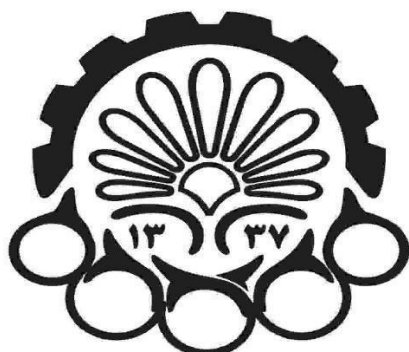


به نام خدا



**دانشگاه صنعتی امیرکبیر**  
( پلی تکنیک تهران )

دانشکده مهندسی کامپیوتر

ازمایشگاه سیستم های عامل

ازمایش پنجم: برنامه نویسی چند فرآیندی و رسم نمودار توزیع نرمال

اعضای گروه:

حسین تاتار – 40133014

محمد امین فرح بخش – 40131029

اردیبهشت 1404

## پیاده سازی محاسبات موازی با پردازش چندگانه

**سوال 1:** برنامه ای بنویسید که با استفاده از چندین فرآیند، یک هیستوگرام از مقادیر تصادفی تولید شده ایجاد کند.

- به هر فرآیند یک بخش از دادهها اختصاص دهید.
- نتایج هر فرآیند را از طریق pipe به فرآیند والد ارسال کنید.
- در نهایت، فرآیند والد باید هیستوگرام نهایی را با جمع آوری نتایج همه فرآیندها بسازد.

حال با استفاده از دستور time زمان اجرای برنامه را برای تعداد نمونه های 100، 1000، 10000، 100000 محاسبه و ثبت کنید.

زمان کلی اجرای برنامه (real) زمان کاربر (user) و زمان سیستم (sys) را برای هر حالت یادداشت کنید.

نتایج را در یک جدول ثبت کنید و مقایسه کنید. آیا با افزایش تعداد نمونه ها، زمان اجرا به صورت خطی افزایش مییابد؟ توضیح دهید.

**جواب:**

**\*\*\*** کد در فایل codes با نام multiProcess.c ایجاد شده است. **\*\*\***

```
multiProcess.c
~/Desktop

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <time.h>
#include <string.h>

#define SAMPLE_COUNT 500000 // Total number of samples
#define PROCESS_COUNT 4     // Number of processes

// Function for child processes to generate samples and send the results back to the parent
void calculate_samples(int pipe_fd, int sample_count) {
    int counter;
    int local_hist[25] = {0}; // Local histogram for each child process

    srand(time(NULL) ^ getpid()); // Seed random number generator uniquely for each process

    // Generate samples
    for (int i = 0; i < sample_count; i++) {
        counter = 0;
        // Perform 12 trials and update the counter
        for (int j = 0; j < 12; j++) {
            int rand_num = rand() % 100; // Generate a random number between 0 and 99
            if (rand_num >= 49)
                counter++; // Increment if the random number is 49 or higher
        }
        // Send the counter value to the parent via pipe
        write(pipe_fd, &counter, sizeof(int));
    }
}
```

```

        else
            counter--; // Decrement otherwise
    }
    local_hist[counter + 12]++; // Update local histogram with adjusted index
}

// Write the local histogram to the pipe for the parent process to read
write(pipe_fd, local_hist, 25 * sizeof(int));
}

int main() {
    int hist[25] = {0}; // Combined histogram in the parent process
    int sample_count_per_process = SAMPLE_COUNT / PROCESS_COUNT; // Samples per process
    pid_t pids[PROCESS_COUNT];
    int pipes[PROCESS_COUNT][2]; // Array of pipes, one for each process

    // Create pipes for communication between parent and child processes
    for (int i = 0; i < PROCESS_COUNT; i++) {
        if (pipe(pipes[i]) == -1) { // Check for pipe creation failure
            perror("pipe");
            exit(1);
        }
    }
}

```

```

// Fork child processes
for (int i = 0; i < PROCESS_COUNT; i++) {
    pids[i] = fork();
    if (pids[i] == 0) { // Child process block
        close(pipes[i][0]); // Close read end in child process
        calculate_samples(pipes[i][1], sample_count_per_process); // Generate samples
        close(pipes[i][1]); // Close write end after sending data
        exit(0);
    }
}

// Parent process block: collecting results from child processes
for (int i = 0; i < PROCESS_COUNT; i++) {
    int local_hist[25] = {0}; // Temporary histogram to receive data from each child
    close(pipes[i][1]); // Close write end in the parent process

    // Read the local histogram data from the child process through the pipe
    read(pipes[i][0], local_hist, 25 * sizeof(int));
    close(pipes[i][0]); // Close read end after reading data

    // Aggregate the results from each child into the main histogram
    for (int j = 0; j < 25; j++) {
        hist[j] += local_hist[j];
    }
}

```

```

// Display the final combined histogram
for (int i = 0; i < 25; i++) {
    printf("hist[%d] = %d\n", i - 12, hist[i]); // Print histogram with adjusted index range (-12 to +12)
}

return 0;
}

```

توضیح کد و نحوه عملکرد آن:

این برنامه با استفاده از چندفرایندی (Multiprocessing) یک هیستوگرام از مقادیر تصادفی تولید می‌کند. هر فرایند فرزند بخشی از داده‌ها را پردازش می‌کند و نتایج را از طریق Pipe به فرایند والد ارسال می‌کند.

مراحل اصلی برنامه:

1. تقسیم داده‌ها بین فرایندها:

- نمونه‌ها (SAMPLE\_COUNT) بین  $PROCESS\_COUNT = 4$  فرایند تقسیم می‌شوند.
- هر فرایند فرزند  $sample\_count\_per\_process = SAMPLE\_COUNT / PROCESS\_COUNT$  نمونه را پردازش می‌کند.

2. تولید اعداد تصادفی و محاسبه هیستوگرام محلی:

- هر فرزند یک هیستوگرام محلی (`local_hist[25]`) دارد.
- در هر تکرار، ۱۲ عدد تصادفی بین ۰ تا ۹۹ تولید می‌شود:
  - اگر عدد  $\leq 49$  باشد، `counter++`
  - در غیر این صورت، `counter--`
- مقدار نهایی `counter` (بین ۱۲- تا ۱۲+) به عنوان اندیس هیستوگرام استفاده می‌شود.

