

## CSE 344

# HOMEWORK 5 REPORT

### Summary of Changes from HW4 to HW5

In HW5, the directory copying utility "MWCp" from HW4 has been enhanced with the following features:

#### 1. Condition Variables:

- Used to signal when the buffer is not empty and when the buffer is not full (so the manager can add more items).
- Implemented to ensure that the manager waits when the buffer is full and workers wait when the buffer is empty.

```
34 pthread_cond_t cond_full; // Condition variable to signal buffer is not full
35 pthread_cond_t cond_empty; // Condition variable to signal buffer is not empty
```

#### 2. Barriers:

- Used to ensure that all worker threads wait at a certain point before proceeding. This ensures synchronization of all threads before moving to the next phase of processing.

```
36 pthread_barrier_t barrier; // Barrier for synchronization
```

### 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.

```

98     if (argc != 5) {
99         printf("Usage: %s <buffer size> <number of workers> <source dir> <destination dir>\n", argv[0]);
100        exit(EXIT_FAILURE);
101    }
102
103    // Parse command-line arguments
104    int buffer_size = atoi(argv[1]);
105    num_workers = atoi(argv[2]);
106    char *src_dir = argv[3];
107    char *dest_dir = argv[4];

```

- Starts worker threads and waits for their completion.

```

134    // Create worker threads
135    worker_threads = malloc(num_workers * sizeof(pthread_t));
136    for (int i = 0; i < num_workers; i++) {
137        pthread_create(&worker_threads[i], NULL, worker, (void *)&buffer);
138    }

```

```

152    // Wait for all worker threads to finish
153    for (int i = 0; i < num_workers; i++) {
154        pthread_join(worker_threads[i], NULL);
155    }

```

- Measures execution time to copy files in the directory.

```

120    // Start measuring time
121    struct timespec start, end;
122    clock_gettime(CLOCK_MONOTONIC, &start);
123

```

```

157    // Stop measuring time
158    clock_gettime(CLOCK_MONOTONIC, &end);

```

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

```

295 void print_statistics(int num_workers, int buffer_size, struct timespec start, struct timespec end) {
296     long seconds = end.tv_sec - start.tv_sec;
297     long nanoseconds = end.tv_nsec - start.tv_nsec;
298     long milliseconds = (seconds * 1000) + (nanoseconds / 1000000);
299     long minutes = seconds / 60;
300     seconds = seconds % 60;
301
302     printf("\n-----STATISTICS-----\n");
303     printf("Consumers: %d - Buffer Size: %d\n", num_workers, buffer_size);
304     printf("Number of Regular Files: %d\n", regular_files);
305     printf("Number of FIFO Files: %d\n", fifo_files);
306     printf("Number of Directories: %d\n", directories);
307     printf("TOTAL BYTES COPIED: %ld\n", total_bytes_copied);
308     printf("TOTAL TIME: %02ld:%02ld.%03ld (min:sec.mili)\n", minutes, seconds, milliseconds);
309 }

```

## Manager

- Single manager thread.

```
130 // Create the manager thread
131 pthread_t manager_thread;
132 pthread_create(&manager_thread, NULL, manager, (void *)&args);
133
```

- Reads source and destination directory paths.

```
124 // Set up arguments for the manager thread
125 manager_args args;
126 strncpy(args.src_dir, src_dir, sizeof(args.src_dir) - 1);
127 strncpy(args.dest_dir, dest_dir, sizeof(args.dest_dir) - 1);
128 args.buffer = &buffer;
129
```

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

```
206 // Handle regular files
207 int src_fd = open(src_path, O_RDONLY);
208 if (src_fd < 0) {
209     perror("open src");
210     continue;
211 }
212 int dest_fd = open(dest_path, O_WRONLY | O_CREAT | O_TRUNC, 0644);
213 if (dest_fd < 0) {
214     perror("open dest");
215     close(src_fd);
216     continue;
217 }
218
```

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

```
212 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.

```
208 if (src_fd < 0) {
209     perror("open src");
210     continue;
211 }
212 int dest_fd = open(dest_path, O_WRONLY | O_CREAT | O_TRUNC, 0644);
213 if (dest_fd < 0) {
214     perror("open dest");
215     close(src_fd);
216     continue;
217 }
```

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

```

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

```

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

```

219 // Lock the buffer and add file information
220 pthread_mutex_lock(&buffer->mutex);
221 while (buffer->count == buffer->buffer_size) {
222     pthread_cond_wait(&buffer->cond_full, &buffer->mutex);
223 }
224
225 file_info info = {src_fd, dest_fd, "", ""};
226 strncpy(info.src_name, src_path, sizeof(info.src_name) - 1);
227 strncpy(info.dest_name, dest_path, sizeof(info.dest_name) - 1);
228 buffer->buffer[buffer->in] = info;
229 buffer->in = (buffer->in + 1) % buffer->buffer_size;
230 buffer->count++;
231
232 // Signal that the buffer is not empty
233 pthread_cond_signal(&buffer->cond_empty);
234 pthread_mutex_unlock(&buffer->mutex);

```

## Worker

- Reads file information from the buffer.

