FATİH DOĞAÇ
1901042654

CSE344 – SYSTEM PROGRAMMING
HOMEWORK #4 REPORT

1. Introduction

Objective: Develop a directory copying utility called "MWCp" that copies files and sub-directories in
parallel. Use a worker-manager approach to synchronize thread activity. Use POSIX and Standard C libraries.

Main Program:
• Accepts buffer size, number of workers, and source/destination directories as command-line arguments.
• Starts worker threads and waits for completion.
• Measures execution time to copy files in the directory. Keep statistics about the number and types of files copied.

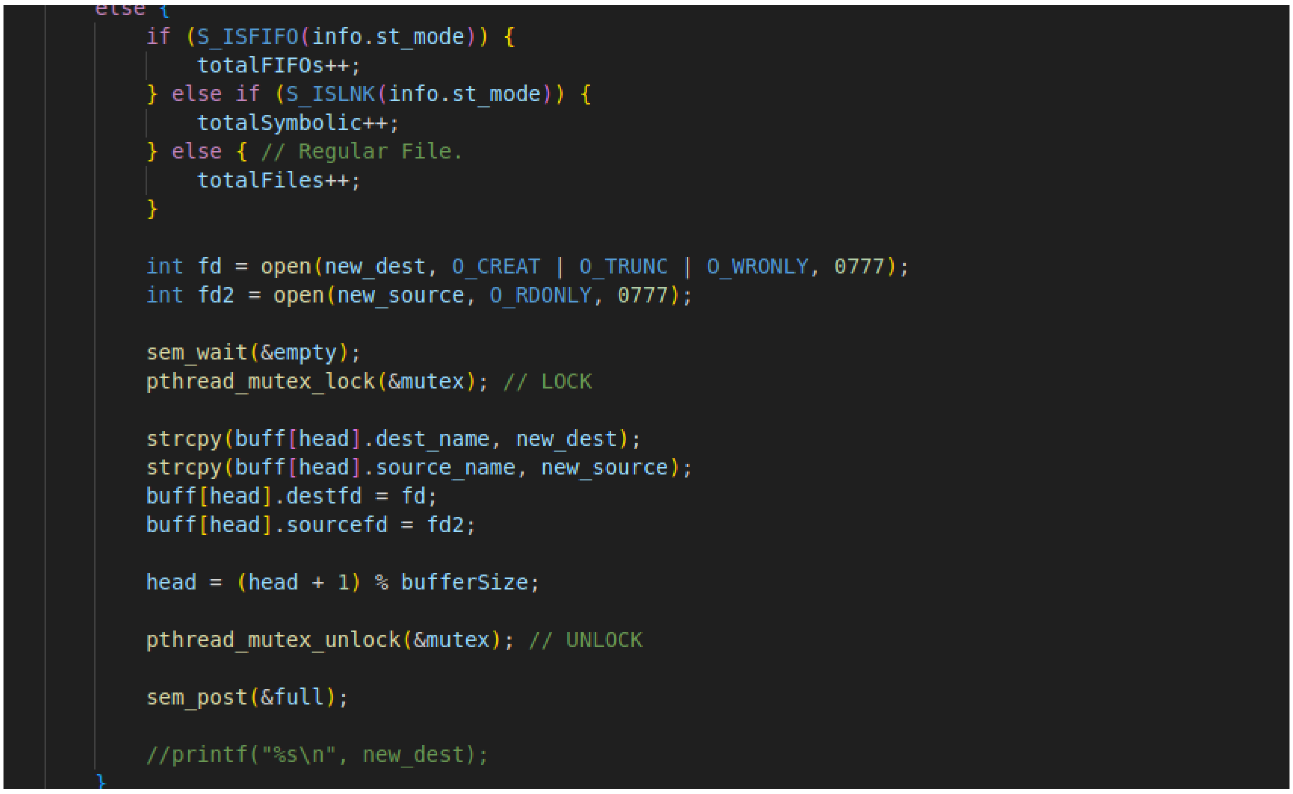
Manager:
• You should have only one manager thread.
• Reads source & destination directory paths.
• Opens files for reading and creates corresponding files in the destination directory.
• If a file already exists in the destination directory with the same name, the file should be opened and truncated.
• If an error occurs in opening either file, both files are closed, and an informative message is sent to standard output. Then, two open file descriptors and names are passed into a buffer.
• Buffer structure: The manager waits to fill the reserved buffer, and worker waits for the buffer to empty.
• You manage the buffer (is it empty or full, is it okay to access the buffer or should the execution wait until it is available) so that the threads can be terminated gracefully.
• Notifies program completion when the producer finishes filling the buffer with file names for the given directories, it should set a done flag and exits.

