دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

گزارش کار آزمایشگاه سیستم عامل – شماره ۴

فرآیندها و نخ‌ها

حسنا اویارحسینی – ۹۸۲۳۰۱۰

استاد درس: جناب آقای مهندس کیخا

نیمسال دوم سال تحصیلی ۰۱-۱۴۰۰

بخش ۱)

ابتدا کد برنامهای که در تعریف مسئله شرح داده شد را در حالت سریال بنویسید و زمان اجرا شدن برنامه خود را در جدول زیر گزارش دهید.

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

void printHisogram(int *hist){
   int i,j;
   for(i = 0; i < 25; i ++){
       for (j = 0; j < (int)(hist[i]); j ++){
           printf("*");
       }
       printf("\n");
   }
}
int main(){
   clock_t begin = clock();

   srand(time(NULL));
   int hist [25] = {0};
   int n = 5000;
   for (int j = 0; j < n; j ++){
       int counter = 0;
       for (int i = 0; i < 12; i ++){
           int random_number = rand() % 100;
           if (random_number >= 49)
               counter ++;
           else
               counter --;
       }
       hist[counter + 12] ++;
   }

   clock_t end = clock();

   printf("Number of samples= %d\n", n);
   double time_spend = (double)(end - begin) * 1000.0 / CLOCKS_PER_SEC;
   printf("Time= %f\n", time_spend);
   printHisogram(hist);

