به نام خدا

دانشکده مهندسی کامپیوتر

آزمایشگاه سیستم های عامل

آزمایش ششم : برنامه نویسی چند نخی

اعضای گروه :

محمد امین فرح بخش - (40131029)

حسین تاتار – (40133014)

اردیبهشت 1404

سوال 1) ضرب ماتریسی با استفاده از Multithreading و pthread

برنامه ای بنویسید که ضرب دو ماتریس را با استفاده از pthread ها انجام دهد. هر نخ باید یک بخش از محاسبات را انجام دهد .

ابعاد ماتریس10000*10000

برای تعداد نخ های2 و4 و 8 زمان اجرا را بررسی و گزارش کنید.

توجه : به دلیل محدودیت حافظه و زمان اجرای بسیار بالا ابعاد ابعاد ماتریس را 1000*1000 در نظر گرفته ایم .

برای این سوال ابتدا کتابخانه ها و مقادیر مورد نیاز را تعریف میکنیم و سپس سه ماتریس خود را به صورت گلوبال تعریف میکنیم تا در دسترس هر نخ باشد. در این برنامه ما به کمک تقسیم محاسبات میان تعدادی نخ مقدارA * B را محاسبه کرده و نتیجه را درC ذخیره میکنیم .(در بخش دوم این سوال هنگام تحلیل زمان اجرا کافیست مقدار THREAD_NUM را تغییر داده و کد را کامپایل و اجرا نماییم.)

matrix_multiply.c

home > amin > matrix_multiply.c > ...

```c
#define _POSIX_C_SOURCE 199309L
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>

#define MATRIX_SIZE 1000
#define THREAD_NUM 2

int A[MATRIX_SIZE][MATRIX_SIZE];
int B[MATRIX_SIZE][MATRIX_SIZE];
int C[MATRIX_SIZE][MATRIX_SIZE];

void initialize_matrices()
{
    srand(time(NULL));
    for (int i = 0; i < MATRIX_SIZE; i++)
    {
        for (int j = 0; j < MATRIX_SIZE; j++)
        {
            A[i][j] = rand() % 10;
            B[i][j] = rand() % 10;
        }
    }
}
```

مطابق تصویر بالا تابع مقداردهی اولیه ماتریس ها راتعریف میکنیم که پیش از هرچیز آن را در تابع اصلی فراخوانی میکنیم .

اکنون تابع اجرا شده برای هر نخ را تعریف میکنیم ؛ در این تابع تقسیم محاسبات را بر اساس شناسه نخ انجام میدهیم که آرگومان ورودی این تابع است . این تقسیم وظایف بر روی سطر های ماتریس A انجام میشود به این صورت که هر نخ تعدادی از سطر های Aرا در ماتریس Bضرب میکند تا نتایج درایه های مربوط به آن درC بدست آید به عنوان مثال برای THREAD_NUM = 2 :

- Tid = 0 :

$$start = \frac{(0 * 1000)}{2} = 0\,, \quad end = \frac{(0+1) * 1000}{2} = 500$$

- Tid = 1 :

$$start = \frac{1 * 1000}{2} = 500\,, \quad end = \frac{(1+1) * 1000}{2} = 1000$$

```c
void *multiply(void *arg)
{
    int tid = *(int *)arg;
    int start = (tid * MATRIX_SIZE) / THREAD_NUM;
    int end = ((tid + 1) * MATRIX_SIZE) / THREAD_NUM;

    for (int i = start; i < end; i++)
    {
        for (int j = 0; j < MATRIX_SIZE; j++)
        {
            for (int k = 0; k < MATRIX_SIZE; k++)
            {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }

    return NULL;
}
```

در نهایت در تابع اصلی متغیر های مورد نیاز را تعریف میکنیم (threadsنشان دهنده هر نخthread_ids نشان دهنده شناسه هر نخ و به کمک استرایک timespec زمان اجرایی را محاسبه کرده و در متغیر elapsed_time ذخیره میکنیم)؛ تابع مقداردهی ماتریس ها را صدا زده و زمان شروع را ثبت می کنیم