Worker:
• Reads file information from the buffer.
• Copies files from source to destination.
• Writes completion status to standard output.
• Critical section: the producers and the multiple consumers write the standard output.
• Terminates when signaled.
• Worker thread pool: to regulate the number of

2. Mechanism

Program accepts the buffer size, worker number, source and destination folder from the main arguments and creates the worker threads according to this information.
My buffer array is a circular array it has head and tail indexes.

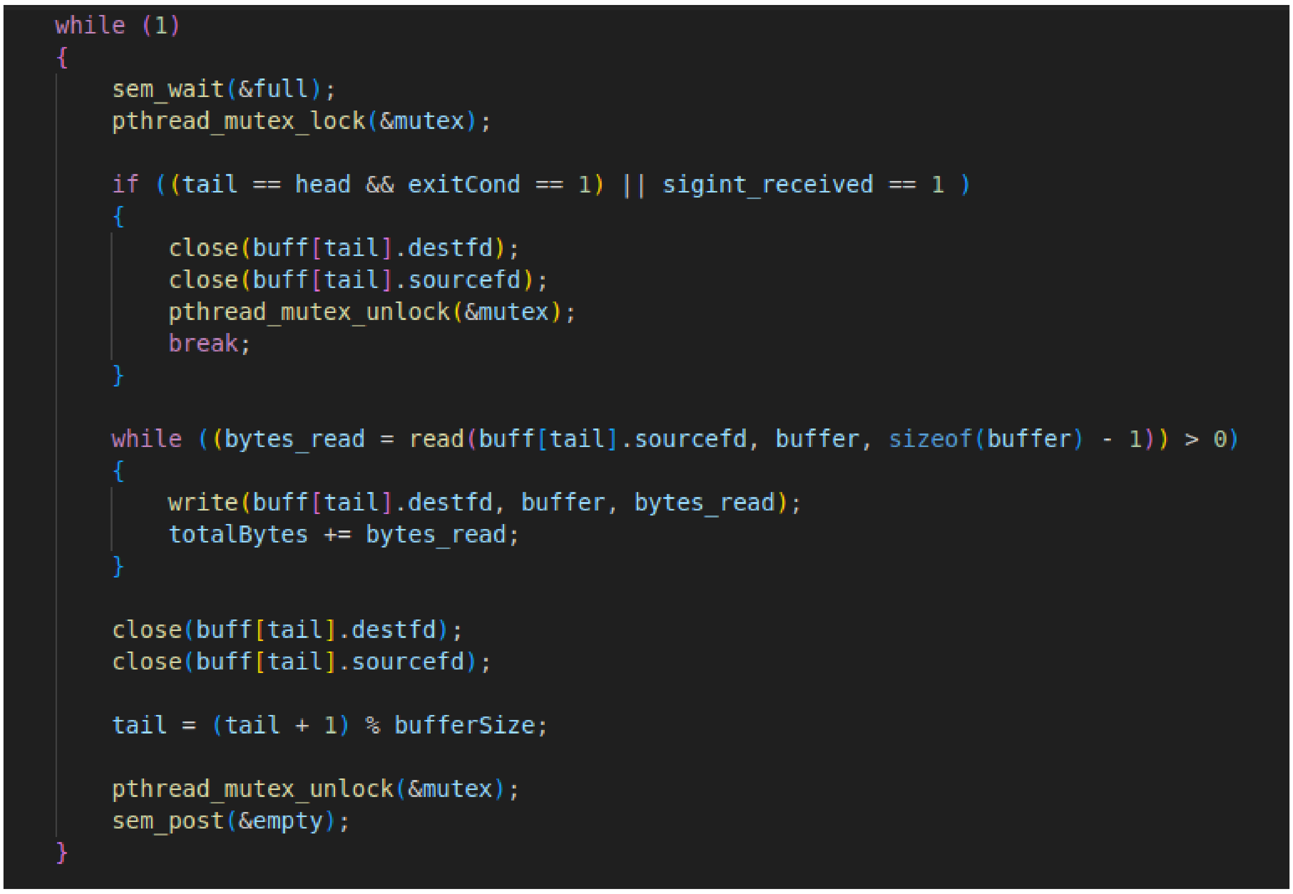
Manager:



```
        if (S_ISFIFO(info.st_mode)) {
            totalFIFOs++;
        } else if (S_ISLNK(info.st_mode)) {
            totalSymbolic++;
        } else { // Regular File.
            totalFiles++;
        }

        int fd = open(new_dest, O_CREAT | O_TRUNC | O_WRONLY, 0777);
        int fd2 = open(new_source, O_RDONLY, 0777);

        sem_wait(&empty);
        pthread_mutex_lock(&mutex); // LOCK

        strcpy(buff[head].dest_name, new_dest);
        strcpy(buff[head].source_name, new_source);
        buff[head].destfd = fd;
        buff[head].sourcefd = fd2;

        head = (head + 1) % bufferSize;

        pthread_mutex_unlock(&mutex); // UNLOCK

        sem_post(&full);

        //printf("%s\n", new_dest);
```

