_tmp _tmp _tmp _tmp _tmp _tmp _tmp _tmp _tmp

Gebze Technical University
Computer Engineering

SYSTEM PROGRAMMING

Emre YILMAZ: 1901042606

Homework 4 and 5, May 2024

Profesor: Erkan Zergeroglu

1901042606

May 25, 2024

Contents

1 Important Notes

1. **This homework's source code and report will be uploaded for both Homework-4 and Homework-5. As requested in HW-5, Barriers and Condition Variables have been used. You can find their usage in the report.**

2. All cases and signal handling described in the homework PDF are working successfully. You can find them in the Test section.

3. The output of first test case using Valgrind has been saved as a text file and submitted with the homework.

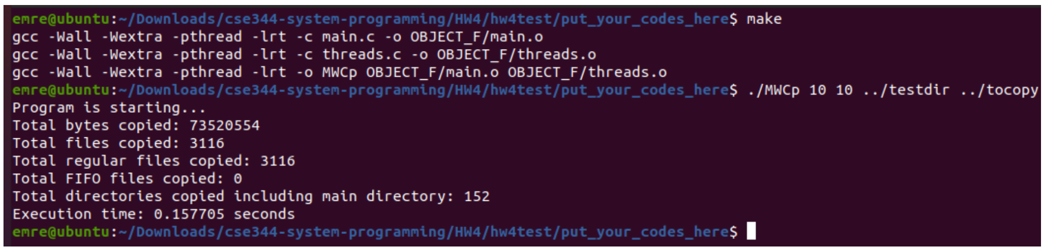
4. The third test case, along with Signal Handling, has also been tested and the Valgrind output has been saved as a text file and submitted with the homework.

5. Particular attention has been paid to ensure there are no leaks during CTRL-C signal handling. This case has been recorded on video and the link has been placed in the relevant section (section 8.5).

6. Although PDF says Valgrind will be tested with test case-1, if you want to try to get Valgrind output of test case-3, it may take a few seconds due to the high number of threads and Valgrind usage. Please be patient and wait, it works. Similarly, I recommend waiting a few seconds if you use Valgrind for the CTRL-C signal test in this case.

7. While showing number of directories that are copied, it is included the folder itself. So, it starts with 1.

2 How to Run



```
emre@ubuntu:~/Downloads/cse344-system-programming/HW4/hw4test/put_your_codes_here$ make
gcc -Wall -Wextra -pthread -lrt -c main.c -o OBJECT_F/main.o
gcc -Wall -Wextra -pthread -lrt -c threads.c -o OBJECT_F/threads.o
gcc -Wall -Wextra -pthread -lrt -o MWCp OBJECT_F/main.o OBJECT_F/threads.o
emre@ubuntu:~/Downloads/cse344-system-programming/HW4/hw4test/put_your_codes_here$ ./MWCp 10 10 ../testdir ../tocopy
Program is starting...
Total bytes copied: 73520554
Total files copied: 3116
Total regular files copied: 3116
Total FIFO files copied: 0
Total directories copied including main directory: 152
Execution time: 0.157705 seconds
emre@ubuntu:~/Downloads/cse344-system-programming/HW4/hw4test/put_your_codes_here$
```

Fig-0: How to Run

3 Mutexes

3.1 buffer_mutex

This mutex, in addition to providing synchronization between workers, also facilitates synchronization between workers and the manager. All entries and exits to critical sections are made through this mutex. Details are in the Threads section of the report.

4 Condition Variables

4.1 buffer_not_full

If there is no space for a new entry in the buffer, the manager thread is blocked and waits through this condition. When a worker takes an entry from the buffer and frees up space, it signals through this condition to wake the manager thread, allowing it to enter a new entry into the buffer and continue working. Details are in the Threads section of the report.

4.2 buffer_not_empty

If there is no entry in the buffer, the worker thread is blocked and waits through this condition. When the manager inputs a new entry into the buffer, they signal through this condition to wake the worker thread, allowing it to retrieve the necessary variable from the buffer and continue working. Details are in the Threads section of the report.

