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

1. 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.

2. string_utils contains:

```
3.
4.    char strip(char c);
5.
6.    bool string_starts_with(char* str, char* prefix);
7.
8.    void get_substring(char** str_ptr, char** sub_str_ptr, int ind);
9.
10.   int count_words(char* line);
11.
12.   char** split(char* line);
13.
14.   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.
    a. If the command is a dispatcher type, handle it using dispatcher functions.
    b. 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.