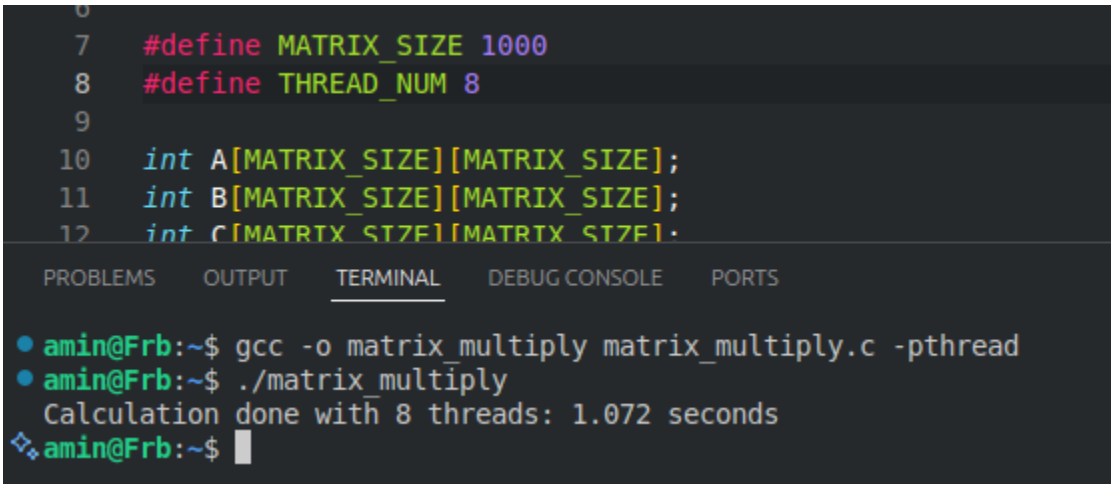
سپس هر نخ را در یک حلقه از شناسه 0تا thread_num میسازیم و در حلقه ای دیگر تمامی آنها راjoinمیکنیم؛ در نهایت زمان اتمام محاسبات را ثبت کرده و زمان elapsed_timeرا محاسبه کرده و در خروجی به همراه تعداد نخ ها چاپ میکنیم .

```c
int main()
{
    pthread_t threads[THREAD_NUM];
    int thread_ids[THREAD_NUM];
    struct timespec start_time, end_time;
    double elapsed_time;

    initialize_matrices();

    clock_gettime(CLOCK_MONOTONIC, &start_time);

    for (int i = 0; i < THREAD_NUM; i++)
    {
        thread_ids[i] = i;
        pthread_create(&threads[i], NULL, multiply, &thread_ids[i]);
    }

    for (int i = 0; i < THREAD_NUM; i++)
    {
        pthread_join(threads[i], NULL);
    }

    clock_gettime(CLOCK_MONOTONIC, &end_time);

    elapsed_time = (end_time.tv_sec - start_time.tv_sec) +
                   (end_time.tv_nsec - start_time.tv_nsec) / 1e9;

    printf("Calculation done with %d threads: %.3f seconds\n", THREAD_NUM, elapsed_time);
    return 0;
}
```

حال این برنامه را با سه مقدار THREAD_NUM = 2 , 4 , 8 کامپایل و اجرا میکنیم :

```c
7   #define MATRIX_SIZE 1000
8   #define THREAD_NUM 2
9
```

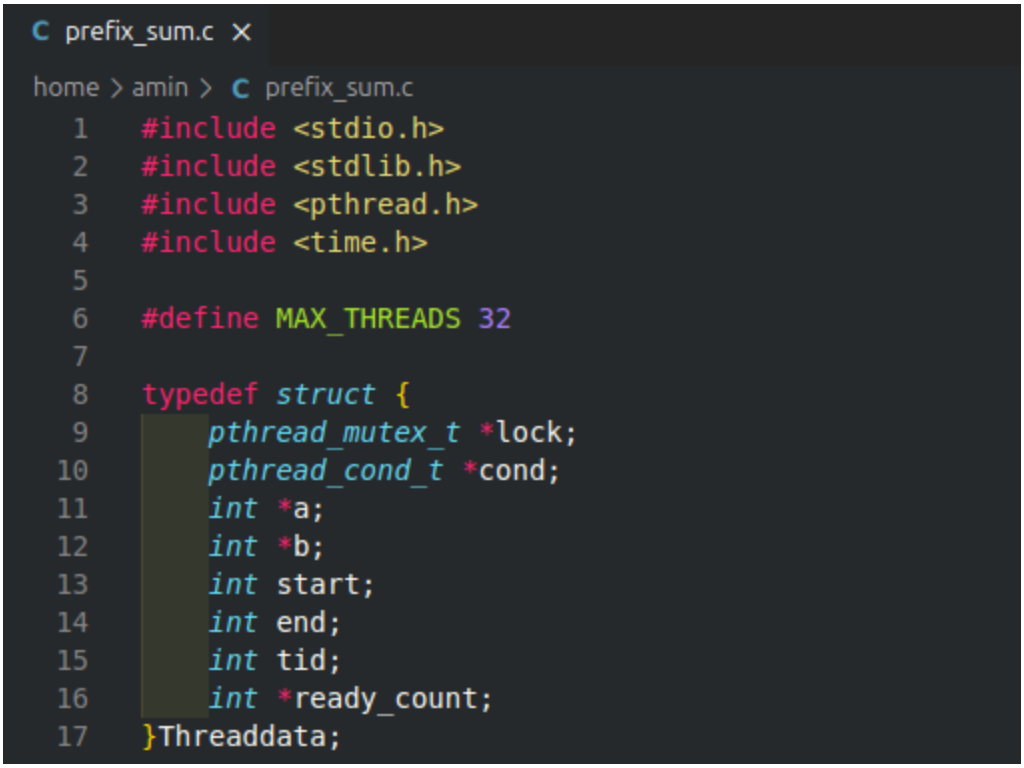
```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  PORTS

amin@Frb:~$ gcc -o matrix_multiply matrix_multiply.c -pthread
amin@Frb:~$ ./matrix_multiply
Calculation done with 2 threads: 2.321 seconds
amin@Frb:~$
```



