گزارش‌کار آزمایش 5

استاد: دکتر حسینی
دانشجو: فرشید نوشی- 9831068

بخش ۱:

در این بخش قصد داریم تا مسئله مطرح شده در دستور کار را به صورت سریال بنویسیم.

ابتدا تابع printHistogram را تعریف می‌کنیم، این تابع آرایه hist و تعداد نمونه را می‌گیرد، اگر تعداد نمونه کمتر از ۴۰۰ بود، به صورت ستاره مقدار هر خانه hist را نشان می‌دهد و اگر بیشتر بود مقدار مورد نظر آن خانه از آرایه hist را چاپ میکند. i را هم منهای ۱۲ کردیم تا طبق دستور کار بین منفی ۱۲ تا مثبت ۱۲ بشود.

```
void print(int *hist, int sample) {
    for (int i = 0; i < 25; i++)
    {
        printf("%d\t", i - 12);
        if(sample <= 400) {
            for (int j = 0; j < hist[i]; j++){
                printf("*");
            }
            printf("\n");
        }
        else
            printf(" %d\n", hist[i]);
    }
}
```

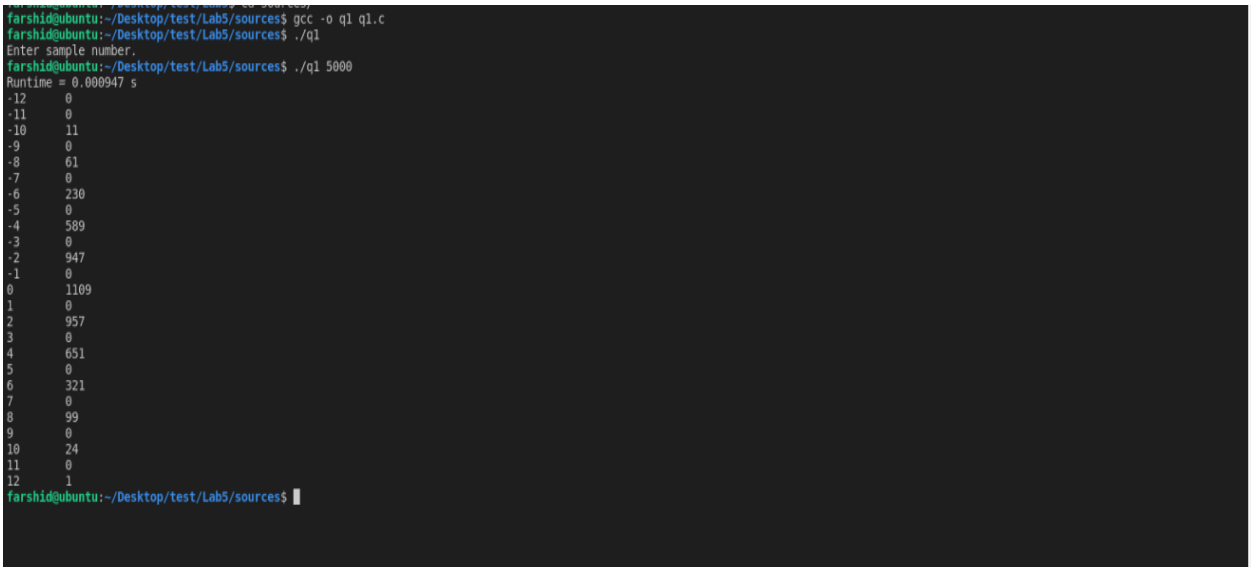
در ابتدای main، چک می‌کنیم که اگر کاربر ورودی نداده بود، پیغام مناسب برای کاربر چاپ شود تا تعداد نمونه را وارد کند. در ادامه نیز متغیر های مورد نیاز و آرایه hist را تعریف می‌کنیم. سپس تعداد نمونه را از آرگومان ورودی می‌خوانیم.

```
int main(int argc, char* argv[])
{
    if (argc != 2)
    {
        printf("Enter sample number.\n");
        return 0;
    }
    int sample, counter, random, hist[25] = {0};
    sscanf(argv[1], "%d", &sample);
```

