آزمایشگاه سیستم عامل

گزارش آزمایش شماره 5

نام استاد: مهندس حمیدرضا کیخا

تاریخ: 1400 / 10 / 16

نام و نام‌خانوادگی: نگار کرمی

شماره دانشجویی: 9722039

نام و نام‌خانوادگی: مجید نامی

شماره دانشجویی: 9728095

دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

مرحله اول: ابتدا کد برنامه ای که در تعریف مسئله شرح داده شد را در حالت سریال بنویسید و زمان اجرا شدن برنامه خود را در جدول گزارش دهید .

ابتدا کد مورد نظر را به زبان C لازم است تا بنویسیم. کد موردنظر به صورت زیر میباشد:

serial.c

D: > 00_1 > Operating Systems Lab > 5 > serial.c

```c
#include <stdio.h>
#include <sys/time.h>
#include "time.h"
#include <stdlib.h>
#include <string.h>


int main(int argc, char const *argv[])
{
    int iterations = 5000;
    int hist[25] = {0};
    int counter = 0;
    int random;
    int sum = 0;
    struct timeval stop, start;
    gettimeofday(&start, NULL);

    if (argc >= 2)
        iterations = atoi(argv[1]);

    srand(time(NULL));

    for (int i = 0; i < iterations; i++)
    {
        counter = 0;
        for (int j = 0; j < 12; j++)
        {
            random = rand() % 99 + 1;
            if (random >= 49)
                counter++;
            else
                counter--;
        }
        hist[counter + 12]++;
    }
    gettimeofday(&stop, NULL);
    printf("The runtime is %lu us\n\n", (stop.tv_sec - start.tv_sec) * 1000000 + stop.tv_usec - start.tv_usec);
```

```c

    int scale = iterations / 100;
    for (int i = 0; i < 25; i++)
    {
        printf("%3d: %7d - ", i - 12, hist[i]);
        sum += hist[i];
        int max = hist[i] / scale;
        for(int j = 0; j < max; j++)
            printf("*");
        printf("\n");
    }
    printf("The number of iterations is %d.\n", sum);

    return 0;
}

```

پس از include کردن کتابخانه‌های لازم ، به سراغ تعریف متغیرها میرویم.

یک متغیر iteration داریم که مقدار اولیه‌ی آن 5000 است ولی درصورتی که کاربر مقدار آرگومان ورودی را دهد درون آن میریزیم. یک آرایه‌ی 25 تایی hist درست میکنیم و در همه‌ی درایه‌های آن مقدار 0 قرار میدهیم. یک مقدار counter نیاز داریم طبق صورت مسئله تا زمانی که عدد رندوم ما بزرگ تر مساوی49 بود یکی اضافه شود و در غیر این صورت یکی کم شود.

همچنین یک متغیر sum داریم که مقدار ذخیره شده در هرخانه‌ی آرایه hist را در انتهال باهم جمع میکنیم که قاعدتا باید برابر با مقدار ورودی iteration باشد تا صحت را نشان دهد.(در حالت سریال این قاعده برقرار است).

یک استراکت timeval هم برای نگهداری زمان شروع و پایان لازم است تا بسازیم.

به کمک تابع atoi مقدار آرگومان ورودی که تعداد iteration های ما را مشخص میکند را به عدد تبدیل میکنیم و در متغیر iterations نگه میداریم.

سپس یک for تو در تو نیاز داریم. حلقه‌ی بیرونی باید به اندازه‌ی مقدار موجود در iterations که از به عنوان ورودی مقدار آن داده میشود و تعداد نمونه میباشد، iterate کند.

همچنین در صورت سوال گفته شده هر نمونه در 12 مرحله، حلقه‌ی داخلی باید عدد رندوم بین 0 تا 100 تولید کند پس یک حلقه‌ی 12تایی نیاز داریم. برای عدد رندوم نیز چون بین 0 تا 100 خواسته شده است، پس باید در نظر بگیریم که به صورت (random = rand() % 99 +1;) بنویسیم.

