**گزارش دستورکار پنجم آزمایشگاه سیستم­های عامل**

نگار موقتیان، 9831062

1. **انجام محاسبات به صورت سریال**

در این قسمت از آزمایش می­خواهیم برنامه­­ای بنویسیم که به صورت سریال مطابق دستورکار آرایۀ hist را پر کند، بنابراین در این مرحله برای اجرای تمام محاسبات تنها از یک پردازه استفاده می­کنیم. ساختار برنامۀ نوشته شده بدین منظور در ادامه آمده­است.

در ابتدای برنامه کتابخانه­های مورد نیاز اضافه شده­اند.

پس از آن مطابق توضیحات دستورکار یک آرایۀ 25 تایی به نام hist و یک متغیر به نام samples تعریف شده­است که تعداد نمونه­های آزمایش را در خود ذخیره می­کند (این متغیر با توجه به ورودی کاربر مقداردهی خواهد شد).

در ادامه تابع printHistogram نیز مانند آنچه در دستورکار آمده تعریف شده­است، با این تفاوت که یک عدد صحیح scale را به عنوان ورودی خود می­پذیرد. زمانی که تعداد نمونه­های آزمایش زیاد می­شود مقدار هر خانه از آرایۀ hist نیز به طبع افزایش خواهد یافت. به ازای تعداد نمونه­های بسیار بزرگ مقدار بعضی از خانه­های این آرایه به قدری زیاد می­شود که نمایش آن در یک خط ترمینال ممکن نیست. بنابراین یک متغیر scale تعریف شده­است تا تمام این داده­ها را به نسبتی کوچک کند. برای انتخاب این ضریب نیز به این صورت عمل شده­است که اگر تعداد نمونه­ها کم­تر از 500 بود به آن­ها ضریب 1 می­دهیم و در غیر این صورت آن­ها را تقسیم بر  
(500 / تعداد نمونه­ها) می­کنیم. در این صورت مطمئن هستیم تعداد ستاره­های چاپ شده از 500 عدد بیش­تر نمی­شود.

پس از آن در تابع main ابتدا تابع srand() را صدا می­زنیم تا یک هسته برای تولیدکنندۀ اعداد تصادفی تعیین کنیم. در غیر این صورت با هر بار اجرای برنامه اعداد تصادفی تولید شده یکسان خواهند بود.

در ادامه مقدار آرگومان اول داده شده به برنامه را به عنوان تعداد نمونه­ها تعریف می­کنیم و در صورتی که این آرگومان به برنامه داده نشده بود پیغام مناسب داده و برنامه را خاتمه می­دهیم.

سپس در یک حلقۀ تو در تو به ازای هر نمونه 12 عدد تصادفی تولید کرده و مطابق دستور کار مقدار counter را تغییر می­دهیم تا نهایتاً این نمونه را در یکی از خانه­های آرایۀ hist قرار دهیم.

در نهایت نیز با استفاده از تابع printHistogram نمودار متناظر با مقادیر موجود در آرایۀ hist را رسم می­کنیم.

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<sys/time.h>

int hist[25], samples;

// print the final histogram. the variable "scale" is used as to make sure the printed stars fit in the terminal window

void printHistogram(int scale) {

printf("\n=====================================");

for (int i = 0; i < 25; i++) {

for (int j = 0; j < hist[i] / scale; j++)

printf("\*");

printf("\n");

}

printf("=====================================\n");

}

int main(int argc, char \*argv[]) {

srand(time(0));

if (argc < 2) { // check if all the required parameters are passed

printf("Please pass the number of samples as argument.\n");

return -1;

}

samples = atoi(argv[1]);

// record the starting time

struct timeval begin, end;

gettimeofday(&begin, 0);

// fill the hist array

for (int i = 0; i < samples; i++) {

int counter = 0;

for (int j = 0; j < 12; j++)

counter += (rand() % 101 >= 49) ? 1 : -1;

hist[counter + 12]++;

}

// compute the elapsed time in milliseconds and print it

gettimeofday(&end, 0);

long seconds = end.tv\_sec - begin.tv\_sec;

long microseconds = end.tv\_usec - begin.tv\_usec;

double elapsed = (seconds + microseconds\*1e-6) \* 1e3;

printf("\nFinished Processing in %.3f milliseconds.\n", elapsed);

// plot the final result

printHistogram((samples >= 500) ? samples / 500 : 1);

