Prelab: 06
Demo: 34
Report: 45
Delay: 0
Total: 85

Lab2-Report

Haoxiang Zhang

September 23, 2019

Contents

1 Week-1 **1**

1.1 Assignment 1

2 Week-2 **2**

2.1 Assignment 2

I Code Section **3**

3 Lab-2 Week-1 **3**

4 Lab-2 Week-2 **9**

Postlab

1 Week-1

1.1 Assignment

Objectives and Lab Description

Read a matrix in the .txt file. Then create some threads to searching the specific number. Then return the execution time. In the .txt file. the first line is the size of the matrix, and start from second line. there is a matrix.

Implementation

First of all, create 4 threads that will do the four different searching ways. Secondly, in the main function, read the matrix from file. In order to split the numbers from the whole line. I use ***strtok(string, split_ symbol)***. This function is from ***string.h***. The pointer will stop each time when meet a white space. After storing size and matrix data, start to create threads. Each thread should build every thing separately. In the thread part, first, declear a thread id, the n create a thread with function **pthread_create(*threadid, *attribution, thread_fun, *param)**. Remember to use **pthread_join(threadid, status)** to check the thread whether is done or not. In each thread function, there is a param passed from main with type void. first convert param to **long** type. the searching the number in the matrix.

Did you use - i) different for-loop to join all threads a ii) global variable to "count" the number of elements found in each thread.

Experiments and Results

When run the code, it will print the size of the matrix and whole matrix data. Then will ask you which number you want to search. After that, print all the position of the number that found and the sum of the require number. At last, it gives a execution time for searching.

Discussion and Postlab Questions

Case\Avg. Search Time	(2x100)	(15x15)	(20x10)
One thread search the entire matrix	0.000722s	0.000753s	0.000298s
One thread to search each row	0.000693s	0.002071s	0.002380s
One thread to search each col	0.009771s	0.001939s	0.001338s
One thread search each element	0.015582s	0.018228s	0.016144s

i) Add some explanation to this table

ii) Also add output displayed on terminal.

For the data above, it is obvious that usually to searching, one thread is enought. For some of situation, like 2x100 matrix, I think to create 4 threads that each one search 2x25 size. The execution time should be short. One thread search each element. It is not effeciency, because when create thread, then start to searching, it takes more time. After finish searching, each one should be terminated, which wastes many time.

2 Week-2

2.1 Assignment

Objectives and Lab Description

Create a real time task that have three thread (two read and one write) and one buffer. Each reader will read the content from a file and put into buffer, then the writer will put the content from buffer to an array. The thread order should be (reader1 -> writer -> reader2 -> writer ...etc). At the last step print the data in the array, then the content will be a reasonable story.

Implementation

First of all, declare three threads, one buffer, and one character array. The priority should be set asd 51 because the kernel's priority is 50. If the priority is less than kernel's, then the reading and writing will not be executed expectly. Second step will be create three thread functions and create the threads in the main (pthread_create() and ptherad_join()). In the thread function, first of all we should configure a timer for the thread, which allowed the schedulor to count the time. For this part, first, we should setup the period time which is the cycle time; then, we should set up the start time which means the thread will be start in this time in each period time; at last, set the time with using **timerfd_settime(timer_fd, 0, timerspec, NULL)**. The second part is to initialize current thread. In this part, first, we should setup priority; then create a loop and put the **sche_setscheduler(0, schedule_mode, param)** which set the priority to the thread. The third part will be setup **read(timer_fd, number of periods, size of periods)**. This part is in order to check the number of bytes read. If the reader read to end of file, it will return **ZERO**. Between ***setscheduler*** and ***read*** there is the code for reading or writing.

-2

Also say how did you synchronize the threads ??

↳ Which file ?

Experiments and Results

-3

When run the code, at beginning, the execution order will be reader1, reader2, writer, then will start the schedule, so the result will not print every thing and the order may not correct as well. So I check the code and give each thread a start point. Then the result printed well.

↳ How do I know?

Discussion and Postlab Questions

Add some figures with explanation of the output.

