

# 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.

```
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);

```

```

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;

```

- 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 }

```

- 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.

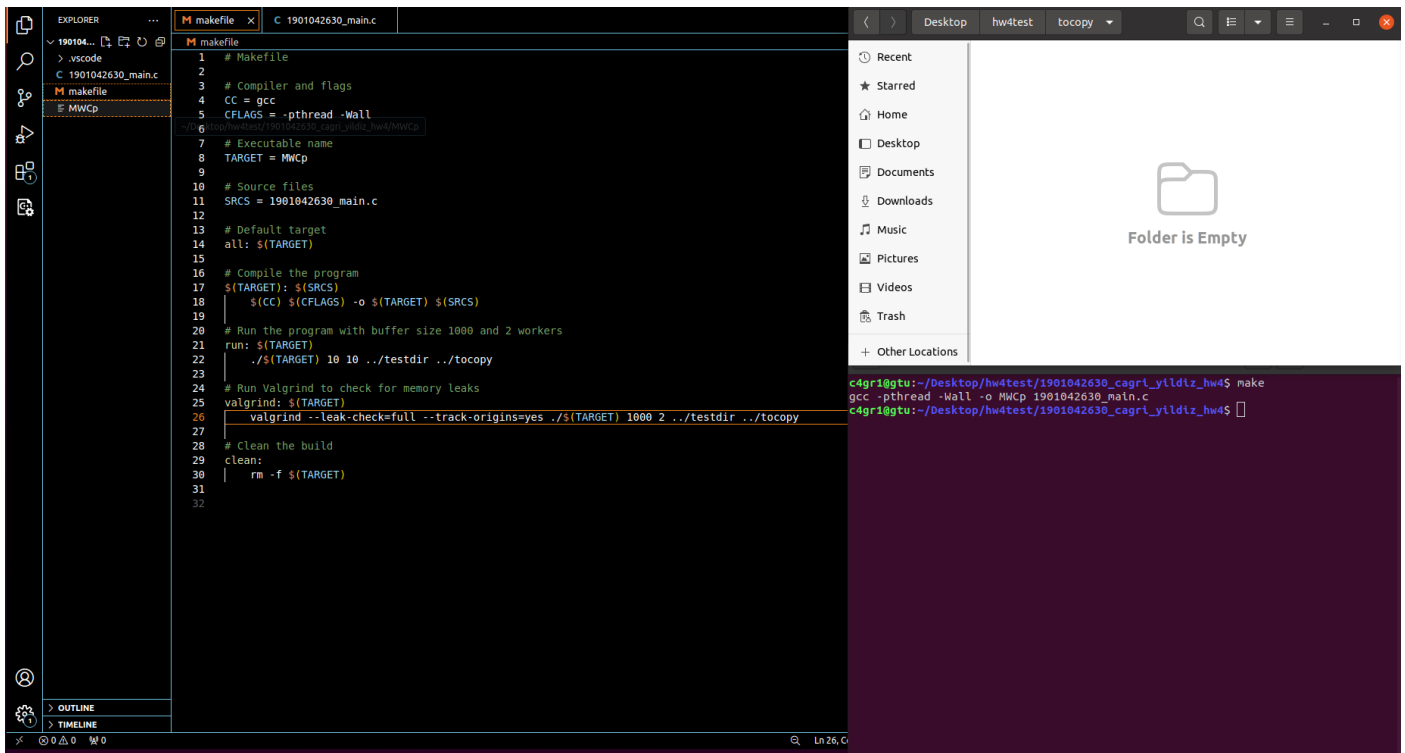
```
65 // Function to clean up resources
66 void clean_up() {
67     if (buffer.buffer) {
68         free(buffer.buffer); // Free the buffer memory
69     }
70     if (worker_threads) {
71         free(worker_threads); // Free the worker threads array
72     }
73     pthread_mutex_destroy(&buffer.mutex); // 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



