Gebze Technical University

Department of Computer Engineering

CSE344 System Programming

Homework 5 Report

Burak Yıldırım
1901042609

Contents

1 Introduction

1.1 Project Description

The task of this project is to design and implement a directory copying system using threads with condition variables and barriers.

1.2 Compilation

```
CC = gcc
CFLAGS = -w
DFLAGS = -g
TFLAGS = -pthread
UTILS_SRCDIR = utils/src
UTILS_INCDIR = utils/include
TARGET = MWCp
SOURCES = ./1901042609_main.c $(wildcard $(UTILS_SRCDIR)/*.c)
HEADERS = ./1901042609_main.h $(wildcard $(UTILS_INCDIR)/**/*.h)
INCDIRS = -I. -I$(UTILS_INCDIR)
V_BUFFER =
V_WORKER =
V_SRC =
V_DEST =

export V_BUFFER
export V_WORKER
export V_SRC
export V_DEST

OS := $(shell uname)

ifeq ($(OS), Darwin)
        TFLAGS =
endif

all: $(TARGET)

test: $(TARGET)
        ./test.sh $(V_BUFFER) $(V_WORKER) $(V_SRC) $(V_DEST)

$(TARGET): $(SOURCES) $(HEADERS)
        $(CC) $(CFLAGS) -o $@ $(SOURCES) $(INCDIRS) $(TFLAGS)

debug: CFLAGS += $(DFLAGS)
debug: $(TARGET)

clean:
        rm -rf $(TARGET)

valgrind_memory: debug
        valgrind  ./$(TARGET) $(V_BUFFER) $(V_WORKER) $(V_SRC) $(V_DEST)

valgrind_thread: debug
        valgrind --tool=helgrind ./$(TARGET) $(V_BUFFER) $(V_WORKER) $(V_SRC) $(V_DEST)
```

Executing **make** compiles the project, **make clean** removes the executable, **make test** pressure tests the code by executing it 10 times in a row, **make valgrind_memory** launches the programs in Valgrind for memory leak checks, and **make valgrind_thread** launches the programs in Valgrind for thread errors.

2 General Structure

```
Hw5
├── 1901042609_main.c
├── 1901042609_main.h
├── utils
│   ├── include
│   │   └── utils
│   │       ├── array.h
│   │       ├── io.h
│   │       ├── string.h
│   │       └── type.h
│   └── src
│       ├── array.c
│       ├── io.c
│       └── string.c
├── Makefile
└── test.sh
```

This is the folder structure of the project.

- utils: utility functions and macros used by multiple files. The reason for adding an extra utils directory inside the include directory is to be able to call the utils headers like 'utils/io.h', etc.
- 1901042609_main: main functions and macros.
- Makefile: compile project and clean the executable.
- test.sh: pressure test the code by executing it 10 times in a row.

3 Implementation

3.1 Structs and Enums

```
typedef enum {
    REGULAR,
    DIRECTORY,
    FIFO,
    UNKNOWN,
    INVALID
} FileType;
```

```
typedef enum {
    FALSE = 0,
    TRUE = 1
} Bool;


typedef struct {
    int src_fd;
    int dest_fd;
    char src[PATH_MAX];
    char dest[PATH_MAX];
} CopyInfo;
```

3.2 Steps

The main program is implemented following these steps:

1. The command line arguments are verified.
2. `SIGINT` is handled.
3. `cleanup` is registered with `atexit`.
4. Barrier is initialized using `pthread_barrier_init`.
5. Source and destination directories are opened. If destination directory doesn't exist, it's created.
6. Manager thread is created using `pthread_create`.
7. Worker threads are created `pthread_create`.
8. Manager thread is waited using `pthread_join`.
9. Worker threads are waited using `pthread_join`.
10. Elapsed time is calculated and the statistics are printed.
11. `cleanup` function is called to free all the dynamically allocated variables, close the original source and destination folders, and destroy all the mutexes, the condition variables and the barrier.
12. Program finishes.

3.3 Details

All access to the global variables are done between `pthread_mutex_lock(&general_mutex)` and `pthread_mutex_unlock(&general_mutex)`.

3.3.1 manager

Manager thread runs in a loop until all the files in the source directory and its subdirectories are opened and their file descriptors as well as their names are written to the buffer or until the `is_finished` flag is set. While reading the source directory if the current entry is a regular file, it calls the `handle_regular` function which does the following:

- open the source file for reading with `open`.
- create or truncate the destination file with `open`.
- lock the `general_mutex`.
- check if the current occupancy of the buffer is equal to the buffer size or if there is no available workers.
- if one of them is true wait for `buffer_not_full_cond`.
- if none of them is true or if the `buffer_not_full_cond` is signaled, put the file descriptor and name information of the source and the destination files in the buffer and increment the current occupancy of the buffer by 1.
- signal the `buffer_not_empty_cond` so that a worker can wake up and do the copying.
- unlock the `general_mutex`.