سپس اگر که عدد رندوم ما بزرگ تر مساوی49 بود یکی به کانتر اضافه شود و در غیر این صورت یکی کم شود. در نهایت میدانیم که عدد موجود در کانتر عددی از 12- تا 12+ میباشد. بنابراین اگر بخواهیم خانه‌ی متناظر با آن را در آرایه پیدا کنیم و یکی بهش اضافه کنیم باید به صورت (hist[counter + 12]++;) بنویسیم.

همچنین لازم به ذکر است، برای اینکه بتوانیم تایم عملیات خواسته شده را به دست آوریم یک بار قبل از شروع عملیات زمان فعلی سیستم را در یک متغیر start ذخیره کنیم. یک بار هم این کار را بعد از پایان حلقه‌ی تو در تو انجام میدهیم و تایم فعلی سیستم را در یک متغیر stop ذخیره میکنیم. برای این کار از تابع gettimemodify کمک گرفیتم.

در آخر هم مقدار تایم های start و stop را از هم کم میکنیم تا مقدار زمان عملیات به دست آید و چاپ میکنیم.

برای چاپ هیستوگرام هم میتوان از کد موجو در دستور کار کمک گرفت که حالا ما کمی آن را تغییر دادیم.

برای این کار از یک scale یک صدم استفاده کردیم چون تعداد ستاره ها در صورتی که scale ما یک به یک باشد خیلی زیاد میشود.

در آخر هم پس از محاسبه ی مقدار sum آن را چاپ میکنیم.

خروجی ها برای 3 مقدار نمونه‌ی 5000 و 50000 و 500000 به صورت زیر میباشد:

برای iterations=5000 :

```
majid@majid-virtual-machine:~/Desktop/os_lab/5$ gcc serial.c -o main; ./main 5000
The runtime is 807 us

-12:       1 -
-11:       0 -
-10:      14 -
 -9:       0 -
 -8:      62 - *
 -7:       0 -
 -6:     253 - *****
 -5:       0 -
 -4:     528 - **********
 -3:       0 -
 -2:     893 - *****************
 -1:       0 -
  0:    1093 - *********************
  1:       0 -
  2:     977 - ******************
  3:       0 -
  4:     746 - **************
  5:       0 -
  6:     313 - ******
  7:       0 -
  8:     100 - **
  9:       0 -
 10:      18 -
 11:       0 -
 12:       2 -
The number of iterations is 5000.
majid@majid-virtual-machine:~/Desktop/os_lab/5$
```

برای iterations=50000 :

```
majid@majid-virtual-machine:~/Desktop/os_lab/5$ gcc serial.c -o main; ./main 50000
The runtime is 8829 us

-12:       2 -
-11:       0 -
-10:     110 -
 -9:       0 -
 -8:     640 - *
 -7:       0 -
 -6:    2360 - ****
 -5:       0 -
 -4:    5178 - **********
 -3:       0 -
 -2:    9109 - ******************
 -1:       0 -
  0:   11214 - **********************
  1:       0 -
  2:   10130 - ********************
  3:       0 -
  4:    6781 - *************
  5:       0 -
  6:    3207 - ******
  7:       0 -
  8:    1051 - **
  9:       0 -
 10:     193 -
 11:       0 -
 12:      25 -
The number of iterations is 50000.
majid@majid-virtual-machine:~/Desktop/os_lab/5$
```

برای iterations=500000 :

```
majid@majid-virtual-machine:~/Desktop/os_lab/5$ gcc serial.c -o main; ./main 500000
The runtime is 82327 us

-12:       88 -
-11:        0 -
-10:     1049 -
 -9:        0 -
 -8:     6149 - *
 -7:        0 -
 -6:    22402 - ****
 -5:        0 -
 -4:    52883 - **********
 -3:        0 -
 -2:    90296 - ******************
 -1:        0 -
  0:   111878 - **********************
  1:        0 -
  2:   102553 - ********************
  3:        0 -
  4:    68283 - *************
  5:        0 -
  6:    32208 - ******
  7:        0 -
  8:     9999 - *
  9:        0 -
 10:     2048 -
 11:        0 -
 12:      164 -
The number of iterations is 500000.
majid@majid-virtual-machine:~/Desktop/os_lab/5$
```