4.3 fd_limit

As mentioned in the PDF, there is a certain limit to opening files in Linux. In this case, if a large number like 1024 is given for the buffer parameter, the manager will not be able to open files and will have to skip some. Therefore, it is important to track the number of open files. For this reason, the fd_limit condition variable has been defined and used. When the number of open files reaches a certain limit, the manager is blocked and waits for the workers to close some files. For this waiting process, a condition variable has been used, and workers send a signal to the relevant variable when they close files, thus waking the manager thread.

5 Barriers

In my design, the functions that initialize and destroy the condition variables and mutexes are managed by the manager thread. Therefore, the use of a barrier is essential for synchronizing the initialization, destruction, and the start times of the threads.

5.1 startBarrier

Worker threads should not start running before the mutexes and condition variables are initialized. Therefore, the startBarrier barrier is initialized to the total number of worker threads plus one for the manager. Worker threads get stuck at the barrier and wait before they start running. After the manager thread completes the initialization processes and is ready to work, it also reaches the barrier point. From this point, both the manager and workers begin working, thus ensuring synchronization.

5.2 endBarrier

Similarly, before the manager thread finishes working, it must ensure that all worker threads have completed their tasks and processed the final entries in the buffer. For this purpose, an endBarrier barrier has been initialized for the total number of worker threads plus one for the manager. After the manager thread writes the necessary files to the buffer, it exits the loop and gets stuck at the barrier before destroying the mutexes and condition variables. Worker threads, after clearing the last elements in the buffer, also exit the loop and reach the barrier point. Once all active threads reach this point, the manager thread completes the cleanup tasks, and all threads exit. This ensures synchronization during the termination of the threads.

6 Threads

6.1 Manager Thread

First, the parameters coming from the argument are solved and stored in the necessary variables. Immediately afterwards, the condition variables and mutexes are initialized and wait at the barrier point. When the worker threads also reach the barrier point, the process_directory function is called and the actual work begins.

In the process_directory function, we recursively process the source directory to copy all regular files and FIFO files to the destination directory. We ensure that directory creation, file opening, and copying are done in a synchronized manner using mutexes to prevent race conditions. We also count the number of directories, regular files, and FIFO files copied. But, let me explain this function step by step:

- Open source directory
- Create destination directory.
- Increment directory count. But it is critical section. Use mutexes while doing it. Since this process_directory function is recursively called when a new directory is detected in the target directory, we can count the directories in this way.
- Read directory entries.
- Generate destination and source paths. Put them into a string.
- If a new directory is encountered, make a recursive call to handle that directory.
- If a FIFO or regular file is encountered, go on.
- Prepare the buffer. Put the source and destination strings into the struct file_info
- Open source file, if the source file is FIFO file, add the flag NON_BLOCK since we must not be blocked while reading FIFO.
- Open destination file with flags `O_WRONLY | O_CREAT | O_TRUNC`.
- If the source file is FIFO file, increase FIFO counter. If the source file is a regular file, increase reguler file counter. But, this step is critical section. We used mutex while increasing counters.
- Check the buffer index; if there is no space in the buffer, use the cond_wait function and the buffer_not_full condition to block and wait until space is available in the buffer.
- Check the number of open files. If it has reached the limit, use the cond_wait function and the fd_limit condition to block and wait until some files are closed and a signal is received.
- Add file descriptors to buffer.

After process_directory function is finished, make the variable `done` 1. It means that manager thread is finished, there will be no buffer anymore. After, that if there is a worker that sleeps and waits for buffer they are signaled. They will get the signal and stops working since the variable `done` is 1. This operation is crittical section. We used mutex while changing the variable `done`.

After that, this manager thread waits in the barrier to all worker threads are reached the barrier point. After all worker threads finished their task and reached the barrier, manager thread cleans up all condition variables and mutex. Then exit.

6.2 Worker Threads

The worker_thread function is responsible for processing files from a shared buffer and copying them from the source to the destination directory. It uses barriers to synchronize the start and end of the worker threads and mutexes to ensure thread-safe access to shared resources. The function waits at the start barrier until all worker threads are ready, processes files in a loop until signaled to stop, and waits at the end barrier before exiting. Let me explain it step by step.