```

252 // Lock the buffer and wait if it's empty
253 pthread_mutex_lock(&buffer->mutex);
254 while (buffer->count == 0 && !buffer->done) {
255     pthread_cond_wait(&buffer->cond_empty, &buffer->mutex);
256 }
257 if (buffer->count == 0 && buffer->done) {
258     pthread_mutex_unlock(&buffer->mutex);
259     break;
260 }
261
262 // Get file information from the buffer
263 file_info info = buffer->buffer[buffer->out];
264 buffer->out = (buffer->out + 1) % buffer->buffer_size;
265 buffer->count--;
266
267 // Signal that the buffer is not full
268 pthread_cond_signal(&buffer->cond_full);
269 pthread_mutex_unlock(&buffer->mutex);

```

- Copies files from source to destination.

```

277 void copy_file(file_info *info) {
278     char buffer[BUFFER_SIZE];
279     ssize_t bytes_read, bytes_written;
280     while ((bytes_read = read(info->src_fd, buffer, BUFFER_SIZE)) > 0) {
281         bytes_written = write(info->dest_fd, buffer, bytes_read);
282         if (bytes_written != bytes_read) {
283             perror("write");
284             break;
285         }
286         total_bytes_copied += bytes_written;
287     }
288
289     // Close file descriptors
290     close(info->src_fd);
291     close(info->dest_fd);
292     regular_files++;
293 }

```

- Writes completion status to standard output.

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

- Handles critical section for writing to standard output.

```
219 // Lock the buffer and add file information
220 pthread_mutex_lock(&buffer->mutex);
221 while (buffer->count == buffer->buffer_size) {
222     pthread_cond_wait(&buffer->cond_full, &buffer->mutex);
223 }
224
225 file_info info = {src_fd, dest_fd, "", ""};
226 strncpy(info.src_name, src_path, sizeof(info.src_name) - 1);
227 strncpy(info.dest_name, dest_path, sizeof(info.dest_name) - 1);
228 buffer->buffer[buffer->in] = info;
229 buffer->in = (buffer->in + 1) % buffer->buffer_size;
230 buffer->count++;
231
232 // Signal that the buffer is not empty
233 pthread_cond_signal(&buffer->cond_empty);
234 pthread_mutex_unlock(&buffer->mutex);
```

- Terminates when signaled.

```
257 if (buffer->count == 0 && buffer->done) {
258     pthread_mutex_unlock(&buffer->mutex);
259     break;
260 }
```

- Manages the worker thread pool.

```
134 // Create worker threads
135 worker_threads = malloc(num_workers * sizeof(pthread_t));
136 for (int i = 0; i < num_workers; i++) {
137     pthread_create(&worker_threads[i], NULL, worker, (void *)&buffer);
138 }
```

## Error Handling

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