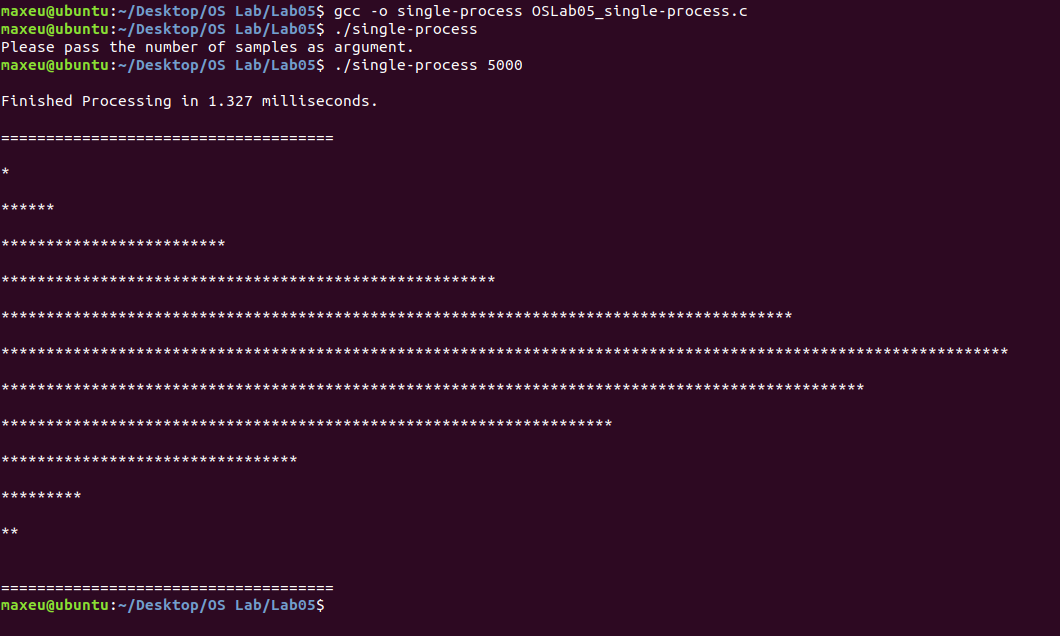
return 0;

}

همچنین زمان را در شروع فرآیند برنامه خوانده و در یک struct از جنس timeval ذخیره می­کنیم. همین کار را در انتهای فرآیند نیز انجام داده و اختلاف این دو زمان را پیدا می­کنیم تا زمان سپری شده برای اجرای عملیات برنامه را محاسبه کنیم. برای اندازه­گیری زمان در این قسمت از برنامه از لینک زیر کمک گرفته شده­است:

https://levelup.gitconnected.com/8-ways-to-measure-execution-time-in-c-c-48634458d0f9

در نهایت خروجی این برنامه به ازای مقادیر خواسته شده در دستورکار به صورت زیر است:

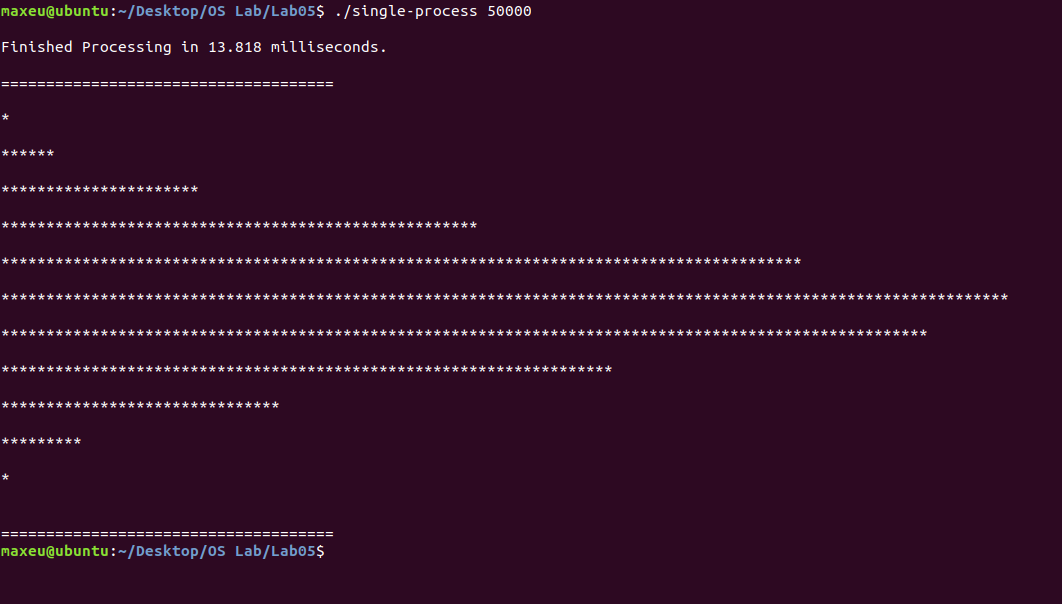


در ابتدا برنامه را کامپایل کرده­ایم.

