CSE344 Homework 5 Report

Akif Safa Angi – 200104004079

1. Problem Definition

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.

2. General Structure

Information about condition variables and barrier

I used same condition structure with homework 4. Used one barrier. In barrier there are 3 possible waits. Two of them inside the while loop in worker thread.

First wait -> If file not opened correctly, thread will wait here until others copy files and wait in other barrier.

Second wait -> Copied file and wait after copied file in barrier

Third wait -> If any threads wait after copy and no other files left, it may stuck in first wait. If any threads stuck in first wait, others will wait in third wait to make sure they left correctly.

Code changes:

Added barrier_wait end of while loop in worker.

Modified if(file descriptors) for barrier_wait(explained above)

Added barrier_wait and its conditions after while loop in worker

Simplified version of the code. For detailly (signal handler, closing file etc.) implementation you can look code file.

pthread_mutex_t the_mutex; // Mutex for synchronization between threads

pthread_cond_t condc, condp; // Condition variables for controlling if buffer is full or empty

pthread_mutex_t flag; // Mutex for SIGINT

pthread_barrier_t workerBarrier; // Barrier for workers

typedef struct {

    int srcFd; // File descriptor for source file

    int destFd; // File descriptor for destination file

    char srcName[PATH_SIZE]; // Source file name

    char destName[PATH_SIZE]; // Destination file name

} Files;

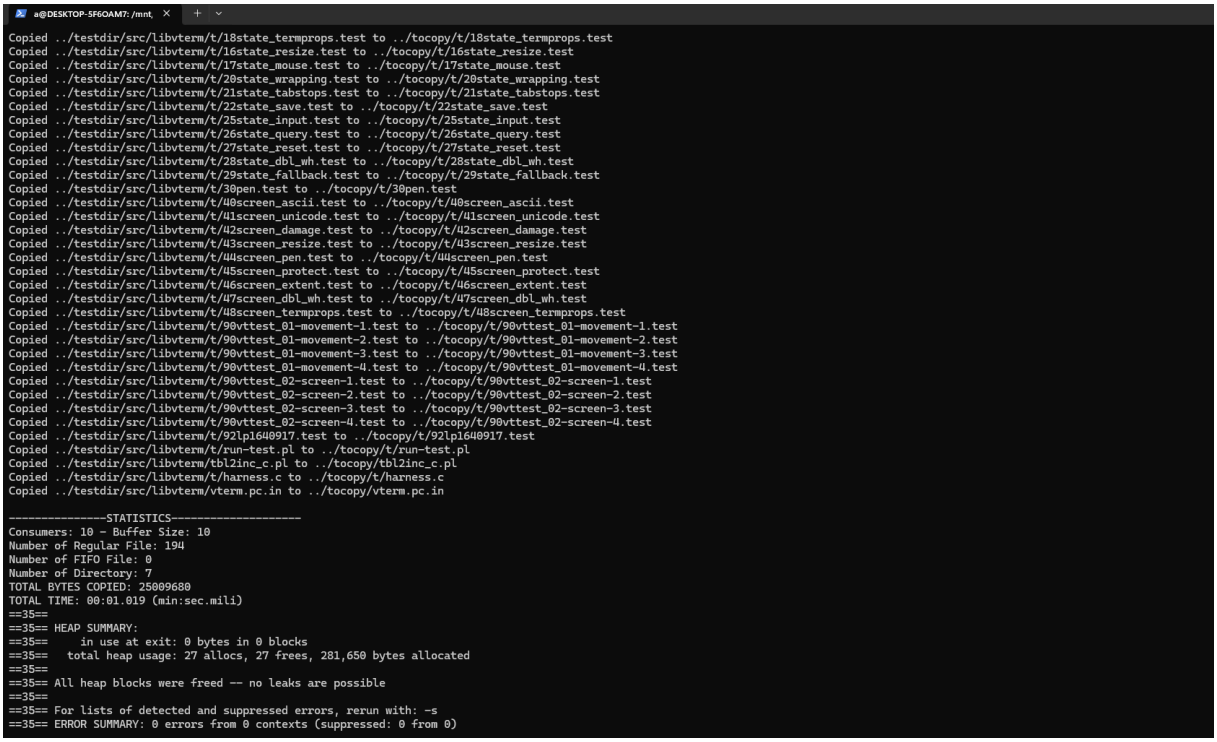
```
typedef struct {
    int bufferSize; // Buffer size Files *buffer; // Buffer
    int count; // Number of items in buffer
    int head; // Head of buffer int last; // Last of buffer
    int doneFlag; // Done flag
} Buffer;

manager(arg) {
    char **argv = (char **) arg;
    createDirectory(argv[3], argv[4]);
    lock(&thread_mutex);
    buffer.doneFlag = 1;
    broadcast(&condc);
    unlock(&thread_mutex);
}
worker(arg) {
    while(1) {
        item = remove_item();
        if(item.srcFd == -1 or item.dest == -1) {
            lock(&the_mutex);
            if(activeWorker == numberOfWorkers) {
                unlock(&the_mutex);
                barrier_wait(&workerBarrier);
            } else {
                unlock(&the_mutex);
            }
            if(item.srcFd == -1 and item.dest == -1) {
                lock(&the_mutex);
                if(activeWorker == numberOfWorkers) wait = 1;
                unlock(&the_mutex);
                break;
```