}
```

برای انجام این کار از ۲ for تو در تو استفاده میکنیم در حلقه داخلی ۱۲ بار عدد رندوم تولید کرده و اگر این عدد بزرگتر یا مساوی ۴۹ بود به counter یکی اضافه کرده و در غیر این صورت از counter یکی کم میکنیم، سپس این کار را به تعداد نمونهها (n) بار انجام داده و هر بار در آرایه hist مشخص میکنیم که counter چه

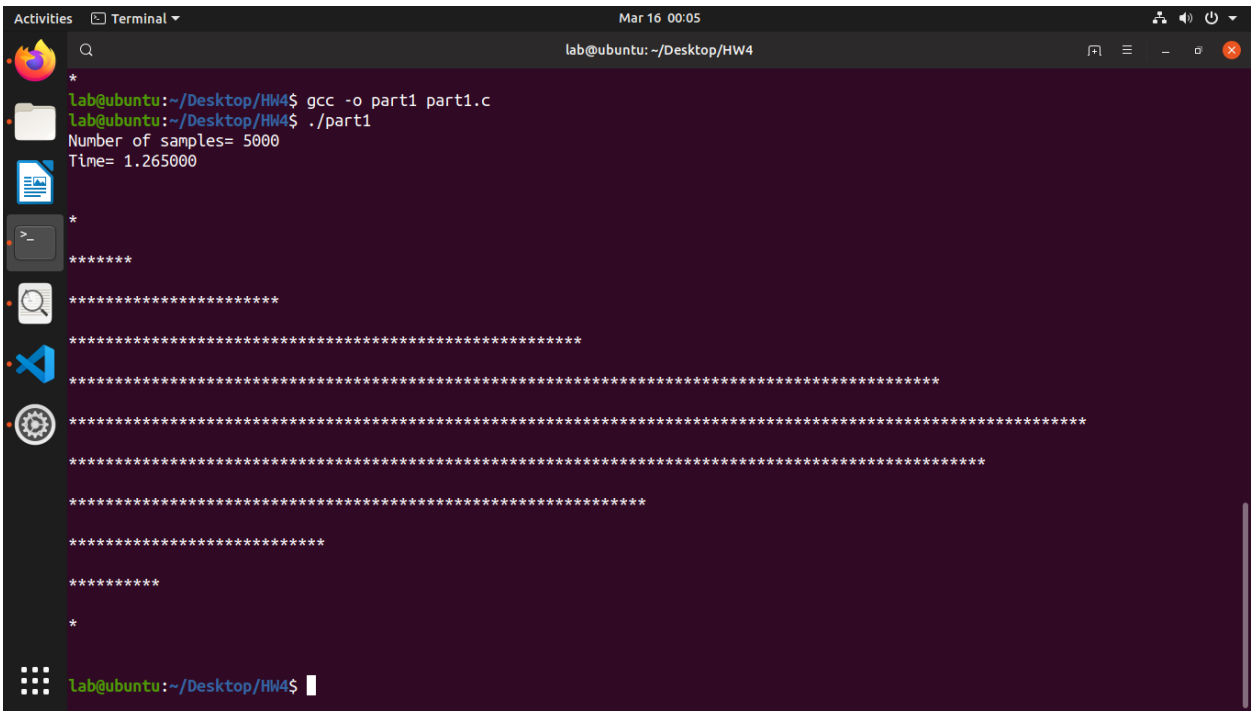
مقداری داشته است و خانه متناظر با مقدار counterرا در آرایه hist یکی زیاد میکنیم. با تمام شدن حلقه بیرونی، آرایه hist داده‌هایی مربوط به یک توزیع نرمال تصادفی را خواهد داشت.

سپس به کمک تابع printHisogram نمودار هیستوگرام این آرایه را رسم میکنیم. (چون تعداد داده‌ها زیاد می باشد و رسم یک * برای هر یک در ترمینال جا نمیشد [مانند شکل ۴] به ازای هر n/500 نمونه یک * رسم شده است تا شکل واضح تر باشد)

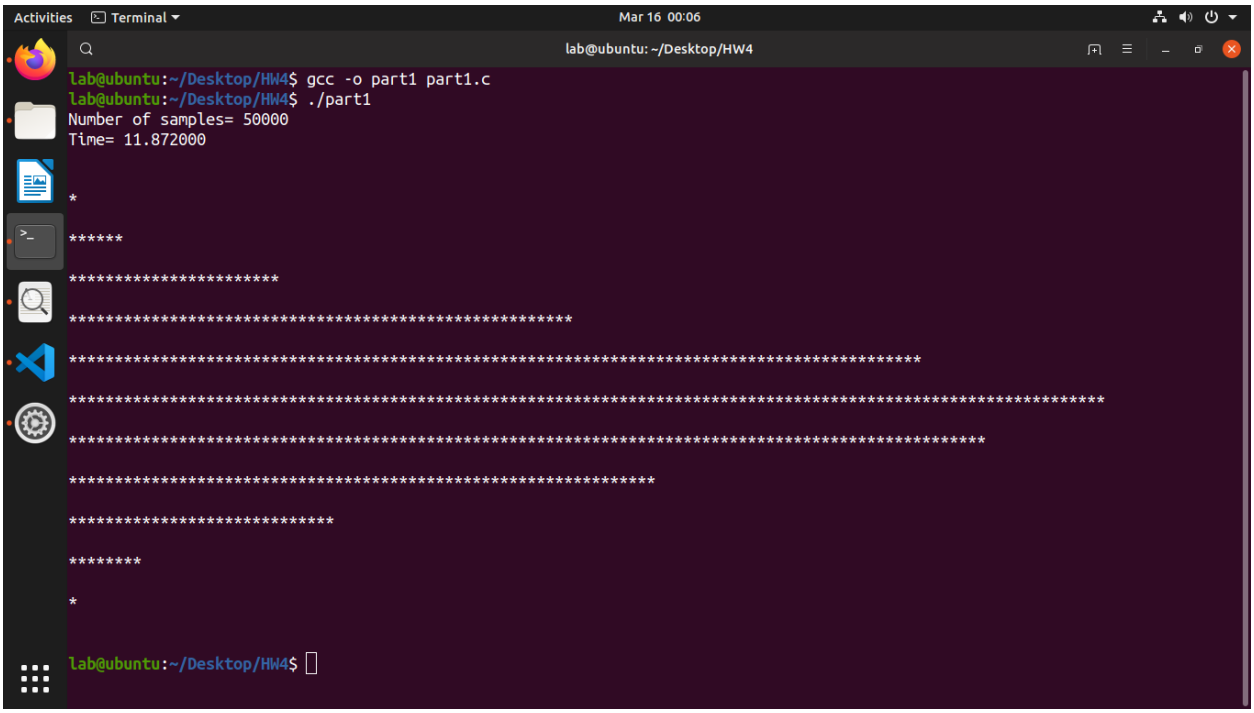
برای اندازه گیری زمان اجرای برنامه از تابع clock() از کتابخانه time.h استفاده شده است، به این صورت که یکبار در ابتدا و یکبار در انتهای برنامه این تابع را صدا میزنیم و در دو متغیر begin, end ذخیره میکنیم سپس اختلاف این دو عدد که برحسب کلاک میباشد را ضربدر ۱۰۰۰و تقسیم بر CLOCKS_PER_SEC میکنیم تا خروجی بر حسب ms بدست آید.