سپس در اجرای اول به برنامه آرگومانی نداده­ایم و همانطور که دیده می­شود پیغام مناسبی چاپ شده­است.

پس از آن تعداد نمونه­ها 5.000 عدد در نظر گرفته شده و محاسبات در 327/1 میلی ثانیه انجام شده­است.

این بار تعداد نمونه­ها 50.000 عدد در نظر گرفته شده و محاسبات در 818/13 میلی ثانیه انجام شده­است.



و در نهایت تعداد نمونه­ها 500.000 عدد در نظر گرفته شده و برنامه در 588/123 میلی ثانیه انجام شده­است.

بنابراین جدول خواسته شده را می­توان با مقادیر زیر پر کرد:

|  |  |  |  |
| --- | --- | --- | --- |
| تعداد نمونه | 5.000 | 50.000 | 500.000 |
| زمان اجرا (میلی ثانیه) | 327/1 | 818/13 | 588/123 |

1. **انجام محاسبات به صورت موازی**

حال در این قسمت از آزمایش می­خواهیم برنامه­­ را به صورتی بنویسیم که در آن چندین پردازه به صورت موازی بر روی آرایۀ hist کار کنند. در این صورت با تقسیم کار میان پردازه­ها می­تواند سرعت اجرای برنامه را افزایش داد. بدین منظور این بار آرایۀ hist را به صورت shared memory تعریف می­کنیم تا تمامی پردازه­ها به آن دسترسی داشته باشند و بتوانند به طور همزمان بر روی آن داده بنویسند و در نهایت پردازۀ پدر نیز بتواند مقدار آن را بخواند تا پاسخ نهایی را چاپ کند. ساختار برنامۀ نوشته شده بدین منظور در ادامه آمده­است.

در ابتدای برنامه کتابخانه­های مورد نیاز اضافه شده­اند.

پس از آن متغیرهای samples، تعداد نمونه­ها، processes، تعداد پردازه­ها، pid، pid ای که تابع fork در ادامه برمی­گرداند (دلیل مقداردهی اولیۀ آن به 1 توضیح داده خواهد شد) و memoryID که مسئول ذخیره­سازی id مربوط به shared memory ساخته شده است تعریف شده­اند.

در ادامه تابع printHistogram نیز درست مانند قسمت قبل تعریف شده­است، با این تفاوت که این بار آرایۀ hist به آن پاس داده شده­است، زیرا دیگر یک آرایۀ global نیست و آن را در طول اجرای برنامه از طریق shared memory دریافت می­کنیم.

پس از آن در تابع main باز هم ابتدا تابع srand() را صدا می­زنیم تا نتایج تولید شده کاملاً تصادفی باشند.

سپس در ادامه مقدار آرگومان اول و دوم داده شده به برنامه را به عنوان تعداد نمونه­ها و تعداد پردازه­ها تعریف می­کنیم و در صورتی که این آرگومان­ها به برنامه داده نشده بودند پیغام مناسب داده و برنامه را خاتمه می­دهیم.

حال shared memory ای که قرار است پردازه­ها به طور مشترک بر روی آن کار کنند را تعریف می­کنیم. برای این کار از متن آزمایش قبلی دستورکار و همچنین سایت زیر برای تکمیل ورود­ی­های تابع shmget استفاده شده­است:

<https://www.mkssoftware.com/docs/man3/shmget.3.asp>

سپس به تعداد پردازه­ها (که آن را به عنوان ورودی دریافت کرده­ایم) تابع fork را بر روی پردازۀ اصلی صدا می­زنیم تا فرزندان آن به تعداد مناسب ایجاد شوند. برای تشخیص این که پردازه­ای که در آن هستیم پردازۀ اصلی برنامه و والد باقی پردازه­هاست از متغیر pid استفاده می­شود. این متغیر ابتدا با 1 (که عددی بزرگ­تر از صفر است) مقداردهی شده­است تا بار اول وارد if شده و fork را اجرا کنیم. از آن به بعد با هر بار اجرای fork متغیر pid را آپدیت می­کنیم، از این طریق در پردازه­های فرزند همواره pid برابر با صفر و در پردازۀ اصلی این عدد هر بار یک عدد بزرگ­تر از صفر خواهد بود (به شرط اجرای موفقیت­آمیز fork).