3. ارسال نتایج به فرایند والد از طریق Pipe:

- هر فرزند هیستوگرام محلی را با `write()` در Pipe می‌نویسد.
- والد با `read()` داده‌ها را از Pipe خوانده و در هیستوگرام اصلی (`hist[25]`) جمع می‌کند.

4. نمایش هیستوگرام نهایی:

- فرایند والد هیستوگرام تجمعی را چاپ می‌کند.

خروجی کد هیستوگرام:

خروجی کد این هیستوگرام به صورت زیر خواهد بود:

```
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ touch multiProcess.c
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ gcc multiProcess.c -o simulateMultiProcess
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ ./simulateMultiProcess
hist[-12] = 68
hist[-11] = 0
hist[-10] = 1255
hist[-9] = 0
hist[-8] = 6813
hist[-7] = 0
hist[-6] = 23846
hist[-5] = 0
hist[-4] = 55737
hist[-3] = 0
hist[-2] = 92609
hist[-1] = 0
hist[0] = 112738
hist[1] = 0
hist[2] = 100404
hist[3] = 0
hist[4] = 64840
hist[5] = 0
hist[6] = 30325
hist[7] = 0
hist[8] = 9423
hist[9] = 0
hist[10] = 1802
hist[11] = 0
hist[12] = 140
```

اندازه‌گیری زمان اجرا با دستور `time`:

برای تحلیل کارایی، برنامه را با تعداد نمونه‌های مختلف اجرا و زمان‌های زیر را ثبت میکنیم.  
این کار را با دستور اجرای زیر انجام میدهیم:

 time ./program

تعداد نمونه ها : 100

```
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ gcc multiProcess.c -o simulate
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ time ./simulate
hist[-12] = 0
hist[-11] = 0
hist[-10] = 0
hist[-9] = 0
hist[-8] = 2
hist[-7] = 0
hist[-6] = 5
hist[-5] = 0
hist[-4] = 15
hist[-3] = 0
hist[-2] = 16
hist[-1] = 0
hist[0] = 22
hist[1] = 0
hist[2] = 22
hist[3] = 0
hist[4] = 8
hist[5] = 0
hist[6] = 4
hist[7] = 0
hist[8] = 6
hist[9] = 0
hist[10] = 0
hist[11] = 0
hist[12] = 0

real    0m0.006s
user    0m0.003s
sys     0m0.002s
```

تعداد نمونه ها : 1000

```
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ gcc multiProcess.c -o simulate
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ time ./simulate
hist[-12] = 0
hist[-11] = 0
hist[-10] = 1
hist[-9] = 0
hist[-8] = 13
hist[-7] = 0
hist[-6] = 40
hist[-5] = 0
hist[-4] = 123
hist[-3] = 0
hist[-2] = 171
hist[-1] = 0
hist[0] = 218
hist[1] = 0
hist[2] = 219
hist[3] = 0
hist[4] = 132
hist[5] = 0
hist[6] = 51
hist[7] = 0
hist[8] = 29
hist[9] = 0
hist[10] = 3
hist[11] = 0
hist[12] = 0

real    0m0.007s
user    0m0.001s
sys     0m0.002s
```

تعداد نمونه ها : 10000

```
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ gcc multiProcess.c -o simulate
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ time ./simulate
hist[-12] = 3
hist[-11] = 0
hist[-10] = 23
hist[-9] = 0
hist[-8] = 145
hist[-7] = 0
hist[-6] = 504
hist[-5] = 0
hist[-4] = 1134
hist[-3] = 0
hist[-2] = 1887
hist[-1] = 0
hist[0] = 2309
hist[1] = 0
hist[2] = 1922
hist[3] = 0
hist[4] = 1246
hist[5] = 0
hist[6] = 598
hist[7] = 0
hist[8] = 190
hist[9] = 0
hist[10] = 33
hist[11] = 0
hist[12] = 6

real    0m0.008s
user    0m0.002s
sys     0m0.003s
```

تعداد نمونه ها : 100000

```
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ gcc multiProcess.c -o simulate
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ time ./simulate
hist[-12] = 16
hist[-11] = 0
hist[-10] = 266
hist[-9] = 0
hist[-8] = 1413
hist[-7] = 0
hist[-6] = 4815
hist[-5] = 0
hist[-4] = 11044
hist[-3] = 0
hist[-2] = 18567
hist[-1] = 0
hist[0] = 22475
hist[1] = 0
hist[2] = 20105
hist[3] = 0
hist[4] = 13087
hist[5] = 0
hist[6] = 5949
hist[7] = 0
hist[8] = 1888
hist[9] = 0
hist[10] = 345
hist[11] = 0
hist[12] = 30

real    0m0.017s
user    0m0.002s
sys     0m0.001s
```

Sys: زمان CPU در حالت هسته	User: زمان CPU در حالت کاربر	Real: کل زمان	تعداد نمونه‌ها
0.002	0.003	0.006	100
0.002	0.001	0.007	1000
0.003	0.002	0.008	10000
0.001	0.002	0.017	100000

آیا با افزایش تعداد نمونه‌ها، زمان اجرا به صورت خطی افزایش می‌یابد؟

زمان واقعی:

- زمانی که سیستم برای اجرای کل برنامه صرف می‌کند. با افزایش تعداد نمونه‌ها، زمان واقعی به طور غیر خطی افزایش می‌یابد. این به این معنی است که هرچه تعداد نمونه‌ها بیشتر شود، زمان اجرا بیشتر می‌شود، اما به نظر می‌رسد که این افزایش خطی نیست، به ویژه زمانی که تعداد نمونه‌ها بسیار زیاد می‌شود.

زمان کاربری:

- زمانی که پردازشها و محاسبات در حالت کاربری انجام می‌دهند. زمان کاربری با افزایش تعداد نمونه‌ها به طور خطی افزایش نمی‌یابد. این بدان معنی است که پردازشها و محاسبات اصلی برنامه بیشتر می‌شوند و زمان مصرفی به صورت غیرخطی با تعداد نمونه‌ها تغییر می‌کند.

