

CSE-344 SYSTEM PROGRAMMING

SPRING 2024

HOMEWORK 3

REPORT

ONUR ATASEVER

210104004087

I applied worker manager approach for threads as requested. I created the Manager, the desired number of threads and the desired size buffer. The manager calls the copy\_directory function and adds it to the buffer. Since the place where the buffer is added is a region that can be accessed by all threads, it is considered a critical region and is protected with a mutex. The buffer size was checked with each addition to the buffer. If the buffer is full, the thread goes to sleep and continues to block until it receives the buffer is not full signal. When the buffer is not full, every time an element is added, the buffer is not empty signal is sent. Thus, if there is a worker waiting by blocking it because the buffer is empty, it is woken up. If what is wanted to be written to the buffer is a folder rather than a file, this folder is recursively sent back to the copy\_directory function.

Worker threads check the buffer size when they start running. If the buffer is empty, they are blocked and start waiting. After receiving the Buffer is not empty signal, it wakes up and continues to do its job. Retrieves data from the buffer and deletes it. According to the information they receive from Buffer, they determine the file type and increase the necessary counters. Mutex protections are used in every operation regarding the buffer as it is a critical area and to provide synchronization. Then, copying is done from the source path to the destination path. It is copying byte by byte. So with valgrind it takes time to run.

Signal handler created for CTRL+C. Each thread runs when the running variable is 1. If this signal comes, the running variable becomes 0 and the threads stop running. In addition, blocked threads are woken up in the signal handler so that the program reaches the end properly, performs the freeing operations and ends.

To run test 1 with valgrind you can use “make valgrind1” command, for test 2 “make valgrind2” and for test 3 “make valgrind3”.

For each read, write, open system calls for file, error check is done. For opening directory or getting file status error checks are done. For sigaction function, error checks is done and for the command line arguments error check is done.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

==32148== Memcheck, a memory error detector
==32148== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==32148== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==32148== Command: ./MwCp
==32148==
Usage: ./MwCp <buffer_size> <num_workers> <src_dir> <dest_dir>
==32148==
==32148== HEAP SUMMARY:
==32148==   in use at exit: 0 bytes in 0 blocks
==32148== total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==32148==
==32148== All heap blocks were freed -- no leaks are possible
==32148==
==32148== For lists of detected and suppressed errors, rerun with: -s
==32148== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
make: *** [Makefile:30: runV3] Error 1
root@DESKTOP-PLJPMRB:/home/hw4/code#
```

### Pseudo Code For Main:

```
FUNCTION main(argc, argv)

  IF argc != 5 THEN

    PRINT "Usage: %s <buffer_size> <num_workers> <src_dir> <dest_dir>"

    EXIT_FAILURE

  SETUP signal handler for SIGINT to call intHandler

  PARSE command line arguments to get buffer_size, num_workers, src_dir,
dest_dir

  ALLOCATE memory for worker threads

  INITIALIZE buffer with buffer_size

  GET current time as start time

  CREATE manager thread to run manager function

  FOR each worker thread

    CREATE worker thread to run worker function

  JOIN manager thread

  FOR each worker thread

    JOIN worker thread

  GET current time as end time

  CALCULATE execution time

  PRINT statistics: number of files, directories, total bytes copied, execution time

  FREE allocated memory for worker threads

  DESTROY buffer

  RETURN 0

END FUNCTION
```

### **Pseudo Code For Manager:**

```
FUNCTION manager(arg)

    CALL copy_directory(src_dir, dest_dir)

    LOCK buffer.mutex

    SET buffer.done to 1

    BROADCAST buffer.not_empty // Wake up all worker threads

    UNLOCK buffer.mutex

    RETURN NULL

END FUNCTION
```

### **Pseudo Code For copy\_directory:**

```
FUNCTION copy_directory(src, dest)

    OPEN source directory

    CREATE destination directory

    WHILE there are more entries in the source directory AND running is TRUE

        IF entry is not "." or ".." THEN

            CONSTRUCT src_path and dest_path

            GET file status

            IF file is a regular file THEN

                ADD file pair to buffer

            ELSE IF file is a directory THEN

                INCREMENT directory count

                RECURSIVELY call copy_directory(src_path, dest_path)