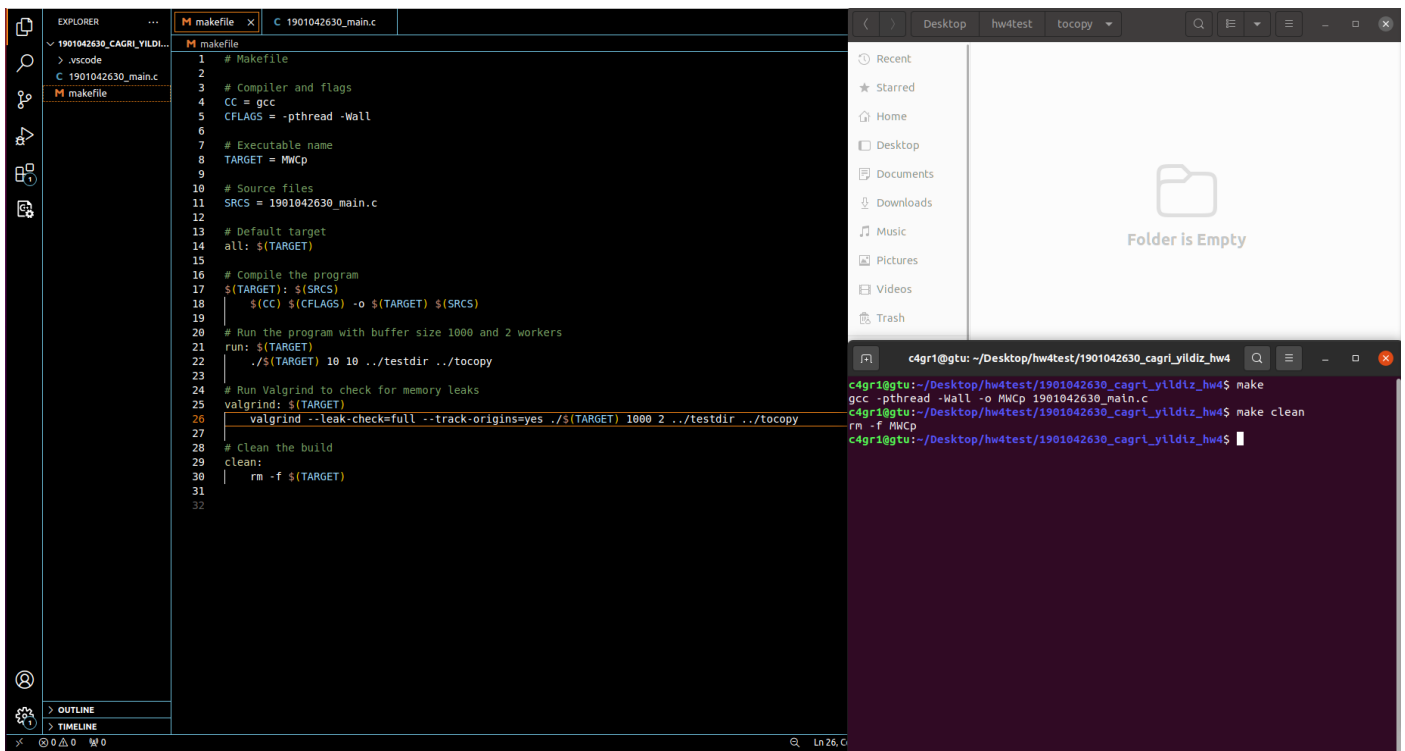
The screenshot shows the VS Code interface with a Makefile open in the editor. The Makefile content is as follows:

```
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) 1000 2 ../testdir ../tocopy
27
28 # Clean the build
29 clean:
30 | rm -f $(TARGET)
31
32
```

The terminal on the right shows the execution of the 'make' command:

```
c4gr1@gtu:~/Desktop/hw4test/1901042630_cagri_yildiz_hw4$ make
gcc -pthread -Wall -o MWCp 1901042630_main.c
c4gr1@gtu:~/Desktop/hw4test/1901042630_cagri_yildiz_hw4$
```