```
97 // Check for correct number of command-line arguments
98 if (argc != 5) {
99     printf("Usage: %s <buffer size> <number of workers> <source dir> <destination dir>\n", argv[0]);
100     exit(EXIT_FAILURE);
101 }
```

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

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

- Memory Management:

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

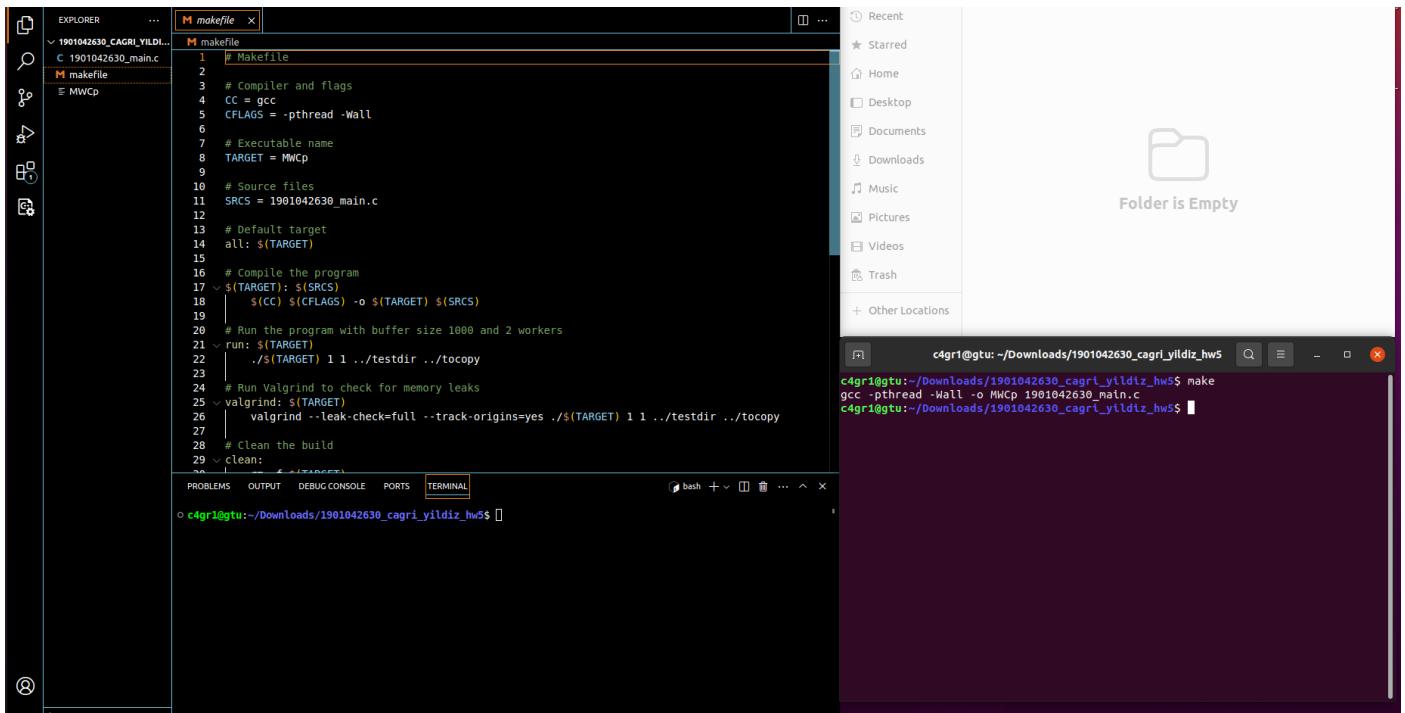
```
65 // Function to clean up resources
66 void clean_up() {
67     if (buffer.buffer) {
68         free(buffer.buffer); // Free the buffer memory
69     }
70     if (worker_threads) {
71         free(worker_threads); // Free the worker threads array
72     }
73     pthread_mutex_destroy(&buffer.mutex); // Destroy the mutex
74     pthread_cond_destroy(&buffer.cond_full); // Destroy the full condition variable
75     pthread_cond_destroy(&buffer.cond_empty); // Destroy the empty condition variable
76 }
```

## Conclusion

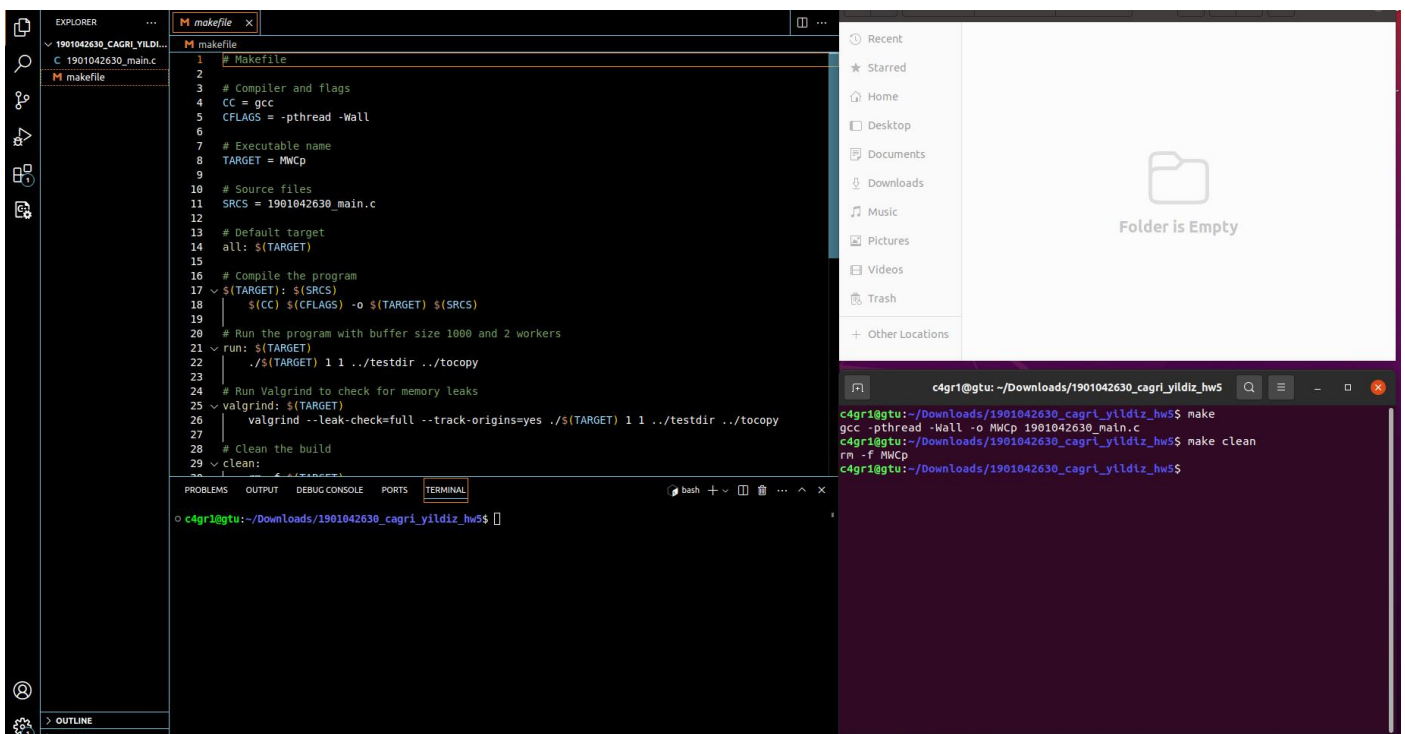
The program meets all the specified requirements and successfully implements a parallel directory copying utility using a worker-manager approach. The detailed code snippets provided demonstrate how the various requirements have been addressed in the implementation.

## 2. Terminal Outputs

- make or make all



- make clean



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