در ادامه خروجی این برنامه به‌ازای تعداد نمونه های 5000, 50000, 500000 آمده است:

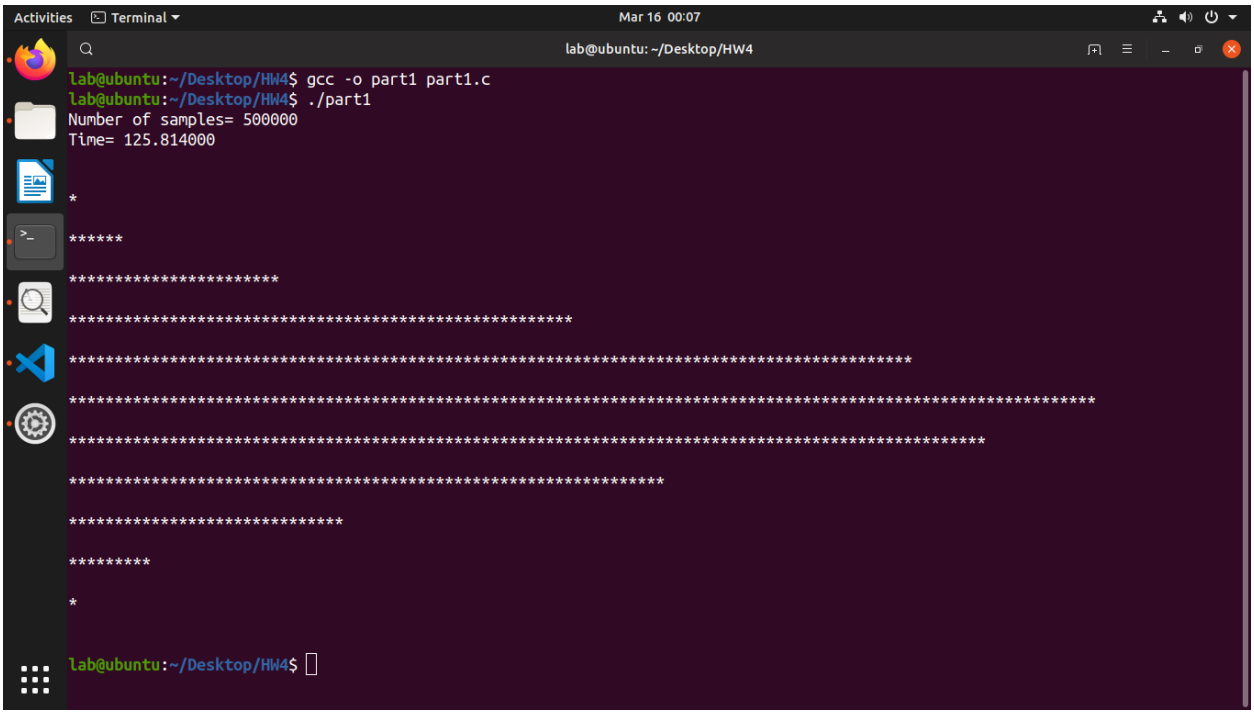
تعداد نمونه	۵۰۰۰	۵۰۰۰۰	۵۰۰۰۰۰
زمان اجرا(ms)	۱,۲۶۵	۱۱,۸۷۲	۱۲۵,۸۱۴



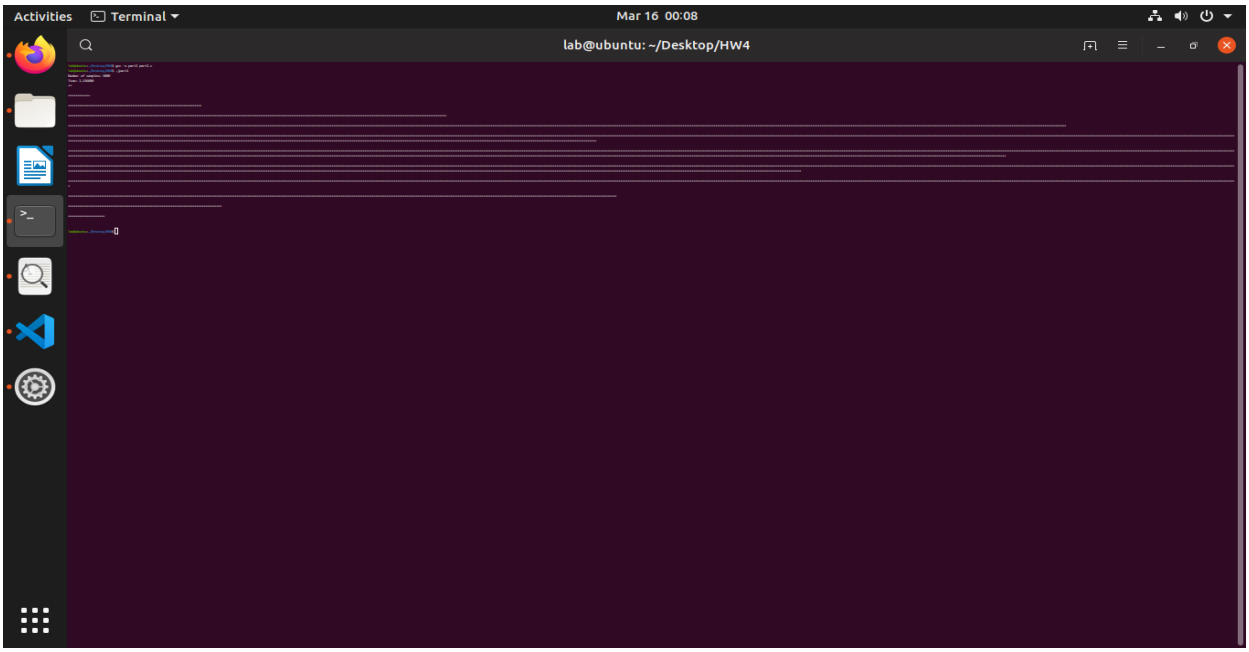
شکل 1 خروجی سریال به ازای n=5000



شکل2 خروجی سریال به ازای n=50000



شکل3 خروجی سریال به ازای n=500000



شکل4 خروجی سریال به ازای n=5000 در حالتی که به ازای هر نمونه یک * گذاشته ایم

بخش ۲)

حال برنامه‌ای بنویسید که با استفاده از () fork و یا ()exec تعدادی فرآیند فرزند ایجاد شود و کارها را پخش کنید.

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<sys/types.h>
#include<unistd.h>
#include<sys/shm.h>
#include<sys/stat.h>
#include<sys/wait.h>

enum { ARRAY_SIZE = 25 };
int segmentId;
const int shareSize = sizeof(int) * (ARRAY_SIZE);
int *hist;

void calculate_smaple(int end){
   for (int j = 0; j < end; j ++){
       int counter = 0;
       for (int i = 0; i < 12; i ++){
           int random_number = rand() % 100;
           if (random_number >= 49)
               counter ++;
           else
               counter --;
       }
       hist[counter + 12] ++;
```

```
    }
}

void printHisogram(int *hist){
    int i,j;
    for(i = 0; i < 25; i ++){
        for (j = 0; j < (int)(hist[i] / 10); j ++){
            printf("*");
        }
        printf("\n");
    }
}

int main(){

    clock_t begin = clock();

    segmentId = shmget(IPC_PRIVATE, shareSize, S_IRUSR | S_IWUSR);
    hist = (int *) shmat(segmentId, NULL, 0);
    srand(time(NULL));
    int n = 5000;

    int n1 = fork();
    int n2 = fork();
    calculate_smaple(n/4);
    while(wait(NULL) != -1);

    /* parent process */
    if (n1 != 0 && n2 != 0){

        clock_t end = clock();

        int sum = 0;
        for(int i=0; i<25; i++)
            sum += hist[i];


        printf("Number of samples= %d\n", sum);
        double time_spend = (double)(end - begin) * 1000 / CLOCKS_PER_SEC;
        printf("Time= %f\n", time_spend);
        printHisogram(hist);
    }

}
```