- make clean



The screenshot shows the VS Code interface with the same Makefile open. The terminal on the right shows the execution of the 'make clean' command:

```
c4gr1@gtu:~/Desktop/hw4test/1901042630_cagri_yildiz_hw4$ make clean
rm -f MWCp
c4gr1@gtu:~/Desktop/hw4test/1901042630_cagri_yildiz_hw4$
```

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

The screenshot shows a VS Code editor with a file explorer on the left displaying the project structure. The main editor shows a `Makefile` with the following content:

```
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) 1000 2 ../testdir ../tocopy
27
28 # Clean the build
29 clean:
30 | rm -f $(TARGET)
31
32
```

The terminal window on the right shows the execution of the `make run` command. The output includes the following statistics:

```
-----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: 00:01.1210 (min:sec.mill)
c4gr1@gtu:~/Desktop/hw4test/1901042630_cagri_yildiz_hw4$
```

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

The screenshot shows a VS Code editor with a file explorer on the left displaying the project structure. The main editor shows a `Makefile` with the following content:

```
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) 1000 2 ../testdir ../tocopy
27
28 # Clean the build
29 clean:
30 | rm -f $(TARGET)
31
32
```

The terminal window on the right shows the execution of the `make run` command. The output includes the following statistics:

```
-----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: 00:02.1419 (min:sec.mill)
c4gr1@gtu:~/Desktop/hw4test/1901042630_cagri_yildiz_hw4$
```



- make valgrind ( 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) 10 10 ../testdir ../tocopy
27
28 # Clean the build
29 clean:
30 | rm -f $(TARGET)
31
32

```

```

c4gr1@gtu: ~/Desktop/hw4test/1901042630_cagri_yildiz_hw4
==7970== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==7970== Command: ./MWCP 10 10 ../testdir ../tocopy
==7970==
-----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: 00:08.7599 (min:sec.mll)
==7970==
==7970== HEAP SUMMARY:
==7970==   in use at exit: 1,654 bytes in 4 blocks
==7970==   total heap usage: 171 allocs, 167 frees, 5,075,838 bytes allocated
==7970==
==7970== LEAK SUMMARY:
==7970==   definitely lost: 0 bytes in 0 blocks
==7970==   indirectly lost: 0 bytes in 0 blocks
==7970==   possibly lost: 0 bytes in 0 blocks
==7970==   still reachable: 1,654 bytes in 4 blocks
==7970==   suppressed: 0 bytes in 0 blocks
==7970== Reachable blocks (those to which a pointer was found) are not shown.
==7970== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==7970==
==7970== For lists of detected and suppressed errors, rerun with: -s
==7970== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
c4gr1@gtu: ~/Desktop/hw4test/1901042630_cagri_yildiz_hw4$

```

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

```

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) 10 10 ../testdir ../tocopy
27
28 # Clean the build
29 clean:
30 | rm -f $(TARGET)
31
32

```

```

c4gr1@gtu: ~/Desktop/hw4test/1901042630_cagri_yildiz_hw4
==7970== LEAK SUMMARY:
==7970==   definitely lost: 0 bytes in 0 blocks
==7970==   indirectly lost: 0 bytes in 0 blocks
==7970==   possibly lost: 0 bytes in 0 blocks
==7970==   still reachable: 1,654 bytes in 4 blocks
==7970==   suppressed: 0 bytes in 0 blocks
==7970== Reachable blocks (those to which a pointer was found) are not shown.
==7970== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==7970==
==7970== For lists of detected and suppressed errors, rerun with: -s
==7970== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
c4gr1@gtu: ~/Desktop/hw4test/1901042630_cagri_yildiz_hw4$ make run
./MWCP 10 10 ../testdir ../tocopy
Signal received. Cleaning up and exiting...
c4gr1@gtu: ~/Desktop/hw4test/1901042630_cagri_yildiz_hw4$

```



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