1. Wait in the barrier. When manager thread prepares condition variables and mutex, it is going to reach the barrier, too and all workers and manager will continue.
2. Increment active thread count. It is critical section. Lock and unlock mutex while doing this.
3. Go to loop for process the buffer.
4. Go to critical section and lock the mutex.
5. Wait the buffer to become not empty. Use condition variable to do this.
6. Check all files are handled successfuly, I mean, check the variable `done` is 1 or not. If it is 1, it means that manager thread made it 1, there will be no buffer. Unlock the mutex and break the loop.
7. If manager thread is not finished yet, take the next entry from buffer. Awake the manager, signal that the buffer is not full for the manager thread to add more files. Then, exit the critical section by unlocking the mutex.
8. Then call the copy_file function. This function will copy the soure to destination.
9. Inside the copy_file function, Within this function, use the target file descriptor to read 4096 bytes from the source file, and then write the bytes you read using the target file descriptor. Since the write operation may not be atomic, continue writing until you are sure that all the bytes read have been written. Break the loop when the Read function does not read any bytes, and enter the critical section to update the total number of bytes read. Then unlock the mutex and exit the critical section. Return the worker function after copying file successfuly.
10. After that, close the buffer target and source file descriptors. After closing file descriptors, send signal to fd_limit condition variable. If manager is not able to open new files, let it continue since it closed some files.
11. If the manager thread is finished its task and it made the variable `done` 1, break the main loop and go into critical section. Decrease active threads and exit the critical section.

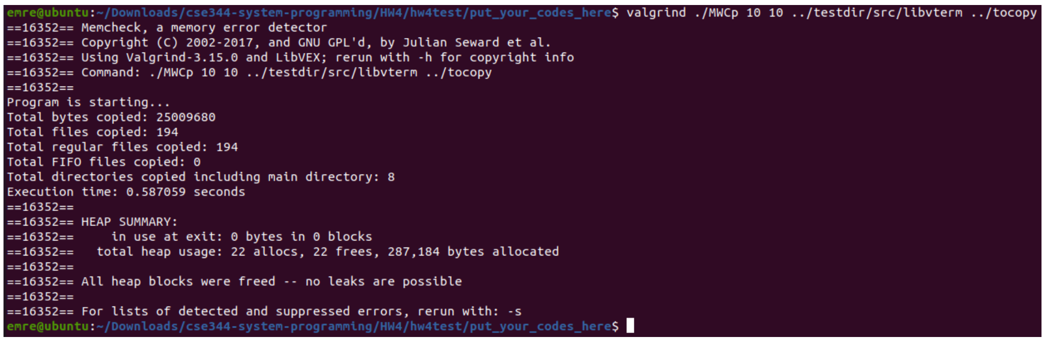
12. Then, wait in the barrier to exit the thread. When all workers reached this barrier, manager thread will know all workers are finished their task and it is going to destroy condition variables and mutex. Then all threads exit.

7 SIGINT Handling

When the program receives a CTRL-C signal, it enters a critical section and sets the sigint_received variable to 1. The manager thread, while traversing directories or files within a directory in the process_directory function, checks this variable. If this variable is set to 1, the process_directory function returns as if no files are left, and the manager thread waits at the barrier point for the worker threads to finish the buffer at hand and reach the barrier. Afterwards, they all terminate together. In this way, memory leaks are prevented, and the program is successfully terminated.

8 Test Results

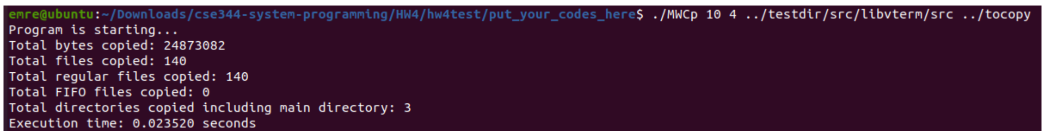
8.1 Test Case - 1



```
emre@ubuntu:~/Downloads/cse344-system-programming/HW4/hw4test/put_your_codes_here$ valgrind ./MWCp 10 10 ../testdir/src/libvterm ../tocopy
==16352== Memcheck, a memory error detector
==16352== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==16352== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==16352== Command: ./MWCp 10 10 ../testdir/src/libvterm ../tocopy
==16352==
Program is starting...
Total bytes copied: 25009680
Total files copied: 194
Total regular files copied: 194
Total FIFO files copied: 0
Total directories copied including main directory: 8
Execution time: 0.587059 seconds
==16352==
==16352== HEAP SUMMARY:
==16352==     in use at exit: 0 bytes in 0 blocks
==16352==   total heap usage: 22 allocs, 22 frees, 287,184 bytes allocated
==16352==
==16352== All heap blocks were freed -- no leaks are possible
==16352==
==16352== For lists of detected and suppressed errors, rerun with: -s
emre@ubuntu:~/Downloads/cse344-system-programming/HW4/hw4test/put_your_codes_here$ 
```