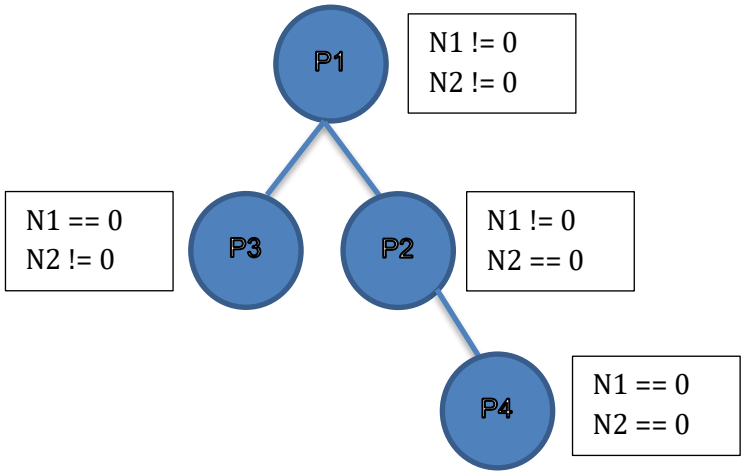
در این قسمت سعی میکنیم با تقسیم کردن کار بین چند پردازه سرعت انجام عملیات را بیشتر کنیم. برای اینکار ابتدا حافظه‌ای به اندازه ۲۵ عدد صحیح در shared memory به صورت زیر میگیریم در واقع این حافظه همان آرایه hist خواهد بود ولی چون میخواهیم همزمان چند پردازه به آن دسترسی داشته باشند آن را در حافظه اشتراکی قرار میدهیم.