```

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) 10 10 ../testdir ../tocopy
27
28 # Clean the build
29 clean:
30 | rm -f $(TARGET)
31
32

```

```

c4gr1@gtu: ~/Desktop/hw4test/1901042630_cagri_yildiz_hw4
c4gr1@gtu:~/Desktop/hw4test/1901042630_cagri_yildiz_hw4$ make valgrind
valgrind --leak-check=full --track-origins=yes ./MWCP 10 10 ../testdir ../tocopy
==8051== Memcheck, a memory error detector.
==8051== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==8051== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==8051== Command: ./MWCP 10 10 ../testdir ../tocopy
==8051==
Signal received. Cleaning up and exiting...
==8051==
==8051== HEAP SUMMARY:
==8051==    in use at exit: 133,190 bytes in 9 blocks
==8051==    total heap usage: 100 allocs, 91 frees, 2,745,902 bytes allocated
==8051==
==8051== 272 bytes in 1 blocks are possibly lost in loss record 3 of 9
==8051==    at 0x483D099: calloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-and64-linux.so)
==8051==    by 0x40149DA: allocate_dtv (dl-tls.c:286)
==8051==    by 0x40149DA: _dl_allocate_tls (dl-tls.c:532)
==8051==    by 0x4865322: allocate_stack (allocatetest.c:622)
==8051==    by 0x4865322: pthread_create@@GLIBC_2.2.5 (pthread_create.c:660)
==8051==    by 0x109875: main (in /home/c4gr1/Desktop/hw4test/1901042630_cagri_yildiz_hw4/MWCP)
==8051==
==8051== LEAK SUMMARY:
==8051==    definitely lost: 0 bytes in 0 blocks
==8051==    indirectly lost: 0 bytes in 0 blocks
==8051==    possibly lost: 272 bytes in 1 blocks
==8051==    still reachable: 132,918 bytes in 8 blocks
==8051==    suppressed: 0 bytes in 0 blocks
==8051== Reachable blocks (those to which a pointer was found) are not shown.
==8051== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==8051==
==8051== For lists of detected and suppressed errors, rerun with: -s
==8051== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)

```

**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**

```

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) 10 10 ../testdir ../tocopy
27
28 # Clean the build
29 clean:
30 | rm -f $(TARGET)
31
32