        CLOSE source directory

    END FUNCTION
```

## Pseudo Code For Worker:

FUNCTION worker(arg)

    WHILE running IS TRUE

        REMOVE file pair from buffer

        IF file pair is empty THEN

            BREAK

        OPEN source and destination files

        COPY file contents from source to destination

        CLOSE source and destination files

    RETURN NULL

END FUNCTION

## Mutex Protection And Conditional wait Examples:

```
// Add a file pair to the buffer
void add_to_buffer(Buffer *buffer, const char *src, const char *dest)
{
    // printf("Locktan önce\n");
    pthread_mutex_lock(&buffer->mutex); // Lock the buffer
    // printf("Locktan sonra\n");
    while (buffer->count == buffer->buffer_size && running)
    {
        pthread_cond_wait(&buffer->not_full, &buffer->mutex); // If the buffer is full, wait for it to be not full
        if (!running) // If the program is not running, unlock the mutex to prevent other mutexes wait and return
        {
            pthread_mutex_unlock(&buffer->mutex);
            return;
        }
    }
    // printf("While'dan sonra\n");
    strncpy(buffer->buffer[buffer->tail].src, src, MAX_FILENAME_LENGTH);
    strncpy(buffer->buffer[buffer->tail].dest, dest, MAX_FILENAME_LENGTH);
    buffer->tail = (buffer->tail + 1) % buffer->buffer_size; // Update the tail
    buffer->count++; // Increment the count
    pthread_cond_signal(&buffer->not_empty); // Signal that the buffer is not empty
    pthread_mutex_unlock(&buffer->mutex); // Unlock the buffer
}
```

```
// Remove a file pair from the buffer
FilePair remove_from_buffer(Buffer *buffer)
{
    pthread_mutex_lock(&buffer->mutex); // Lock the buffer
    while (buffer->count == 0 && !buffer->done && running) // If the buffer is empty, wait for it to be not empty
    {
        pthread_cond_wait(&buffer->not_empty, &buffer->mutex);
        if (!running) // If the program is not running, unlock the mutex to prevent other mutexes wait and return
        {
            // printf("Running 0, return edilecek\n");
            pthread_mutex_unlock(&buffer->mutex);
            return;
        }
    }
}
```

### Signal Handler Example For CTRL+C:

```
// All the workers are not ending because of the signal handler. I tried to use pthread_cancel but it didn't work.
void intHandler(int dummy)
{
    running = 0; // Set running to 0 to signal all threads to exit from while loop
    pthread_cond_signal(&buffer.not_full);
    pthread_cond_signal(&buffer.not_empty);
    buffer.done = 1;
    pthread_mutex_unlock(&buffer.mutex);
    pthread_cond_broadcast(&buffer.not_empty); // Wake up all workers
    pthread_cond_broadcast(&buffer.not_full); // Wake up all workers
}
```

Test 1 Output:

The screenshot displays a Windows Subsystem for Linux (WSL) environment. The Explorer pane on the left shows the file structure of the project, including `main2.c`, `Makefile`, and `valgrind-out.txt`. The Code editor shows the C program `main2.c`, which implements a buffer management system using mutexes and condition variables. The program includes a `remove_from_buffer` function and a `main` function that tests the buffer operations. The Terminal pane shows the output of the compilation and execution of the program. The output includes the Valgrind error checker results, which show that the program executed successfully without any memory errors. The output also includes statistics such as the number of regular files, FIFO files, directories, and the total bytes copied. The output concludes with a summary of detected and suppressed errors, indicating that no errors were found.