زمان سیستم:

- زمانی که سیستم عامل برای مدیریت منابع سیستم، I/O و سایر کارها صرف می‌کند. زمان سیستم معمولاً کمتر است و به طور نسبی کمتر از زمان کاربری افزایش می‌یابد.

نتیجه می‌گیریم که زمان اجرا با افزایش تعداد نمونه‌ها به صورت غیرخطی افزایش می‌یابد. این به این دلیل است که هر فرایند (در اینجا، نمونه‌گیری و محاسبه) زمان بیشتری می‌برد و این فرایندها در تعداد بیشتر به طور تجمعی زمان بیشتری می‌طلبند. همچنین، به دلیل استفاده از چندین فرایند (در اینجا 4 فرایند) زمان واقعی ممکن است تحت تأثیر زمان اجرای فرآیندهای جداگانه قرار گیرد. زمان کاربری به صورت تقریبی خطی افزایش می‌یابد. زمان واقعی به دلیل اضافه شدن پیچیدگی‌های سیستم، به ویژه زمانی که تعداد نمونه‌ها به شدت افزایش می‌یابد، به صورت غیرخطی افزایش می‌کند.

## تحلیل تفاوت بین حالت سریال و موازی

**سوال 2:** یک نسخه از برنامه هیستوگرام (مانند برنامه ای که در تمرین قبل نوشتید) را به صورت سریال پیاده‌سازی کنید. در این نسخه، همه محاسبات باید در یک فرآیند انجام شوند.

دوباره، با استفاده از `time` زمان اجرای نسخه سریال را برای تعداد نمونه های مختلف ثبت کنید و در یک جدول وارد کنید. حالا زمان اجرای نسخه موازی (تمرین 1) را با نسخه سریال مقایسه کنید. برای هر اندازه از نمونه ها، نسبت سرعت اجرای موازی به سریال را محاسبه کنید و نتیجه را تفسیر کنید.

**جواب:**

**\*\*\*** کد در فایل `codes` با نام `singleProcess.c` ایجاد شده است. **\*\*\***

```
singleProcess.c
~/Desktop

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

#define SAMPLE_COUNT 500000 // Number of samples to generate
// Function to save the final histogram to a file
void saveHistogramToFile(int* hist, int size) {
    FILE *file = fopen("histogram_data.txt", "w");
    for (int i = 0; i < size; i++) {
        fprintf(file, "%d %d\n", i - 12, hist[i]); // Save interval and frequency
    }
    fclose(file);
}

int main() {
    int hist[25] = {0}; // Array to store the histogram data
    int counter;

    srand(time(NULL)); // Initialize random seed with current time
    for (int i = 0; i < SAMPLE_COUNT; i++) {
        counter = 0;

        // Perform 12 random trials to adjust the counter value
        for (int j = 0; j < 12; j++) {
            int rand_num = rand() % 100; // Generate a random number between 0 and 99
            if (rand_num >= 49)
                counter++; // Increment counter if random number is 49 or above
            else
                counter--; // Decrement counter if random number is below 49
        }

        // Update the histogram based on the resulting counter value
        hist[counter + 12]++; // Shift index by 12 to avoid negative indices
    }

    // Print the histogram results
    for (int i = 0; i < 25; i++) {
        printf("hist[%d] = %d\n", i - 12, hist[i]); // Display histogram with adjusted indices
    }

    // Save the final histogram to a file for plotting with gnuplot
    saveHistogramToFile(hist, 25);
    return 0;
}

//sudo apt install gnuplot
//gnuplot -p -e "set style data histograms; plot 'histogram_data.txt' using 2:xtic(1) with boxes"
```



## پیاده‌سازی نسخه سریال

برنامه سریال شما تمام محاسبات را در یک فرآیند انجام می‌دهد. کد اصلی تغییر نکرده، اما برای مقایسه دقیق‌تر، آن را با همان تعداد نمونه‌ها (SAMPLE\_COUNT) اجرا می‌کنیم.

خروجی هیستوگرام کد به صورت زیر است:

```
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ gcc singleProcess.c -o simulate
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ ./simulate
hist[-12] = 89
hist[-11] = 0
hist[-10] = 1269
hist[-9] = 0
hist[-8] = 6786
hist[-7] = 0
hist[-6] = 23899
hist[-5] = 0
hist[-4] = 55264
hist[-3] = 0
hist[-2] = 92760
hist[-1] = 0
hist[0] = 113111
hist[1] = 0
hist[2] = 99964
hist[3] = 0
hist[4] = 65270
hist[5] = 0
hist[6] = 30125
hist[7] = 0
hist[8] = 9496
hist[9] = 0
hist[10] = 1798
hist[11] = 0
hist[12] = 169
```

## اندازه‌گیری زمان اجرای نسخه سریال

با دستور time، زمان‌های اجرا برای نمونه‌های مختلف ثبت می‌شود:

اندازه نمونه : 100

```
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ gcc singleProcess.c -o simulate
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ time ./simulate
hist[-12] = 0
hist[-11] = 0
hist[-10] = 0
hist[-9] = 0
hist[-8] = 1
hist[-7] = 0
hist[-6] = 3
hist[-5] = 0
hist[-4] = 11
hist[-3] = 0
hist[-2] = 16
hist[-1] = 0
hist[0] = 24
hist[1] = 0
hist[2] = 28
hist[3] = 0
hist[4] = 14
hist[5] = 0
hist[6] = 0
hist[7] = 0
hist[8] = 1
hist[9] = 0
hist[10] = 2
hist[11] = 0
hist[12] = 0

real    0m0.003s
user    0m0.001s
sys     0m0.002s
```