Fig-1: Test Case-1 Output

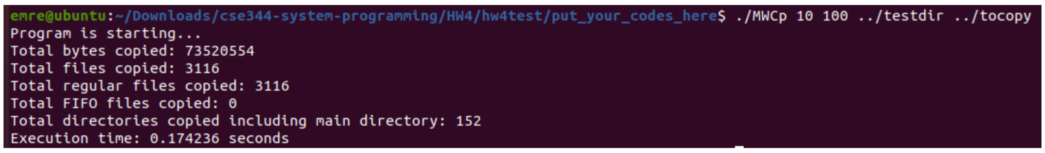
8.2 Test Case - 2



```
emre@ubuntu:~/Downloads/cse344-system-programming/HW4/hw4test/put_your_codes_here$ ./MWCp 10 4 ../testdir/src/libvterm/src ../tocopy
Program is starting...
Total bytes copied: 24873082
Total files copied: 140
Total regular files copied: 140
Total FIFO files copied: 0
Total directories copied including main directory: 3
Execution time: 0.023520 seconds
```

Fig-3: Test Case-2 Output

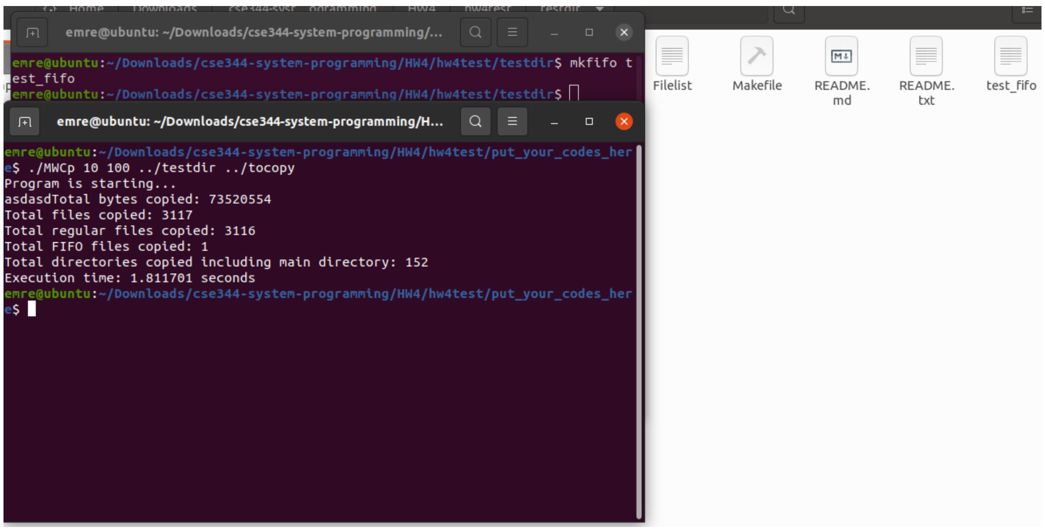
8.3 Test Case - 3



```
emre@ubuntu:~/Downloads/cse344-system-programming/HW4/hw4test/put_your_codes_here$ ./MWCp 10 100 ../testdir ../tocopy
Program is starting...
Total bytes copied: 73520554
Total files copied: 3116
Total regular files copied: 3116
Total FIFO files copied: 0
Total directories copied including main directory: 152
Execution time: 0.174236 seconds
```

Fig-4: Test Case-3 Output

8.4 Fifo Test



```
emre@ubuntu:~/Downloads/cse344-system-programming/HW4/hw4test/testdir$ mkfifo t
est_fifo
emre@ubuntu:~/Downloads/cse344-system-programming/HW4/hw4test/testdir$
```