در بخش زیر، ابتدا کلاک آغاز را می‌گیریم، سپس به کمک ۲ حلقه ی تو در تو، عدد رندوم بین صفر تا ۱۰۰ را تولید کرده و با منطق گفته شده در دستور کار، آرایه hist را به روز می‌کنیم و مقدار می‌دهیم. در آخر کلاک اتمام را گرفته و با توجه به کلاک آغاز، مدت زمان اجرای برنامه را چاپ می‌کنیم و با فراخوانی تابع printHistogram، آرایه hist را چاپ می‌کنیم.

```
    clock_t begin = clock();
    srand(time(0));
    for (int i = 0; i < sample; i++)
    {
        counter = 12;
        for (int j = 0; j < 12; j++)
        {
            random = rand() % (100 + 1 - 0) + 0;
            if (random >= 49 && counter <= 23)
                counter++;
            else if (random < 49 && counter >= 1)
                counter--;
        }
        hist[counter]++;
    }
    clock_t end = clock();
    printf("Runtime = %f s\n", ((double)end - begin) / CLOCKS_PER_SEC);
    print(hist, sample);
    return 0;
}
```

خروجی برنامه به ازای نمونه‌ی ۵۰۰۰ عضوی:



Index های فرد به خاطر این صفر هستند، چون که با 12 بار عدد تصادفی ایجاد کردن، با هر دو بار انجام اینکار یا دوبار counter++ داریم یا دوبار counter-- یا یکبار از هر کدام که یعنی یا ۲ بار اضافه می‌شود یا دو بار کم می‌شود یا 0 ، پس روی index های فرد هیچ وقت نمی‌افتد.

خروجی برنامه به ازای نمونه‌ی ۵۰۰۰۰ عضوی:

خروجی برنامه به ازای نمونه‌ی ۵۰۰۰۰۰ عضوی:

زمان اجرای برنامه در حالت سریال به ازای تعداد نمونه‌های مختلف:

تعداد نمونه	۵۰۰۰	۵۰۰۰۰	۵۰۰۰۰۰
زمان اجرا (ms)	**0.9**	**11**	**117**

بخش ۲:

در این بخش قصد داریم تا به کمک IPC و ()fork تعدادی پردازه‌ی فرزند ایجاد کرده و کارهای گفته شده در دستور کار را بین این پردازه‌ها تقسیم کنیم.

ابتدا مانند قسمت قبل اگر ورودی نداشتیم به کاربر گفته می‌شود تا تعداد نمونه را وارد کند. سپس متغیرهای مورد نیاز را تعریف کرده و با کمک shmget یک آدرس در حافظه از مکان مشخص و به حجم ۳۲ بایت با قابلیت خواندن و نوشتن تعریف می‌کنیم و با کمک shmat، حافظه را attach می‌کنیم. سپس تعداد نمونه را از آرگومان ورودی می‌خوانیم.

```
int main(int argc, char* argv[])
{
    if (argc != 2)
    {
        printf("Enter sample number.\n");
        return 0;
    }
    int sample, counter, random, hist[25] = {0}, pid, status = 0;
    int segment_id = shmget(IPC_PRIVATE, 32, S_IRUSR | S_IWUSR);
    int *shared_memory = (int *)shmat(segment_id, NULL, 0);
    sscanf(argv[1], "%d", &sample);
```

