FATİH DOĞAÇ
1901042654

CSE344 – SYSTEM PROGRAMMING
HOMEWORK #5 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.

Condition variables will be used to signal when the buffer is not empty (so workers can start processing) and when the buffer is not full (so the manager can add more items). When the buffer is full, the manager thread will wait on a condition variable. When a worker thread removes an item from
the buffer, it will signal the condition variable to wake up the manager thread. Similarly, when the buffer
is empty, worker threads will wait on a different condition variable. When the manager thread adds an
item to the buffer, it will signal this condition variable to wake up the worker threads.
Barriers to ensure that all worker threads wait at a certain point before proceeding. This can be useful to ensure that all threads have completed a phase of processing before moving on to the next phase.

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

        while (count == bufferSize) {
            pthread_cond_wait(&not_full, &mutex);
        }

        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;
        count++;

        pthread_cond_signal(&not_empty);
        pthread_mutex_unlock(&mutex); // UNLOCK

        //sem_post(&full);
```

Manager locks the mutex and checks if the buffer is full. If yes, it waits via condition variable “not_full”. When the not full condition is met, manager continues to fill the buffer with file descriptors. And then signals **one** of the threads that buffer is not empty.

Workers:

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

    while (count == 0 && !exitCond) {
        pthread_cond_wait(&not_empty, &mutex);
    }

    if ((count == 0 && 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;
    count--;
    pthread_cond_signal(&not_full);
    pthread_mutex_unlock(&mutex);

    //sem_post(&empty);
}

pthread_barrier_wait(&barrier);
pthread_mutex_lock(&mutex2);
printf("Worker finished..\n");
pthread_mutex_unlock(&mutex2);
```

If the mutex is available, the worker enters the critical region. Checks if the buffer is empty. If yes, waits for the not_empty condition. When it got signalled, it checks if should it exit. If not, copies the current source, then closes the file descriptors. Moves the tail once and signals the manager that buffer is not_full right now. Then continues to loop.

When a worker finishes, it hits the barrier and waits. When the manager finishes, it hits the barrier and waits. When they are all done, they continue and exit the program.

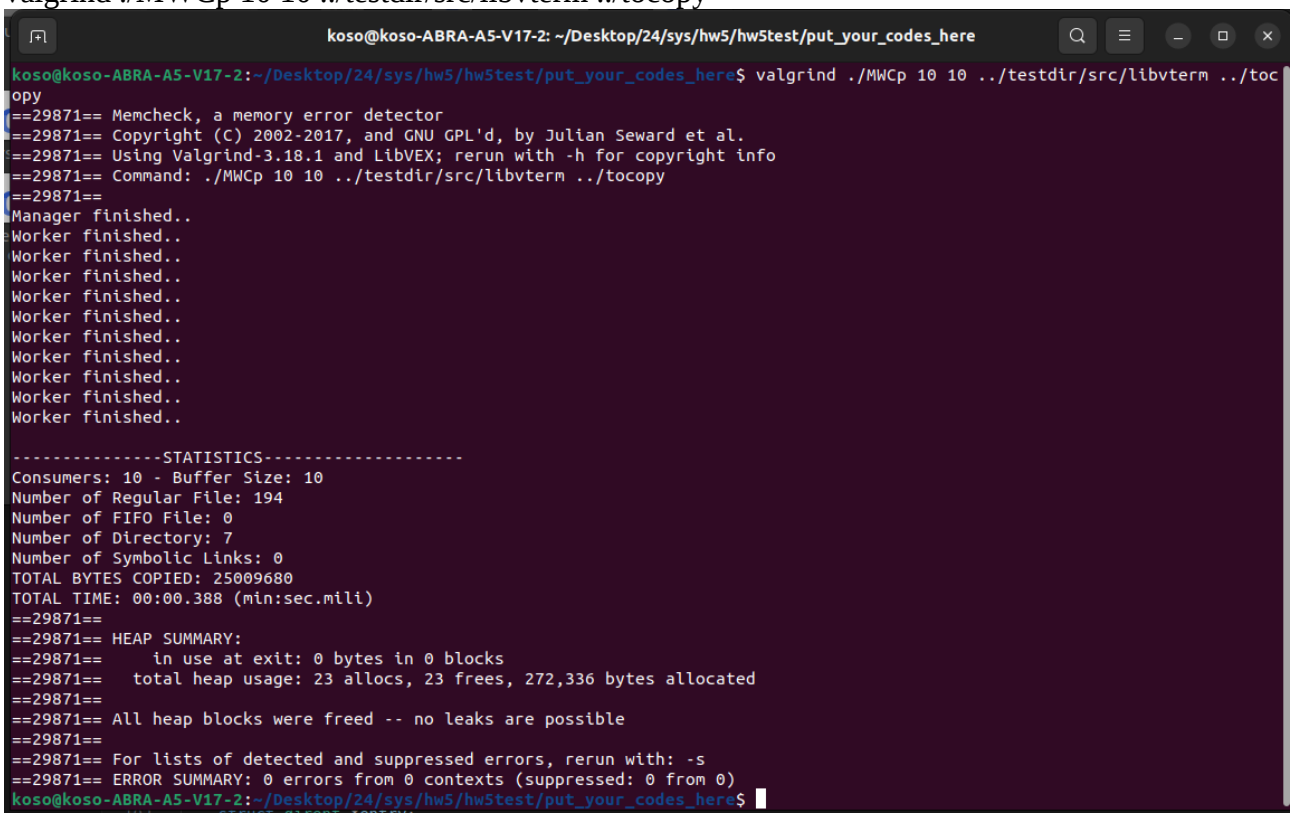
```
void* manager_func(void *args){
    struct manager_thread_params *params = args;

    open_files_in_dest(params->source, params->destination);

    pthread_mutex_lock(&mutex);

    exitCond = 1;
    pthread_cond_broadcast(&not_empty);

    pthread_mutex_unlock(&mutex);

    pthread_barrier_wait(&barrier);

    pthread_mutex_lock(&mutex2);
    printf("Manager finished..\n");
    pthread_mutex_unlock(&mutex2);

    return NULL;
}
```