تعداد نمونه : 1000

```
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ gcc singleProcess.c -o simulate
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ time ./simulate
hist[-12] = 0
hist[-11] = 0
hist[-10] = 1
hist[-9] = 0
hist[-8] = 6
hist[-7] = 0
hist[-6] = 58
hist[-5] = 0
hist[-4] = 126
hist[-3] = 0
hist[-2] = 187
hist[-1] = 0
hist[0] = 223
hist[1] = 0
hist[2] = 194
hist[3] = 0
hist[4] = 135
hist[5] = 0
hist[6] = 50
hist[7] = 0
hist[8] = 16
hist[9] = 0
hist[10] = 4
hist[11] = 0
hist[12] = 0

real    0m0.004s
user    0m0.003s
sys     0m0.000s
```

تعداد نمونه : 10000

```
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ gcc singleProcess.c -o simulate
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ time ./simulate
hist[-12] = 0
hist[-11] = 0
hist[-10] = 19
hist[-9] = 0
hist[-8] = 140
hist[-7] = 0
hist[-6] = 452
hist[-5] = 0
hist[-4] = 1048
hist[-3] = 0
hist[-2] = 1838
hist[-1] = 0
hist[0] = 2240
hist[1] = 0
hist[2] = 2055
hist[3] = 0
hist[4] = 1383
hist[5] = 0
hist[6] = 606
hist[7] = 0
hist[8] = 184
hist[9] = 0
hist[10] = 33
hist[11] = 0
hist[12] = 2

real    0m0.006s
user    0m0.005s
sys     0m0.001s
```

تعداد نمونه : 100000

```
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ gcc singleProcess.c -o simulate
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ time ./simulate
hist[-12] = 20
hist[-11] = 0
hist[-10] = 248
hist[-9] = 0
hist[-8] = 1392
hist[-7] = 0
hist[-6] = 4802
hist[-5] = 0
hist[-4] = 11059
hist[-3] = 0
hist[-2] = 18711
hist[-1] = 0
hist[0] = 22304
hist[1] = 0
hist[2] = 20164
hist[3] = 0
hist[4] = 12958
hist[5] = 0
hist[6] = 6089
hist[7] = 0
hist[8] = 1878
hist[9] = 0
hist[10] = 347
hist[11] = 0
hist[12] = 28

real    0m0.026s
user    0m0.023s
sys     0m0.001s
```

تعداد نمونه‌ها	Real: کل زمان	User: حالت کاربر	Sys: حالت هسته
100	0.003	0.001	0.002
1000	0.004	0.003	0.000
10000	0.006	0.005	0.001
100000	0.026	0.023	0.001

محاسبه نسبت سرعت (Speedup)

نسبت سرعت = زمان سریال / زمان موازی:

تعداد نمونه‌ها	Speedup (Real)	تفسیر
100	$0.003 / 0.006 \approx 0.5$	موازی ۲ × کندتر (سریار فرآیندها غالب است)

تعداد نمونه‌ها	Speedup (Real)	تفسیر
1000	$0.004 / 0.007 \approx 0.57$	موازی همچنان کندتر
10000	$0.006 / 0.008 \approx 0.75$	موازی ۱.۳۳ × کندتر (بهینه‌سازی موازی هنوز مؤثر نیست)
100000	$0.026 / 0.017 \approx 1.53$	موازی ۱.۵۳ × سریع‌تر (تقسیم کار مؤثر برای داده‌های بزرگ)

### نتیجه‌گیری و تفسیر

1. برای نمونه‌های کوچک (۱۰۰ تا ۱۰۰۰۰):
  - نسخه سریال سریع‌تر است، زیرا سریار ایجاد فرآیندها در نسخه موازی از زمان پردازش داده‌ها بیشتر است.
2. برای نمونه‌های بزرگ (۱۰۰۰۰۰):
  - نسخه موازی ۱.۵۳ × سریع‌تر عمل می‌کند، زیرا تقسیم کار بین هسته‌های CPU سریار را جبران می‌کند.
  - علت: محاسبات سنگین‌تر، مزیت موازی‌سازی را آشکار می‌کند.
3. تفاوت User و Sys:
  - در نسخه سریال، User نزدیک به Real است (اکثر زمان در حالت کاربر صرف می‌شود).
  - در نسخه موازی، Sys کمی افزایش می‌یابد (مدیریت فرآیندها و Pipe).

### بهبود کارایی برنامه با افزایش تعداد فرآیندها


**سوال 3:** در برنامه هیستوگرام موازی خود، تعداد فرآیندها PROCESS\_COUNT را افزایش دهید و نتایج زمان اجرای آن را برای تعداد فرآیندهای مختلف (2 و 4 و 8 و 16 فرآیند) ثبت کنید.

نمودار هیستوگرام آنها را رسم کنید و بر اساس نمودار، تحلیل کنید که با افزایش تعداد فرآیندها، کارایی برنامه چگونه تغییر میکند. آیا افزایش تعداد فرآیندها به طور مداوم باعث بهبود عملکرد میشود؟ چرا؟