حال باتوجه به خروجی ها به سراغ تکمیل جدول میرویم:

تعداد نمونه	5000	50000	500000
زمان اجرا	0.807ms	8.829ms	82.327ms

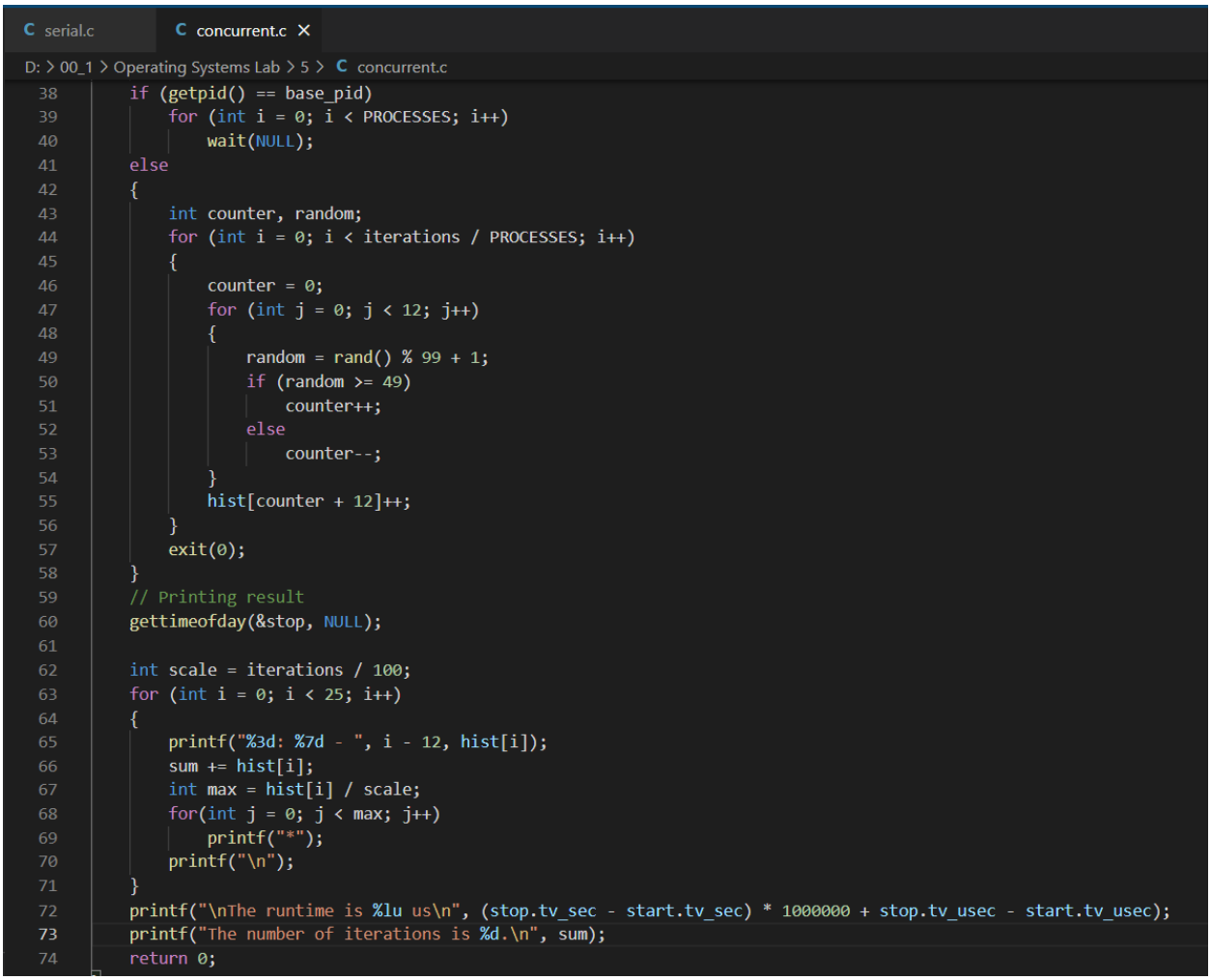
مرحله دوم: حال برنامه ای بنویسید که با استفاده از()fork و یا()exec تعدادی فرآیند فرزند ایجاد شود و کارها را پخش کنید.