حال که تمام پردازه­هایی که می­خواستیم ایجاد شدند، برای هر یک از آن­ها آرایۀ hist را از shared memory از طریق ID آن بازیابی کرده و ذخیره می­کنیم.

پس از آن بررسی می­کنیم که داخل پردازۀ والد هستیم یا یکی از پردازه­های فرزند.

1. اگر داخل پردازۀ والد بودیم ابتدا زمان فعلی را به عنوان شروع زمان اجرا رکورد می­کنیم (پیش از این در حال ایجاد پردازه­ها بودیم و در این برنامه تنها زمان نیاز برای انجام محاسبات اندازه­گیری شده­است). سپس به ازای تمام فرزندان wait می­کنیم تا تمام محاسبات مربوطه پایان پذیرند. پس از آن مانند بخش قبلی آزمایش زمان سپری شده در این مدت را اندازه­گیری کرده و چاپ می­کنیم. در نهایت نیز آرایۀ پر شده توسط پردازه­های فرزند را توسط تابع printHistogram چاپ کرده و shareed memory ای که در حافظۀ اصلی رزور کرده بودیم را آزاد می­کنیم.
2. در غیر این صورت اگر داخل پردازۀ فرزند بودیم سهم خود از محاسبات را انجام می­دهیم (در این حالت تعداد نمونه­هایی که باید ایجاد شوند تعداد نمونه­ها تقسیم بر تعداد پردازه­ها خواهد بود).

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<unistd.h>

#include<sys/time.h>

#include<sys/types.h>

#include<sys/wait.h>

#include<sys/ipc.h>

#include<sys/shm.h>

int samples, processes, memoryID, pid = 1;

// print the final histogram. the variable "scale" is used as to make sure the printed stars fit in the terminal window

void printHistogram(int \*hist, int scale) {

printf("\n=====================================");

for (int i = 0; i < 25; i++) {

for (int j = 0; j < hist[i] / scale; j++)

printf("\*");

printf("\n");

}

printf("=====================================\n");

}

int main(int argc, char \*argv[]) {

srand(time(0));

if (argc < 3) { // check if all the required parameters are passed

printf("Please pass the number of samples and processes as argument.\n");

return -1;

}

samples = atoi(argv[1]);

processes = atoi(argv[2]);

// initiate the shared memory

memoryID = shmget(IPC\_PRIVATE, 25 \* sizeof(int), IPC\_CREAT | 0666);

// create child processes

for (int i = 0; i < processes; i++)

if (pid > 0) // fork only if we are in the main process

pid = fork();

else

break;

// retrieve the shared memory

int \*hist = (int \*) shmat(memoryID, NULL, 0);

if (pid > 0) { // we are in the main process

// record the starting time

struct timeval begin, end;

gettimeofday(&begin, 0);

// wait for all children to finish their process

for (int i = 0; i < processes; i++)

wait(NULL);

// compute the elapsed time in milliseconds and print it

gettimeofday(&end, 0);

long seconds = end.tv\_sec - begin.tv\_sec;

long microseconds = end.tv\_usec - begin.tv\_usec;

double elapsed = (seconds + microseconds\*1e-6) \* 1e3;

printf("\nFinished Processing in %.3f milliseconds.\n", elapsed);

// plot the final result

printHistogram(hist, (samples >= 500) ? samples / 500 : 1);

// free the shared memory

shmctl(memoryID, IPC\_RMID, NULL);

}

else { // we are in one of the child processes, so we need to fill the hist array

for (int i = 0; i < samples / processes; i++) {

int counter = 0;

for (int j = 0; j < 12; j++)

counter += (rand() % 101 >= 49) ? 1 : -1;

\*(hist + counter + 12) = \*(hist + counter + 12) + 1;