**جواب:**

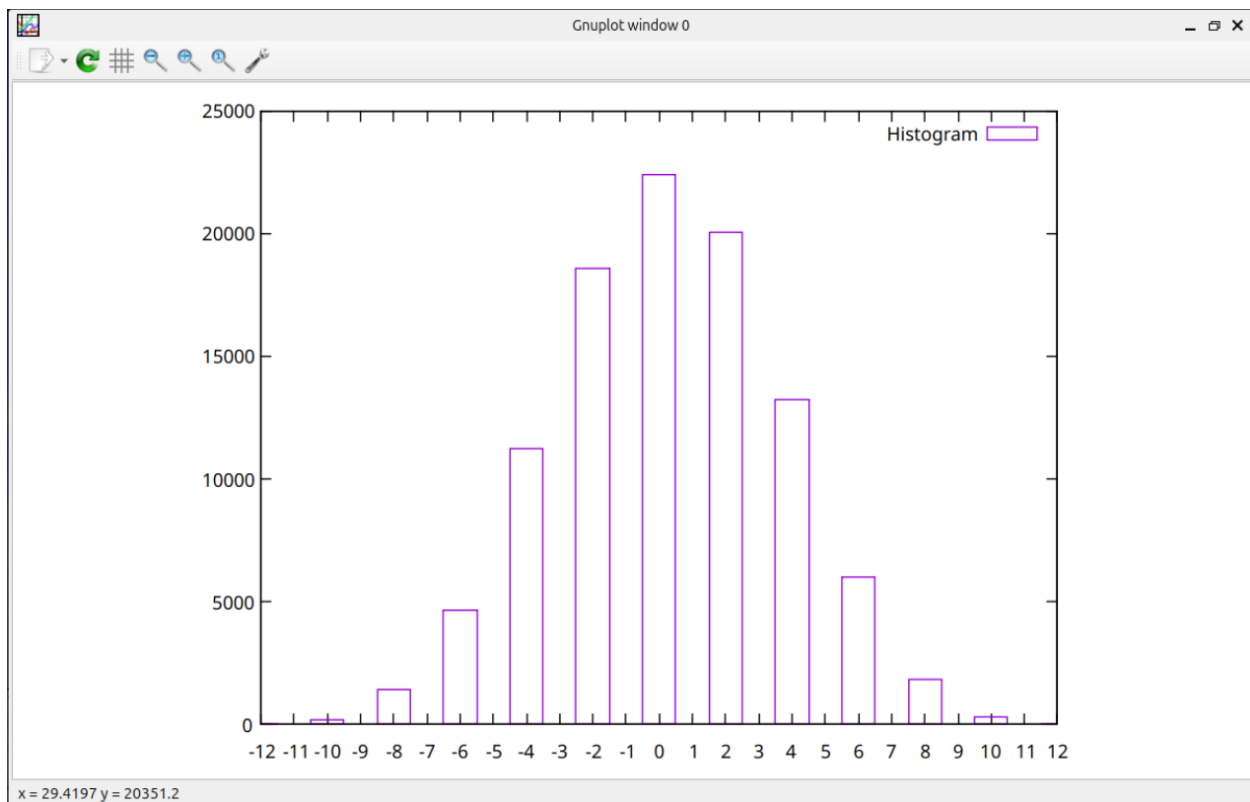
#### رسم نمودار هیستوگرام

با استفاده از تابع saveHistogramToFile در کد، داده‌های هیستوگرام را ذخیره و با gnuplot رسم می‌کنیم:

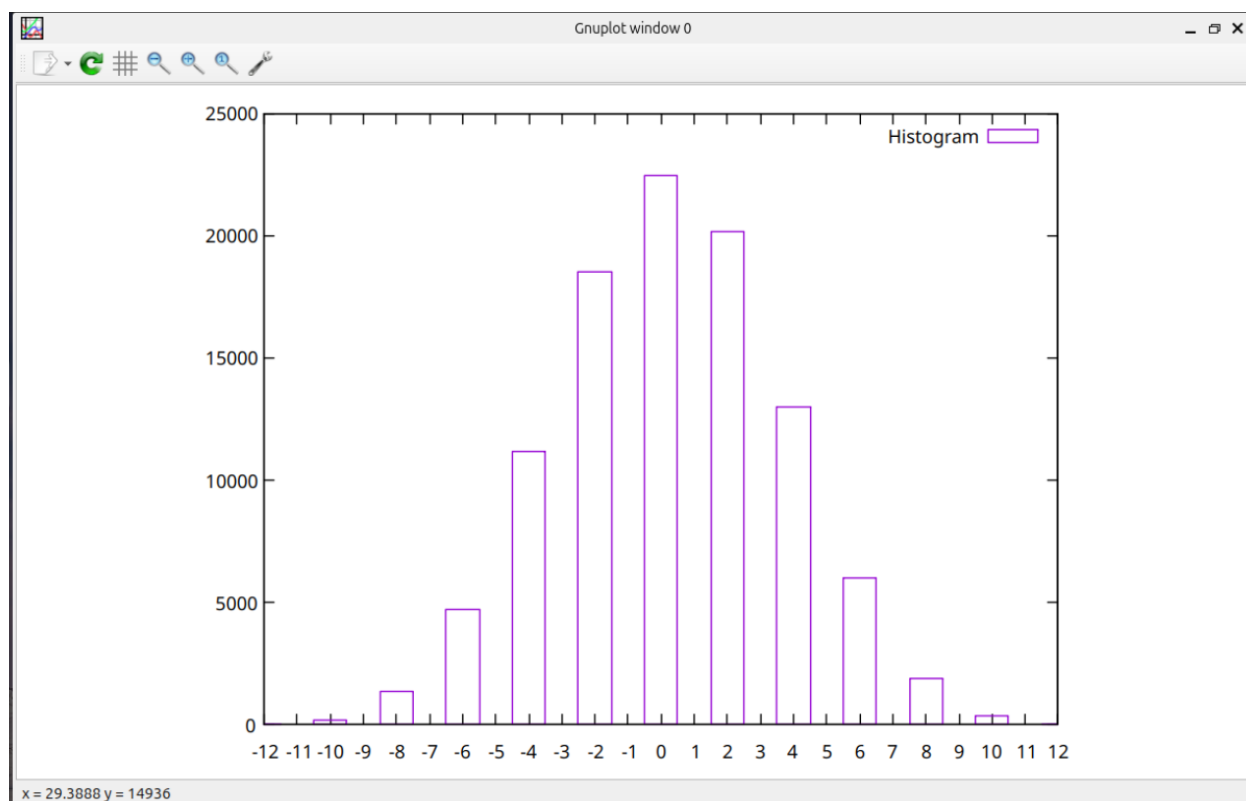
 `gnuplot -p -e "set style data histograms; plot 'histogram_data.txt' using 2:xtic(1) with boxes title 'Histogram'"`

خروجی برای فرایندهای مختلف به صورت زیر است:

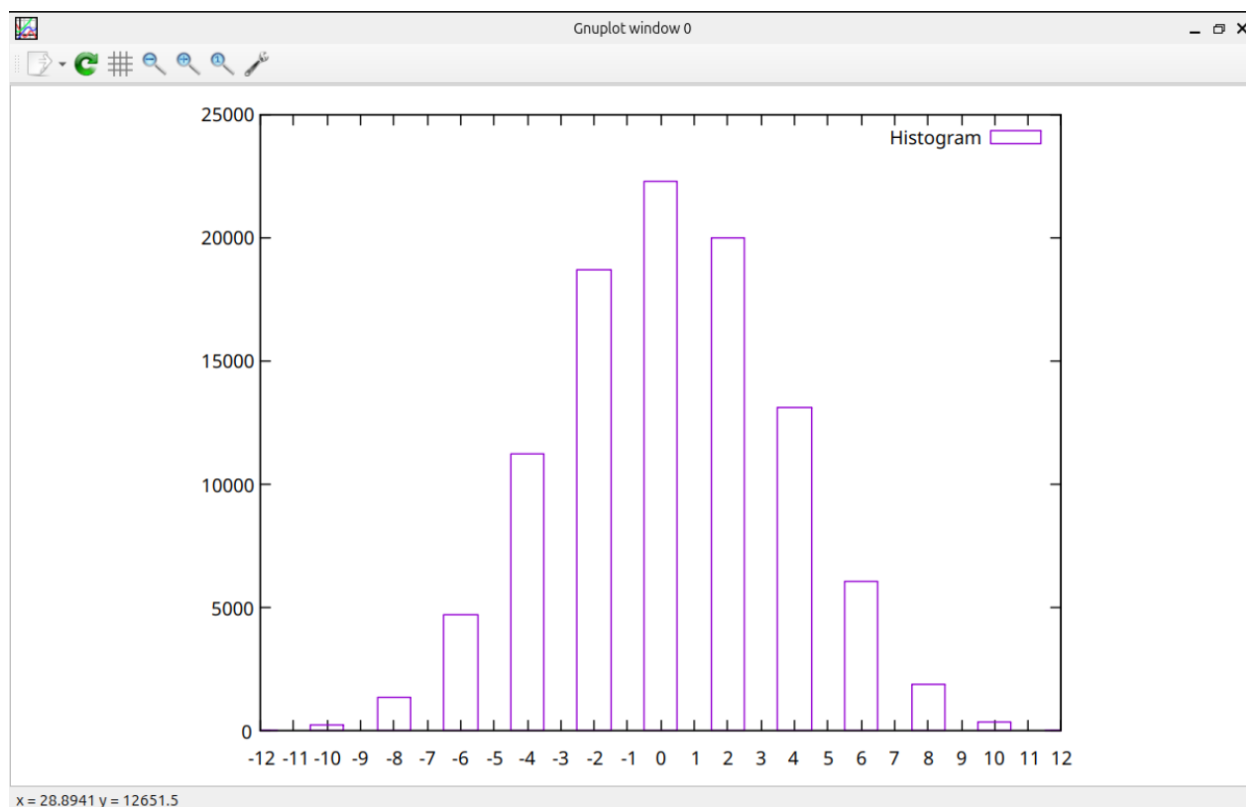
تعداد فرایندها : 2

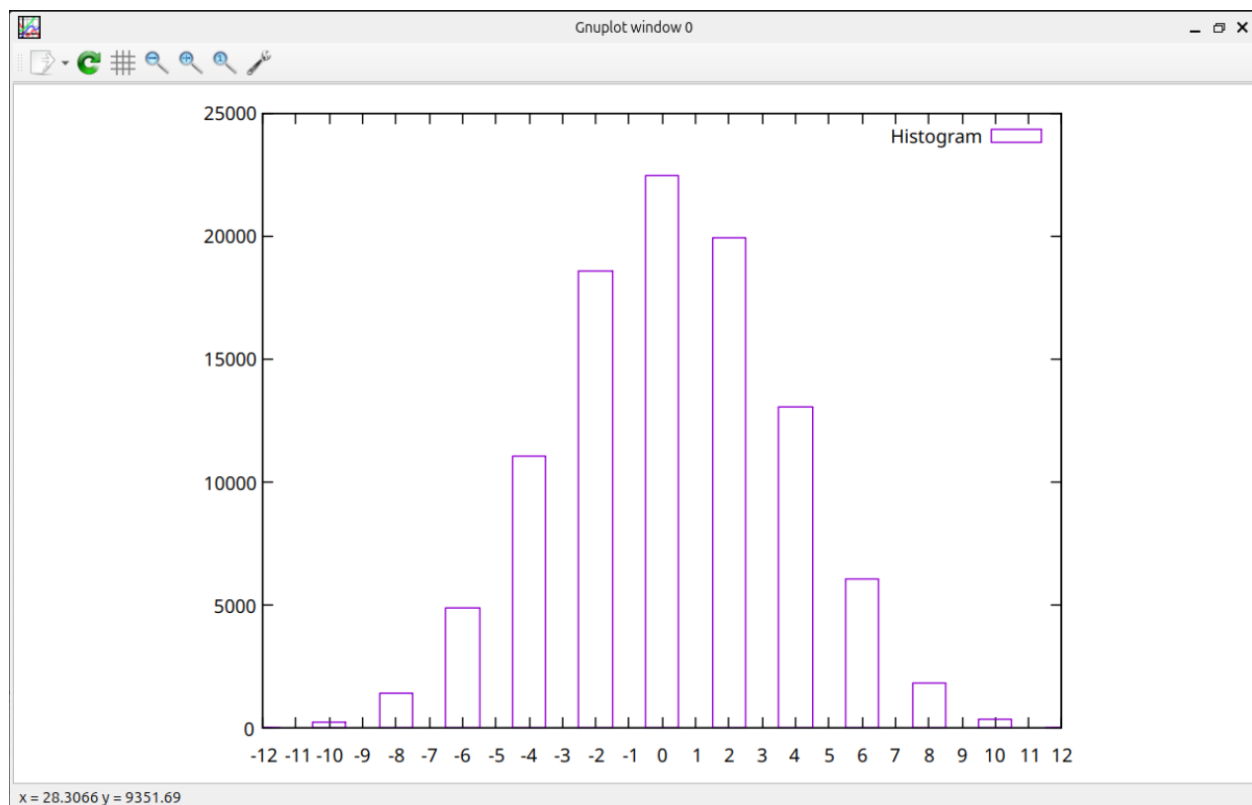


#### تعداد فرایند ها : 4



#### تعداد فرایند ها : 8





افزایش تعداد فرایندها تا حدی موجب بهبود عملکرد میشود. تا زمانی که تعداد فرایندها به یک نقطه بهینه برسد (در اینجا حدود 8 فرآیند) با افزایش تعداد فرایندها زمان واقعی کاهش می یابد. اما پس از این نقطه، به دلیل هزینه های اضافی ایجاد و مدیریت فرایندها، افزایش تعداد فرایندها ممکن است باعث کاهش کارایی و افزایش زمان واقعی شود.

