Report hw5

```
if(task_index >= task_capacity) pthread_cond_wait(&cond_task_copied, &mutex);                    //bir önceki ödevde buranın kontrolü yapılıyor yer yoksa array büyütülüyordu
snprintf(tasks[task_index], (strlen(source) + strlen(destination) + 2), "%s|%s", source, destination);
task_count++;
task_index++;
pthread_cond_broadcast(&cond_task_added);
pthread_mutex_unlock(&mutex);
```

In the previous assignment, a conditional variable was used in only one place: the `add_task` function. Here, it was checking the control and, if necessary, expanding the array. In the new version, it makes threads wait there. Unlike the previous assignment, a barrier has been added. I added a barrier to the worker threads' function. Threads wait for each other and restart from there once the appropriate number is reached.Previous report:

```
char** tasks;
int task_count = 0;
int task_index = 0;
int task_capacity = 0;
int worker_count;
int fifo_file_count = 0;
int directory_count = 0;
off_t total_bytes_copied = 0;
pthread_mutex_t mutex;
pthread_cond_t cond_task_added;
int ctrl_c = 0;
```

Here are the global variables of the program. The paths of the files to be copied are stored in a char array named tasks. Values to be held for the statistics section are globally assigned. Mutex and conditional variables are assigned. The Ctrl_c variable checks whether an interrupt signal has been received or not.

```
pthread_mutex_init(&mutex, NULL);
pthread_cond_init(&cond_task_added, NULL);

pthread_t manager;
pthread_t workers[worker_count];

char* manager_args[] = {source_directory, destination_directory};
pthread_create(&manager, NULL, manager_thread, manager_args);

pthread_join(manager, NULL);



for (int i = 0; i < worker_count; ++i) {
        pthread_create(&workers[i], NULL, copy_file, destination_directory);
}

for (int i = 0; i < worker_count; ++i) {
        pthread_join(workers[i], NULL);
}
```

Here is the part in the main function where the threads start. The manager thread completes its work and joins, and then the worker threads start.

```
void* manager_thread(void* arg) {
        char** args = (char**)arg;
        char* source_directory = args[0];
        char* destination_directory = args[1];

        list_files_recursive(source_directory, destination_directory);
        pthread_mutex_lock(&mutex);
        pthread_cond_broadcast(&cond_task_added);
        pthread_mutex_unlock(&mutex);

        return NULL;
}
```

The manager thread fills the tasks array with all the files in the directory recursively. If the list_files_recursive function finds a file, it populates it using the add_task function. If it encounters a directory, it calls itself with this directory as the parameter.

```
void* copy_file(void* arg) {
        while (1) {
                if(ctrl_c) return;

                pthread_mutex_lock(&mutex);

                if (task_index ≥ task_count) {
                        pthread_mutex_unlock(&mutex);
                        break;
                }
                char* task = get_task();
                pthread_mutex_unlock(&mutex);

                if (task == NULL) continue;
                char* task_copy = strdup(task);
                if (task_copy == NULL) {
                        perror("strdup");
                        continue;
                }
```

The `copy_file` function, which is the function of the worker threads, retrieves a path from the task array under the protection of mutexes to prevent race conditions. Each element of this `tasks` array contains both the source and the destination. Within the function, the source and destination are separated.

```
char* get_task() {

        while (task_index ≥ task_count) {
                pthread_cond_wait(&cond_task_added, &mutex);
        }
        char* task = tasks[task_index];
        task_index++;
        return task;
}
```

The `get_task` function retrieves the source and destination paths based on the current index.

OUTPUTS:

```
┌──(kali㉿kali)-[~/Desktop/system/hw4]
└─$ valgrind ./MWCp.out 10 10 hw4test/testdir/src/libvterm hw4test/tocopy
==39314== Memcheck, a memory error detector
==39314== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==39314== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==39314== Command: ./MWCp.out 10 10 hw4test/testdir/src/libvterm hw4test/tocopy
==39314==
--------------STATISTICS--------------------
Consumers: 10 - Buffer Size: 10
Number of Regular Files: 195
Number of FIFO Files: 0
Number of Directories: 8
TOTAL BYTES COPIED: 25009690
TOTAL TIME: 00:00.456 (min:sec.milli)
==39314==
==39314== HEAP SUMMARY:
==39314==     in use at exit: 0 bytes in 0 blocks
==39314==   total heap usage: 614 allocs, 614 frees, 533,420 bytes allocated
==39314==
==39314== All heap blocks were freed -- no leaks are possible
==39314==
==39314== For lists of detected and suppressed errors, rerun with: -s
==39314== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

```
┌──(kali㉿kali)-[~/Desktop/system/hw4]
└─$ ./MWCp.out 10 4 hw4test/testdir/src/libvterm/src hw4test/toCopy
--------------STATISTICS--------------------
Consumers: 4 - Buffer Size: 10
Number of Regular Files: 140
Number of FIFO Files: 0
Number of Directories: 2
TOTAL BYTES COPIED: 24873082
TOTAL TIME: 00:00.046 (min:sec.milli)
```

```
┌──(kali㉿kali)-[~/Desktop/system/hw4]
└─$ ./MWCp.out 10 10 hw4test/testdir hw4test/toCopy
--------------STATISTICS--------------------
Consumers: 10 - Buffer Size: 10
Number of Regular Files: 3117
Number of FIFO Files: 0
Number of Directories: 152
TOTAL BYTES COPIED: 73520564
TOTAL TIME: 00:00.309 (min:sec.milli)
```