```
File Edit Selection View Go Run Terminal Help
hwa [WSL: Ubuntu]

EXPLORER
hwa [WSL: Ubuntu]
  code
  main2.c
  Makefile
  valgrind-out.txt
  testdir
  Yeri Masor
  main2.c
  Makefile
  MWCpa

code
main2.c
151 void add_to_buffer(Buffer *buffer, const char *src, const char *dest)
152 {
153     pthread_mutex_lock(&buffer->mutex); // Lock the buffer
154     pthread_mutex_unlock(&buffer->mutex); // Unlock the buffer
155 }
156
157 // Remove a file pair from the buffer
158 filePair remove_from_buffer(Buffer *buffer)
159 {
160     pthread_mutex_lock(&buffer->mutex); // Lock the buffer
161     while (buffer->count == 0 && !buffer->done && running) // If the buffer is empty, wait for it to be not empty
162     {
163         pthread_cond_wait(&buffer->not_empty, &buffer->mutex);
164         if (!running) // If the program is not running, unlock the mutex to prevent other mutexes wait and return
165         {
166             // printf("Running 0, return nillock\n");
167             pthread_mutex_unlock(&buffer->mutex);
168         }
169     }
170 }

TERMINAL
root@DESKTOP-PL3PWB: /home/hwa/code#
==24617== Memcheck, a memory error detector
==24617== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==24617== Using Valgrind-3.18.1 and LDBEX; rerun with -h for copyright info
==24617== Command: ./main2
==24617==
-----STATISTICS-----
Consumers: 10 - Buffer Size: 10
Number of Regular File: 194
Number of FIFO File: 0
Number of Directory: 7
TOTAL BYTES COPIED: 25009680
Execution time: 61 seconds and 6034878 micros
==24617==
==24617== HEAP SUMMARY:
==24617==    in use at exit: 0 bytes in 0 blocks
==24617==   total heap usage: 22 allocs, 22 frees, 271,744 bytes allocated
==24617==
==24617== All heap blocks were freed -- no leaks are possible
==24617==
==24617== For lists of detected and suppressed errors, rerun with: -s
==24617== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
root@DESKTOP-PL3PWB: /home/hwa/code#
```

### Test 1 Output With CTRL+C:

The image shows a Visual Studio Code editor window with a C program named 'main2.c' open. The code is a multi-threaded program that uses pthreads and Valgrind for memory checking. The code is as follows:

```

357 }
358
359 // All the workers are not ending because of the signal handler. I tried to use pthread_cancel but it didn't work.
360 void inthandler(int dummy)
361 {
362     printf("CTRL-C caught. Exiting...\n");
363     running = 0; // Set running to 0 to signal all threads to exit from while loop
364     pthread_cond_signal(&buffer_not_full);
365     pthread_cond_signal(&buffer_not_empty);
366     buffer_done = 1;
367     pthread_mutex_unlock(&buffer_mutex);
368     pthread_cond_broadcast(&buffer_not_empty); // Wake up all workers
369     pthread_cond_broadcast(&buffer_not_full); // Wake up all workers
370 }

```

The terminal output shows the execution of the program using Valgrind. The output is as follows:

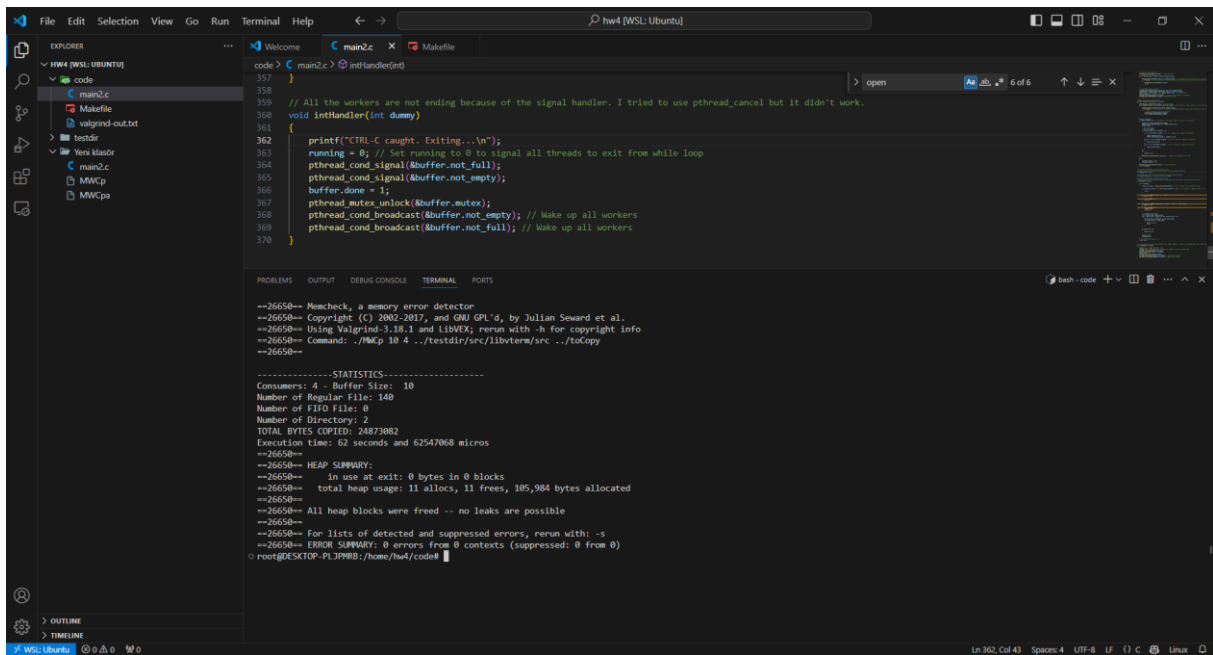
```