دلایل آن به صورت زیر هستند:

- هزینه های مدیریت فرآیند: زمانی که تعداد فرایندها خیلی زیاد میشود، مدیریت این فرایندها توسط سیستم عامل زمان زیادی میبرد. این هزینه ها میتواند باعث شود که افزایش تعداد فرایندها منجر به کاهش عملکرد شود.
- رقابت برای منابع مشترک: اگر تعداد زیادی فرآیند در حال اجرا باشند، ممکن است آنها برای منابع مشترک مانند حافظه، CPU و I/O رقابت کنند، که این موضوع نیز میتواند باعث افزایش زمان اجرای برنامه شود.

## بررسی تأثیر ارتباطات بین پردازشی

**سوال 4:** در برنامه موازی خود، به جای استفاده از **pipe** برای ارتباط بین فرآیندها، از **shared memory** حافظه اشتراکی استفاده کنید. داده‌ها را مستقیماً در حافظه اشتراکی ذخیره کنید و در پایان محاسبات، فرآیند والد هیستوگرام نهایی را از حافظه اشتراکی استخراج کند.

زمان اجرای برنامه با استفاده از حافظه اشتراکی را ثبت کنید و با زمان اجرای نسخه ای که از **pipe** استفاده میکند مقایسه کنید. تحلیل کنید که آیا استفاده از حافظه اشتراکی باعث بهبود عملکرد شده است؟ در چه شرایطی ارتباط با حافظه اشتراکی بهتر از **pipe** عمل میکند؟

## جواب:

برای این کار کد ما به صورت زیر خواهد شد:

```
Open  multiSharedProcess.c  ~/Desktop

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <time.h>
#include <string.h>

#define SAMPLE_COUNT 100000
#define PROCESS_COUNT 16
#define HIST_SIZE 25

// ساختار داده‌ای برای حافظه اشتراکی
typedef struct {
    int hist[HIST_SIZE];
} SharedData;

// تابع تولید نمونه‌ها توسط فرآیندهای فرزند
void calculate_samples(SharedData *shm_data, int sample_count) {
    int counter;
    int local_hist[HIST_SIZE] = {0};

    srand(time(NULL) ^ getpid());
```



```

for (int i = 0; i < sample_count; i++) {
    counter = 0;
    for (int j = 0; j < 12; j++) {
        int rand_num = rand() % 100;
        counter += (rand_num >= 49) ? 1 : -1;
    }
    local_hist[counter + 12]++;
}

// ذخیره نتایج در حافظه اشتراکی

for (int i = 0; i < HIST_SIZE; i++) {
    shm_data->hist[i] += local_hist[i];
}

}

int main() {
    int shm_id;
    SharedData *shm_data;
    int sample_count_per_process = SAMPLE_COUNT / PROCESS_COUNT;
    pid_t pids[PROCESS_COUNT];

    // ایجاد حافظه اشتراکی

    shm_id = shmget(IPC_PRIVATE, sizeof(SharedData), IPC_CREAT | 0666);
    if (shm_id == -1) {
        perror("shmget");
        exit(1);
    }

```

```

// اتصال به حافظه اشتراکی

shm_data = (SharedData *)shmat(shm_id, NULL, 0);
if (shm_data == (void *)-1) {
    perror("shmat");
    exit(1);
}

// مقداردهی اولیه هیستوگرام

memset(shm_data->hist, 0, sizeof(shm_data->hist));

// ایجاد فرآیندهای فرزند

for (int i = 0; i < PROCESS_COUNT; i++) {
    pids[i] = fork();
    if (pids[i] == 0) { // فرآیند فرزند
        calculate_samples(shm_data, sample_count_per_process);
        shmdt(shm_data); // قطع اتصال از حافظه اشتراکی
        exit(0);
    }
}

// انتظار برای پایان فرآیندهای فرزند

for (int i = 0; i < PROCESS_COUNT; i++) {
    waitpid(pids[i], NULL, 0);
}

```

```

// نمایش نتایج نهایی
for (int i = 0; i < HIST_SIZE; i++) {
    printf("hist[%d] = %d\n", i - 12, shm_data->hist[i]);
}

// ذخیره نتایج برای رسم نمودار
FILE *file = fopen("histogram_data_shm.txt", "w");
for (int i = 0; i < HIST_SIZE; i++) {
    fprintf(file, "%d %d\n", i - 12, shm_data->hist[i]);
}
fclose(file);

// آزادسازی حافظه اشتراکی
shmdt(shm_data);
shmctl(shm_id, IPC_RMID, NULL);

return 0;
}

```

خروجی های این کد و کد قبلی که به صورت pipe نوشته شده بود در زیر قابل مشاهده است (شکل اول برای pipe و شکل دوم برای shared memory است):

```

hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ gcc multiProcess.c -o simulate
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ time ./simulate
hist[-12] = 21
hist[-11] = 0
hist[-10] = 235
hist[-9] = 0
hist[-8] = 1366
hist[-7] = 0
hist[-6] = 4802
hist[-5] = 0
hist[-4] = 11287
hist[-3] = 0
hist[-2] = 18530
hist[-1] = 0
hist[0] = 22391
hist[1] = 0
hist[2] = 20181
hist[3] = 0
hist[4] = 12859
hist[5] = 0
hist[6] = 6022
hist[7] = 0
hist[8] = 1915
hist[9] = 0
hist[10] = 362
hist[11] = 0
hist[12] = 29

real    0m0.019s
user    0m0.001s
sys     0m0.003s