For getting the faster execution time, at first I set the two reader threads' period as 4 second, and writer's period as 2 second. it runs well but takes a lot of time. then I set the readers' period as 1 second and start time is 0.25 second and 0.75 second, then set writer's period and start time as 0.5 second. it went to faster. Because these operations are really fast, the execution time could be more faster.

Part I

Code Section

3 Lab-2 Week-1

Comments: 5/5
Code: 34/40
i) Outputs are not consistent among all the threads. (-2)
ii) Synchronizing among the threads explanation is not correct. (-3)
iii) Using "global" sum variable to count the numer of hits for given search number. (-1)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <sys/time.h>
#define MAX_LINE 500
int row;
int col;
int *matrix;
int num;
int sum=0;
struct timeval start, end;
/*
*(matrix+i*row + j is the address of point(i,j)
 the address of matrix is an 1_D array.
 like row1col1 | row1col2 | row1col3 | row2col1 | row2col2 | row2col3
*/
// One thread for searching the entire matrix
void* oneT_all(){
    printf("prepare to searching %d\n", num);
    int i;
    int j;
    printf("the number found in: \n");
    for (i=0; i<row; i++) {
        for (j=0; j<col; j++) {
            if(*(matrix+i*row + j) == num) {
              printf("(%d, %d) ", i, j);
               sum += 1;
                        }
       }
       printf("\n");
    }
    printf("Totally found %d numbers!\n", sum);
    pthread_exit(0);
}
// One thread for searching each row of the matrix
void* oneT_row(void *tid){
    long row_num;
```

```
    row_num = (long)tid;
    printf("\nthread #%ld...\n", row_num);
    int j;
    for (j = 0; j < col; j++) {
        if (*(matrix + row_num*row + j) == num) {
            printf("(%ld, %d) ", row_num, j);
            sum += 1;
        }
    }
    pthread_exit(0); }
// One thread for searching each column of the matrix
void* oneT_col(void *tid){
    long col_num;
    col_num = (long)tid;
    printf("\nthread #%ld...\n", col_num);
    int i;
    for (i = 0; i < col; i++) {
        if (*(matrix + i*row + col_num) == num) {
            printf("(%d, %ld) ", i, col_num);
            sum += 1;
        }
    }
    pthread_exit(0);
}
// One thread for searching each element of the matrix
void* oneT_one(void *tid){
    long n;
    n = (long)tid;
    if (*(matrix + n) == num) {
        int r = (int)n/(col-1);
        int c = (int)n%(col-1);
        printf("(%d, %d)\n", r,c);
        sum += 1;
    }
    pthread_exit(0);
}
/*
    main function
 */
int main (int argc, char **argv){
    /*
     * Task1: reading data from file
    */
    FILE *fptr;
    char line[MAX_LINE];
```

```
char *result; // line
char *p;
char filename[15] = "20x10.txt";
printf("reading from <%s> .....\n", filename);
fptr = fopen(filename, "r");
// check the access for file
if(fptr == NULL){
   printf("Cannot open file <%s> \n", filename);
   exit(0);
}
result  =fgets(line, MAX_LINE, fptr);
// reading
/*
     strtok(string, split_value)
     strtok read one word each time. (everytime meet space will stop)
*/
p = strtok(result, " ");
if(p != NULL){
    row = atoi(p);
    p = strtok(NULL, " ");
    col = atoi(p);
}
printf("%d x %d\n", row, col);
int m[row][col];
// reading matrix
int i;
int j;
for (i = 0; i < row; i++){
   result = fgets(line, MAX_LINE, fptr);
   p = strtok(result, "\t");
   if(p != NULL){
        m[i][0] = atoi(p);
   }
   for (j = 1; j < col; j++){
       p = strtok(NULL, "\t");
      if(p != NULL){
         m[i][j] = atoi(p);
       }
   }
}
matrix = &m[0][0];
     // printing m
for(i = 0; i < row; i++){
    for(j = 0; j < col; j++){
       printf("%d   ", m[i][j]);
```