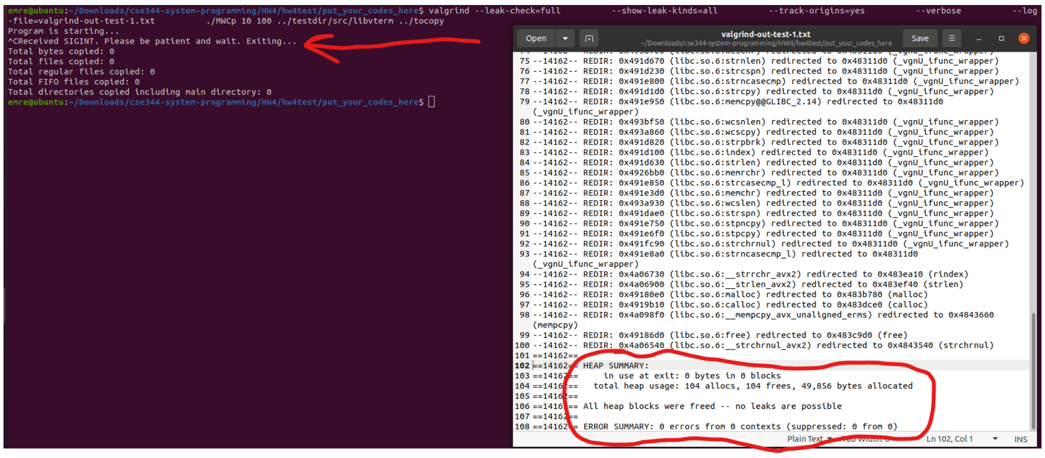
```
emre@ubuntu:~/Downloads/cse344-system-programming/HW4/hw4test/put_your_codes_her
e$ ./MWCp 10 100 ../testdir ../tocopy
Program is starting...
asdasdTotal bytes copied: 73520554
Total files copied: 3117
Total regular files copied: 3116
Total FIFO files copied: 1
Total directories copied including main directory: 152
Execution time: 1.811701 seconds
emre@ubuntu:~/Downloads/cse344-system-programming/HW4/hw4test/put_your_codes_her
e$
```

emre@ubuntu: ~/Downloads/cse344-system-programming/...
emre@ubuntu: ~/Downloads/cse344-system-programming/H...
Filelist
Makefile
README.
md
README.
txt
test_fifo

Fig-5: Test Case for FIFO

There is no FIFO file in test directory that is sent. So, to test it, firstly it is created an empty FIFO file. Then program run. As you see in the result, it copied it successfuly.

8.5 CTRL-C Signal Test



```
emre@ubuntu:~/Downloads/cse344-system-programming/HW4/hw4test/put_your_codes_here$ valgrind --leak-check=full         --show-leak-kinds=all          --track-origins=yes          --verbose          --log
-file=valgrind-out-test-1.txt          ./MWCp 10 100 ../testdir/src/libvterm ../tocopy
Program is starting...
^CReceived SIGINT. Please be patient and wait. Exiting...
Total bytes copied: 0
Total files copied: 0
Total regular files copied: 0
Total FIFO files copied: 0
Total directories copied including main directory: 0
emre@ubuntu:~/Downloads/cse344-system-programming/HW4/hw4test/put_your_codes_here$
```