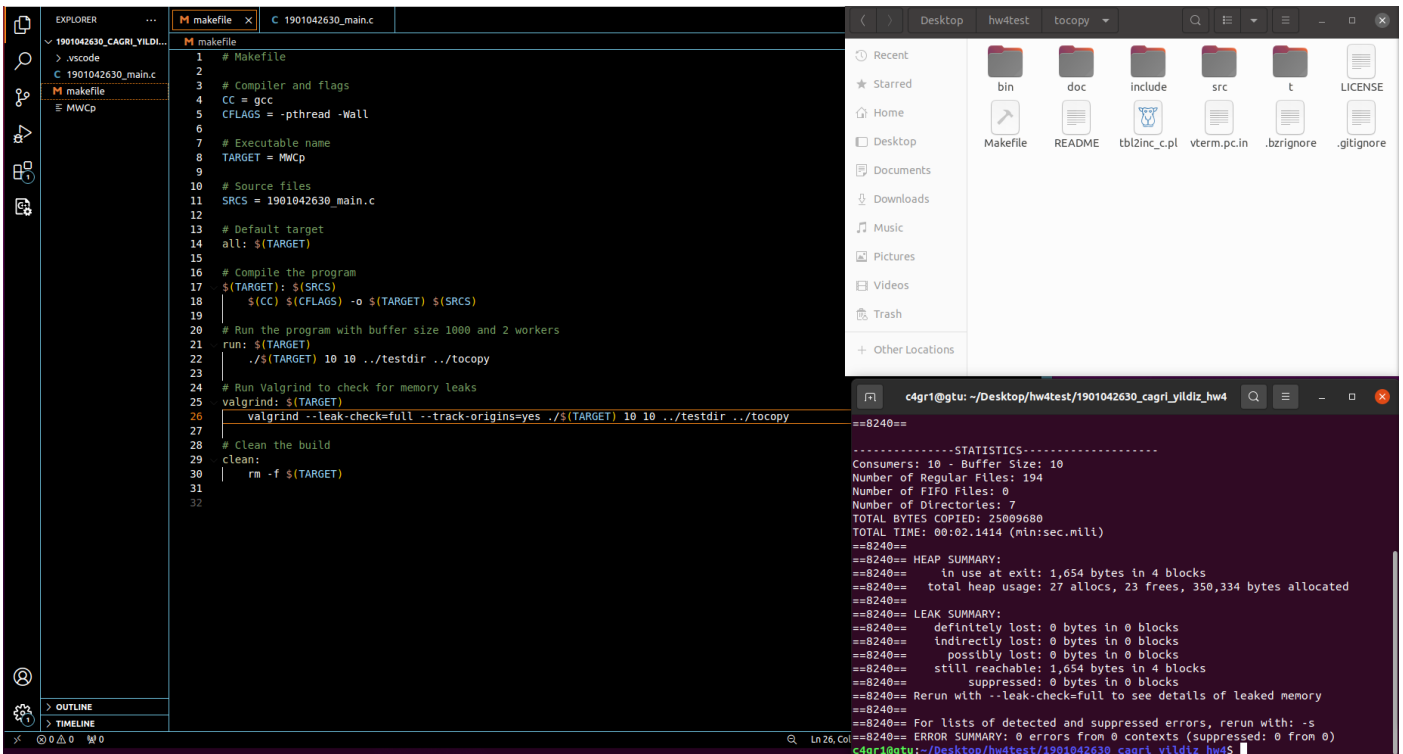
```

```

c4gr1@gtu: ~/Desktop/hw4test/1901042630_cagri_yildiz_hw4
c4gr1@gtu:~/Desktop/hw4test/1901