```

1 # Makefile
2
3 # Compiler and flags
4 CC = gcc
5 CFLAGS = -pthread -Wall
6
7 # Executable name
8 TARGET = MWCP
9
10 # Source files
11 SRCS = 1901042630_main.c
12
13 # Default target
14 all: $(TARGET)
15
16 # Compile the program
17 $(TARGET): $(SRCS)
18 | $(CC) $(CFLAGS) -o $(TARGET) $(SRCS)
19
20 # Run the program with buffer size 1000 and 2 workers
21 run: $(TARGET)
22 | ./$(TARGET) 10 10 ../testdir ../tocopy
23
24 # Run Valgrind to check for memory leaks
25 valgrind: $(TARGET)
26 | valgrind --leak-check=full --track-origins=yes ./$(TARGET) 1 1 ../testdir ../tocopy
27
28 # Clean the build
29 clean:
30 | rm -f $(TARGET)

```

```

c4gr1@gtu: ~/Downloads/1901042630_cagri_yildiz_hw5$ make
gcc -pthread -Wall -o MWCP 1901042630_main.c
c4gr1@gtu: ~/Downloads/1901042630_cagri_yildiz_hw5$ make clean
rm -f MWCP
c4gr1@gtu: ~/Downloads/1901042630_cagri_yildiz_hw5$ make run
gcc -pthread -Wall -o MWCP 1901042630_main.c
./MWCP 10 10 ../testdir ../tocopy
-----STATISTICS-----
Consumers: 10 - Buffer Size: 10
Number of Regular Files: 3110
Number of FIFO Files: 0
Number of Directories: 151
TOTAL BYTES COPIED: 73520554
TOTAL TIME: 00:01.904 (min:sec.milli)
c4gr1@gtu: ~/Downloads/1901042630_cagri_yildiz_hw5$

```

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

```

1 # Makefile
2
3 # Compiler and flags
4 CC = gcc
5 CFLAGS = -pthread -Wall
6
7 # Executable name
8 TARGET = MWCP
9
10 # Source files
11 SRCS = 1901042630_main.c
12
13 # Default target
14 all: $(TARGET)
15
16 # Compile the program
17 $(TARGET): $(SRCS)
18 | $(CC) $(CFLAGS) -o $(TARGET) $(SRCS)
19
20 # Run the program with buffer size 1000 and 2 workers
21 run: $(TARGET)
22 | ./$(TARGET) 10 10 ../testdir ../tocopy
23
24 # Run Valgrind to check for memory leaks
25 valgrind: $(TARGET)
26 | valgrind --leak-check=full --track-origins=yes ./$(TARGET) 1 1 ../testdir ../tocopy
27
28 # Clean the build
29 clean:
30 | rm -f $(TARGET)

```

```

c4gr1@gtu: ~/Downloads/1901042630_cagri_yildiz_hw5$ make clean
rm -f MWCP
c4gr1@gtu: ~/Downloads/1901042630_cagri_yildiz_hw5$ make run
gcc -pthread -Wall -o MWCP 1901042630_main.c
./MWCP 10 10 ../testdir ../tocopy
-----STATISTICS-----
Consumers: 10 - Buffer Size: 10
Number of Regular Files: 3110
Number of FIFO Files: 0
Number of Directories: 151
TOTAL BYTES COPIED: 73520554
TOTAL TIME: 00:01.904 (min:sec.milli)
c4gr1@gtu: ~/Downloads/1901042630_cagri_yildiz_hw5$ make run
./MWCP 10 10 ../testdir ../tocopy
-----STATISTICS-----
Consumers: 10 - Buffer Size: 10
Number of Regular Files: 3110
Number of FIFO Files: 0
Number of Directories: 151
TOTAL BYTES COPIED: 73520554
TOTAL TIME: 00:01.1376 (min:sec.milli)
c4gr1@gtu: ~/Downloads/1901042630_cagri_yildiz_hw5$

```



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

The screenshot shows a VS Code editor with a Makefile open. The Makefile defines a target named 'MWCP' and includes instructions for compilation, execution, and memory leak checking using Valgrind. The terminal output shows the successful execution of the program and the results of the Valgrind run, which indicates no memory leaks were found.

