CSE344 SYSTEMS PROGRAMMING

HOMEWORK 4

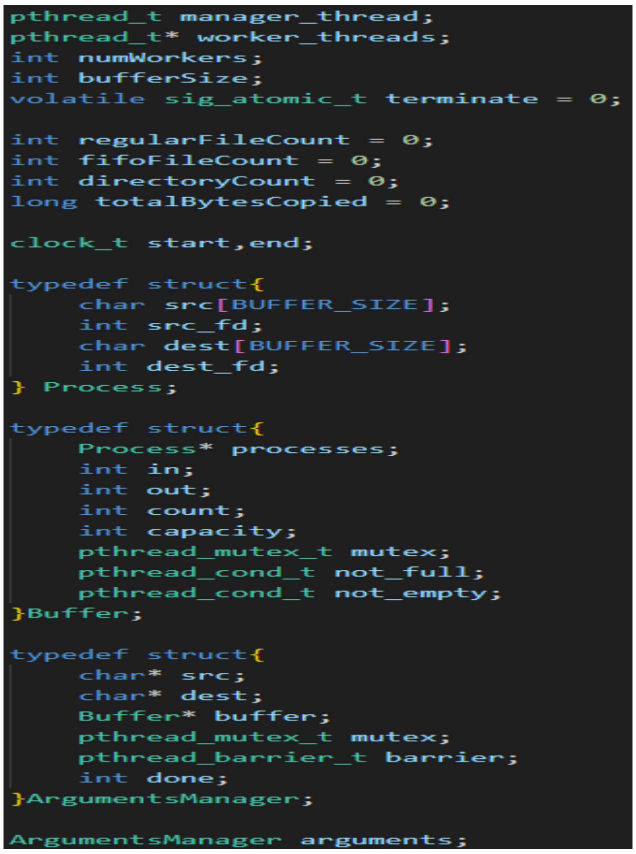
REPORT

Burak Demirkaya

210104004274

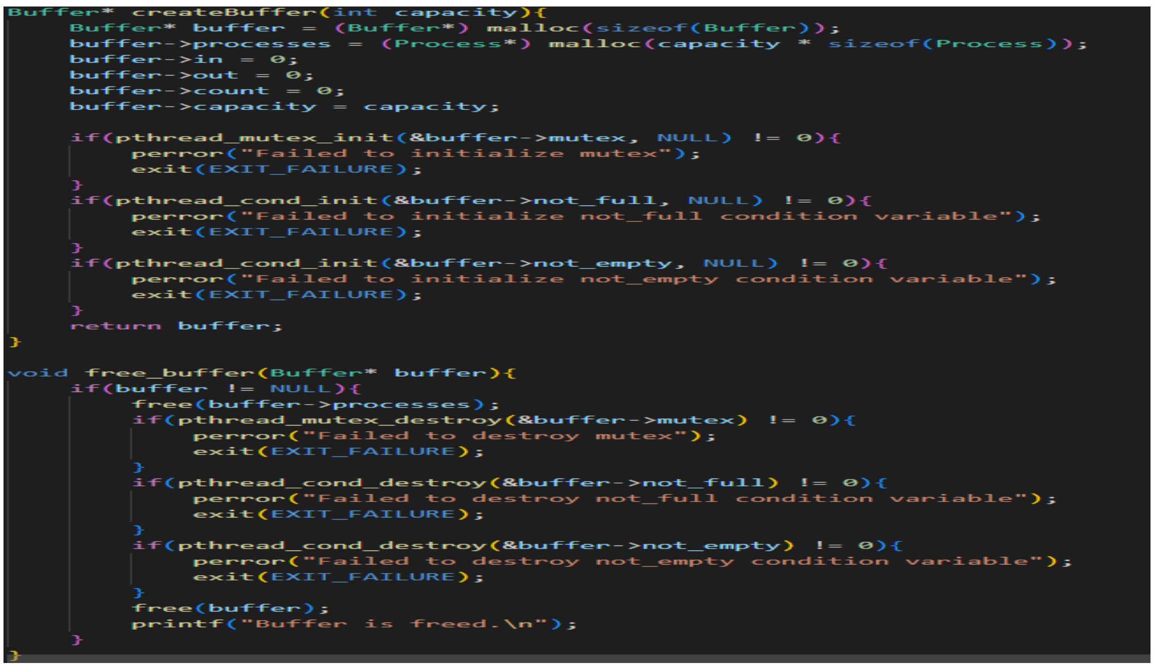
24/05/2024

1. Design Decisions



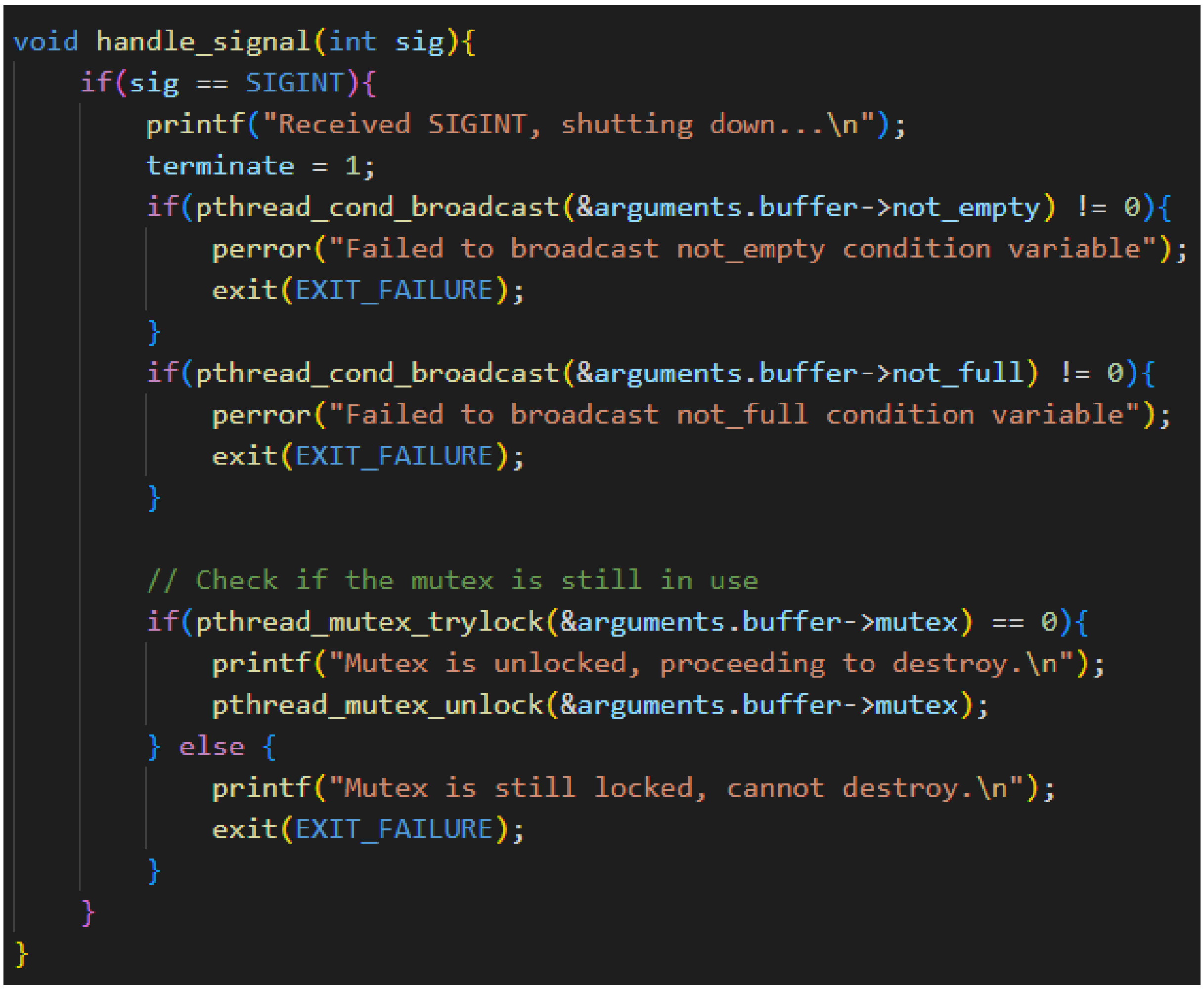
```c
pthread_t manager_thread;
pthread_t* worker_threads;
int numWorkers;
int bufferSize;
volatile sig_atomic_t terminate = 0;

int regularFileCount = 0;
int fifoFileCount = 0;
int directoryCount = 0;
long totalBytesCopied = 0;

clock_t start,end;

typedef struct{
    char src[BUFFER_SIZE];
    int src_fd;
    char dest[BUFFER_SIZE];
    int dest_fd;
} Process;

typedef struct{
    Process* processes;
    int in;
    int out;
    int count;
    int capacity;
    pthread_mutex_t mutex;
    pthread_cond_t not_full;
    pthread_cond_t not_empty;
}Buffer;

typedef struct{
    char* src;
    char* dest;
    Buffer* buffer;
    pthread_mutex_t mutex;
    pthread_barrier_t barrier;
    int done;
}ArgumentsManager;

ArgumentsManager arguments;
```