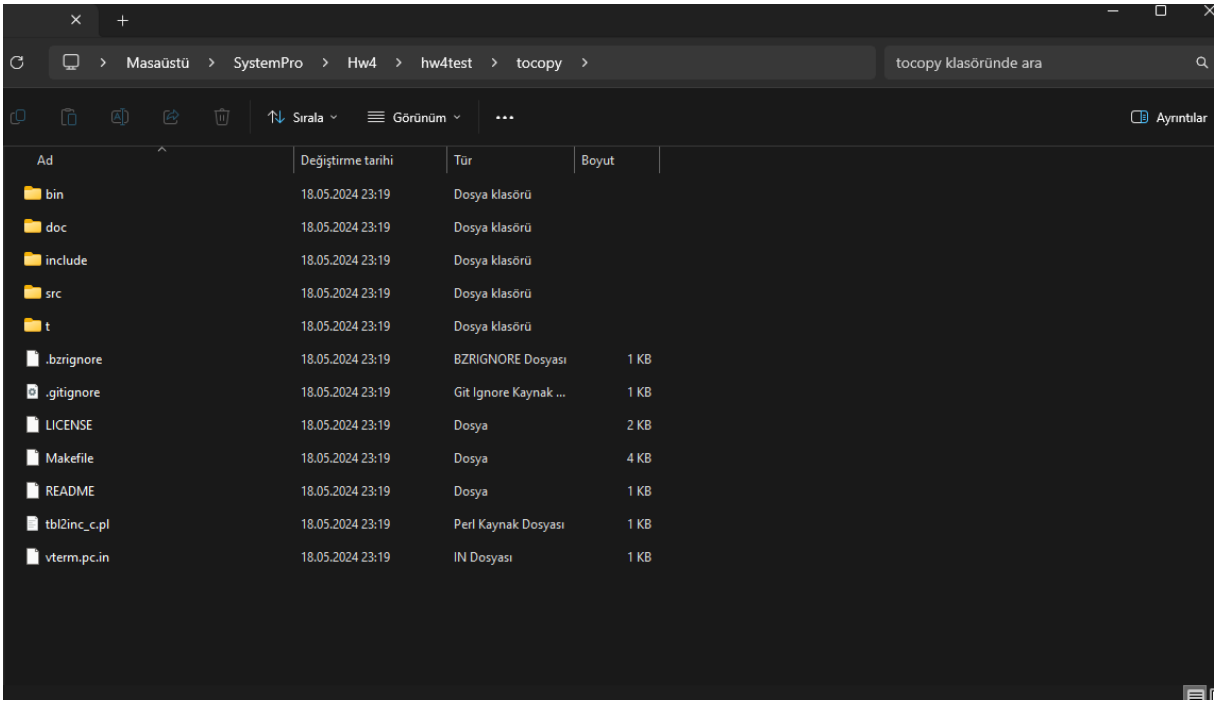
```
                }
                continue;
            }
            while(bytes_read = read(item.src) > 0) {
                bytes_write = write(item.destFd);
                totalWritten += bytes_write;
            }
            close(item.srcFd);
            close(item.dstFd);
            lock(&thread_mutex);
            totalBytes += totalWritten;
            unlock(&thread_mutex);
            barrier_wait(&workerBarrier);
        }
        lock(&the_mutex);
        activeWorker--;
        if(wait == 1 && buffer.count == 0) {
            unlock(&the_mutex);
            barrier_wait(&workerBarrier);
        } else {
            unlock(&the_mutex);
        }
        pthread_exit(0);
}
add_item(Files item) {
        lock(&thread_mutex);
        while(buffer.count == maxBuffer) cond_wait(&condp, &thread_mutex);
        buffer.buffer[buffer.last] = item;
        buffer.last = (buffer.last + 1) % buffer.bufferSize;
        buffer.count++;
        cond_signal(&condc);
```

```
        unlock(&thread_mutex);
}
remove_item() {
        lock(&thread_mutex);
        while(buffer.count == 0 && buffer.doneFlag == 0) cond_wait(&condc, &thread_mutex);
        if(buffer.count > 0) {
                item = buffer.buffer[buffer.head];
                buffer.head = (buffer.head + 1) % buffer.bufferSize;
                buffer.count--;
                cond_signal(&condp);
                unlock(&thread_mutex);
                return item;
        }
        unlock(&thread_mutex);
        return (Files) {-1, -1, “”, “”};
}
```