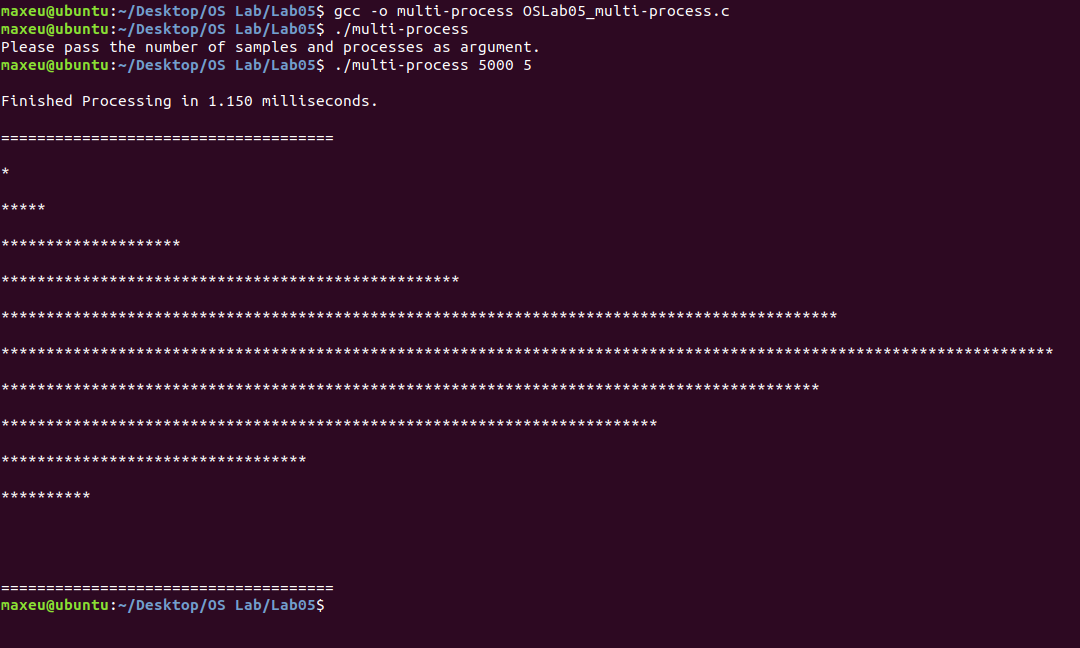
}

}

return 0;

}

در نهایت خروجی این برنامه به ازای مقادیر خواسته شده در دستورکار به صورت زیر است:

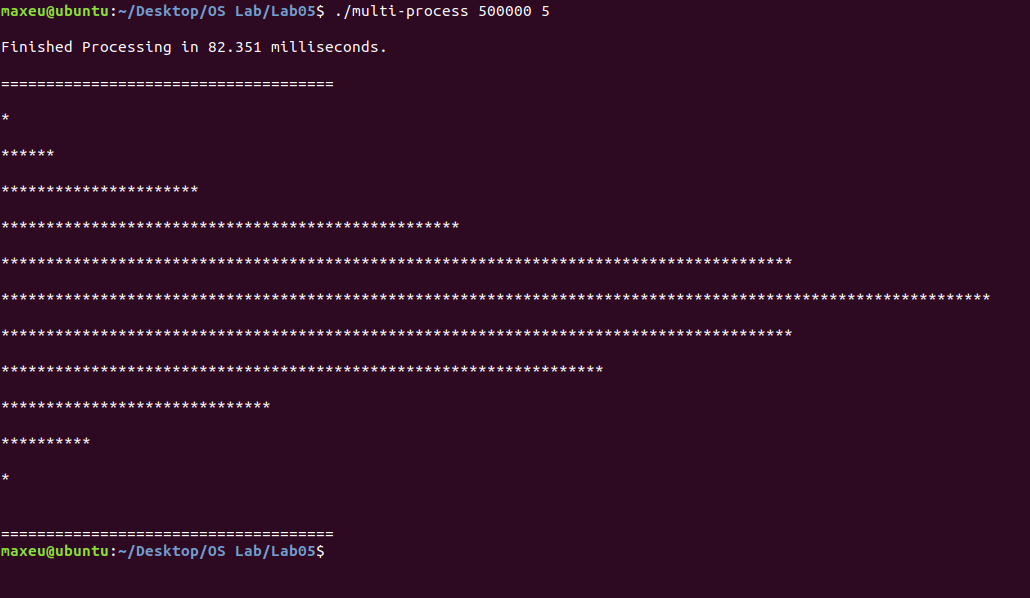


در ابتدا برنامه را کامپایل کرده­ایم.

سپس در اجرای اول به برنامه آرگومانی نداده­ایم و همانطور که دیده می­شود پیغام مناسبی چاپ شده­است.

پس از آن تعداد نمونه­ها 5.000 عدد و تعداد پردازه­ها 5 عدد در نظر گرفته شده و محاسبات در 150/1 میلی ثانیه انجام شده­است.