```

4 CC = gcc
5 CFLAGS = -pthread -Wall
6
7 # Executable name
8 TARGET = MWCP
9
10 # Source files
11 SRCS = 1901042630_main.c
12
13 # Default target
14 all: $(TARGET)
15
16 # Compile the program
17 $(TARGET): $(SRCS)
18 | $(CC) $(CFLAGS) -o $(TARGET) $(SRCS)
19
20 # Run the program with buffer size 1000 and 2 workers
21 run: $(TARGET)
22 | ./$(TARGET) 10 10 ../testdir ../tocopy
23
24 # Run Valgrind to check for memory leaks
25 valgrind: $(TARGET)
26 | valgrind --leak-check=full --track-origins=yes ./$(TARGET) 10 10 ../testdir ../tocopy
27
28 # Clean the build
29 clean:
30 | rm -f $(TARGET)
31
32

```

```

c4gr1@gtu:~/Downloads/1901042630_cagri_yildiz_hw$
-----STATISTICS-----
Consumers: 10 - Buffer Size: 10
Number of Regular Files: 3116
Number of FIFO Files: 0
Number of Directories: 151
TOTAL BYTES COPIED: 73520554
TOTAL TIME: 00:10.9640 (min:sec.mill)
==5783==
==5783== HEAP SUMMARY:
==5783==   in use at exit: 1,654 bytes in 4 blocks
==5783==   total heap usage: 171 allocs, 167 frees, 5,075,838 bytes allocated
==5783==
==5783== LEAK SUMMARY:
==5783==   definitely lost: 0 bytes in 0 blocks
==5783==   indirectly lost: 0 bytes in 0 blocks
==5783==   possibly lost: 0 bytes in 0 blocks
==5783==   still reachable: 1,654 bytes in 4 blocks
==5783==   suppressed: 0 bytes in 0 blocks
==5783== Reachable blocks (those to which a pointer was found) are not shown.
==5783== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==5783==
==5783== For lists of detected and suppressed errors, rerun with: -s
==5783== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
c4gr1@gtu:~/Downloads/1901042630_cagri_yildiz_hw$

```

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

The screenshot shows the same VS Code editor setup as before, but with the terminal output showing the execution of the program being interrupted by a Ctrl+C signal. The output shows the program's statistics and the results of the Valgrind run, which indicates no memory leaks were found.

```

4 CC = gcc
5 CFLAGS = -pthread -Wall
6
7 # Executable name
8 TARGET = MWCP
9
10 # Source files
11 SRCS = 1901042630_main.c
12
13 # Default target
14 all: $(TARGET)
15
16 # Compile the program
17 $(TARGET): $(SRCS)
18 | $(CC) $(CFLAGS) -o $(TARGET) $(SRCS)
19
20 # Run the program with buffer size 1000 and 2 workers
21 run: $(TARGET)
22 | ./$(TARGET) 10 10 ../testdir ../tocopy
23
24 # Run Valgrind to check for memory leaks
25 valgrind: $(TARGET)
26 | valgrind --leak-check=full --track-origins=yes ./$(TARGET) 10 10 ../testdir ../tocopy
27
28 # Clean the build
29 clean:
30 | rm -f $(TARGET)
31
32

```

```

c4gr1@gtu:~/Downloads/1901042630_cagri_yildiz_hw$
==5844==   definitely lost: 0 bytes in 0 blocks
==5844==   indirectly lost: 0 bytes in 0 blocks
==5844==   possibly lost: 0 bytes in 0 blocks
==5844==   still reachable: 1,654 bytes in 4 blocks
==5844==   suppressed: 0 bytes in 0 blocks
==5844== Reachable blocks (those to which a pointer was found) are not shown.
==5844== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==5844==
==5844== For lists of detected and suppressed errors, rerun with: -s
==5844== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
c4gr1@gtu:~/Downloads/1901042630_cagri_yildiz_hw$ make run
./MWCP 10 10 ../testdir ../tocopy
Signal received. Cleaning up and exiting...
-----STATISTICS-----
Consumers: 10 - Buffer Size: 10
Number of Regular Files: 2695
Number of FIFO Files: 0
Number of Directories: 145
TOTAL BYTES COPIED: 63445848
TOTAL TIME: 00:01.1125 (min:sec.mill)
c4gr1@gtu:~/Downloads/1901042630_cagri_yildiz_hw$

```

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