==26280== Memcheck, a memory error detector
==26280== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==26280== Using Valgrind-3.18.1 and LibVex; rerun with -h for copyright info
==26280== Command: ./main2Valgrind 10 10 ../testdir/src/libvex ../tucopy4
==26280==
*CTRL-C caught. Exiting...

-----STATISTICS-----
Consumers: 10 - Buffer Size: 10
Number of Regular File: 70
Number of FIFO File: 0
Number of Directory: 4
TOTAL BYTES COPIED: 913239
Execution time: 3 seconds and 3777726 micros
==26280==
==26280== HEAP SUMMARY:
==26280==   in use at exit: 0 bytes in 0 blocks
==26280==   total heap usage: 19 allocs, 19 frees, 173,296 bytes allocated
==26280==
==26280== All heap blocks were freed -- no leaks are possible
==26280==
==26280== For lists of detected and suppressed errors, rerun with: -s
==26280== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

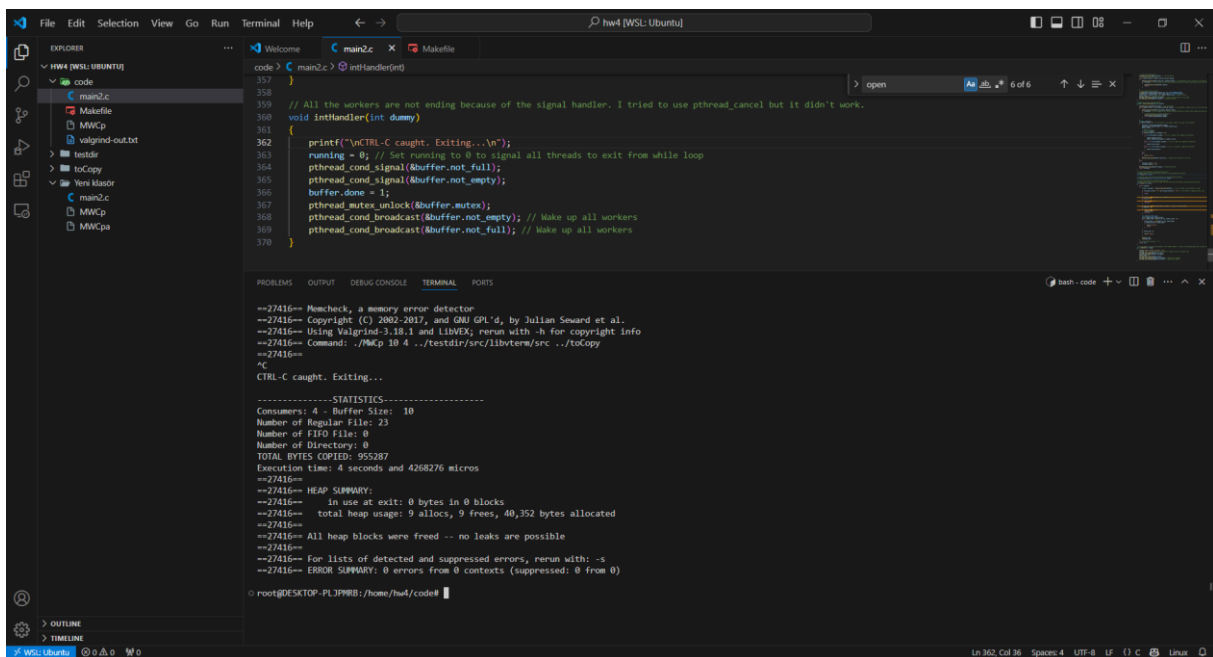
## Test 2 Output:



```
code > C main2.c > intHandler(int dummy)
357 }
358
359 // All the workers are not ending because of the signal handler. I tried to use pthread_cancel but it didn't work.
360 void intHandler(int dummy)
361 {
362     printf("CTRL-C caught. Exiting...\n");
363     running = 0; // Set running to 0 to signal all threads to exit from while loop
364     pthread_cond_signal(&buffer.not_full);
365     pthread_cond_signal(&buffer.not_empty);
366     buffer.done = 1;
367     pthread_mutex_unlock(&buffer.mutex);
368     pthread_cond_broadcast(&buffer.not_empty); // Wake up all workers
369     pthread_cond_broadcast(&buffer.not_full); // Wake up all workers
370 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
--26658-- Memcheck, a memory error detector
--26658-- Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
--26658-- Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
--26658-- Command: ./Mkcp 10 4 ../testdir/src/libbterm/src ../toCopy
--26658--
-----STATISTICS-----
Consumers: 4 - Buffer Size: 10
Number of Regular File: 140
Number of FIFO File: 0
Number of Directory: 2
TOTAL BYTES COPIED: 24873082
Execution time: 62 seconds and 62547068 micros
--26658--
--26658-- HEAP SUMMARY:
--26658--   in use at exit: 0 bytes in 0 blocks
--26658--   total heap usage: 11 allocs, 11 frees, 105,984 bytes allocated
--26658--
--26658-- All heap blocks were freed -- no leaks are possible
--26658--
--26658-- For lists of detected and suppressed errors, rerun with: -s
--26658-- ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
root@DESKTOP-PLJPMWB:/home/hw4/code#
```