ابتدا به کمک تابع shmget حافظه مشترک را ایجاد کرده و شناسه حافظه را در shegmentId میریزیم، سپس در خط دوم این حافظه مشترک را تحت عنوان hist به فضای آدرس پردازه فعلی اضافه میکنیم (این کار

قبل از fork انجام شده است ولی چون fork کردن بدون استفاده از exec فضای آدرس دهی پردازه را کپی میکند این حافظه مشترک در پرداز های فرزند که در ادامه اضافه خواهند شد، در دسترس خواهد بود):

```
segmentId = shmget(IPC_PRIVATE, shareSize, S_IRUSR | S_IWUSR);
hist = (int *) shmat(segmentId, NULL, 0);
```

سپس از دو دستور fork استفاده میکنیم تا ۳ پردازه دیگر بسازیم در واقع پس از دو خط مربوط به فورک وضعیت پردازه ها به صورت زیر خواهد بود:

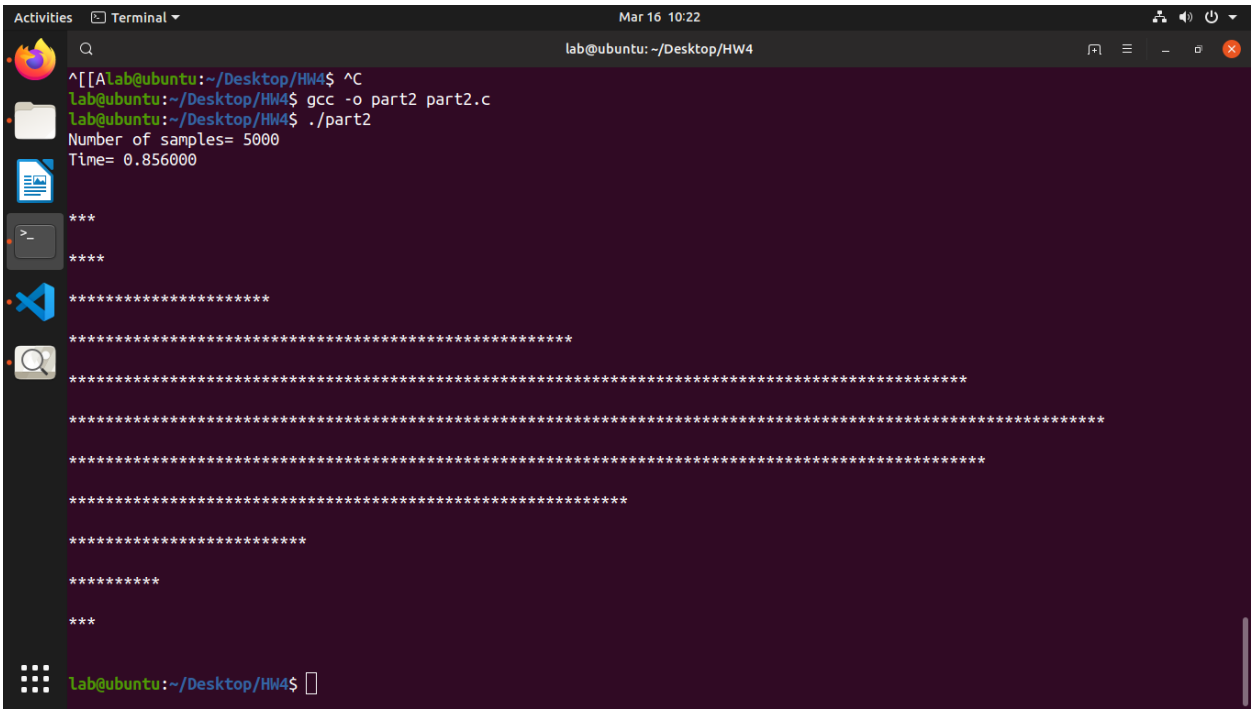


سپس در هر پردازه تابع `calculate_smaple(n/4);` صدا زده میشد که عملیات نمونه برداری را برای n/4 از داده ها همانند روال توضیح داده شده در آزمایش ۱ انجام میدهد و نتیجه را در hist که در حافظه اشتراکی وجود دارد ذخیره میکند. و پس از انجام این کار با دستور `while(wait(NULL) != -1);` هر پردازه صبر میکند تا پردازنده های فرزندش تمام شوند (زیرا دستور wait() در صورتی که هیچ پردازه ای برای منتظر بودن روی آن را نداشته باشد عدد -۱ و در غیر اینصورت عدد مثبتی برمیگرداند).

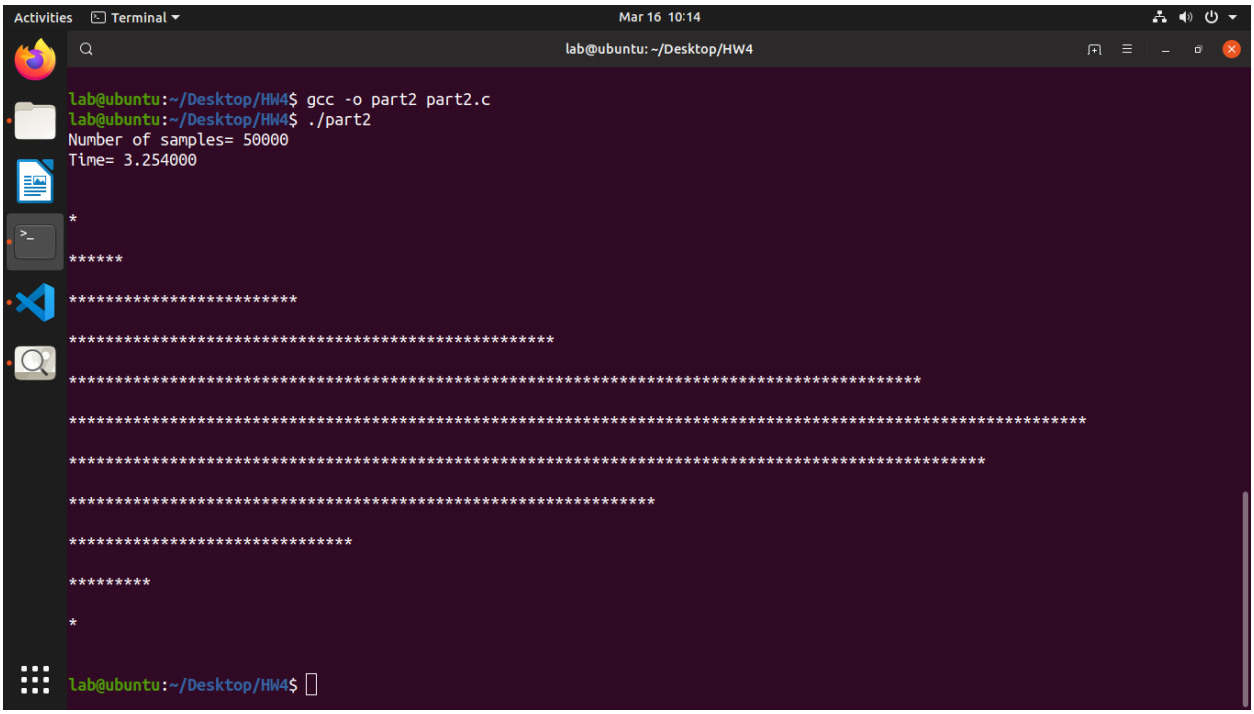
در نهایت پس از اتمام تمام پردازه ها پردازه پدر وارد if میشود و زمان اجرای برنامه و هیستوگرام مربوط به آرایه مشترک hist که توسط هر ۴ پردازه پر شده است را رسم میکند.