If the current entry is a FIFO, it calls `handle_fifo` which follows the same steps as `handle_regular` with one addition. This time before opening the source file, the FIFO is created with `mkfifo` at the destination path. The rest is the same. If the current entry is another directory, it calls the `handle_directory` which opens the directory, runs in a loop until all of its entries are read or until the `is_finished` flag is set, and acts according to the type of the file as mentioned before, e.g. calls `handle_regular` for regular files, `handle_fifo` for FIFOs, and `handle_directory` for directories. This way all of the files in the source directory is copied to the destination directory recursively. After all the files are read, it sets the `is_finished` flag to `TRUE` and waits for the barrier using `pthread_barrier_wait(&barrier)`.

3.3.2 worker

Worker thread runs in an infinite loop until the `is_finished` flag is set. At each iteration it waits the `buffer_not_empty_cond`. After waiting, it checks if the `is_sigint` flag is set. If it is, it unlocks the `general_mutex` and breaks the loop. If not, it checks if the current occupancy of the buffer is 0. If it is, it unlocks the `general_mutex`, checks whether the `is_finished` flag is set. If the flag is set then it breaks the loop, if the flag is not set then it goes back to the beginning of the loop waiting for `buffer_not_empty_cond`. If the current occupancy of the buffer is not 0, then it decrements the current occupancy and the available thread number by 1. Then it gets the first element of the buffer, shifts the buffer to the left, and unlocks the `general_mutex`. Then it copies the source file to destination file in loop where it increments the `total_bytes` inside a critical region made by `general_mutex`. After the copy loop it locks the `general_mutex`, increments the available thread number by 1, signals the `buffer_not_full_cond`, and unlocks the `general_mutex`. After the loop it waits for the barrier using `pthread_barrier_wait(&barrier)`.

3.3.3 Signal Handling

3.3.3.1 SIGINT

In the `SIGINT` handler, first `is_finished` and `is_sigint` flags are set to `TRUE`, and `buffer_not_empty_cond` and `buffer_not_full_cond` are broadcasted. Then the manager thread and the worker threads are joined. Then the elapsed time is calculated and the statistics are printed. Finally, the `cleanup` function frees all the dynamically allocated variables, closes the original source and destination folders, and destroys all the mutexes, the condition variables and the barrier.

4 Testing

4.1 Test 1

```
---------------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: 00:01.130 (min:sec.mili)
==6842==
==6842== HEAP SUMMARY:
==6842==     in use at exit: 0 bytes in 0 blocks
==6842==   total heap usage: 39 allocs, 39 frees, 610,168 bytes allocated
==6842==
==6842== All heap blocks were freed -- no leaks are possible
==6842==
==6842== For counts of detected and suppressed errors, rerun with: -v
==6842== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

4.2 Test 2

```
---------------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: 00:00.073 (min:sec.mili)
```

4.3 Test 3

```
---------------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: 00:01.083 (min:sec.mili)
```

4.4 SIGINT

```
SIGINT signal received. Printing the statistics so far and exiting...

Copied ../testdir/src/textprop.c to ../toCopy/src/textprop.c
Copied ../testdir/src/xdiff/COPYING to ../toCopy/src/xdiff/COPYING
Copied ../testdir/src/gui.c to ../toCopy/src/gui.c
Copied ../testdir/src/mark.c to ../toCopy/src/mark.c
Copied ../testdir/src/evalfunc.c to ../toCopy/src/evalfunc.c
Copied ../testdir/src/auto/configure to ../toCopy/src/auto/configure
Copied ../testdir/src/libvterm/src/isl1551098739430.pdf to ../toCopy/src/libvterm/src/isl1551098739430.pdf

---------------STATISTICS--------------------
Consumers: 10 - Buffer Size: 10
Number of Regular File: 396
Number of FIFO File: 0
Number of Directory: 17
Bytes copied so far: 36961236
Elapsed time so far: 00:01.408 (min:sec.mili)
==6921==
==6921== HEAP SUMMARY:
==6921==     in use at exit: 0 bytes in 0 blocks
==6921==   total heap usage: 59 allocs, 59 frees, 1,266,488 bytes allocated
==6921==
==6921== All heap blocks were freed -- no leaks are possible
==6921==
==6921== For counts of detected and suppressed errors, rerun with: -v
==6921== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```