If the managar can get through the empty semaphore, it will insert the informations about the files into the buffer's head. And it will move the head once. Then it will inform the workers (sem_post(full)) that there is files that needs to be processed.

Workers:

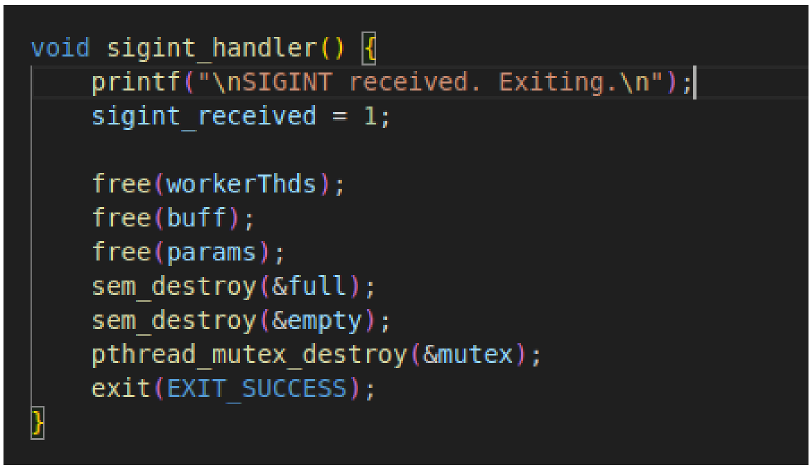


```
while (1)
{
    sem_wait(&full);
    pthread_mutex_lock(&mutex);

    if ((tail == head && exitCond == 1) || sigint_received == 1 )
    {
        close(buff[tail].destfd);
        close(buff[tail].sourcefd);
        pthread_mutex_unlock(&mutex);
        break;
    }

    while ((bytes_read = read(buff[tail].sourcefd, buffer, sizeof(buffer) - 1)) > 0)
    {
        write(buff[tail].destfd, buffer, bytes_read);
        totalBytes += bytes_read;
    }

    close(buff[tail].destfd);
    close(buff[tail].sourcefd);

    tail = (tail + 1) % bufferSize;

    pthread_mutex_unlock(&mutex);
    sem_post(&empty);
}
```

If it can go through full semaphore, that means there is files that needs to be processed. It reads from source file descriptor and writes to destination file descriptor. Then moves the tail once. Then informs the manager that there is empty space in buffer.

The exit condition of workers is if tail equals to head, that means buffer is empty. But threads can work with different speeds so the tail can be equal to the head in anytime (but can’t exceed it). So there is a exitCond variable to check if the thread should exit. That means manager is done creating files, empty the buffer and leave. Or they got a SIGINT.