ابتدا کد برنامه را در C میزنیم که به صورت زیر میباشد:

serial.c | concurrent.c

D: > 00_1 > Operating Systems Lab > 5 > concurrent.c

```c
#include <time.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#define PROCESSES 5


int main(int argc, char const *argv[])
{
    int id;
    int base_pid;
    int *hist;
    int sum = 0;
    int iterations = 5000;
    struct timeval stop, start;
    gettimeofday(&start, NULL);

    if (argc >= 2)
        iterations = atoi(argv[1]);

    srand(time(NULL));

    id = shmget(IPC_PRIVATE, sizeof(int) * 25, IPC_CREAT | 0666);
    hist = (int *) shmat(id, NULL, 0);

    base_pid = getpid();
    for (int i = 0; i < PROCESSES; i++)
        if (getpid() == base_pid)
            fork();
        else
            break;

```



در این کد نیز پس از مشخص کردن لایبری ها به سراغ تعریف متغیرها میرویم.

در اینجا نیز مانند کد قبل متغیرهای sum و iteration و استراکت timeval را داریم. همچنین یک متغیر اشاره گر hist هم تعریف میکنیم.

همچنین یک متغیر id برای ذخیره‌ی id حاصل از حافظه‌ی اشتراکی و درواقع خروجی shmget میباشد و همچنین یک متغیر base_pid برای ذخیره‌ی pid پراسس پدر میباشد.

در اینجا در خط 28 و 29 ما اومدیم ابتدا به کمک shmget یک حافظه‌ی اشتراکی را درست کردیم در حافظه به اندازه آرایه‌ی موردنظرمان که یک آرایه‌ی 25 تایی int میباشد، و بعد به کمک shmat این حافظه را به اشاره‌گر hist مپ میکنیم.

سپس به کمک getpid() مقدار pid پراسس پدر را در base_pid ذخیره میکنیم. سپس به اندازه متغیر PROCESS که بالاتر آن را define کردیم که 5 باشد ، از فورک استفاده میکنیم و پراسس child میسازیم.

برای اینکه خود بچه ها بچه‌ای نداشته باشند و فقط پدر 5 تا فرزند داشته باشد از شرط موجود در خط 33 استفاده کردیم.

سپس در پردازه‌ی پدر برای فرزندان به کمک تابع wait صبر میکنیم و آن‌ها را دریافت میکنیم.

در ضمن کاری که قبل تر داشتیم و به صورت سریال انجام میدادیم اکنون بین فرندان پخش میشود. همانطور که در for خط 44 دیده میشود هر فرزند به اندازه‌ی iterations/PROCESS تا iteration را انجام میدهد فقط.

بقیه قسمت های کد نکته‌ی خاصی ندارند و مانند قسمت قبل میباشند و در این حالت نیز زمان عملیات و مقدار sum و همچنین هیستوگرام را در خروجی نمایش میدهیم.

خروجی ها برای 3 مقدار نمونه‌ی 5000 و 50000 و 500000 به صورت زیر میباشد:

برای iterations=5000 :

```
majid@majid-virtual-machine:~/Desktop/os_lab/5$ gcc concurrent.c -o main; ./main 5000
-12:       5 -
-11:       0 -
-10:      20 -
 -9:       0 -
 -8:      45 -
 -7:       0 -
 -6:     194 - ***
 -5:       0 -
 -4:     497 - *********
 -3:       0 -
 -2:     857 - *****************
 -1:       0 -
  0:    1082 - **********************
  1:       0 -
  2:    1209 - ************************
  3:       0 -
  4:     650 - *************
  5:       0 -
  6:     245 - ****
  7:       0 -
  8:     140 - **
  9:       0 -
 10:      15 -
 11:       0 -
 12:       0 -

The runtime is 1630 us
The number of iterations is 4959.
majid@majid-virtual-machine:~/Desktop/os_lab/5$
```