```
        }
      printf("\n");
    }
    fclose(fptr);

    printf("============================== Start ==============================\n");
    /*
     *  Task2
  */
    printf("Please enter a number you want to search:\n");
    scanf("%d", &num);
    printf("Searching %d ...\n", num);
    printf("argc: %d\n", argc);
    gettimeofday(&start, NULL);
    if (argc == 1) {
      /*
       * One thread to search the entire matrix
       */
      // thread id
      pthread_t thread;
       // create attributes
       pthread_attr_t attr;
       pthread_attr_init(&attr);
            pthread_create(&thread, &attr, oneT_all, NULL);
       pthread_join(thread, NULL);
                    }
     else if (argc == 2){
      /*
       * One thread for searchingeach row of the matrix
       */
      // thread id
       pthread_t thread[row];
       // create attributes
      pthread_attr_t attr;
      pthread_attr_init(&attr);
       long t;
      for (t=0; t<row; t++) {
          pthread_create(&thread[t], &attr, oneT_row, (void *)t);
      }
      for (t=0; t<row; t++) {
           pthread_join(thread[t], NULL);
      }
     }
    else if (argc == 3){
       /*
```

```
        * One   thread   for   searchingeach column of the matrix
        */
       // thread id
       pthread_t thread[col];
       // create attributes
       pthread_attr_t attr;
       pthread_attr_init(&attr);
       long t;
       for (t=0; t<col; t++) {
           pthread_create(&thread[t], &attr, oneT_col, (void *)t);
       }
       for (t=0; t<col; t++) {
          pthread_join(thread[t], NULL);
       }
    }
    else if (argc == 4){
       /*
       * One thread for each elementof the matrix
       */
      long l = row * col;
       //struct Point *pp;
       // thread id
      pthread_t thread[l];
       // create attributes
      pthread_attr_t attr;
       pthread_attr_init(&attr);
       long t;
        // pass the id of thread.
        // because the id of thread is same position of the index of matrix pointer.
       for (t=0; t<l; t++) {
           pthread_create(&thread[t], &attr, oneT_one, (void *)t);
       }
       for (t=0; t<l; t++) {
           pthread_join(thread[t], NULL);
      }

         } else {
       printf("Wrong argc... done!");
       exit(0);
   }
   printf("\nsum is %d.....\n",sum);
   gettimeofday(&end, NULL);
float time = ((float)end.tv_sec - (float)start.tv_sec) + ((float)end.tv_usec - (float
    printf("time used: %lfs\n", time);
    // exit
```