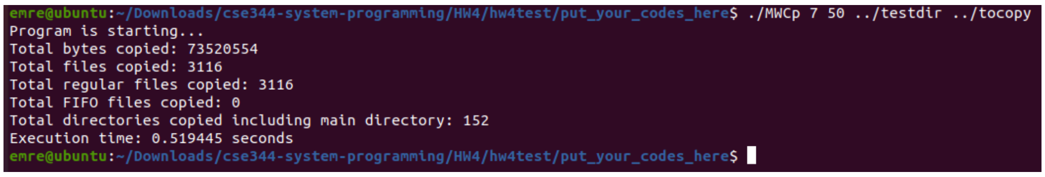
```
valgrind-out-test-1.txt
~/Downloads/cse344-system-programming/HW4/hw4test/put_your_codes_here

--14162-- REDIR: 0x491d670 (libc.so.6:strnlen) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--14162-- REDIR: 0x491d230 (libc.so.6:strcspn) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--14162-- REDIR: 0x491e800 (libc.so.6:strncasecmp) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--14162-- REDIR: 0x491d1d0 (libc.so.6:strcpy) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--14162-- REDIR: 0x491e950 (libc.so.6:memcpy@@GLIBC_2.14) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--14162-- REDIR: 0x493bf50 (libc.so.6:wcsnlen) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--14162-- REDIR: 0x493a860 (libc.so.6:wcscpy) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--14162-- REDIR: 0x491d820 (libc.so.6:strpbrk) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--14162-- REDIR: 0x491d100 (libc.so.6:index) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--14162-- REDIR: 0x491d630 (libc.so.6:strlen) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--14162-- REDIR: 0x4926bb0 (libc.so.6:memrchr) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--14162-- REDIR: 0x491e850 (libc.so.6:strcasecmp_l) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--14162-- REDIR: 0x491e3d0 (libc.so.6:memchr) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--14162-- REDIR: 0x493a930 (libc.so.6:wcslen) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--14162-- REDIR: 0x491dae0 (libc.so.6:strspn) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--14162-- REDIR: 0x491e750 (libc.so.6:stpncpy) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--14162-- REDIR: 0x491e6f0 (libc.so.6:stpcpy) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--14162-- REDIR: 0x491fc90 (libc.so.6:strchrnul) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--14162-- REDIR: 0x491e8a0 (libc.so.6:strncasecmp_l) redirected to 0x48311d0 (_vgnU_ifunc_wrapper)
--14162-- REDIR: 0x4a06730 (libc.so.6:__strrchr_avx2) redirected to 0x483ea10 (rindex)
--14162-- REDIR: 0x4a06900 (libc.so.6:__strlen_avx2) redirected to 0x483ef40 (strlen)
--14162-- REDIR: 0x49180e0 (libc.so.6:malloc) redirected to 0x483b780 (malloc)
--14162-- REDIR: 0x4919b10 (libc.so.6:calloc) redirected to 0x483dce0 (calloc)
--14162-- REDIR: 0x4a098f0 (libc.so.6:__mempcpy_avx_unaligned_erms) redirected to 0x4843660 (mempcpy)
--14162-- REDIR: 0x49186d0 (libc.so.6:free) redirected to 0x483c9d0 (free)
--14162-- REDIR: 0x4a06540 (libc.so.6:__strchrnul_avx2) redirected to 0x4843540 (strchrnul)
==14162==
==14162== HEAP SUMMARY:
==14162==     in use at exit: 0 bytes in 0 blocks
==14162==   total heap usage: 104 allocs, 104 frees, 49,856 bytes allocated
==14162==
==14162== All heap blocks were freed -- no leaks are possible
==14162==
==14162== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Fig-6: Test Case for CTRL-C Signal

As you can see, shortly after the signal is sent, it is caught, and the program is terminated. Also, as seen in the figure, care has been taken to ensure that there are no leaks while the SIGINT signal is being processed. **Due to this leak prevention measure, capturing and processing the signal may take a few seconds. Please be patient.**

Click here to watch CTRL-C test video

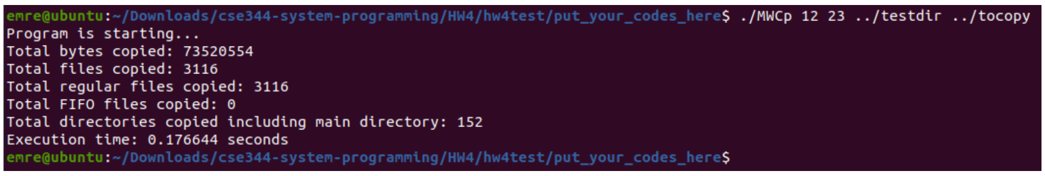
8.6 Additional Test Case - 1