SIGINT HANDLER:

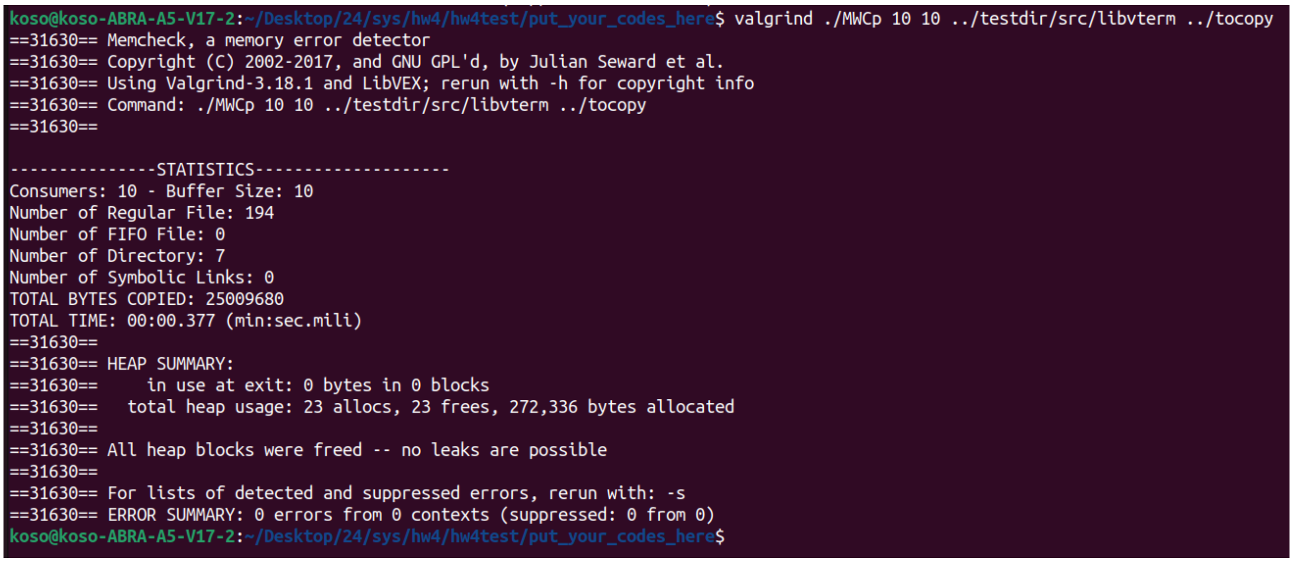


```
void sigint_handler() {
    printf("\nSIGINT received. Exiting.\n");
    sigint_received = 1;

    free(workerThds);
    free(buff);
    free(params);
    sem_destroy(&full);
    sem_destroy(&empty);
    pthread_mutex_destroy(&mutex);
    exit(EXIT_SUCCESS);
}
```

It frees every pointer and cleans up then leaves.

