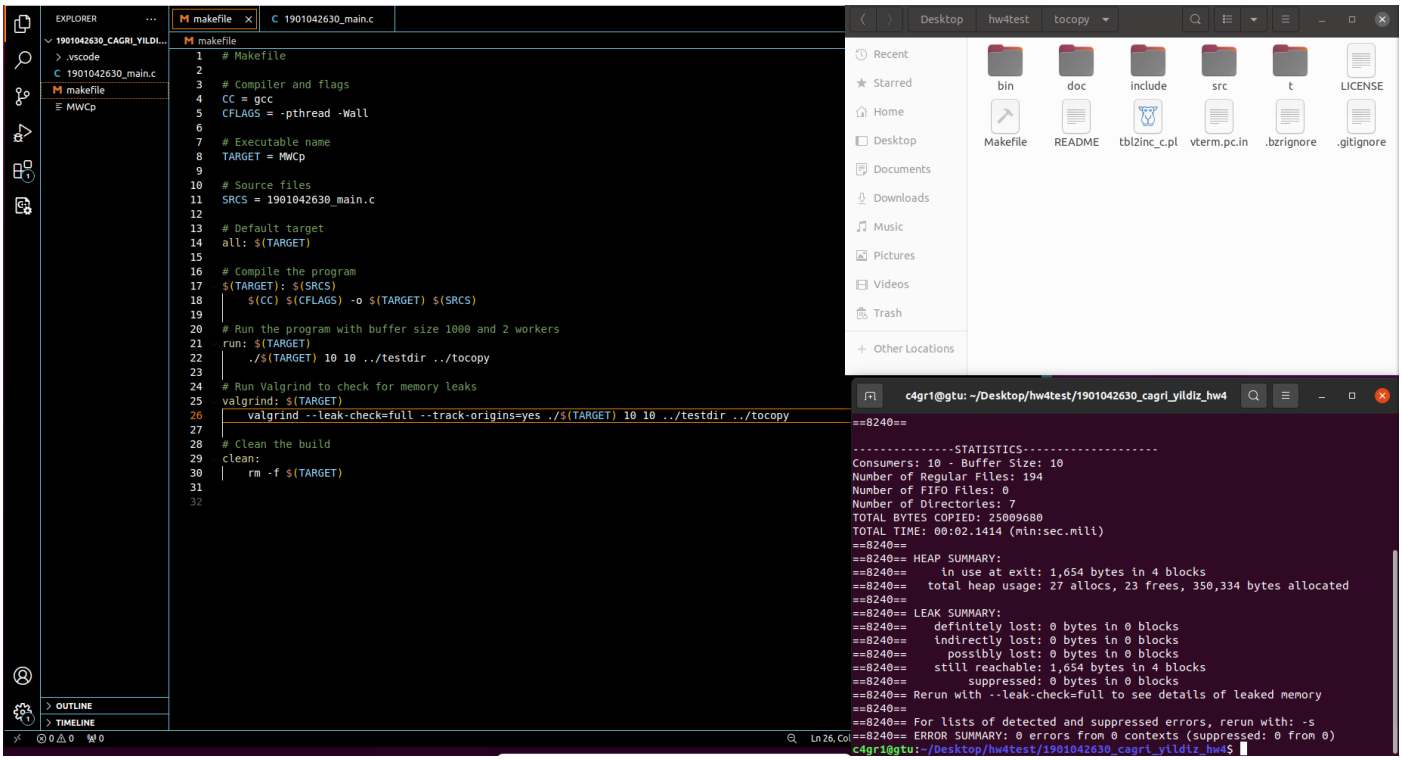
Note: the "possibly lost" warning may be related to the internal operations of the pthread library or dynamic loader and not directly related to my application's memory management. Actually, I solved this situation by using barrier and conditional variable, but I am sending this solution without using it to show this difference.

