**0512.4402: Introduction to Programming Systems**

**HW2**

Unfortunately, my partner decided to leave the course while working on the current submission.

My test command is:

* make clean
* make
* ./hw2 test.txt 400 2 1

In the end:

* count00.txt contains 0.
* count00.txt contains 1.

**Modules**

I used to modules:

1. dispatcher.h, dispatcher.c
2. string\_utils.h, string\_utils.c
3. dispatcher module contains:

struct MyThread {

// Threads

long long jobs\_counter;

int64\_t start\_time;

int64\_t total\_start\_time;

int64\_t total\_run\_time;

int64\_t min\_time;

int64\_t max\_time;

pthread\_t tid;

int index;

bool busy;

int commands\_amount;

bool log\_enabled;

char\*\* commands;

} Thread;

The struct receives commands from the dispatcher and execute them till no more commands are pending.

The flag busy is set when the thread is executing a command.

typedef struct MyDispatcher {

// Dispatcher

int64\_t start\_time;

int64\_t total\_start\_time;

char\* filename;

int num\_threads;

int num\_counters;

bool log\_enabled;

Thread\*\* threads;

} Dispatcher;

The struct hold all the threads created based on the input.

The dispatcher contains init, free and print (debug purposes) for both structs.

Both structs keep track on the time since the beginning of the program execution to update the log files.

The functions which are used from the main flow are:

int get\_unbusy\_thread(Dispatcher\* dispatcher);

void msleep(char\* command);

void thread\_change\_counter(int i, int change);

void dispatcher\_wait(Dispatcher\* dispatcher);

Dispatcher\* setup\_dispatcher(char \*argv[], time\_t total\_start\_time);

void free\_dispatcher(Dispatcher\* dispatcher);

void print\_thread(Thread\* thread);

void print\_threads(Thread\*\* threads, int num\_threads);

void print\_dispatcher(Dispatcher\* dispatcher);

Other ones are inner functions.

1. string\_utils contains:
2. char strip(char c);
3. bool string\_starts\_with(char\* str, char\* prefix);
4. void get\_substring(char\*\* str\_ptr, char\*\* sub\_str\_ptr, int ind);
5. int count\_words(char\* line);
6. char\*\* split(char\* line);
7. void replace\_repeat(char\*\*\* commands\_ptr, int\* len\_ptr);

All the functions are used to parse the input commands.

**Syncing:**

I used mutex locks to lock the used counter files:

pthread\_mutex\_t locks[MAX\_NUM\_COUNTERS];

Whenever some thread had to use the counter file the sequence was:

1. Locking the required lock.
2. Updating the counter file.
3. Unlocking the required lock.
4. pthread\_mutex\_lock(&locks[i]);
5. thread\_change\_counter(i, 1);
6. pthread\_mutex\_unlock(&locks[i]);

Mutex are initiated with default settings:

for (i=0; i<MAX\_NUM\_COUNTERS; i++){

pthread\_mutex\_init(&locks[i], NULL);

}

They are killed at the end of the program:

for (i=0; i<MAX\_NUM\_COUNTERS; i++){

pthread\_mutex\_destroy(&locks[i]);

}

**Commands Parsing:**

I used string\_utils module to tranfrom the input line into matrix which holds a command for each basic command in the line.

Whenever I faced a "repeat x" instruction I extracted it to it's longer version, for example:

Line = repeat 2; msleep 1; increment 1

Matrix = [msleep 1, incement, msleep1, increment 1]

If the line started with worker, the matrix is being sent to a single thread, otherwise each thread is being send to some unbusy thread.

**Waiting:**

Whenever a thread was not busy, it was sent to sleep for a single millisecond.

void\* run\_thread(void\* args)

{

// Runs the thread commands if exist, o.w sleeps

Thread\* thread = args;

char\* command = NULL;

while (true){

if (thread->busy){

busy\_thread(thread);

}

usleep(1000);

}

}

I was instructed not to kill threads while running the main flow, therefore, the thread is being sent to sleep whenever it does not have pending commands.

**Time Reading:**

Reading time in millisecond was not easy, at first I used clock() although I have found that is not accurate at all.

Therefore, I used:

int64\_t millis()

{

// Return current time in millisecond since EPOCH

struct timespec now;

timespec\_get(&now, TIME\_UTC);

return ((int64\_t) now.tv\_sec) \* 1000 + ((int64\_t) now.tv\_nsec) / 1000000;

}

Using both seconds and nanosecond the resolution was much better.

**The Main Flow:**

The main flow is long, my partner left me, so I didn't have time to split it to multiple functions, so I'll break it to blocks:

1. Timing the beginning of the program.
2. Setting up the variables, dispatcher, threads.
3. For each line int the file, parse in into commands matrix.
   1. If the command is a dispatcher type, handle it using dispatcher functions.
   2. If the command is thread type, dispatcher will allocate the commands for unbusy threads.
4. When the file is over the dispatcher will wait till all the threads are done, using the busy bool flag for each thread.
5. The dispatcher will kill all the threads.
6. The dispatcher will disable each lock mutex.
7. The statistics will be printed into the stats file.
8. Free all the memory being used.

**File Names:**

1. threadxxxx.txt
2. countxx.txt
3. stats.txt