SIGINT HANDLER:

I couldn't managed to handle SIGINT properly. There would be leaks. So I blocked the SIGINT. Program can't be interrupted.

3. Test Cases

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



```
koso@koso-ABRA-A5-V17-2: ~/Desktop/24/sys/hw5/hw5test/put_your_codes_here
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw5/hw5test/put_your_codes_here$ valgrind ./MWCp 10 10 ../testdir/src/libvterm ../toc
opy
==29871== Memcheck, a memory error detector
==29871== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==29871== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==29871== Command: ./MWCp 10 10 ../testdir/src/libvterm ../tocopy
==29871==
Manager finished..
Worker finished..
Worker finished..
Worker finished..
Worker finished..
Worker finished..
Worker finished..
Worker finished..
Worker finished..
Worker finished..
Worker finished..

---------------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: 00:00.388 (min:sec.mili)
==29871==
==29871== HEAP SUMMARY:
==29871==     in use at exit: 0 bytes in 0 blocks
==29871==   total heap usage: 23 allocs, 23 frees, 272,336 bytes allocated
==29871==
==29871== All heap blocks were freed -- no leaks are possible
==29871==
==29871== For lists of detected and suppressed errors, rerun with: -s
==29871== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw5/hw5test/put_your_codes_here$
```

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

```
koso@koso-ABRA-A5-V17-2: ~/Desktop/24/sys/hw5/hw5test/put_your_codes_here
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw5/hw5test/put_your_codes_here$ ./MWCp 10 4 ../testdir/src/libvterm/src ../toCopy
Worker finished..
Manager finished..
Worker finished..
Worker finished..
Worker finished..

---------------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: 00:00.050 (min:sec.mili)
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw5/hw5test/put_your_codes_here$
```

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

```
koso@koso-ABRA-A5-V17-2: ~/Desktop/24/sys/hw5/hw5test/put_your_codes_here
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw5/hw5test/put_your_codes_here$ ./MWCp 10 10 ../testdir ../toCopy
Manager finished..
Worker finished..
Worker finished..
Worker finished..
Worker finished..
Worker finished..
Worker finished..
Worker finished..
Worker finished..
Worker finished..
Worker finished..

---------------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: 00:00.255 (min:sec.mili)
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw5/hw5test/put_your_codes_here$
```

Case 4:
Bad input.