این بار تعداد نمونه­ها 50.000 عدد و تعداد پردازه­ها 5 عدد در نظر گرفته شده و محاسبات در 891/10 میلی ثانیه انجام شده­است.



و در نهایت تعداد نمونه­ها 500.000 عدد و تعداد پردازه­ها 5 عدد در نظر گرفته شده و برنامه در 351/82 میلی ثانیه انجام شده­است.

بنابراین جدول خواسته شده را می­توان با مقادیر زیر پر کرد:

|  |  |  |  |
| --- | --- | --- | --- |
| تعداد نمونه | 5.000 | 50.000 | 500.000 |
| زمان اجرا (میلی ثانیه) | 150/1 | 891/10 | 351/82 |

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

بله؛ از آن­جایی که تمام پردازه­های فرزند به طور همروند بر روی آرایۀ hist مقدار می­نویسند امکان به وجود آمدن race condition وجود دارد. به طور مشخص در یکی از خطوط برنامه داریم:

\*(hist + counter + 12) = \*(hist + counter + 12) + 1;

حال اگر مقدار counter در دو پردازه در یک زمان یکسان باشد، یکی مقدار این خانه از hist را خوانده، آن را تغییر دهد، سپس پیش از این که مقدار جدید این خانه را در آن بریزد پردازه­ای دیگر نیز همین خانه از آرایه را بخواند تا آن را تغییر دهد، نتیجۀ مورد انتظار را نخواهیم داشت.

البته از آن­جایی که حجم نمونه­ها در این آزمایش زیاد بوده و احتمال این که برای دو پردازه به طور همزمان متغیر counter یک مقدار یکسان پیدا کند کم است، خلل جدی­ای در روند آزمایش ایجاد نخواهد شد. اما برای رفع این مشکل می­توان از یکی از مکانیزم­های همگام سازی استفاده کرد.

برای مثال یکی از معروف­ترین مکانیزم­هایی که برای همگام­سازی پردازه­ها استفاده می­شود استفاده از semaphore است. برای استفاده از آن ابتدا یک سمافور با استفاده از shared memory ID ای که داریم ایجاد می­کنیم. سپس هر زمان که می­خواستیم داده­ای بر روی shared memory بنویسیم ابتدا از تابع sem\_wait() استفاده می­کنیم. این تابع بررسی می­کند که سمافور قفل شده­است یا خیر (قفل بودن یا نبودن آن با مقدار درون آن مشخص می­شود. برای مثال اگر مقدار آن 1 بود آزاد و اگر -1 بود قفل است). اگر قفل نبود آن را قفل کرده و بر روی حافظه مقدار دلخواه را می­نویسد، در غیر این صورت تا زمانی که قفل آن توسط پردازه­ای که آن را قفل کرده و با استفاده از دستور sem\_post() آزاد شود، بلاک می­شود.

لیست کامل­تری از این مکانیزم­ها در لینک زیر آمده­است:

<https://stackoverflow.com/questions/53736985/how-synchronization-is-done-in-shared-memory-data-linux-c>

1. **مقایسۀ روش اول و دوم و بررسی میزان افزایش سرعت:**

|  |  |  |  |
| --- | --- | --- | --- |
| تعداد نمونه | 5.000 | 50.000 | 500.000 |
| افزایش سرعت | %39/15 | %87/26 | %07/50 |

با توجه به جداول پر شده در قسمت اول و دوم می­توان نتایج زیر را بدست آورد:

همانطور که دیده می­شود افزایش سرعت قابل توجهی داشته­ایم. به علاوه به نظر می­رسد هر چه تعداد نمونه­ها بیش­تر شده، استفاده از چند پردازه به جای یک پردازه به صرفه­تر است. زیرا از طرفی استفاده از پردازه­ها به دلیل اجرای موازی کارها سرعت برنامه را افزایش می­دهد و از طرف دیگر کار کردن با چندین پردازه­ به جای یک پردازه برای سیستم سرباری ایجاد می­کند. این سربار با افزایش تعداد نمونه­ها (به شرط ثابت بودن تعداد پردازه­ها) ثابت می­ماند، در حالی که تسریع به دلیل اجرای موازی محاسبات خود را بیش­تر نشان می­دهد. لذا مشاهده می­کنیم که هر چه تعداد نمونه­ها افزایش یافته­اند، میزان افزایش سرعت نیز افزایش یافته­است.