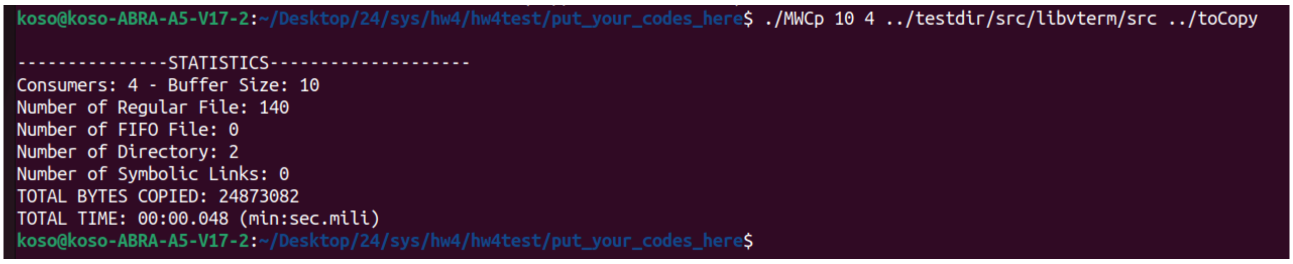
3. Test Cases

Case 1:
valgrind ./MWCp 10 10 ../testdir/src/libvterm ../tocopy



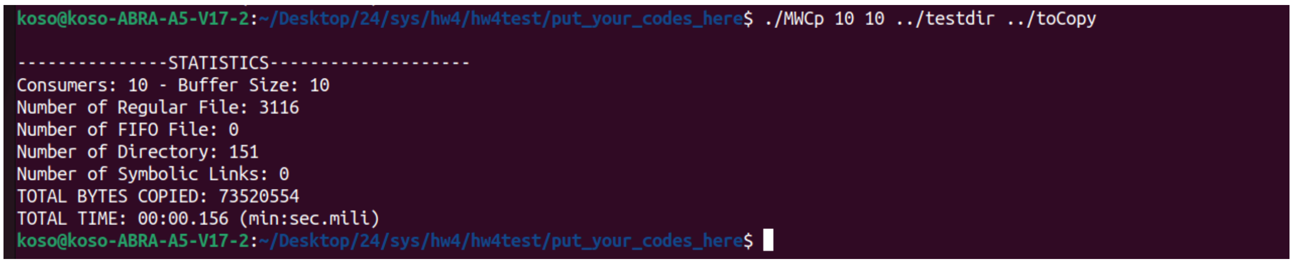
```
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw4/hw4test/put_your_codes_here$ valgrind ./MWCp 10 10 ../testdir/src/libvterm ../tocopy
==31630== Memcheck, a memory error detector
==31630== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==31630== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==31630== Command: ./MWCp 10 10 ../testdir/src/libvterm ../tocopy
==31630==

--------------STATISTICS-------------------
Consumers: 10 - Buffer Size: 10
Number of Regular File: 194
Number of FIFO File: 0
Number of Directory: 7
Number of Symbolic Links: 0
TOTAL BYTES COPIED: 25009680
TOTAL TIME: 00:00.377 (min:sec.mili)
==31630==
==31630== HEAP SUMMARY:
==31630==     in use at exit: 0 bytes in 0 blocks
==31630==   total heap usage: 23 allocs, 23 frees, 272,336 bytes allocated
==31630==
==31630== All heap blocks were freed -- no leaks are possible
==31630==
==31630== For lists of detected and suppressed errors, rerun with: -s
==31630== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw4/hw4test/put_your_codes_here$
```

Case 2:
./MWCp 10 4 ../testdir/src/libvterm/src ../toCopy



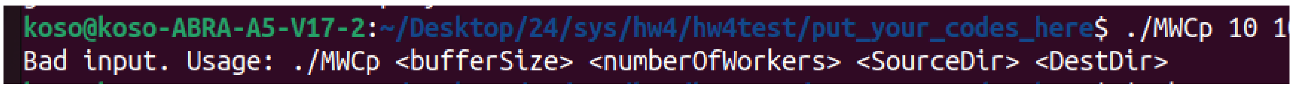
```
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw4/hw4test/put_your_codes_here$ ./MWCp 10 4 ../testdir/src/libvterm/src ../toCopy

--------------STATISTICS-------------------
Consumers: 4 - Buffer Size: 10
Number of Regular File: 140
Number of FIFO File: 0
Number of Directory: 2
Number of Symbolic Links: 0
TOTAL BYTES COPIED: 24873082
TOTAL TIME: 00:00.048 (min:sec.mili)
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw4/hw4test/put_your_codes_here$
```

Case 3:
./MWCp 10 10 ../testdir ../toCopy



```
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw4/hw4test/put_your_codes_here$ ./MWCp 10 10 ../testdir ../toCopy

--------------STATISTICS-------------------
Consumers: 10 - Buffer Size: 10
Number of Regular File: 3116
Number of FIFO File: 0
Number of Directory: 151
Number of Symbolic Links: 0
TOTAL BYTES COPIED: 73520554
TOTAL TIME: 00:00.156 (min:sec.mili)
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw4/hw4test/put_your_codes_here$
```

Case 4:
Bad input.



```
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw4/hw4test/put_your_codes_here$ ./MWCp 10 1
Bad input. Usage: ./MWCp <bufferSize> <numberOfWorkers> <SourceDir> <DestDir>
```