Process struct represents a file copy process. Buffer struct represents the shared buffer with process array, indices, count, capacity, mutex, and condition variables. ArgumentsManager struct passes arguments to the manager thread and worker thread.



```c
Buffer* createBuffer(int capacity){
    Buffer* buffer = (Buffer*) malloc(sizeof(Buffer));
    buffer->processes = (Process*) malloc(capacity * sizeof(Process));
    buffer->in = 0;
    buffer->out = 0;
    buffer->count = 0;
    buffer->capacity = capacity;

    if(pthread_mutex_init(&buffer->mutex, NULL) != 0){
        perror("Failed to initialize mutex");
        exit(EXIT_FAILURE);
    }
    if(pthread_cond_init(&buffer->not_full, NULL) != 0){
        perror("Failed to initialize not_full condition variable");
        exit(EXIT_FAILURE);
    }
    if(pthread_cond_init(&buffer->not_empty, NULL) != 0){
        perror("Failed to initialize not_empty condition variable");
        exit(EXIT_FAILURE);
    }
    return buffer;
}

void free_buffer(Buffer* buffer){
    if(buffer != NULL){
        free(buffer->processes);
        if(pthread_mutex_destroy(&buffer->mutex) != 0){
            perror("Failed to destroy mutex");
            exit(EXIT_FAILURE);
        }
        if(pthread_cond_destroy(&buffer->not_full) != 0){
            perror("Failed to destroy not_full condition variable");
            exit(EXIT_FAILURE);
        }
        if(pthread_cond_destroy(&buffer->not_empty) != 0){
            perror("Failed to destroy not_empty condition variable");
            exit(EXIT_FAILURE);
        }
        free(buffer);
        printf("Buffer is freed.\n");
    }
}
```