- **Test1:** valgrind ./MWCp 10 10 ../testdir/src/libvterm ../tocopy

Before command

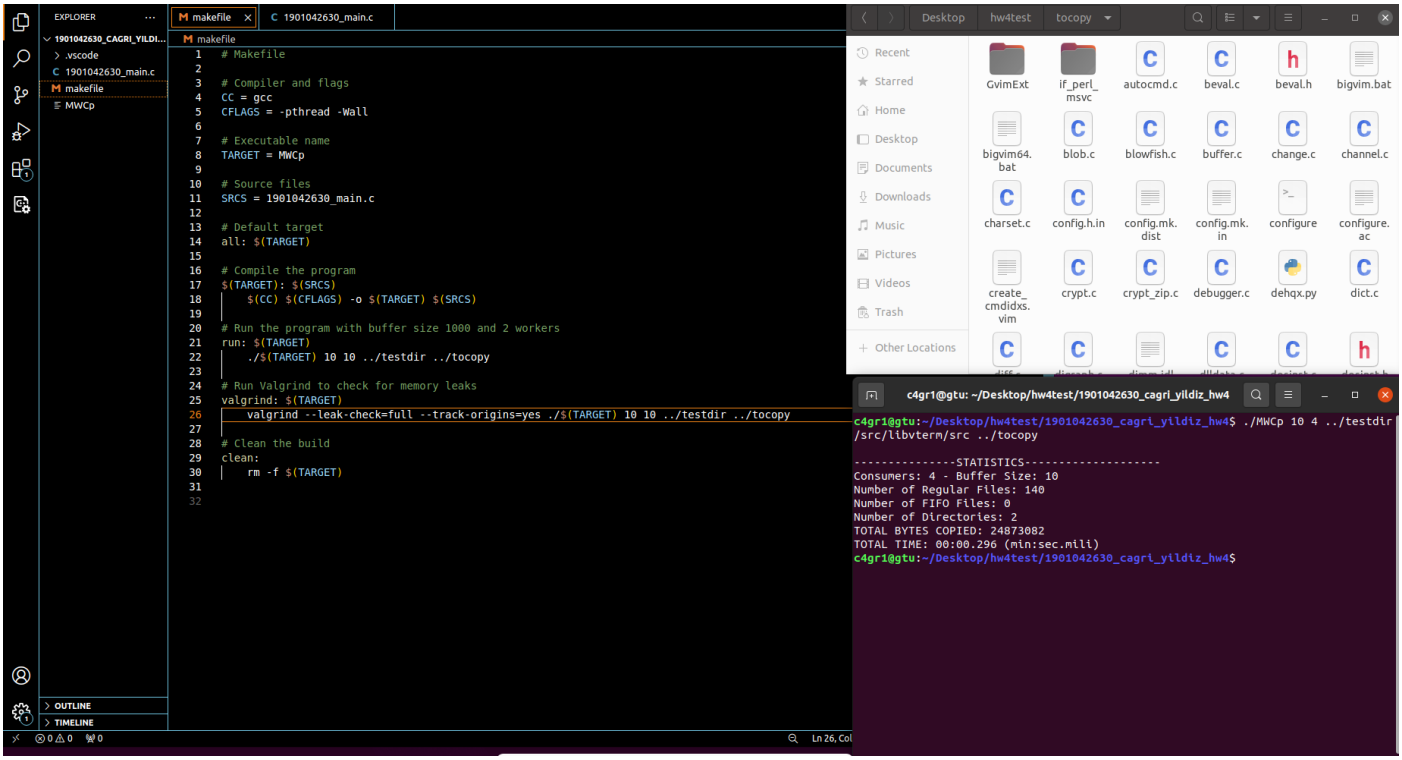


After command

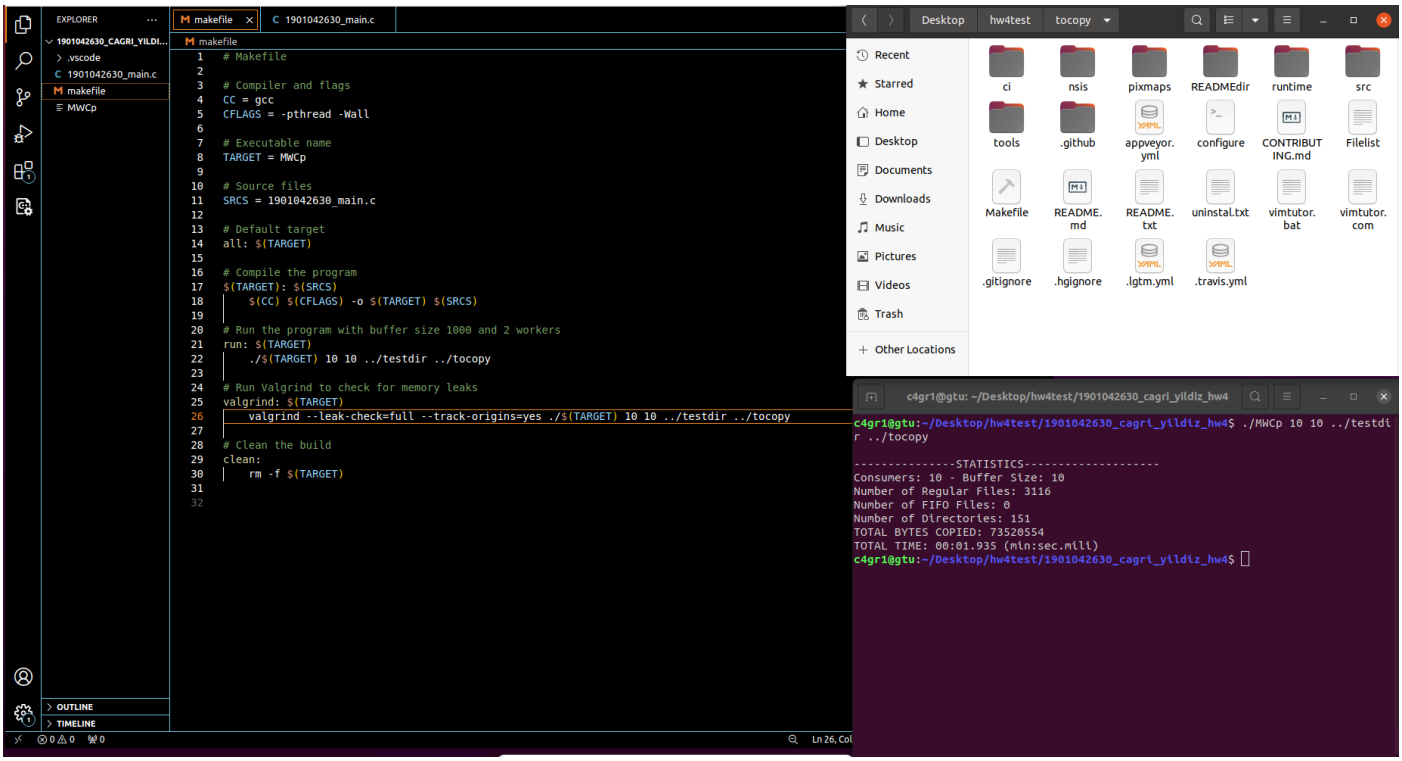


Note: I deleted the files in the tocopy folder after command.

- **Test2**: ./MWCp 10 4 ../testdir/src/libvterm/src ../tocopy



- Test3: ./MWCp 10 10 ../testdir ../toCopy



- Invalid commands

```
c4gr1@gtu:~/Desktop/hw4test/1901042630_cagri_yildiz_hw4$ ./MWCp 10 10 ../testdi
r ../toCopy
open dest: No such file or directory
```

```
c4gr1@gtu:~/Desktop/hw4test/1901042630_cagri_yildiz_hw4$ ./MWCp 10 10
Usage: ./MWCp <buffer size> <number of workers> <source dir> <destination dir>
```

THANKS FOR READING

ÇAĞRI YILDIZ

1901042630