```

4 CC = gcc
5 CFLAGS = -pthread -Wall
6
7 # Executable name
8 TARGET = MWCp
9
10 # Source files
11 SRCS = 1901042630_main.c
12
13 # Default target
14 all: $(TARGET)
15
16 # Compile the program
17 $(TARGET): $(SRCS)
18 | $(CC) $(CFLAGS) -o $(TARGET) $(SRCS)
19
20 # Run the program with buffer size 1000 and 2 workers
21 run: $(TARGET)
22 | ./$(TARGET) 10 10 ../testdir ../tocopy
23
24 # Run Valgrind to check for memory leaks
25 valgrind: $(TARGET)
26 | valgrind --leak-check=full --track-origins=yes ./$(TARGET) 10 10 ../testdir ../tocopy
27
28 # Clean the build
29 clean:
30 | rm -f $(TARGET)
31
32

```

```

c4gr1@gtu: ~/Downloads/1901042630_cagri_yildiz_hw5
Signal received. Cleaning up and exiting...
-----STATISTICS-----
Consumers: 10 - Buffer Size: 10
Number of Regular Files: 1338
Number of FIFO Files: 0
Number of Directories: 99
TOTAL BYTES COPIED: 53363431
TOTAL TIME: 00:04.3529 (min:sec.milli)
==5878==
==5878== HEAP SUMMARY:
==5878==   in use at exit: 1,654 bytes in 4 blocks
==5878== total heap usage: 119 allocs, 115 frees, 3,369,406 bytes allocated
==5878==
==5878== LEAK SUMMARY:
==5878==   definitely lost: 0 bytes in 0 blocks
==5878==   indirectly lost: 0 bytes in 0 blocks
==5878==   possibly lost: 0 bytes in 0 blocks
==5878==   still reachable: 1,654 bytes in 4 blocks
==5878==   suppressed: 0 bytes in 0 blocks
==5878== Reachable blocks (those to which a pointer was found) are not shown.
==5878== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==5878==
==5878== For lists of detected and suppressed errors, rerun with: -s

```

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

Before command

```

4 CC = gcc
5 CFLAGS = -pthread -Wall
6
7 # Executable name
8 TARGET = MWCp
9
10 # Source files
11 SRCS = 1901042630_main.c
12
13 # Default target
14 all: $(TARGET)
15
16 # Compile the program
17 $(TARGET): $(SRCS)
18 | $(CC) $(CFLAGS) -o $(TARGET) $(SRCS)
19
20 # Run the program with buffer size 1000 and 2 workers
21 run: $(TARGET)
22 | ./$(TARGET) 10 10 ../testdir ../tocopy
23
24 # Run Valgrind to check for memory leaks
25 valgrind: $(TARGET)
26 | valgrind --leak-check=full --track-origins=yes ./$(TARGET) 10 10 ../testdir/src/libvterm ../tocopy
27
28 # Clean the build
29 clean:
30 | rm -f $(TARGET)
31
32