```
emre@ubuntu:~/Downloads/cse344-system-programming/HW4/hw4test/put_your_codes_here$ ./MWCp 7 50 ../testdir ../tocopy
Program is starting...
Total bytes copied: 73520554
Total files copied: 3116
Total regular files copied: 3116
Total FIFO files copied: 0
Total directories copied including main directory: 152
Execution time: 0.519445 seconds
emre@ubuntu:~/Downloads/cse344-system-programming/HW4/hw4test/put_your_codes_here$
```

Fig-7: Additional Test Case - 1

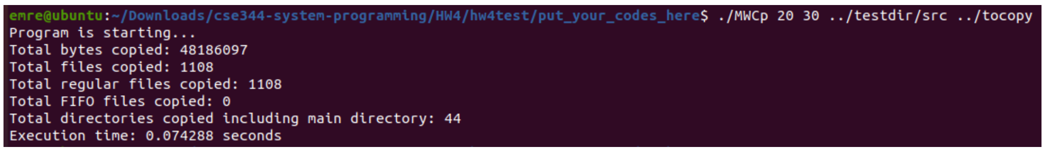
8.7 Additional Test Case - 2



```
emre@ubuntu:~/Downloads/cse344-system-programming/HW4/hw4test/put_your_codes_here$ ./MWCp 12 23 ../testdir ../tocopy
Program is starting...
Total bytes copied: 73520554
Total files copied: 3116
Total regular files copied: 3116
Total FIFO files copied: 0
Total directories copied including main directory: 152
Execution time: 0.176644 seconds
emre@ubuntu:~/Downloads/cse344-system-programming/HW4/hw4test/put_your_codes_here$
```

Fig-8: Additional Test Case - 2

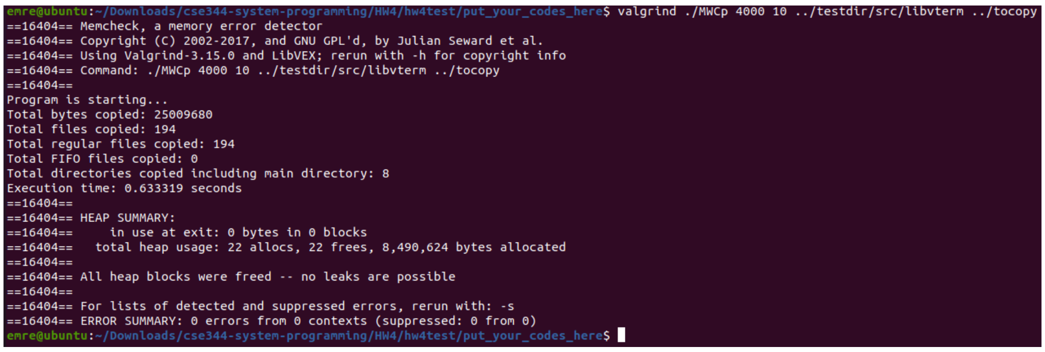
8.8 Additional Test Case - 3



```
emre@ubuntu:~/Downloads/cse344-system-programming/HW4/hw4test/put_your_codes_here$ ./MWCp 20 30 ../testdir/src ../tocopy
Program is starting...
Total bytes copied: 48186097
Total files copied: 1108
Total regular files copied: 1108
Total FIFO files copied: 0
Total directories copied including main directory: 44
Execution time: 0.074288 seconds
```

Fig-9: Additional Test Case - 3

8.9 Additional Test Case - 4 with BIG BUFFER SIZE



```
emre@ubuntu:~/Downloads/cse344-system-programming/HW4/hw4test/put_your_codes_here$ valgrind ./MWCp 4000 10 ../testdir/src/libvterm ../tocopy
==16404== Memcheck, a memory error detector
==16404== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==16404== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==16404== Command: ./MWCp 4000 10 ../testdir/src/libvterm ../tocopy
==16404==
Program is starting...
Total bytes copied: 25009680
Total files copied: 194
Total regular files copied: 194
Total FIFO files copied: 0
Total directories copied including main directory: 8
Execution time: 0.633319 seconds
==16404==
==16404== HEAP SUMMARY:
==16404==     in use at exit: 0 bytes in 0 blocks
==16404==   total heap usage: 22 allocs, 22 frees, 8,490,624 bytes allocated
==16404==
==16404== All heap blocks were freed -- no leaks are possible
==16404==
==16404== For lists of detected and suppressed errors, rerun with: -s
==16404== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
emre@ubuntu:~/Downloads/cse344-system-programming/HW4/hw4test/put_your_codes_here$
```

Fig-10: Additional Test Case - 4 with BIG BUFFER

As you see from this figure-10, open file limit is not an issue for my program. I handle it using condition variables and mutex. It never exceeds the open file limit.