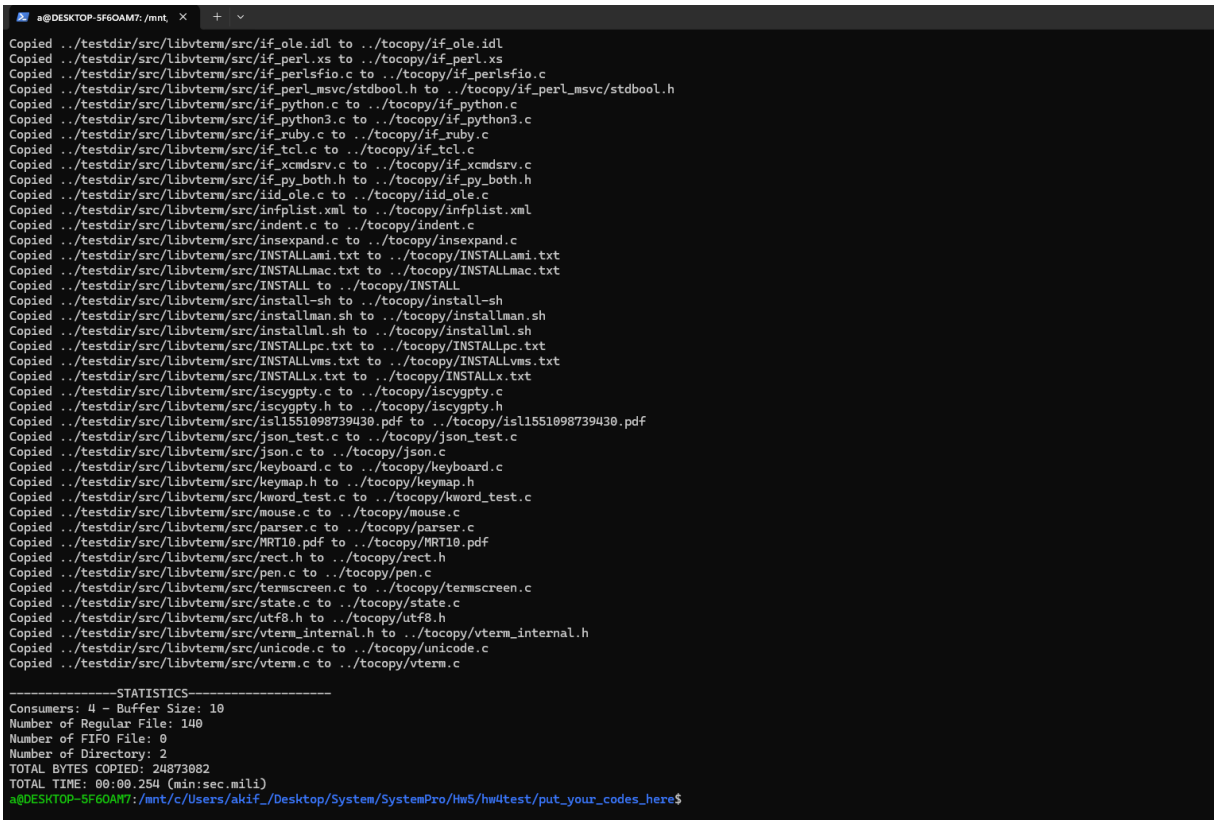
3. Tests

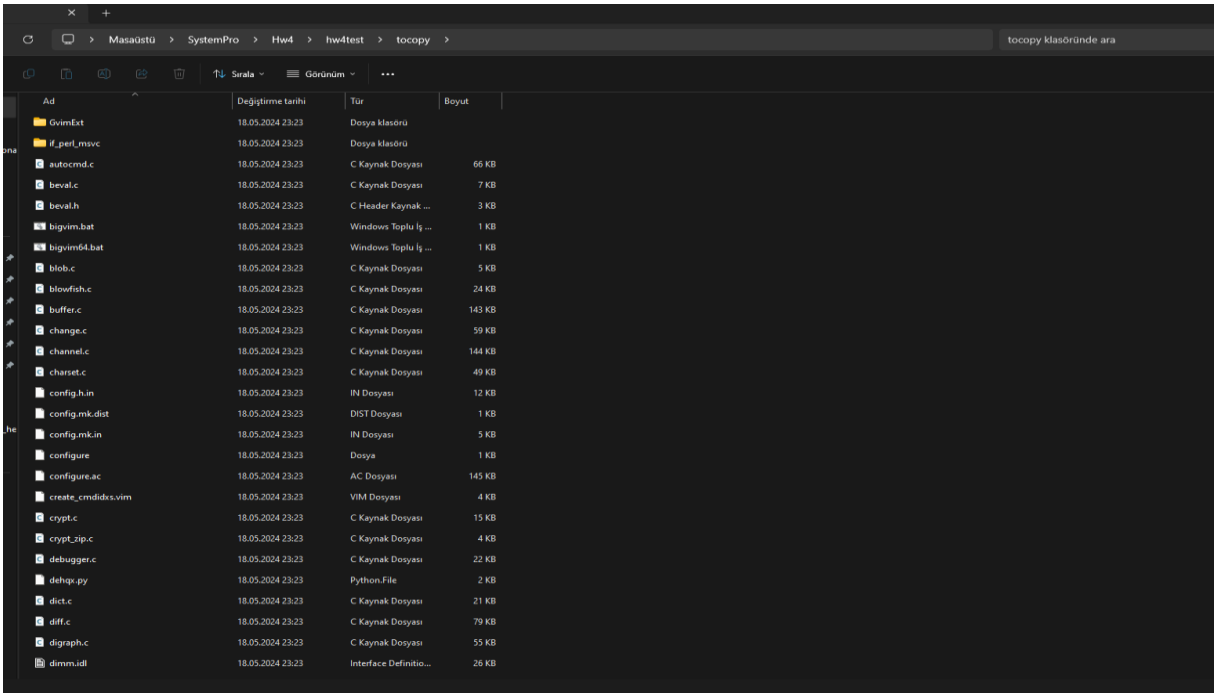
• Test1: valgrind ./MWCp 10 10 ../testdir/src/libvterm ../tocopy, #memory leak checking, buffer size=10, number of workers=10, sourceFile, destinationFile





• Test2: ./MWCp 10 4 ../testdir/src/libvterm/src ../toCopy #buffer size=10, number of workers=4, sourceFile, destinationFile





• Test3: ./MWCp 10 10 ../testdir ../toCopy #buffer size=10, number of workers=10, sourceFile, destinationFile