```
    pthread_exit(0);
        return 0;
}
```

4 Lab-2 Week-2

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sched.h>
#include <stdint.h>
#include <stdlib.h>
#include <sys/timerfd.h>
#include <time.h>
#include <pthread.h>
#include <string.h>
#define MY_PRIORITY 51  // kernel is priority 50

pthread_t thread_r1, thread_r2, thread_w;
char buf[100];          // buffer
char c[20][100];        // store the reading
int line = -3;          // total lines from reading
// set line = -3, because the when just load thread1 and thread2, the order has some p
FILE *fptr1, *fptr2;    // file reading pointers
struct timeval start, end; // timer

void* r1() {
        char fn1[15] = "first.txt";
        fptr1 = fopen(fn1, "r");
        int timer_fd1;
        timer_fd1 = timerfd_create(CLOCK_MONOTONIC, 0);
        if(timer_fd1 == -1) printf("ERROR: timerfd_create1\n");
        struct itimerspec itval;
        // set period time
        itval.it_interval.tv_sec = 1;
        itval.it_interval.tv_nsec = 10;
        // set start time
        itval.it_value.tv_sec = 0;
        itval.it_value.tv_nsec = 250000000;
        // start the timer
        if(timerfd_settime(timer_fd1, 0, &itval, NULL) == -1) printf("ERROR: timeset1\
        // initialize current thread as Real Time       struct sched_param param;

        // wait to timer expire
        uint64_t num_periods = 0;
        ssize_t rr;
        int i;
        for (i =0; i < 10; i++){
                // scheduling
```

```
                sched_setscheduler(0, SCHED_FIFO, &param);
                line += 1;
                if (line >=0 ){
                        printf("thread1 read line %d\n", line);
                        fgets(buf, 100, fptr1);
                }
                rr = read(timer_fd1, &num_periods, sizeof(num_periods));
                //if (rr == -1) printf("ERROR: read1\n");
                if(num_periods > 1) {
                puts("MISSED WINDOW");
         exit(1);
            }
        }
        fclose(fptr1);
        pthread_exit(NULL);
}

void* r2() {
        char fn2[15] = "second.txt";
        fptr2 = fopen(fn2, "r");
        int timer_fd2;
        timer_fd2 = timerfd_create(CLOCK_MONOTONIC, 0);
        if(timer_fd2 == -1) printf("ERROR: timerfd_create2\n");
        struct itimerspec itval;
        // set period time
        itval.it_interval.tv_sec = 1;
        itval.it_interval.tv_nsec = 10;
        // set start time
        itval.it_value.tv_sec = 0;
        itval.it_value.tv_nsec = 750000000;
        // start the timer
        if(timerfd_settime(timer_fd2, 0, &itval, NULL) == -1) printf("ERROR: timeset2\
        // initialize current thread as Real Time
        struct sched_param param;
        // set priority
        param.sched_priority = MY_PRIORITY;
        // wait to timer expire
        uint64_t num_periods = 0;
        ssize_t rr;
        int i;
        for (i =0; i < 10; i++){
                // scheduling
                sched_setscheduler(0, SCHED_FIFO, &param);
                line += 1;
                if (line > 0){
```

```
                        printf("thread2 read line %d\n", line);
                        fgets(buf, 100, fptr2);
                }
                rr = read(timer_fd2, &num_periods, sizeof(num_periods));
                //if (rr == -1) printf("ERROR: read2\n");
                if(num_periods > 1) {
                puts("MISSED WINDOW");
           exit(1);
            }
        }
        fclose(fptr2);
        pthread_exit(NULL);
}

void* w() {
        int timer_fd3;
        timer_fd3 = timerfd_create(CLOCK_MONOTONIC, 0);
        if(timer_fd3 == -1) printf("ERROR: timerfd_create3\n");
        struct itimerspec itval;
        // set period time
        itval.it_interval.tv_sec = 0;
        itval.it_interval.tv_nsec = 500000000;
        // set start time
        itval.it_value.tv_sec = 0;
        itval.it_value.tv_nsec = 500000000;
        // start the timer
        if(timerfd_settime(timer_fd3, 0, &itval, NULL) == -1) printf("ERROR: timeset3\
        // initialize current thread as Real Time
        struct sched_param param;
        // set priority
        param.sched_priority = MY_PRIORITY;
        // wait to timer expire
        uint64_t num_periods = 0;
        ssize_t rr;
        int i;
        for (i =0; i < 18; i++){
                // scheduling
                sched_setscheduler(0, SCHED_FIFO, &param);
                if (line >= 0) {
                        printf("thread3 write line %d\n", line);
                        strcpy(c[line],buf);
                }
                rr = read(timer_fd3, &num_periods, sizeof(num_periods));
                //if (rr == -1) printf("ERROR: read3\n");
                if(num_periods > 1) {
```

```
                    puts("MISSED WINDOW");
                exit(1);
                  }
        }
        pthread_exit(NULL);
}

int main(){
        // set start time
        gettimeofday(&start, NULL);
        int rc1;
        int rc2;
        int rc3;
        if (rc1 = pthread_create(&thread_r1, NULL, r1, NULL)) printf("ERROR: thread_1\
        if (rc2 = pthread_create(&thread_r2, NULL, r2, NULL)) printf("ERROR: thread_2\
        if (rc3 = pthread_create(&thread_w, NULL, w, NULL)) printf("ERROR: thread_3\n

        // check threads termiated
        if(pthread_join(thread_r1, NULL)) printf("ERROR: joint1\n");
        if(pthread_join(thread_r2, NULL)) printf("ERROR: joint2\n");
        if(pthread_join(thread_w, NULL)) printf("ERROR: joint3\n");
        printf("reading...\n");
        // printing
        int i;
        for (i=0; i <= line; i++) {
                printf("%s", c[i]);
        }
        // set end time
        gettimeofday(&end, NULL);
        float time = (end.tv_sec - start.tv_sec) + ((float)end.tv_usec - (float)start
        printf("time used: %lfs\n", time);
        return 0;
}
```