مشاهده میشود که با افزایش تعداد نخ ها زمان محاسبات کاهش می یابد .

سوال 2) برنامه ای بنویسید که عملیات محاسباتی روی یک آرایه یا مجموعه داده را با استفاده ازpthread موازی سازی کند .

برای این سوال، ابتدا کتابخانه و مقادیر مورد نیاز را تعریف کرده و سپس استراکت مربوط به داده‌های هر نخ را مشخص می‌کنیم؛ در این استراکت به دو متغیر cond و lockبرای همگام‌سازی، دو آرایه a و bمشترک میان همه نخ‌ها که نشان‌دهنده ورودی و خروجی هستند، دو اندیس startو end که محدوده وظایف هر نخ را مشخص می‌کند و دو متغیر tid و ready_countکه به ترتیب شناسه نخ و تعداد نخ‌هایی که محاسبات خود را تمام کرده‌اند را در خود نگه می‌دارند که بوسیله‌ی آنها شرط انتظار هر نخ را با pthread_cond_wait مشخص می‌کنیم.



در تابع اجرایی هر نخ، آرگومان ورودی که همان استراکت داده است را دریافت کرده، و سپس در یک حلقه for اقدام به محاسبه prefix_sum می‌کنیم. ابتدا هر تکرار حلقه قفل را به دست گرفته و بررسی می‌کنیم آیا نخ‌های قبلی کار محاسباتی خود را به پایان رسانده‌اند یا خیر (اگر نوبت نخ فعلی برای انجام محاسبات باشد شناسه نخ با تعداد نخ‌های تکمیل شده برابر خواهد شد)؛ اگر همچنان نخی از نخ‌های پیشین کار خود را به اتمام نرسانده باشد به کمک pthread_cond_waitمنتظر می‌مانیم و قفل را آزاد می‌کنیم؛ در غیر این صورت از این حلقه عبور کرده و به محاسبات می‌پردازیم. پس از اتمام محاسبات، مقدار ready_countرا بروز رسانی کرده و بوسیله‌ی pthread_cond_broadcastبه نخ‌های در حال انتظار سیگنال بیدار شدن را می‌دهیم تا شرط نوبت خود را دوباره بررسی کنند.

```c
void *prefix_sum_partial(void *arg) {
    Threaddata *data = (Threaddata *)arg;

    for(int i = data->start; i < data->end; i++) {
        pthread_mutex_lock(data->lock);
        while(data->tid != *data->ready_count) {
            pthread_cond_wait(data->cond, data->lock);
        }

        data->b[i] = (i == 0) ? data->a[i] : data->b[i-1] + data->a[i];

        pthread_mutex_unlock(data->lock);

        pthread_mutex_lock(data->lock);
        (*data->ready_count)++;
        pthread_cond_broadcast(data->cond);
        pthread_mutex_unlock(data->lock);
    }
    return NULL;
}
```