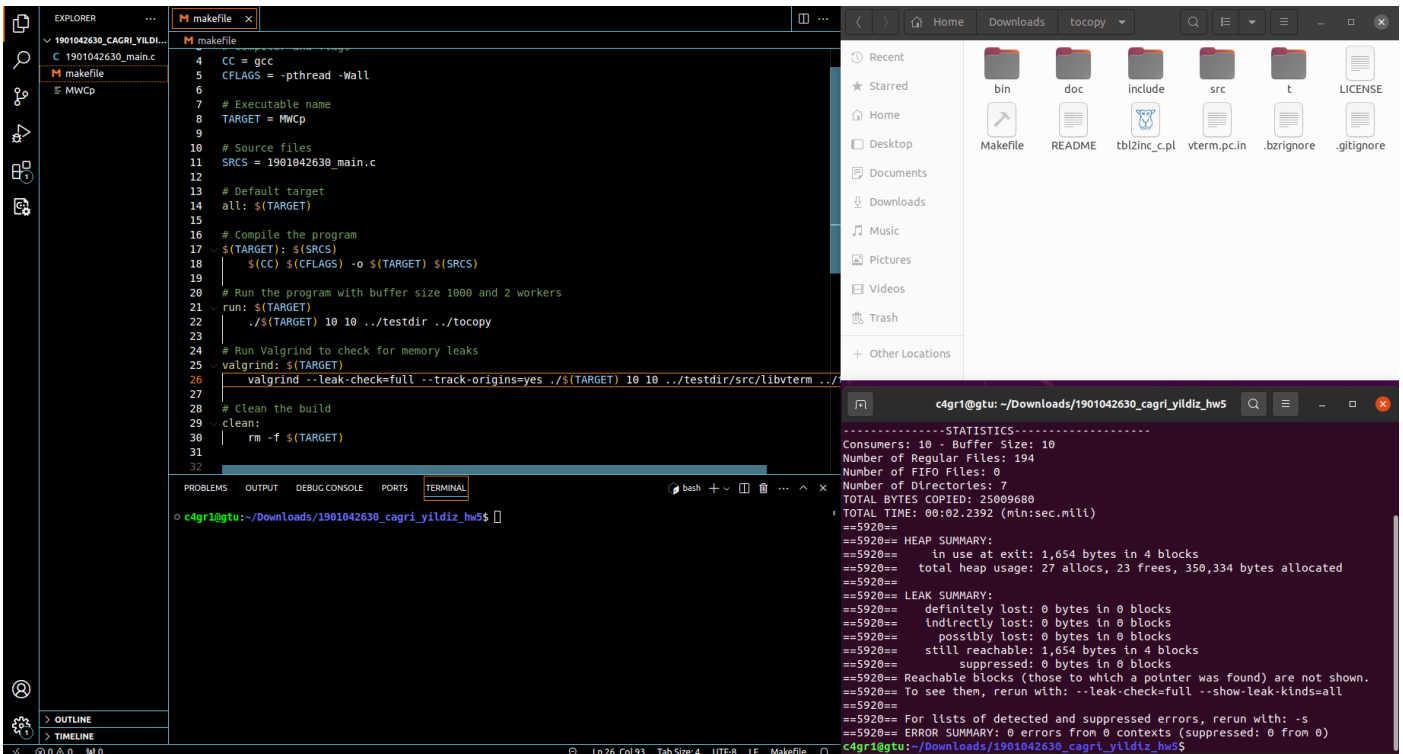
```

```

c4gr1@gtu: ~/Downloads/1901042630_cagri_yildiz_hw5
c4gr1@gtu:~/Downloads/1901042630_cagri_yildiz_hw5$ make valgrind