در ادامه نتایج این برنامه به‌ازای تعداد نمونه های 5000, 50000, 500000 را میبینیم:

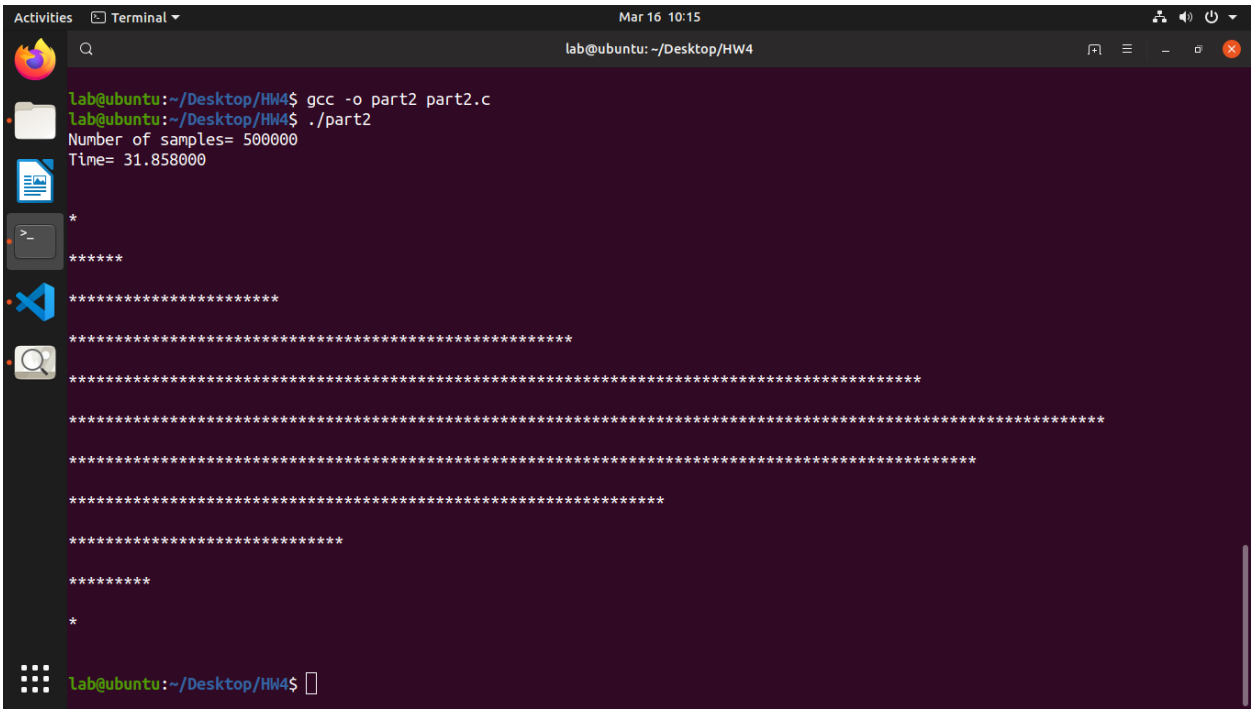
تعداد نمونه	۵۰۰۰	۵۰۰۰۰	۵۰۰۰۰۰
زمان اجرا(ms)	۰,۸۵۶	۳,۲۵۴	۳۱,۸۵۰



شکل5 خروجی چندفرآیندی به ازای n=5000



شکل6 خروجی چندفرآیندی به ازای n=50000



شکل 7 خروجی چندفرآیندی به ازای n=500000

بخش ۳)

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

بله این برنامه دچار race condition میشود زیرا همزمان ۴ پردازه روی یک حافظه مشترک که مربوط به آرایه hist می باشد مینویسند. برای مثال اگر دو پردازه بخواهد همزمان، یکی از مقدار hist[3] کم کند و یکی به آن یک واحد اضافه کند ممکن است حالت زیر پیش بیاید و نتیجه نادرست شود:

- فرض میکنیم مقدار اولیه hist[3] صفر بوده پس انتظار داریم پس از عملیات گفته شده مقدار آن همچنان ۰ بماند (زیرا یکی کم و سپس اضافه میشود) اما اگر register1 را مربوط به محاسبات پردازه اول و register2 را مربوط به محاسبات پردازه دوم بدانیم ممکن است برای مثال حالت زیر پیش بیاید و نتیجه برخلاف انتظار -۱ شود و خطا رخ دهد.