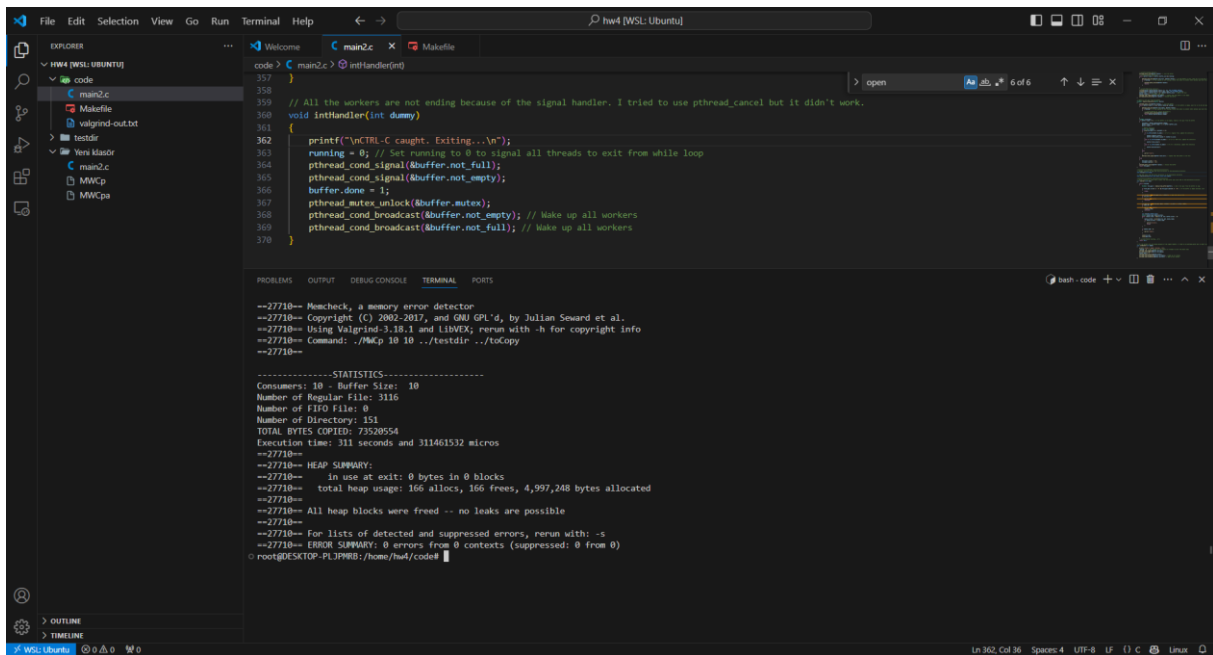
## Test 2 Output With CTRL+C:



```
code > C main2.c > intHandler(int dummy)
357 }
358
359 // All the workers are not ending because of the signal handler. I tried to use pthread_cancel but it didn't work.
360 void intHandler(int dummy)
361 {
362     printf("\nCTRL-C caught. Exiting...\n");
363     running = 0; // Set running to 0 to signal all threads to exit from while loop
364     pthread_cond_signal(&buffer.not_full);
365     pthread_cond_signal(&buffer.not_empty);
366     buffer.done = 1;
367     pthread_mutex_unlock(&buffer.mutex);
368     pthread_cond_broadcast(&buffer.not_empty); // Wake up all workers
369     pthread_cond_broadcast(&buffer.not_full); // Wake up all workers
370 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
--27416-- Memcheck, a memory error detector
--27416-- Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
--27416-- Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
--27416-- Command: ./Mkcp 10 4 ../testdir/src/libbterm/src ../toCopy
--27416--
^C
CTRL-C caught. Exiting...
-----STATISTICS-----
Consumers: 4 - Buffer Size: 10
Number of Regular File: 23
Number of FIFO File: 0
Number of Directory: 0
TOTAL BYTES COPIED: 955287
Execution time: 4 seconds and 4268276 micros
--27416--
--27416-- HEAP SUMMARY:
--27416--   in use at exit: 0 bytes in 0 blocks
--27416--   total heap usage: 9 allocs, 9 frees, 40,352 bytes allocated
--27416--
--27416-- All heap blocks were freed -- no leaks are possible
--27416--
--27416-- For lists of detected and suppressed errors, rerun with: -s
--27416-- ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
root@DESKTOP-PLJPMWB:/home/hw4/code#
```