سپس تابع اصلی را با آرگومان‌های ورودی مناسب تعریف کرده (که به ترتیب اندازه آرایه و تعداد نخ‌ها هستند) و وجود و همچنین صحت هر آرگومان را بررسی کرده و در صورت نیاز پیام خطای مناسب را چاپ می‌کنیم و پس از آن آرایه aرا مقداردهی اولیه می‌کنیم. (در ابتدا زمان شروع را در متغیر startClockنیز ذخیره می‌کنیم).

```c
int main(int argc, char **argv) {
    clock_t startClock = clock();

    if(argc != 3) {
        fprintf(stderr, "usage : %s <size> <thread_num>\n", argv[0]);
        return 1;
    }

    int size = atoi(argv[1]);
    int thread_num = atoi(argv[2]);

    if(size <= 0 || thread_num <= 0 || thread_num > MAX_THREADS) {
        fprintf(stderr, "Invalid input\n");
        return 1;
    }

    int a[size], b[size];
    for(int i = 0; i < size; i++) {
        a[i] = 1;
    }
```

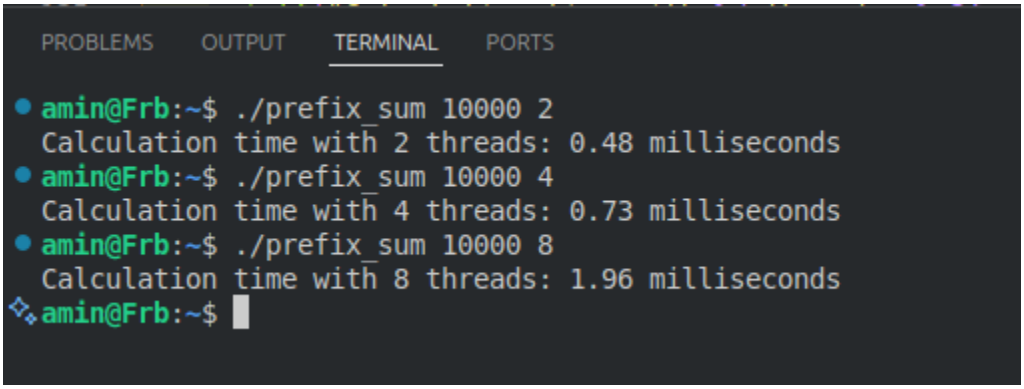
سپس متغیرهای threads وdata برای ذخیره کردن نخ‌ها به همراه داده‌های موردنیاز آن تعریف کرده و داده‌های مشترک میان نخ‌ها که باید در هر اشتراکت قرار گیرد را از جمله lock، cond و ready_count را تعریف می‌کنیم. متغیرهایی همچون elapsed_time و mid به ترتیب زمان اجرا و اندازه‌ی تکه از آرایه که هر نخ مسئول محاسبات آن است را در خود نگه می‌دارند.

```
61      pthread_t threads[thread_num];
62      Threaddata data[thread_num];
63
64      double elapsed_time;
65
66      pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
67      pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
68
69      int ready_count = 0;
70
71      int mid = size / thread_num;
72
```

در نهایت کافی‌ست که هر نخ را با داده‌های مناسب در اشتراکت مربوطه ساخته و تابع prefix_sum_partial را اجرا کنید. پس از آن منتظر join هر نخ مانده و زمان صرف شده محاسبات را بر حسب میلی‌ثانیه را محاسبه کرده و آن را چاپ می‌کنیم.

```
73      for (int i = 0; i < thread_num; i++) {
74          data[i].lock = &lock;
75          data[i].cond = &cond;
76          data[i].a = a;
77          data[i].b = b;
78          data[i].start = i * mid;
79          data[i].end = (i == thread_num - 1) ? size : (data[i].start + mid);
80          data[i].tid = i;
81          data[i].ready_count = &ready_count;
82
83          pthread_create(&threads[i], NULL, prefix_sum_partial, &data[i]);
84      }
85
86      for (int i = 0; i < thread_num; i++) {
87          pthread_join(threads[i], NULL);
88      }
89
90      elapsed_time = ((double) clock() - startClock) * 1000 / CLOCKS_PER_SEC;
91      printf("Calculation time with %d threads: %.1f milli seconds\n", thread_num, elapsed_time);
92
93      return 0;
94  }
```

اکنون کافی است که این برنامه را برای تعداد نخ های مختلف 2 و 4 و 8 اجرا کنیم و خروجی را مشاهده کنیم :



مشاهده می‌شود که این مسئله با افزایش تعداد نخ‌ها، زمان طولانی‌تری طول می‌کشد زیرا با شکسته شدن محاسبات به نخ‌های بیشتر، وظیفه‌ی هر نخ کمتر شده و بنابراین زمان انتظار هر نخ بیشتر می‌شود. علاوه بر این، هزینه ساخت و join نخ‌ها نیز بالا می‌رود.