System Programming, Fall 2023
Homework 4

Due: 23:59:59, Sun, Dec 17, 2023

Late Submission Due: 23:59:59, Thu, Dec 21, 2023

TA E-mail: `ntusp.class2@csie.ntu.edu.tw`

Rules and Instructions

- Any form of cheating, lying, or plagiarism will not be tolerated. Students can get zero scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.

- Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources can be consulted, but not copied from. Previous or classmates' work will be considered as plagiarism.

- Since everyone needs to write the final solutions alone, there is **absolutely no need to lend your homework solutions and/or source codes to your classmates at any time.** In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.

- If you have questions about the homework, please go to the discord channel and discuss (*strongly preferred*, which will provide everyone with a more interactive learning experience). If you really need an email answer, please follow the rules outlined below to get a fast response:

  – The subject should contain tag, `"[hw4]"`, specifying the problem where you have questions. For example, `"[hw4] Is thread num need to be` N `or` N`+1"`. Adding tags allows the TAs to track the status of each email and to provide faster responses to you.

  – If you want to provide your code segments to us as part of your question, please upload it to Gist or similar platforms and provide the link. Please remember to protect your uploaded materials with proper permission settings. Screenshots or code segments directly included in the email are **discouraged** and **may not be reviewed**.

Proper References (0 pts)

For each problem below, please specify the references (the Internet URL you consulted with or the classmates/friends you discussed with) in your PDF (i.e. your report) submitted to NTU COOL. If you have used chatGPT or similar tools, please provide a quick snapshot of your interaction with the tools. *You do not need to list the TAs/instructors.* If you finished any problem all by yourself (or just with the help of TAs/instructors), just say “all by myself.” While we did not allocate any points to this problem, failure to complete this problem can lead to penalty (i.e. negative points). Examples are

- Programming Part: Alice (B12902000), Bob (B11902000), and `https://stackoverflow.com/questions/982388/`
- Hand Written Part: all by myself
- ...

Listing the references in this problem does *not* mean you could copy from them. We cannot stress this enough: *you should always write the final solutions alone and understand them fully.*

Craft a Judge (5 pts)

Problem Description

There are many students in System Programming class. On any given time before deadline, each of the student are able to submit code. After all submissions, PJ and Thisway both want to know the result of the submission. And below comes their implementation:

- The judge runs with $n\_threads + 1$ threads, including one main thread and $n\_threads$ judge threads. For the main thread, it's responsible for accepting the submission from different students with the API `tpool_add`. After that, judge will schedule the work in a FIFO. In other words, the earlier is a job push into job queue via `tpool_add`, the earlier will it be handled by a judge thread. The task will be executed once there's a free judge thread.

Your goal is to help PJ and Thisway to implement the judge library.

Detail

You have to implement and fill the given file(s): `my_pool.h`, `my_pool.c`. Basically, all functions you have to implement is defined in `my_pool.h`. You may also need to add members to each structure defined in `my_pool.h`. The compile command is attached in `Makefile`.

- `tpool *tpool_init(int n_threads)`: You have to create a pointer of thread pool which have $1 + n\_thread$ thread and initialize it with correspond attribute.
- `void tpool_add(tpool *pool, void *(*func)(void *), void *arg)`: The function add a task to the judge `pool` in a **FIFO** job queue, where the task will be called with `func(arg)`.
- `void tpool_wait(tpool *pool)`: The function wait for all tasks on the judge ends in **blocking** way.
- `void tpool_destroy(tpool *pool)`: The function kill all threads and release all memory allocated before.

Note & FAQ

- It's promised that there's a `tpool_wait` before every `tpool_destroy`.
- You can follow the instructions in `README.md` to compile your program.
- **AVOID** busying waiting on taking job from job queue, that is, one thread should be blocked until the desired lock released.
- The judge will be compiled with following command, where `main.c` is the test program which includes `my_pool.h`:
  ```
  gcc main.c my_pool.c -std=c11 -O2 -pthread -o main
  ```
- It's prohibited to use **ANY** global or static variables (except a function), `setjmp` / `longjmp` family function call, and memory leakage (-1 pt if any).
- You **don't** have to free any arguments passed into functions.
- You **don't** have to care about any time limit of functions, i.e., it's promised that the given functions and arguments can be run in finite time.
- You **ONLY** initialize the thread pool with *n_threads* extra threads at the beginning, that is, you **CANNOT** allocate threads and destroy threads during the thread pool works.

Sample Testcase

Sample Program

```
void *func0(void *args) {
  printf("func 0: %d\n", *((int *)args));
  fflush(stdout);
  return NULL;
}
void *func1(void *args) {
  printf("func 1: %d\n", *((int *)args));
  fflush(stdout);
  return NULL;
}
#define N 2
#define M 3
int main() {
  tpool *pool = tpool_init(N);
  void *(*funcs[M])(void *) = {&func0, &func1, &func0};
  int *arg = malloc(M * sizeof(int));
  for (int i = 0; i < M; i++) {
    arg[i] = i;
    tpool_add(pool, funcs[i], (void *)&arg[i]);
  }
  tpool_wait(pool);
  tpool_destroy(pool);
  free(arg);
}
```

Sample Output

```
func 0: 0
func 1: 1
func 0: 2
```

Sample Output(Also acceptable)

```
func 1: 1
func 0: 0
func 0: 2
```

Report (3 + Bonus 0.5pts)

Please submit your report with answering following questions:

1. Proper Reference.
2. Specify why there are many possible answers in Sample Testcase? Provide a solution e.g. a `main.c` that can generate output in deterministic order and explain it. (1pt)
3. Explain the function of each mutex or condition variable if any. (0.5pt, if no mutex, condition variable, why you don't need them)
4. How do you avoid a busy waiting in both main thread (accept submissions, wait for children threads ...) and different judge threads(take jobs...)? (0.5pt)
5. Plot a *t-n_threads* graph with at least 10 data points, where t is real time, user time, and sys time to deal with numerous time-consuming tasks. The data points in x-axis(n_threads) must follow the constraint:{data points} $\supset \{1, 2, 5, 10, 50, 100\}$. Discuss what you observe. Note: `time` command is your friend. And you can test with `bench/main.c` in the repository for testing. (1pt)
6. Bonus: What is ABA problem? Does your implementation resist against ABA problem? Why / Why not?(0.5pt)

Submission

Submit your code to GitHub and report named `report.pdf` via NTU COOL, respectively.

Late submission policy

(D refers to formal deadline, 12/17 23:59)

- If you submit your assignment before D, your final score of this homework is same as your raw score.
- If you submit your assignment on (D, D+1], your final score of this homework is your raw score multiplied by 0.85.
- If you submit your assignment on (D+1, D+2], your final score of this homework is your raw score multiplied by 0.7.

- If you submit your assignment on (D+2, D+3], your final score of this homework is your raw score multiplied by 0.6.
- If you submit your assignment on (D+3, D+4], your final score of this homework is your raw score multiplied by 0.5.

Late submission after 12/21 23:59 will **NOT** be accepted.