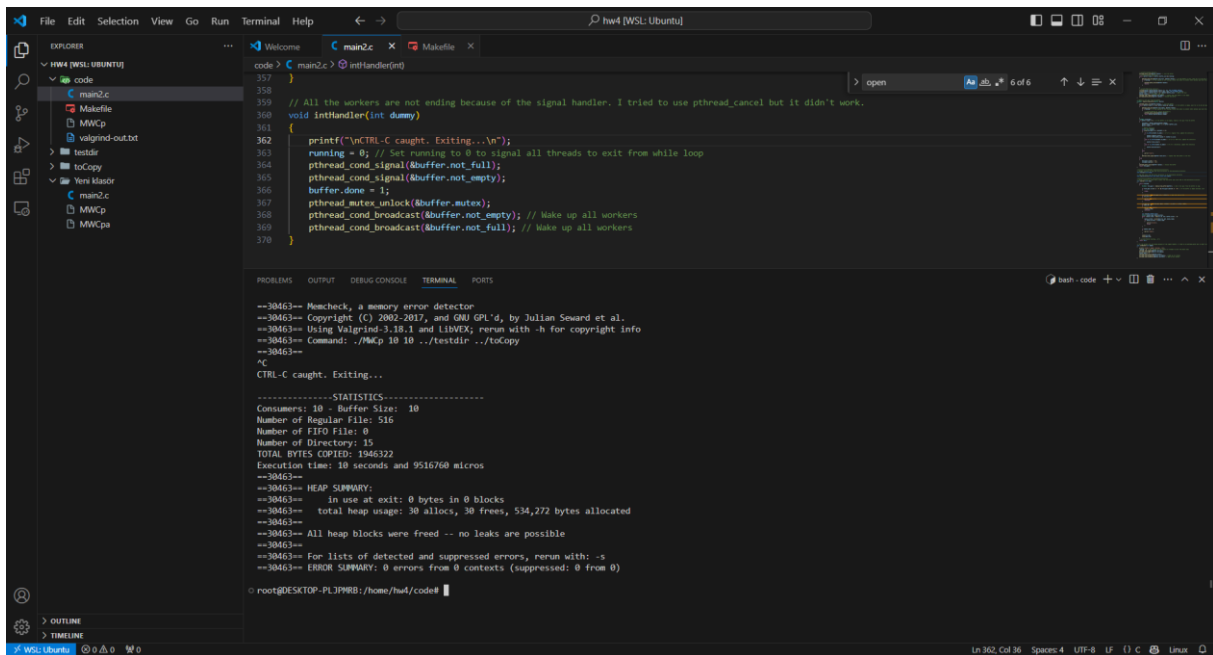
## Test 3 Output:



```
code > C main2.c > intHandler(int dummy)
357 }
358
359 // All the workers are not ending because of the signal handler. I tried to use pthread_cancel but it didn't work.
360 void intHandler(int dummy)
361 {
362     printf("\nCTRL-C caught. Exiting...\n");
363     running = 0; // Set running to 0 to signal all threads to exit from while loop
364     pthread_cond_signal(&bbuffer.not_full);
365     pthread_cond_signal(&bbuffer.not_empty);
366     bbuffer.done = 1;
367     pthread_mutex_unlock(&bbuffer.mutexp);
368     pthread_cond_broadcast(&bbuffer.not_empty); // Wake up all workers
369     pthread_cond_broadcast(&bbuffer.not_full); // Wake up all workers
370 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
--27710-- Memcheck, a memory error detector
--27710-- Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
--27710-- Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
--27710-- Command: ./Mkcp 10 10 ../testdir ../toCopy
--27710--
-----STATISTICS-----
Consumers: 10 - Buffer Size: 10
Number of Regular File: 3116
Number of FIFO File: 0
Number of Directory: 151
TOTAL BYTES COPIED: 73520554
Execution time: 311 seconds and 311461532 micros
--27710--
--27710-- HEAP SUMMARY:
--27710--   in use at exit: 0 bytes in 0 blocks
--27710--   total heap usage: 166 allocs, 166 frees, 4,997,248 bytes allocated
--27710--
--27710-- All heap blocks were freed -- no leaks are possible
--27710--
--27710-- For lists of detected and suppressed errors, rerun with: -s
--27710-- ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
root@DESKTOP-PLJPM8B:/home/hw4/code#
```