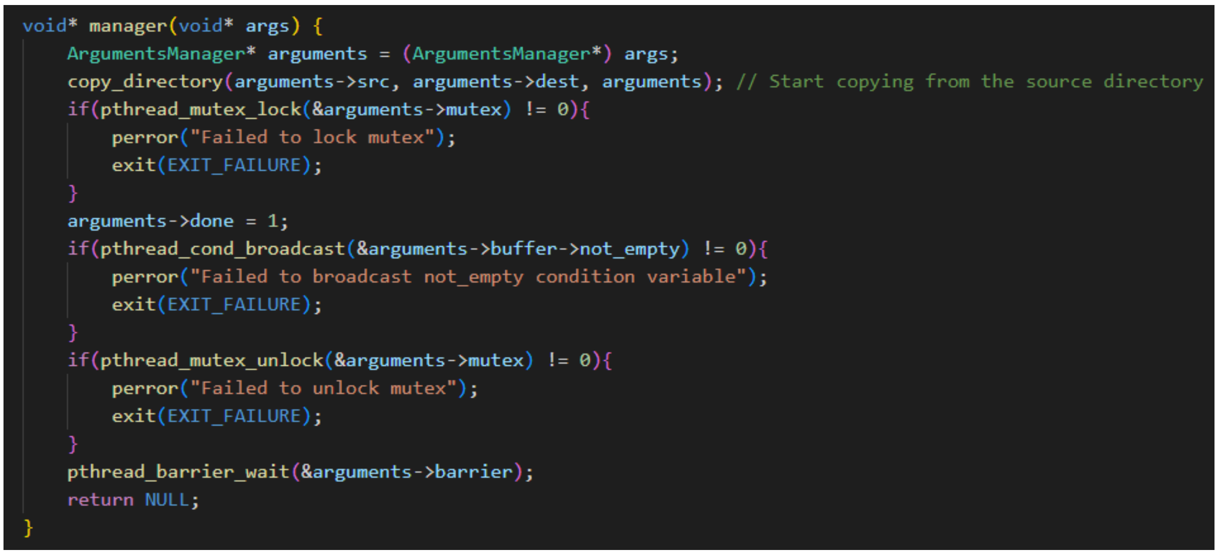
The createBuffer function creates a buffer to store file copying processes shared between threads. It allocates memory for the buffer and its process array, initializes variables, and sets up synchronization mechanisms.
The free_buffer function frees memory allocated for the buffer and its process array, and destroys the synchronization mechanisms.



```c
void handle_signal(int sig){
    if(sig == SIGINT){
        printf("Received SIGINT, shutting down...\n");
        terminate = 1;
        if(pthread_cond_broadcast(&arguments.buffer->not_empty) != 0){
            perror("Failed to broadcast not_empty condition variable");
            exit(EXIT_FAILURE);
        }
        if(pthread_cond_broadcast(&arguments.buffer->not_full) != 0){
            perror("Failed to broadcast not_full condition variable");
            exit(EXIT_FAILURE);
        }

        // Check if the mutex is still in use
        if(pthread_mutex_trylock(&arguments.buffer->mutex) == 0){
            printf("Mutex is unlocked, proceeding to destroy.\n");
            pthread_mutex_unlock(&arguments.buffer->mutex);
        } else {
            printf("Mutex is still locked, cannot destroy.\n");
            exit(EXIT_FAILURE);
        }
    }
}
```

The handle_signal function intercepts the SIGINT interrupt, setting terminate to 1 to indicate threads should stop. It broadcasts condition variables to awaken threads, ensuring no thread stays indefinitely blocked during termination. It checks if the mutex is locked before proceeding to destroy it, and exits the program if it's still locked.

Manager Thread:

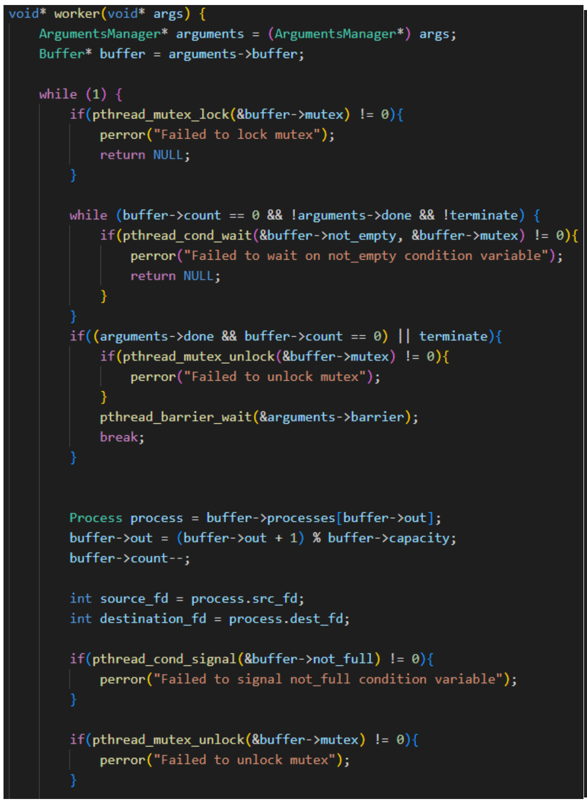


```c
void* manager(void* args) {
    ArgumentsManager* arguments = (ArgumentsManager*) args;
    copy_directory(arguments->src, arguments->dest, arguments); // Start copying from the source directory
    if(pthread_mutex_lock(&arguments->mutex) != 0){
        perror("Failed to lock mutex");
        exit(EXIT_FAILURE);
    }
    arguments->done = 1;
    if(pthread_cond_broadcast(&arguments->buffer->not_empty) != 0){
        perror("Failed to broadcast not_empty condition variable");
        exit(EXIT_FAILURE);
    }
    if(pthread_mutex_unlock(&arguments->mutex) != 0){
        perror("Failed to unlock mutex");
        exit(EXIT_FAILURE);
    }
    pthread_barrier_wait(&arguments->barrier);
    return NULL;
}
```

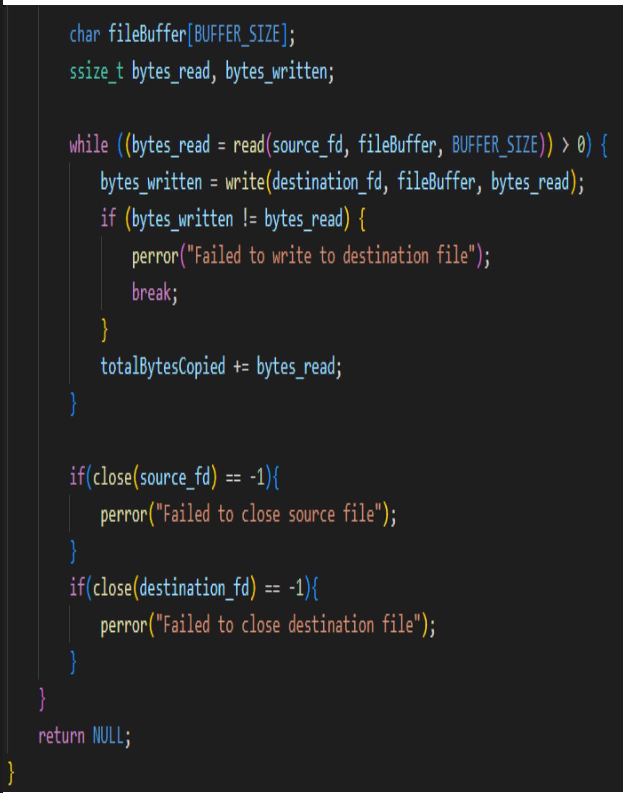
The manager function, main for the manager thread, calls copy_directory to copy files. It locks the mutex for exclusive buffer access, sets done to 1 signaling worker threads that adding processes is complete, and broadcasts the not_empty condition to wake waiting worker threads. After unlocking the mutex, it waits at a barrier to synchronize thread execution start.

The copy_directory function copies directories from source to destination, sending file descriptors to the worker via the buffer. It reads source directory entries, creating corresponding directories in the destination and recursively copying contents. For regular or FIFO files, it opens source and destination files and adds a copy process to the shared buffer, using a mutex and condition variables for synchronization. On termination signal, it closes files and breaks the loop.