برای iterations=50000 :

```
majid@majid-virtual-machine:~/Desktop/os_lab/5$ gcc concurrent.c -o main; ./main 50000
-12:      15 -
-11:       0 -
-10:      85 -
 -9:       0 -
 -8:     570 - *
 -7:       0 -
 -6:    2233 - ****
 -5:       0 -
 -4:    5342 - **********
 -3:       0 -
 -2:    8970 - *****************
 -1:       0 -
  0:   11209 - **********************
  1:       0 -
  2:    9580 - ******************
  3:       0 -
  4:    6564 - *************
  5:       0 -
  6:    3250 - ******
  7:       0 -
  8:    1029 - **
  9:       0 -
 10:     220 -
 11:       0 -
 12:      25 -

The runtime is 5492 us
The number of iterations is 49092.
majid@majid-virtual-machine:~/Desktop/os_lab/5$
```

برای iterations=500000 :

```
majid@majid-virtual-machine:~/Desktop/os_lab/5$ gcc concurrent.c -o main; ./main 500000
-12:     110 -
-11:       0 -
-10:    1109 -
 -9:       0 -
 -8:    6269 - *
 -7:       0 -
 -6:   22020 - ****
 -5:       0 -
 -4:   53158 - **********
 -3:       0 -
 -2:   88387 - *****************
 -1:       0 -
  0:  109721 - **********************
  1:       0 -
  2:  100274 - ********************
  3:       0 -
  4:   67806 - *************
  5:       0 -
  6:   31835 - ******
  7:       0 -
  8:   10201 - **
  9:       0 -
 10:    1970 -
 11:       0 -
 12:     215 -

The runtime is 56275 us
The number of iterations is 493075.
majid@majid-virtual-machine:~/Desktop/os_lab/5$
```

باتوجه به خروجی ها در تصاویر بالا میتوانیم جدول را تکمیل کنیم:

تعداد نمونه	5000	50000	500000
زمان اجرا	1.630ms	5.492ms	56.275ms

در این حالت همانطور که مشاهده میشود ران تایم کاهش یافته است.

همچنین نکته‌ی دیگر آن است که در این حالات چون روی متغیر shared کنترلی نداریم حالت race condition به وجود می آید و دیگر مقدار sum با مقدار iterations که در ورودی دادیم برابر نیست.

همانطور که در کلاس گفته شد یک بار x را global تعریف میکنیم و یک بار هم globalتعریف نمیکنیم که نتایج زیر را خواهیم داشت

گلوبال باشد:

```
majid@majid-virtual-machine:~/Desktop/os_lab/5$ gcc x-global.c -o main; ./main
Parent has x = 0
Child has x = 2
majid@majid-virtual-machine:~/Desktop/os_lab/5$
```

گلوبال نباشد:

```
majid@majid-virtual-machine:~/Desktop/os_lab/5$ gcc x-no-global.c -o main; ./main
Parent has x = 0
Child has x = 2
majid@majid-virtual-machine:~/Desktop/os_lab/5$
```

مرحله سوم: ایا این برنامه درگیر شرایط مسابقه می شود؟چگونه؟اگر جوابتان مثبت بود راه حلی برای آن بیابید.