در ادامه کلاک آغاز را می‌گیریم. سپس با کمک سه fork() ، پنج پردازه می‌سازیم و در ادامه مانند بخش قبلی منطق گزارش‌کار را پیاده می‌کنیم، با این تفاوت که sample را تقسیم بر تعداد پردازه یعنی ۵ کرده و روی shared_memory پیمایش و بروز رسانی را انجام می‌دهیم. بر روی آرایه hist. در آخر هم چک می‌شود که اگر کار پردازه ها تمام شده بود، زمان عملیات برای کاربر چاپ شود و تابع printHistogram فراخوانی شده و تعداد نمونه و shared_memory به آن پاس داده شود تا مقادیر مناسب را چاپ کند. در آخر هم حافظه‌ی اختصاص یافته را از پردازه‌ی پدر و فرزند detach می‌کنیم.

```
    clock_t begin = clock();
    //5 processes
    pid = fork();
    if(pid == 0) {
        fork();
        fork();
    }
    srand(time(0));
    for (int i = 0; i < sample / 5; i++)
    {
        counter = 12;
        for (int j = 0; j < 12; j++)
        {
```

```
            random = rand() % (100 + 1 - 0);
            if (random >= 49 && counter <= 23)
                counter++;
            else if (random < 49 && counter >= 1)
                counter--;
        }
        shared_memory[counter]++;
    }

    if (pid != 0){
        while (wait(&status) > 0);
        clock_t end = clock();
        printf("Runtime = %f s\n", ((double)end - begin) / CLOCKS_PER_SEC);
        print(shared_memory, sample);
        shmdt(shared_memory);
    }
    else{
        shmdt(shared_memory);
    }
    return 0;
}
```

خروجی برنامه به ازای نمونه‌ی ۵۰۰۰ عضوی:

خروجی برنامه به ازای نمونه‌ی ۵۰۰۰۰ عضوی:

خروجی برنامه به ازای نمونه‌ی ۵۰۰۰۰۰ عضوی:

زمان اجرای برنامه در حالت موازی به ازای تعداد نمونه‌های مختلف:

تعداد نمونه	۵۰۰۰	۵۰۰۰۰	۵۰۰۰۰۰
زمان اجرا (ms)	**0.3**	**2**	**33**

بخش ۳:

۳. ایا این برنامه درگیر شرایط مسابقه می‌شود؟چگونه؟اگر جوابتان مثبت بود راه حلی برای آن بیابید.

بله. این برنامه هنگام دسترسی پردازه‌ها به آرایه‌ی shared_memory درگیر شرایط مسابقه می‌شوند. زیرا می‌توانند با توجه به زمانبندی پردازنده توسط سیستم عامل و اختصاص پردازنده به یک ترتیبی، از منبع اختصاص یافته‌ی مشترک استفاده کنند.

برای جلوگیری از این اتفاق می‌توان با استفاده از spin lock ها و یا سمافور، انحصار متقابل در حین دسترسی به آرایه‌ی shared_memory ایجاد کرد. به عنوان مثال می‌توان به ازای هر خانه‌ی آرایه‌ی shared_memory یک سمافور با مقدار اولیه‌ی یک ایجاد کرد و در هنگام ایجاد تغییر در آن، به شکل زیر کد را بازنویسی کرد:

```
sem_wait(&sem[counter]);
```

```
shared_memory[counter]++;

sem_post(&sem[counter]);
```

بخش ۴:

در این بخش شاهد تفاوت سرعت کار در حالت موازی و سری هستیم.

میزان افزایش سرعت در حالت موازی:

تعداد نمونه	۵۰۰۰	۵۰۰۰۰	۵۰۰۰۰۰
افزایش سرعت(ms)	2.439	10.371	93.063
درصد افزایش	71.14	57.66	66.44

نمودار هیستوگرام برای نمونه‌ی ۱۰۰ تایی توسط دو برنامه نمایانگر توزیع نرمال حول خانه صفر است که در حالت موازی این نمودار پهن‌تر شده و ضریب همبستگی‌اش افزایش پیدا کرده است.


12 176
farshid@ubuntu:~/Desktop/test/Lab5/sources$ ./q1 100
Runtime = 0.000027 s
-12
-11
-10
-9
-8 **
-7
-6 ****
-5
-4 **********
-3
-2 ******************
-1
0 ************************
1
2 ******************
3
4 **************
5
6 *******
7
8 *
9
10
11
12


farshid@ubuntu:~/Desktop/test/Lab5/sources$ gcc -o q2 q2.c
farshid@ubuntu:~/Desktop/test/Lab5/sources$ ./q2 100
Runtime = 0.000123 s
-12
-11
-10
-9
-8
-7
-6
-5
-4
-3
-2 *************************
-1
0 ******************
1
2 ******************************
3
4 **********
5
6 **************
7
8
9
10
11
12
farshid@ubuntu:~/Desktop/test/Lab5/sources$