Worker Thread:



```c
void* worker(void* args) {
    ArgumentsManager* arguments = (ArgumentsManager*) args;
    Buffer* buffer = arguments->buffer;

    while (1) {
        if(pthread_mutex_lock(&buffer->mutex) != 0){
            perror("Failed to lock mutex");
            return NULL;
        }

        while (buffer->count == 0 && !arguments->done && !terminate) {
            if(pthread_cond_wait(&buffer->not_empty, &buffer->mutex) != 0){
                perror("Failed to wait on not_empty condition variable");
                return NULL;
            }
        }
        if((arguments->done && buffer->count == 0) || terminate){
            if(pthread_mutex_unlock(&buffer->mutex) != 0){
                perror("Failed to unlock mutex");
            }
            pthread_barrier_wait(&arguments->barrier);
            break;
        }


        Process process = buffer->processes[buffer->out];
        buffer->out = (buffer->out + 1) % buffer->capacity;
        buffer->count--;

        int source_fd = process.src_fd;
        int destination_fd = process.dest_fd;

        if(pthread_cond_signal(&buffer->not_full) != 0){
            perror("Failed to signal not_full condition variable");
        }

        if(pthread_mutex_unlock(&buffer->mutex) != 0){
            perror("Failed to unlock mutex");
        }
```



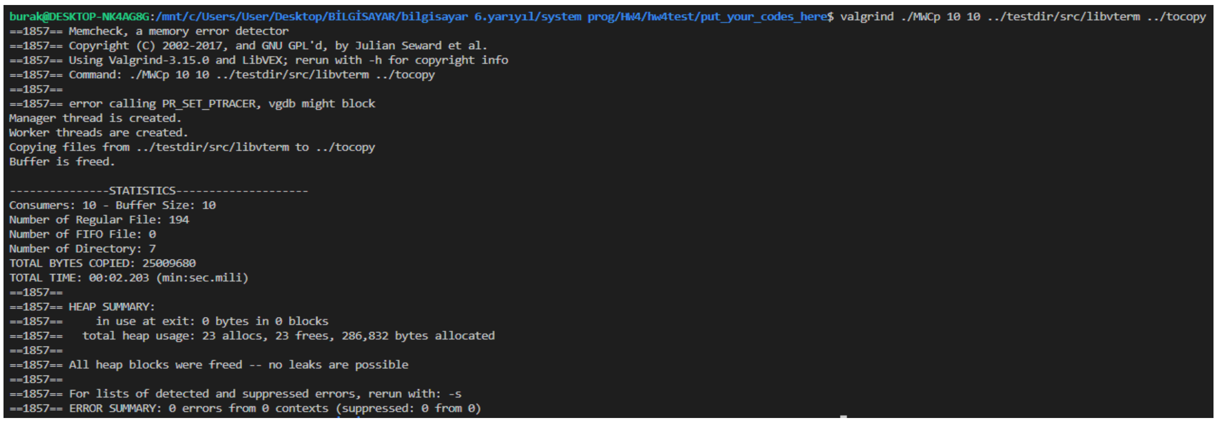
```c
        char fileBuffer[BUFFER_SIZE];
        ssize_t bytes_read, bytes_written;

        while ((bytes_read = read(source_fd, fileBuffer, BUFFER_SIZE)) > 0) {
            bytes_written = write(destination_fd, fileBuffer, bytes_read);
            if (bytes_written != bytes_read) {
                perror("Failed to write to destination file");
                break;
            }
            totalBytesCopied += bytes_read;
        }

        if(close(source_fd) == -1){
            perror("Failed to close source file");
        }
        if(close(destination_fd) == -1){
            perror("Failed to close destination file");
        }
    }
    return NULL;
}
```

The worker function loops indefinitely, locking the mutex for exclusive buffer access. If the buffer is empty, it waits on the not_empty condition or breaks if the manager is done or termination is signaled. If the buffer has processes, it removes one, signals not_full, and unlocks the mutex. It then reads from the source and writes to the destination file, updating the total bytes copied. Finally, it closes the source and destination files.