S0: p0 execute **register1 = hist[3]** {register1 = 0}
S1: p0 execute **register1 = hist[3] + 1** {register1 = 1}
S2: p1 execute **register2 = hist[3]** {register2 = 0}
S3: p1 execute **register2 = hist[3] – 1** {register2 = -1}
S4: p0 execute **hist[3] = register1** {counter = 1 }
S5: p1 execute **hist[3] = register2** {counter = -1}

یک راه حل برای حل این مشکل میتواند این باشد که ۴ آرایه ۲۵ تایی در حافظه مشترک درنظر بگیریم و به کمک دو متغیر n1, n2 تعیین کنیم که هر پردازه در یکی از ۴ آرایه بنویسد سپس در نهایت پردازه پدر مقادیر موجود در این ۴ پردازه را باهم جمع بزند و به عنوان نتیجه نهایی اعلام کند در این حالت چون پردازه ها روی محل یکسانی write نمیکنند مشکل race condition پیش نمیاد.

```
segmentId1 = shmget(IPC_PRIVATE, shareSize, S_IRUSR | S_IWUSR);
hist1 = (int *) shmat(segmentId, NULL, 0);
segmentId2 = shmget(IPC_PRIVATE, shareSize, S_IRUSR | S_IWUSR);
hist2 = (int *) shmat(segmentId, NULL, 0);
segmentId3 = shmget(IPC_PRIVATE, shareSize, S_IRUSR | S_IWUSR);
hist3 = (int *) shmat(segmentId, NULL, 0);
segmentId4 = shmget(IPC_PRIVATE, shareSize, S_IRUSR | S_IWUSR);
hist4 = (int *) shmat(segmentId, NULL, 0);
```

و برای قسمت محاسبه hist ها خواهیم داشت:

```
if (n1 != 0 && n2 !=0){
   calculate_smaple(n/4, hist1);
}else if (n1 == 0 && n2 !=0){
   calculate_smaple(n/4, hist2);
}else if (n1 != 0 && n2 ==0){
   calculate_smaple(n/4, hist4);
}else if (n1 == 0 && n2 ==0){
   calculate_smaple(n/4, hist4);
}
if (n1 != 0 && n2 != 0){

       clock_t end = clock();

       int sum [25] = {};
       for(int i=0; i<25; i++){
           sum[i] += (hist11[i] + hist2[i] + hist3[i] + hist4[i];
       }


       printf("Number of samples= %d\n", sum);
       double time_spend = (double)(end - begin) * 1000 / CLOCKS_PER_SEC;
       printf("Time= %f\n", time_spend);
       printHisogram(sum);
}
```

اما راه بهتر برای حل این مشکل استفاده از سمافور میباشد. سمافور (به انگلیسی: Semaphore) به متغیری گفته می‌شود که در محیط‌های همروند برای کنترل دسترسی فرایندها به منابع مشترک به کار می‌رود. سمافور می‌تواند به دو صورت دودویی (که تنها دو مقدار صحیح و غلط را دارا است) یا شمارنده اعداد صحیح باشد. از سمافور برای جلوگیری از ایجاد وضعیت رقابتی میان فرآیندها استفاده می‌گردد. به این ترتیب، اطمینان حاصل می‌شود که در هر لحظه تنها یک فرایند به منبع مشترک دسترسی دارد و می‌تواند از آن بخواند یا بنویسد.

سمافور در واقع متغیری است که سیستم عامل به ما می دهد و ما فقط میتوانیم دو تابع wait, signal را روی آن صدا بزنیم. هنگامی که تابع wait را روی سمافور فراخوانی بکنیم مقدار آن یک واحد کم می شود و هنگامی که تابع signal را روی آن فراخوانی بکنیم مقدار آن یکی افزایش می یابد. هنگامی که سمافور به صفر رسید دیگر تابع wait نمی تواند مقدار آن را کمتر کند و پردازه ای که wait را فراخوانی کرده است، ادامه‌اش اجرا نمی شود و صبر می کند تا پردازه ی دیگری signal را فراخوانی کند، تا از حالت بلاک در بیاید. به کمک سمافور میتوانیم دسترسی همزمان به یک خانه از آرایه توسط دو پردازه را ببندیم تا شرایط مسابقه پیش نیاید.

بخش۴)

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

تعداد نمونه	۵۰۰۰	۵۰۰۰۰	۵۰۰۰۰۰
افزایش سرعت	٪۳۲,۳۳	٪۷۲,۵۹	٪۷۴,۶۸

مشاهده میکنیم که سرعت برنامه در بخش دوم که کار بین چند پردازه تقسیم شده است بیشتر شده است.