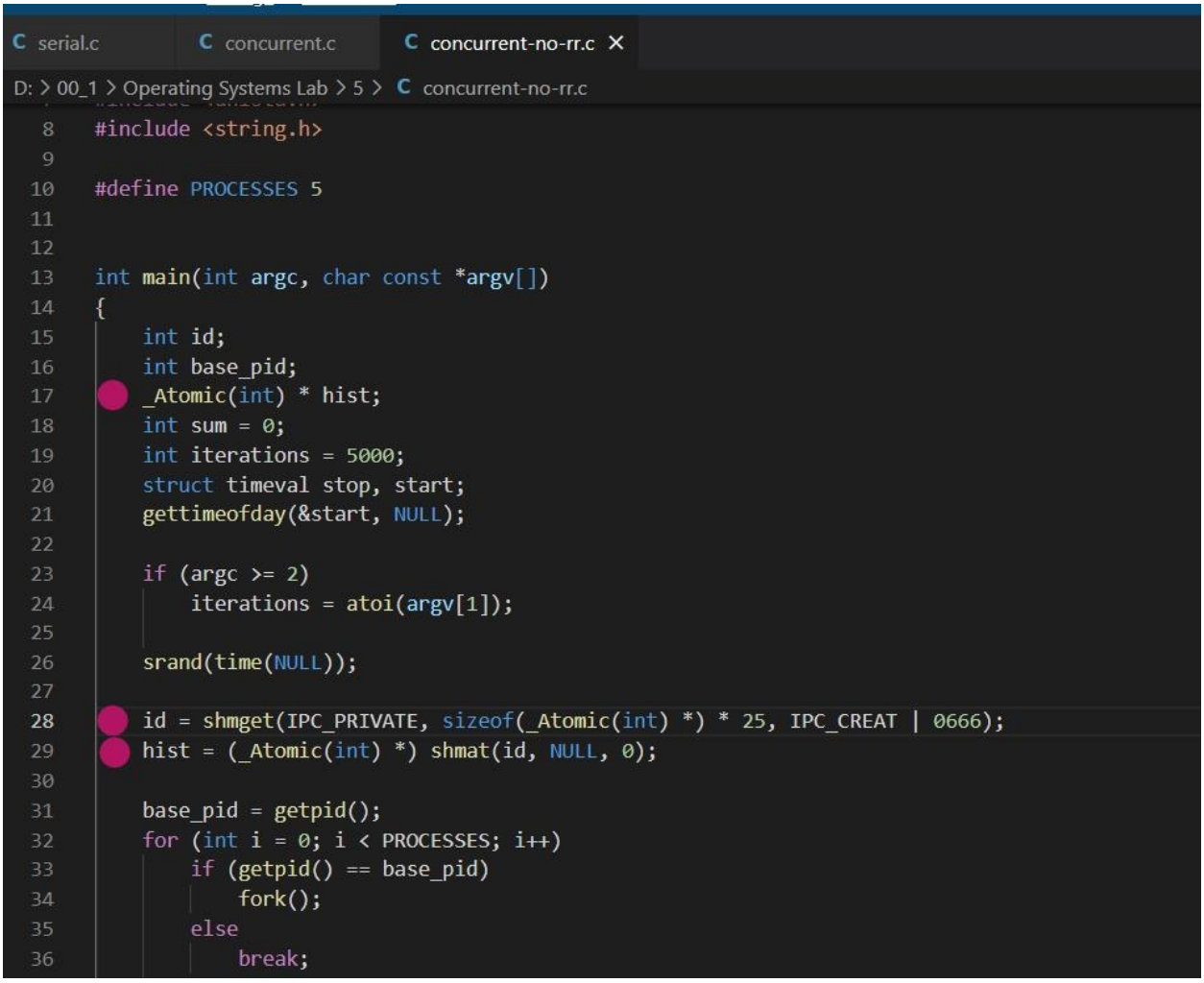
بله. با توجه مجموع اعضای آرایه hist که برابر مقدار ورودی نیست میتوان این موضوع را دریافت.

از آنجایی که آرایه hist را در حافظه اشتراکی تعریف کرده ایم، وقتی مقدار آن را افزایش میدهیم ممکن است به صورت همروند یا همزمان، پردازه دیگر سعی در افزایش آن بکند. دستور افزایش ++ یک دستور اتمیک نیست و از سه دستور اسمبلی تشکیل شده که آخرین دستور آن ریختن مقدار افزایش یافته در حافظه است. اگر پردازه ای قبل از رسیدن به این بخش، CPU را به پردازه دیگری

واگذار کند دچار خطا در این مقدار خواهیم شد. این همان پدیده Race Condition است و به این بخش از کد که این پدیده را به وجود می آورد ناحیه بحرانی گفته می شود.