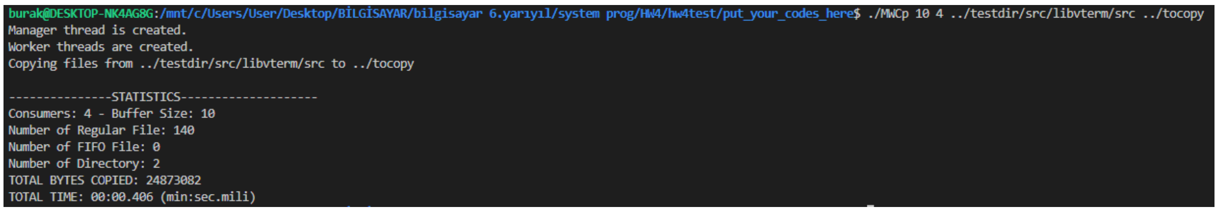
Main Function:

The main function parses command line arguments, sets up a SIGINT signal handler, and initializes the arguments structure and buffer. It creates a mutex and barrier, and then spawns the manager and worker threads. After all threads finish execution, it cleans up resources, calculates execution time, and prints program statistics.

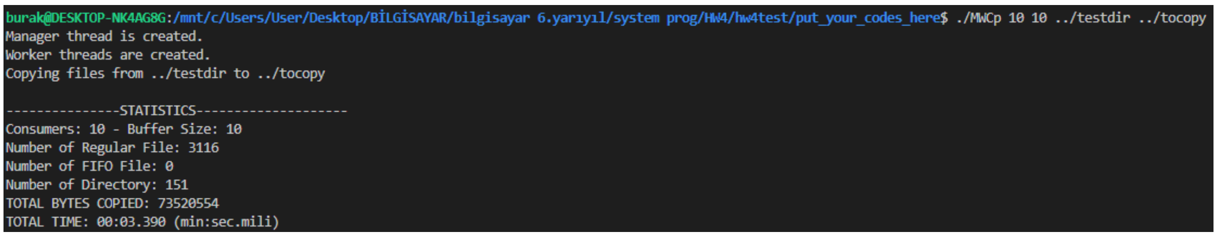
2. Test Results



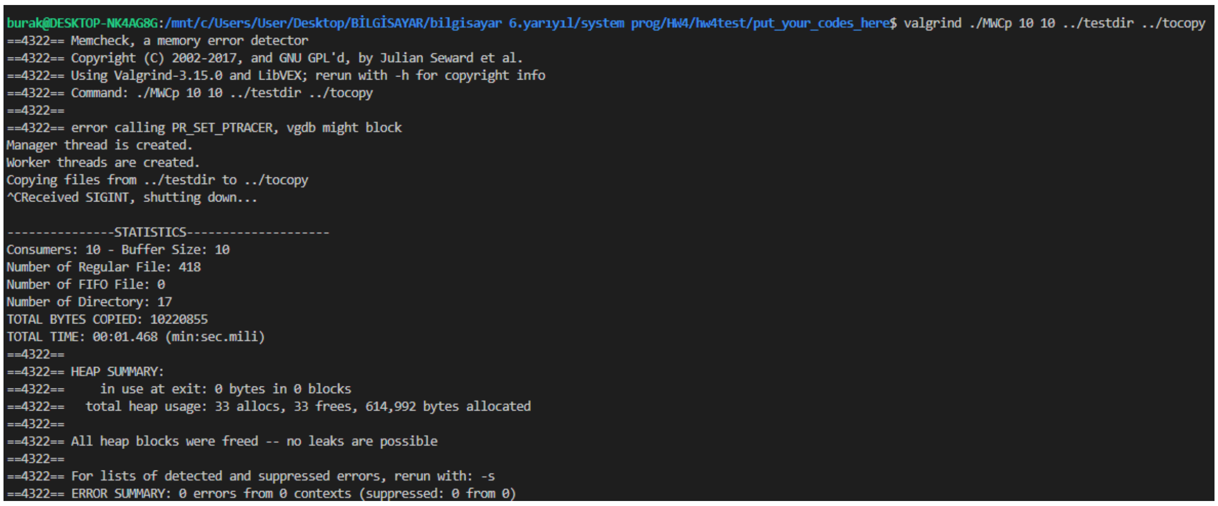
```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/system prog/HW4/hw4test/put_your_codes_here$ valgrind ./MWCp 10 10 ../testdir/src/libvterm ../tocopy
==1857== Memcheck, a memory error detector
==1857== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1857== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==1857== Command: ./MWCp 10 10 ../testdir/src/libvterm ../tocopy
==1857==
==1857== error calling PR_SET_PTRACER, vgdb might block
Manager thread is created.
Worker threads are created.
Copying files from ../testdir/src/libvterm to ../tocopy
Buffer is freed.

--------------STATISTICS-------------------
Consumers: 10 - Buffer Size: 10
Number of Regular File: 194
Number of FIFO File: 0
Number of Directory: 7
TOTAL BYTES COPIED: 25009680
TOTAL TIME: 00:02.203 (min:sec.mili)
==1857==
==1857== HEAP SUMMARY:
==1857==     in use at exit: 0 bytes in 0 blocks
==1857==   total heap usage: 23 allocs, 23 frees, 286,832 bytes allocated
==1857==
==1857== All heap blocks were freed -- no leaks are possible
==1857==
==1857== For lists of detected and suppressed errors, rerun with: -s
==1857== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```



