به نام خدا

گزارش آزمایش پنجم آزمایشگاه سیستم های عامل

زهرا رحیمی

شماره دانشجوئی: ۹۸۳۱۰۲۶

استاد آزمایشگاه: سرکار خانم حسینی

پاییز ۱۴۰۰

بخش اول:

طبق دستور کار پیش می رویم و آرایه hist که نتایج آزمایش برای رسم نمودار توزیع را نگه می دارد می سازیم

client.c | *server.c | sample.c

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

int main() {

    clock_t begin = clock();
    int hist[25];
    int counter = 0;

    for (int j = 0; j < 25; j++) {
        hist[j] = 0;
    }

    srand(time(0));
    int sample_count = 50000;

    for (int i = 0; i < sample_count; i++){
        counter = 0;
        for (int j = 0; j < 12; j++) {
            int r = rand() % 100;
            while (r < 0) {
                r = rand() % 100;
            }
            if (r >= 49) {
                counter++;
            } else {
                counter--;
            }
        }
        hist[counter + 12]++;

    }
    clock_t end = clock();
    float time_spent = (float) (end-begin)/ CLOCKS_PER_SEC;
    printf("%f\n", time_spent);

    return 0;
}
```

Saving file "/home/zahra/Desktop/OS_Lab/az5/sample.c"... C Tab Width: 8 Ln 8, Col 16 INS

تعداد نمونه	۵۰۰۰	۵۰۰۰۰	۵۰۰۰۰۰
زمان اجرا	۰.۰۰۰۷۵۵	۰.۰۰۸۳۱۰	۰.۰۸۴۳۴۹

همانطور که می بینم با افزایش تعداد نمونه ها زمان اجرا هم افزایش می یابد.

بخش دوم:

با پخش کردن کار ها و انداخت نصف کارها بر دوش هر فرآیند به آزمایش سرعت می بخشیم:

```c
#include <time.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <semaphore.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <unistd.h>
#include <wait.h>
#include <fcntl.h>

#define SHM_KEY 102030
#define SHM_SIZE sizeof(int) * 25

sem_t mutex;

int create_segment() {
    int segment_id = shmget (IPC_PRIVATE, 1000,S_IRUSR | S_IWUSR);
    if (segment_id == -1) {
        perror("shmget");
    }
    return segment_id;
}

int* attach_segment(int shmid) {
    int *shm;

    if ((shm = shmat(shmid, NULL, 0)) == (int *) -1) {
        printf("Shared memory did not attached to address!");
    }

    return shm;
}
void detach_segment(int *shm) {
    shmdt(shm);
}
```

```c
void remove_segment(int shmid) {
    if (shmctl(shmid, IPC_RMID, NULL) == -1) {
        printf("Couldn't remove the shared memory!");
    }
}

void print_histogram(int *hist, int number_of_samples) {
    printf("Histogram for sample %d:\n", number_of_samples);
    for (int i = 0; i < 25; i++) {
        for (int j = 0; j < hist[i]; j++) {
            printf("*");
        }
        printf("\n");
    }
}

double calculate(int number_of_samples) {
    clock_t begin = clock();
    int shmid = create_segment();
    int * hist = attach_segment(shmid);
    srand(time(0));
    int rand_num, counter;

    pid_t pid = fork();
    if (pid == 0){
        for (int i = 0; i < number_of_samples / 2; i++){
            counter = 0;
            for (int j = 0; j < 12; j++) {
                rand_num = rand() % 100;
                while (rand_num < 0) {
                    rand_num = rand() % 100;
                }
                if (rand_num >= 49)
                    counter +=1;
                else
                    counter -=1;
            }
            hist[counter +12] += 1;
        }
```

```
        exit(0);
    }
    else {

        for (int i = 0; i < number_of_samples / 2; i++) {
            counter = 0;
            for (int j = 0; j < 12; j++) {
                rand_num = rand() % 100;
                while (rand_num < 0) {
                    rand_num = rand() % 100;
                }
                if (rand_num >= 49)
                    counter += 1;
                else
                    counter -= 1;
            }
            hist[counter + 12] += 1;
        }
       print_histogram(hist, number_of_samples);
    }

    detach_segment(hist);
    remove_segment(shmid);

    clock_t end = clock();



    double time_spend = (double ) (end-begin)/ CLOCKS_PER_SEC;

    return time_spend;
}

int main() {
    double time_spend = calculate(5000);
    printf("%f\n", time_spend);
    return 0;
}
```

خروجی به ازای نمونه ۵۰ تایی:

```
zahra@zahra-virtual-machine:~/Desktop/OS_Lab/az5$ ./code
Histogram for sample 50:



**

****

**********

*********

******

*********

****




0.000229
zahra@zahra-virtual-machine:~/Desktop/OS_Lab/az5$
```

با دوباره اجرا کردن آن زمان اجرای برنامه متفاوت می شود:

```
zahra@zahra-virtual-machine:~/Desktop/OS_Lab/az5$ ./code
Histogram for sample 50:



***

**********

*******

********

********

****




0.000129
zahra@zahra-virtual-machine:~/Desktop/OS_Lab/az5$
```

از آنجا که به دلیل دعوا مقدار دهی یکی از اعضای آرایه hist است که آیا فرآیند پدر زودتر برسد که به ازای counter خودش به آن مقدار دهد یا فرزند، نتیجه می گیریم برنامه درگیر شرایط مسابقه (race condition) شده است و راه حل این موضوع، استفاده از ایجاد lock می باشد به این صورت که خطوطی که می خواهیم در آن فرآیند قفل شود و سوئیچ نکند را قفل کنیم. در این روش با تابع sem_wait(sem_t *mutex) برای لاک کردن یا منتظر ماندن، با تابع sem_post(sem_t *mutex) برای رهایی یا اعلام قفل، sem_init(sem_t *mutex, int pshared, int value) برای مقدار دهی semaphore استفاده می شود که pshared برای فرآیند ها غیر صفر و برای ریسمان ها(thread) صفر قرارداد شده است.

حال با یک بار اجرای این کد به این نتایج می رسیم. (ممکن است در هر بار نتایج متفاوت باشد!)

تعداد نمونه	۵۰۰۰	۵۰۰۰۰	۵۰۰۰۰۰
زمان اجرا	۰.۰۰۰۴۹۳	۰.۰۰۴۱۵۹	۰.۰۵۳۴۵۵

نتیجه مقایسه افزایش سرعت در دو روش سریال و روش استفاده از دو فرآیند در جدول زیر آمده است:

تعداد نمونه	۵۰۰۰	۵۰۰۰۰	۵۰۰۰۰۰
زمان اجرا	۰.۰۰۰۲۶۲	۰.۰۰۴۱۵۱	۰.۰۳۰۸۹۴

قطعه کد عوض شده برای راه حل شرایط مسابقه(با استفاده از semaphore):

```
double calculate(int number_of_samples) {
    clock_t begin = clock();
    int shmid = create_segment();
    int * hist = attach_segment(shmid);
    srand( seed: time( _timer: 0));
    int rand_num, counter;
    if(sem_init(&mutex,  pshared: 1,  value: 1)<0){
        perror("error in semaphore initialization ");
        exit( status: 0);
    }

    pid_t pid = fork();
    if (pid == 0){
        sem_post(&mutex);
        for (int i = 0; i < number_of_samples / 2; i++){
            counter = 0;
            for (int j = 0; j < 12; j++) {
                rand_num = rand() % 100;
                while (rand_num < 0) {
                    rand_num = rand() % 100;
                }
                if (rand_num >= 49)
                    counter +=1;
                else
                    counter -=1;
            }
            hist[counter+12] += 1;
        }
        exit( status: 0);
    }
```

```
    else {
        sem_wait(&mutex);
        for (int i = 0; i < number_of_samples / 2; i++) {
            counter = 0;
            for (int j = 0; j < 12; j++) {
                rand_num = rand() % 100;
                while (rand_num < 0) {
                    rand_num = rand() % 100;
                }
                if (rand_num >= 49)
                    counter += 1;
                else
                    counter -= 1;
            }
            hist[counter + 12] += 1;
        }
        print_histogram(hist, number_of_samples);
        sleep( seconds: 5);
    }
    sem_post(&mutex);
    wait( status: (int *)0);
    sem_destroy(&mutex);
    detach_segment(hist);
    clock_t end = clock();

    double time_spend = (double ) (end-begin)/ CLOCKS_PER_SEC;

    return time_spend;
}
```