راه حل:

یک راه حل تعریف جنس اعضای این آرایه به صورت Atomic است. برای اینکار به جای قرار دادن int در پشت نام متغیر، از کلید واژه _Atomic استفاده می کنیم. در کد زیر این بخش مشاهده می شود:



راه حل دیگر استفاده از کتابخانه semaphore.h است. به این صورت عمل میکنیم که در ابتدای ناحیه بحرانی تابع sem_wait و در انتهای آن تابع sem_post را فراخوانی می کنیم. با رسیدن

یک پردازه به sem_wait سیستم عامل جلوی اجرای این بخش از کد توسط پردازه های دیگر را می گیرد تا زمانی که sem_post صدا زده شود. پس تا زمانی که پردازه قبل کارش تمام نشده و sem_post را صدا نزده هیچ پردازه دیگری حق افزایش این مقدایر را ندارد.

همچنین در صورت تعریف کردن به صورت اتمیک خواهیم دید که این مشکل race برطرف میشود:

برای iterations=5000 :

```
majid@majid-virtual-machine:~/Desktop/os_lab/5$ gcc concurrent-no-rr.c -o main; ./main 5000
-12:       0 -
-11:       0 -
-10:      15 -
 -9:       0 -
 -8:      80 - *
 -7:       0 -
 -6:     240 - ****
 -5:       0 -
 -4:     550 - ***********
 -3:       0 -
 -2:     870 - *****************
 -1:       0 -
  0:    1145 - **********************
  1:       0 -
  2:     940 - ******************
  3:       0 -
  4:     690 - *************
  5:       0 -
  6:     350 - *******
  7:       0 -
  8:     115 - **
  9:       0 -
 10:       5 -
 11:       0 -
 12:       0 -

The runtime is 1092 us
The number of iterations is 5000.
majid@majid-virtual-machine:~/Desktop/os_lab/5$
```

برای iterations=50000 :

```
majid@majid-virtual-machine:~/Desktop/os_lab/5$ gcc concurrent-no-rr.c -o main; ./main 50000
-12:      10 -
-11:       0 -
-10:      90 -
 -9:       0 -
 -8:     590 - *
 -7:       0 -
 -6:    2320 - ****
 -5:       0 -
 -4:    5485 - **********
 -3:       0 -
 -2:    9060 - ******************
 -1:       0 -
  0:   11155 - **********************
  1:       0 -
  2:   10005 - ********************
  3:       0 -
  4:    6755 - *************
  5:       0 -
  6:    3165 - ******
  7:       0 -
  8:    1115 - **
  9:       0 -
 10:     225 -
 11:       0 -
 12:      25 -

The runtime is 5797 us
The number of iterations is 50000.
majid@majid-virtual-machine:~/Desktop/os_lab/5$
```

برای iterations=500000 :

```
majid@majid-virtual-machine:~/Desktop/os_lab/5$ gcc concurrent-no-rr.c -o main; ./main 500000
-12:      95 -
-11:       0 -
-10:    1130 -
 -9:       0 -
 -8:    6435 - *
 -7:       0 -
 -6:   21595 - ****
 -5:       0 -
 -4:   53590 - **********
 -3:       0 -
 -2:   90695 - ******************
 -1:       0 -
  0:  111960 - **********************
  1:       0 -
  2:  101715 - ********************
  3:       0 -
  4:   68355 - *************
  5:       0 -
  6:   32085 - ******
  7:       0 -
  8:   10130 - **
  9:       0 -
 10:    2040 -
 11:       0 -
 12:     175 -

The runtime is 44090 us
The number of iterations is 500000.
majid@majid-virtual-machine:~/Desktop/os_lab/5$
```

مرحله چهارم: نتایج قسمت اول و دوم را مقایسه کنید و میزان افزایش سرعت را در جدول زیر گزارش دهید.

ما در این سوال افزایش سرعت را برابر با $\frac{\text{زمان در حالت االول}}{\text{زمان در حالت دوم}}$ در نظر گرفتیم:

تعداد نمونه	5000	50000	500000
افزایش سرعت	0.495 برابر	1.607 برابر	1.462 برابر

با توجه به مقایسه‌ی نتایج قسمت اول و دوم ، میتوان گفت در حالت دوم با استفاده از fork و پخش کردن کارها بین فرزندان سرعت اجرای برنامه افزایش یافته و مقدار ران تایم کاهش میابد.