042630_cagri_yildiz_hw4$ ./MWCP 10 10 ../testdir/src/libvterm ../tocopy

```

## After command

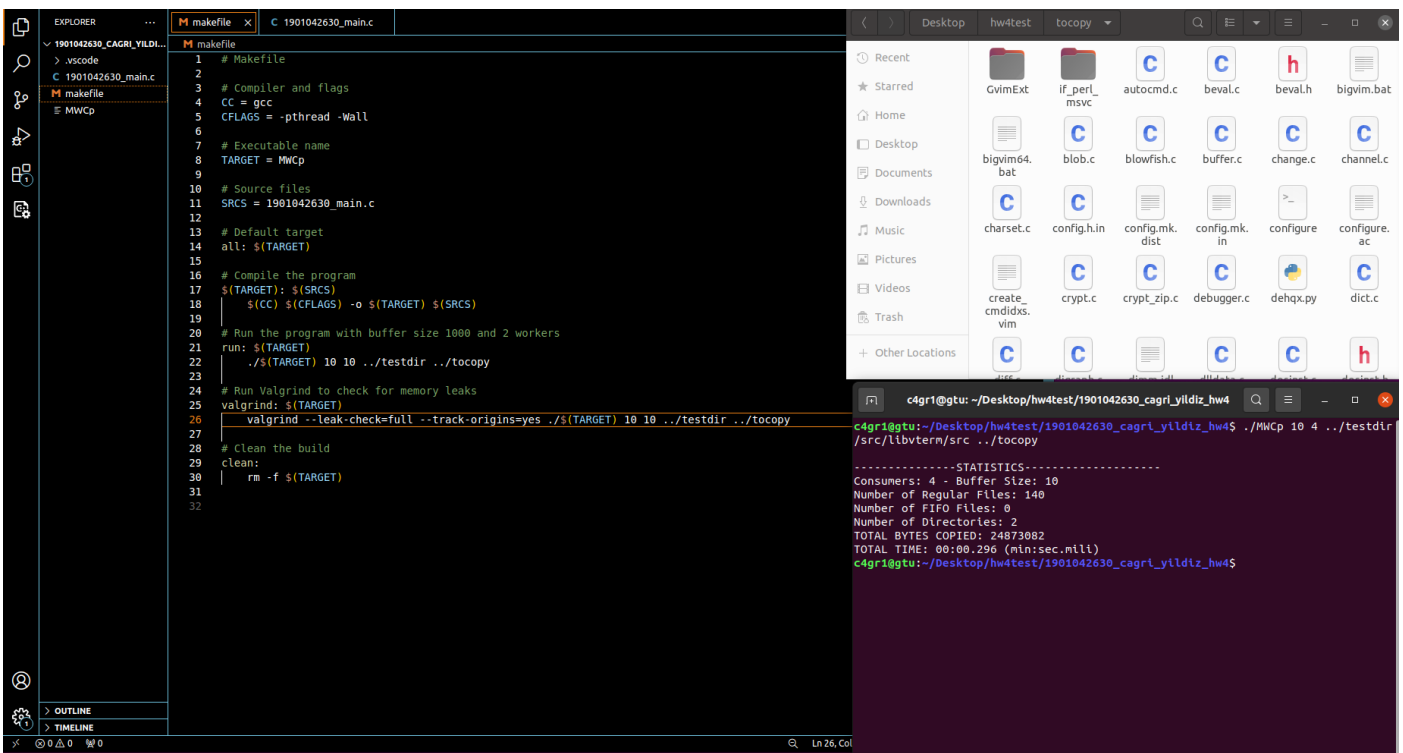


