System Programming Homework 4

鮑鈺文b09902016

4.

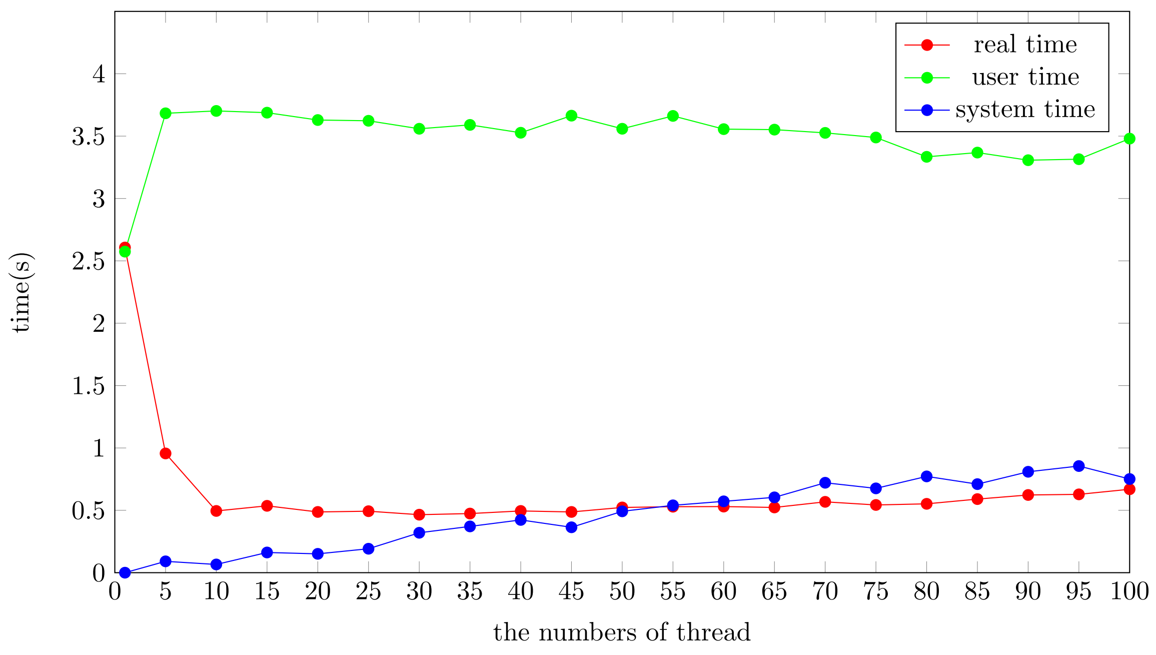
number of threads	2	20
average real time(s)	0.646	0.152
average user time(s)	1.263	2.175
average system time(s)	0.001	0.030

For real execution time, the process with 2 threads is more than the process with 20 threads, because evaluating the next board takes most of time in this program. Hence, the more threads there are evaluating the next board at the same time, the less time it takes theoretically. For user time and system time, process with 2 threads is less than process with 20 threads, because the less threads a process have, the less it have to trap into the system or be blocked by lock.
For multithread, the process first using fgetc read the file and then using two 1-dimensional arrays to record the previous and the next states of the board. Divided the 1-dimensional arrays to the number of threads and assign each parts to the thread respectively. Use *pthread_barrieer* wait the spawned threads after each epoch. Using *fputs* to output the result to the output file.

```
void *spawn_eval(void *arg) {
    // initialize local variables
    for (int i = 0; i < epoch_num; i++) {
        //refresh the board
        pthread_barrier_wait(lk);
    }
}

void thread_mode() {
    long int delta;
    pthread_t tidp[thread_num];
    // initialize global and local variables
    while (((*pre)[ptr] = fgetc(in_fp)) != EOF) ++ptr;
    strcpy(*nxt, *pre);
    (*pre)[ptr] = 0;
    (*nxt)[ptr] = 0;
    seg[0] = 0;
    delta = ptr / thread_num;
    for (int i = 1; i < thread_num; i++) seg[i] = seg[i - 1] + delta;
    seg[thread_num] = ptr;
    pthread_barrier_init(lk, NULL, thread_num);
    for (int j = 0; j < thread_num; j++)
        pthread_create(&(tidp[j]), NULL, spawn_eval, (void *)(long long)j);
    for (int j = 0; j < thread_num; j++)
        pthread_join(tidp[j], NULL);
    if (epoch_num % 2) fputs(*nxt, out_fp);
    else fputs(*pre, out_fp);
}
```

5.



For system time and user time, more threads are created, more time it will take, since the system have to deal with a lot of copying the thread stacks. For real time, the number of threads makes different when the number of threads is around 0 25, because there are more threads calculating the next board at the same time. However, when the number of threads becomes greater, the time cost by handling the locks, spawning the threads and switching between threads becomes significant, the real time it takes to finish the test increases instead. What's more, the threads might finish a task before every thread get a task, which means there will be more idle threads as the number of threads increasing.

6.

type	multithread	multiprocess
average real time(s)	0.145	0.675
average user time(s)	2.176	1.317
average system time(s)	0.063	0.0.000

For execution time, multithread is less then multiprocess in this task, because generating a thread required less memory copying than generating a process even though using mmap() to share memory and to memorize the the boards. For user time, they are much the same because they do much the same amount of time calculating the board.
For multiprocess, it do the same thing as multithread, except using $fork()$ to make child processes and *wait* to retrieve them instead of spawning threads and join them.

```
void child_proc(const int idx) {
    int seg_idx = seg[idx];
    int row = seg[idx] / (col_num + 1), col = seg[idx] % (col_num + 1), count = 0;
    for (int i = 0; i < epoch_num; i++) {
        seg_idx = seg[idx];
        // refresh board
        pthread_barrier_wait(lk);
    }
    exit(0);
}

void proc_mode() {
        // initialize global and local variables
        while (((*pre)[ptr] = fgetc(in_fp)) != EOF) ++ptr;
        strcpy(*nxt, *pre);
        (*pre)[ptr] = 0;
        (*nxt)[ptr] = 0;
        seg[0] = 0;
        delta = ptr / thread_num;
        for (int i = 1; i < thread_num; i++) seg[i] = seg[i - 1] + delta;
        seg[thread_num] = ptr;
        pthread_barrierattr_init(&lk_attr);
        pthread_barrierattr_setpshared(&lk_attr, 1);
        pthread_barrier_init(lk, &lk_attr, thread_num);
        for (int j = 0; j < thread_num; j++)
            pthread_create(&(tidp[j]), NULL, spawn_eval, (void *)(long long)j);
        for (int j = 0; j < thread_num; j++)
            pthread_join(tidp[j], NULL);
        if (epoch_num % 2) fputs(*nxt, out_fp);
        else fputs(*pre, out_fp);
    }
```

簽名：

鮑鈺文