```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/system prog/HW4/hw4test/put_your_codes_here$ ./MWCp 10 4 ../testdir/src/libvterm/src ../tocopy
Manager thread is created.
Worker threads are created.
Copying files from ../testdir/src/libvterm/src to ../tocopy

--------------STATISTICS-------------------
Consumers: 4 - Buffer Size: 10
Number of Regular File: 140
Number of FIFO File: 0
Number of Directory: 2
TOTAL BYTES COPIED: 24873082
TOTAL TIME: 00:00.406 (min:sec.mili)
```



```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/system prog/HW4/hw4test/put_your_codes_here$ ./MWCp 10 10 ../testdir ../tocopy
Manager thread is created.
Worker threads are created.
Copying files from ../testdir to ../tocopy

--------------STATISTICS-------------------
Consumers: 10 - Buffer Size: 10
Number of Regular File: 3116
Number of FIFO File: 0
Number of Directory: 151
TOTAL BYTES COPIED: 73520554
TOTAL TIME: 00:03.390 (min:sec.mili)
```



```
burak@DESKTOP-NK4AG8G:/mnt/c/Users/User/Desktop/BİLGİSAYAR/bilgisayar 6.yarıyıl/system prog/HW4/hw4test/put_your_codes_here$ valgrind ./MWCp 10 10 ../testdir ../tocopy
==4322== Memcheck, a memory error detector
==4322== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==4322== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==4322== Command: ./MWCp 10 10 ../testdir ../tocopy
==4322==
==4322== error calling PR_SET_PTRACER, vgdb might block
Manager thread is created.
Worker threads are created.
Copying files from ../testdir to ../tocopy
^CReceived SIGINT, shutting down...

--------------STATISTICS-------------------
Consumers: 10 - Buffer Size: 10
Number of Regular File: 418
Number of FIFO File: 0
Number of Directory: 17
TOTAL BYTES COPIED: 10220855
TOTAL TIME: 00:01.468 (min:sec.mili)
==4322==
==4322== HEAP SUMMARY:
==4322==     in use at exit: 0 bytes in 0 blocks
==4322==   total heap usage: 33 allocs, 33 frees, 614,992 bytes allocated
==4322==
==4322== All heap blocks were freed -- no leaks are possible
==4322==
==4322== For lists of detected and suppressed errors, rerun with: -s
==4322== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Last one is tested with SIGINT interrupt.