## Test 3 Output With CTRL + C:



```
code > C main2.c > intHandler(int dummy)
357 }
358
359 // All the workers are not ending because of the signal handler. I tried to use pthread_cancel but it didn't work.
360 void intHandler(int dummy)
361 {
362     printf("\nCTRL-C caught. Exiting...\n");
363     running = 0; // Set running to 0 to signal all threads to exit from while loop
364     pthread_cond_signal(&bbuffer.not_full);
365     pthread_cond_signal(&bbuffer.not_empty);
366     bbuffer.done = 1;
367     pthread_mutex_unlock(&bbuffer.mutexp);
368     pthread_cond_broadcast(&bbuffer.not_empty); // Wake up all workers
369     pthread_cond_broadcast(&bbuffer.not_full); // Wake up all workers
370 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
--30463-- Memcheck, a memory error detector
--30463-- Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
--30463-- Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
--30463-- Command: ./Mkcp 10 10 ../testdir ../toCopy
--30463--
CTRL-C caught. Exiting...
-----STATISTICS-----
Consumers: 10 - Buffer Size: 10
Number of Regular File: 516
Number of FIFO File: 0
Number of Directory: 15
TOTAL BYTES COPIED: 1946322
Execution time: 10 seconds and 9516760 micros
--30463--
--30463-- HEAP SUMMARY:
--30463--   in use at exit: 0 bytes in 0 blocks
--30463--   total heap usage: 30 allocs, 30 frees, 534,272 bytes allocated
--30463--
--30463-- All heap blocks were freed -- no leaks are possible
--30463--
--30463-- For lists of detected and suppressed errors, rerun with: -s
--30463-- ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
root@DESKTOP-PLJPM8B:/home/hw4/code#
```