```
1 # Makefile
2
3 # Compiler and flags
4 CC = gcc
5 CFLAGS = -pthread -Wall
6
7 # Executable name
8 TARGET = MWCp
9
10 # Source files
11 SRCS = 1901042630_main.c
12
13 # Default target
14 all: $(TARGET)
15
16 # Compile the program
17 $(TARGET): $(SRCS)
18 | $(CC) $(CFLAGS) -o $(TARGET) $(SRCS)
19
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) 10 10 ../testdir ../tocopy
27
28 # Clean the build
29 clean:
30 | rm -f $(TARGET)
31
32
```

```
====8240====
-----STATISTICS-----
Consumers: 10 - Buffer Size: 10
Number of Regular Files: 194
Number of FIFO Files: 0
Number of Directories: 7
TOTAL BYTES COPIED: 25009600
TOTAL TIME: 00:02.1414 (min:sec.mlll)
====8240====
====8240==== HEAP SUMMARY:
====8240==== in use at exit: 1,654 bytes in 4 blocks
====8240==== total heap usage: 27 allocs, 23 frees, 350,334 bytes allocated
====8240====
====8240==== LEAK SUMMARY:
====8240==== definitely lost: 0 bytes in 0 blocks
====8240==== indirectly lost: 0 bytes in 0 blocks
====8240==== possibly lost: 0 bytes in 0 blocks
====8240==== still reachable: 1,654 bytes in 4 blocks
====8240==== suppressed: 0 bytes in 0 blocks
====8240==== Rerun with --leak-check=full to see details of leaked memory
====8240====
====8240==== For lists of detected and suppressed errors, rerun with: -s
====8240==== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
c4gr1@gtu:~/Desktop/hw4test/1901042630_cagri_yildiz_hw4$
```

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

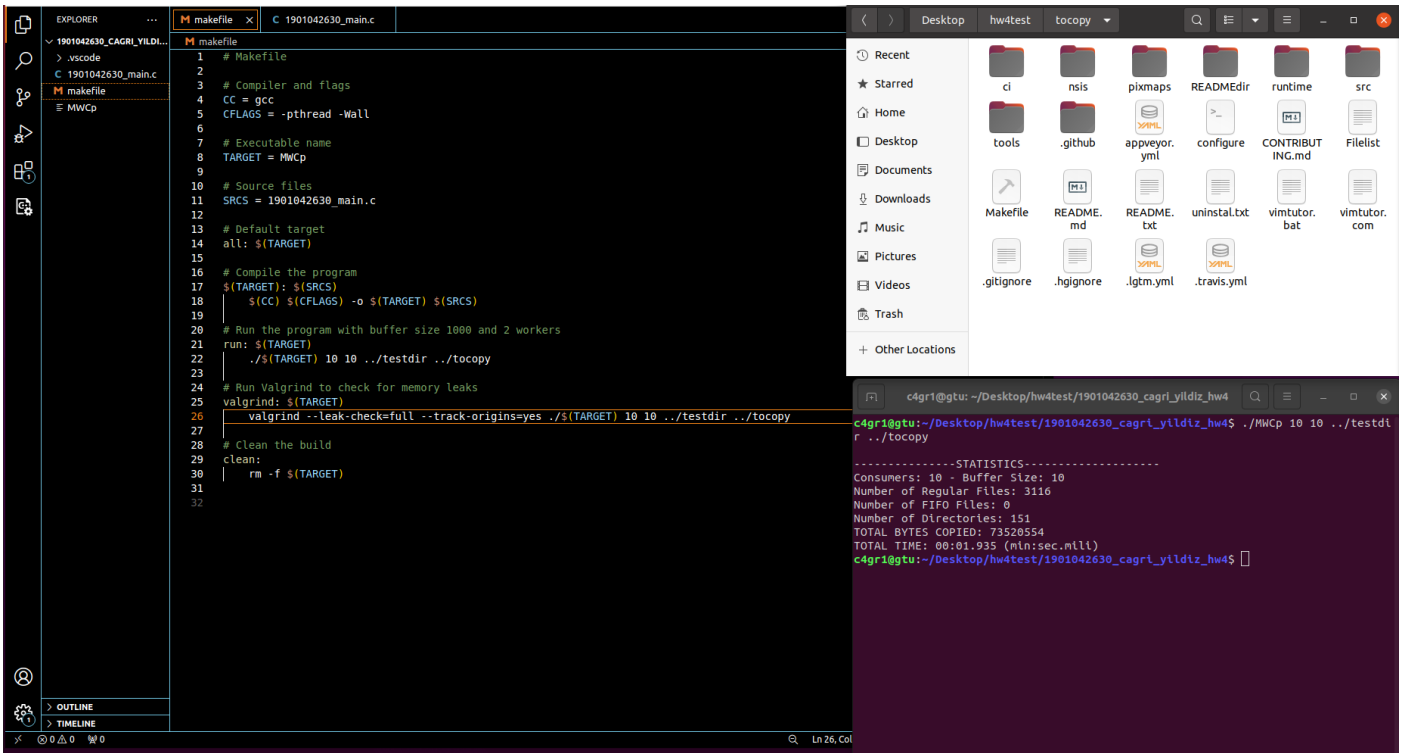
- Test2: ./MWCp 10 4 ../testdir/src/libvterm/src ../tocopy



```
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) 10 10 ../testdir ../tocopy
27
28 # Clean the build
29 clean:
30 | rm -f $(TARGET)
31
32
```

```
====8240====
-----STATISTICS-----
Consumers: 4 - Buffer Size: 10
Number of Regular Files: 140
Number of FIFO Files: 0
Number of Directories: 2
TOTAL BYTES COPIED: 24873082
TOTAL TIME: 00:00.296 (min:sec.mlll)
c4gr1@gtu:~/Desktop/hw4test/1901042630_cagri_yildiz_hw4$
```

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



- Invalid commands

```
c4gr1@gtu:~/Desktop/hw4test/1901042630_cagri_yildiz_hw4$ ./MWCp 10 10 ../testdir ../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