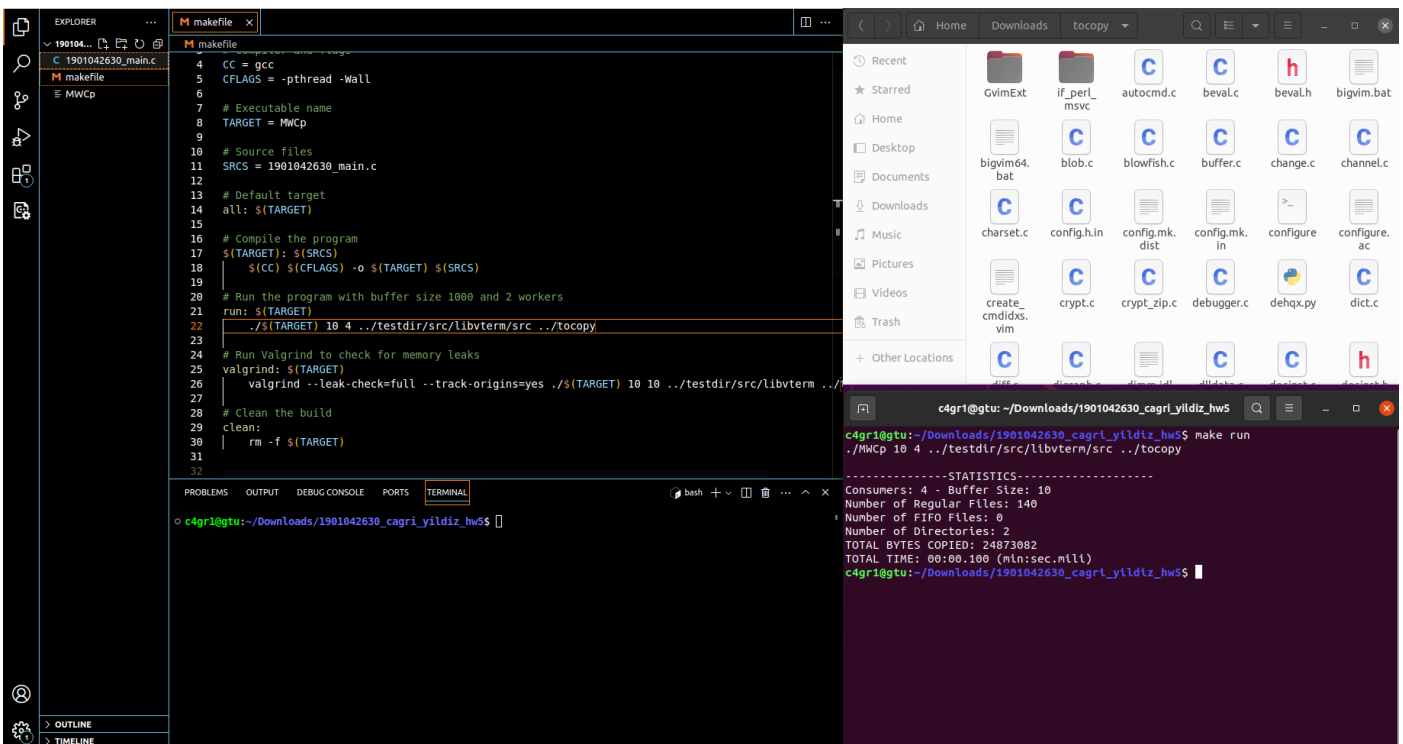
```

## After command

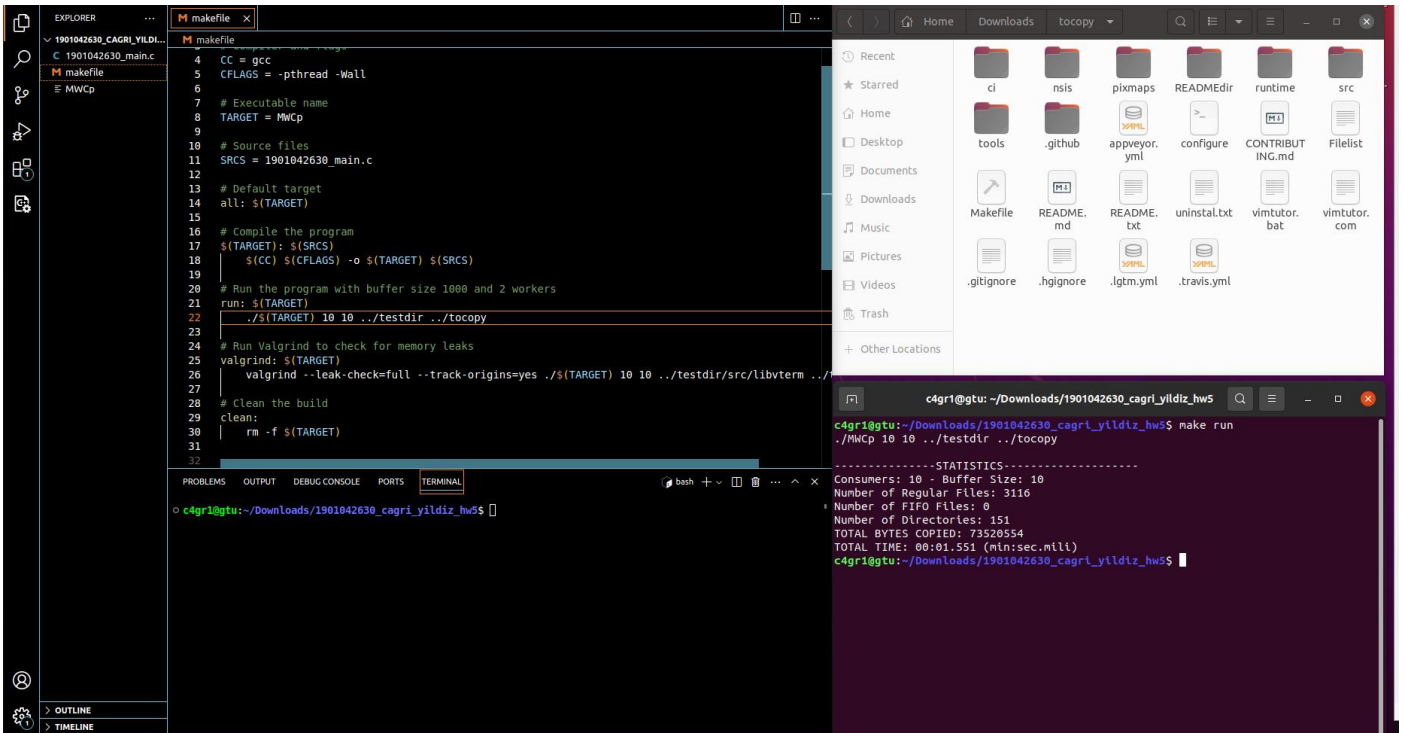


Note: I deleted the files in the tocopy folder after command.

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



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



- Invalid commands

```

c4gr1@gtu:~/Downloads/1901042630_cagri_yildiz_hw5$ make run
./MWCp 10 10 ../testdir ../toCopy
open dest: No such file or directory

```

```

c4gr1@gtu:~/Downloads/1901042630_cagri_yildiz_hw5$ make run
./MWCp 10 10
Usage: ./MWCp <buffer size> <number of workers> <source dir> <destination dir>

```

THANKS FOR READING

ÇAĞRI YILDIZ

1901042630