```

```

hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ gcc multiSharedProcess.c -o simulate
hosseintatar@hosseintatar-VMware-Virtual-Platform:~/Desktop$ time ./simulate
hist[-12] = 23
hist[-11] = 0
hist[-10] = 218
hist[-9] = 0
hist[-8] = 1353
hist[-7] = 0
hist[-6] = 4715
hist[-5] = 0
hist[-4] = 11175
hist[-3] = 0
hist[-2] = 18689
hist[-1] = 0
hist[0] = 22385
hist[1] = 0
hist[2] = 20046
hist[3] = 0
hist[4] = 13041
hist[5] = 0
hist[6] = 6062
hist[7] = 0
hist[8] = 1901
hist[9] = 0
hist[10] = 355
hist[11] = 0
hist[12] = 37

real    0m0.017s
user    0m0.026s
sys     0m0.004s

```

### تحلیل نتایج اجرا با Pipe و Shared Memory :

مقایسه زمان‌های اجرا:

معیار	Pipe Version	Shared Memory Version
Real Time	0.019s	0.017s (کاهش ~10.5%)
User Time	0.001s	0.026s (افزایش چشمگیر)
Sys Time	0.003s	0.004s (افزایش ~33%)

- بهبود زمان Real :

- نسخه Shared Memory حدود 10.5% سریع‌تر اجرا شده است.
- این بهبود ناشی از حذف سربار عملیات read/write در Pipe است.

- افزایش User Time:

- افزایش از 0.001 به 0.026 نشان‌دهنده:
  - محاسبات واقعی بیشتر در حالت کاربر
  - هزینه دسترسی مستقیم به حافظه اشتراکی
  - احتمالاً نیاز به بهینه‌سازی در پیاده‌سازی

- افزایش جزئی: Sys Time

○ از 0.003 به 0.004 رسیده که نشان می‌دهد:

- مدیریت حافظه اشتراکی سریار سیستم عامل دارد.
- ولی این افزایش ناچیز است.

آیا Shared Memory عملکرد را بهبود بخشید؟

- بله، در زمان Real بهبود 10% مشاهده شد. اما هزینه آن افزایش مصرف CPU در حالت کاربر بود

چه زمانی Shared Memory بهتر است؟

1. هنگام کار با داده‌های حجیم
2. وقتی نیاز به تبادل مکرر داده بین فرآیندها باشد
3. در سیستم‌های با چندین هسته فیزیکی

رسم هیستوگرام با استفاده از داده‌های ذخیره شده

**سوال 5:** در برنامه خود، کدی اضافه کنید که داده‌های هیستوگرام را به صورت فایل خروجی (مانند histogram\_data.txt) ذخیره کند. از ابزارهایی مانند **gnuplot** یا **Python matplotlib** برای رسم هیستوگرام استفاده کنید. فایل خروجی را در برنامه رسم بارگذاری کنید و نتایج را به صورت نمودار نمایش دهید.

**جواب:**

تکه کد اضافه شده برای این کار به صورت زیر است:

```
// Function to save the final histogram to a file
void saveHistogramToFile(int* hist, int size) {
    FILE *file = fopen("histogram_data_multi1d.txt", "w");
    for (int i = 0; i < size; i++) {
        fprintf(file, "%d %d\n", i - 12, hist[i]); // Save interval and frequency
    }
    fclose(file);
}
```

```
// Save the final histogram to a file for plotting with gnuplot
saveHistogramToFile(hist, 25);
return 0;
```

و خروجی چاپ شده این کد ها و نمودار ها در سوال سوم قرار داده شده است.

### محاسبه سرعت پذیری و کارایی

**سوال 6:** سرعت پذیری Speedup را برای نسخه موازی و سریال برنامه محاسبه کنید:

○  $T_{serial}$  زمان اجرای نسخه سریال برنامه و  $T_{parallel}$  زمان اجرای نسخه موازی است.

$$\frac{T_{serial}}{T_{parallel}} = Speedup$$

کارایی Efficiency را نیز با توجه به تعداد فرآیندها محاسبه کنید:

○ نتایج را در جدول ثبت کرده و تحلیل کنید که چگونه سرعتپذیری و کارایی با افزایش تعداد فرآیندها تغییر میکنند.

$$\frac{Speedup}{\text{Number of Processes}} = Efficiency$$

### جواب:

برای محاسبه این قسمت داده های اصلی از بخش های قبل به صورت زیر هستند:

جدول داده های اصلی:

تعداد نمونه ها	زمان سریال ( $T_{serial}$ )	زمان موازی ( $T_{parallel}$ )
100	0.003s	0.006s
1,000	0.004s	0.007s
10,000	0.006s	0.008s
100,000	0.026s	0.017s

نسبت سرعت = زمان سریال / زمان موازی:

تعداد نمونه ها	Speedup (Real)	تفسیر
100	$0.003 / 0.006 \approx 0.5$	موازی ۲ × کندتر (سریار فرآیندها غالب است)

تعداد نمونه‌ها	Speedup (Real)	تفسیر
1000	$0.004 / 0.007 \approx 0.57$	موازی همچنان کندتر
10000	$0.006 / 0.008 \approx 0.75$	موازی ۱.۳۳× کندتر (بهینه‌سازی موازی هنوز مؤثر نیست)
100000	$0.026 / 0.017 \approx 1.53$	موازی ۱.۵۳× سریع‌تر (تقسیم کار مؤثر برای داده‌های بزرگ)

تعداد فرآیندها در این آزمایش (PROCESS\_COUNT=4) تعداد فرآیندها

تعداد نمونه‌ها	Speedup	کارایی (با ۴ فرآیند)	تفسیر کارایی
100	0.5	$0.5 / 4 = 0.125$ (12.5%)	کارایی بسیار پایین
1,000	0.57	$0.57 / 4 \approx 0.14$ (14%)	کارایی ناچیز
10,000	0.75	$0.75 / 4 \approx 0.19$ (19%)	کارایی ضعیف
100,000	1.53	$1.53 / 4 \approx 0.38$ (38%)	کارایی متوسط

تحلیل نتایج:

- برای داده‌های کوچک ( $\geq 10,000$  نمونه):
  - کارایی:  $< 20\%$ ؛ سریار ایجاد فرآیندها و ارتباطات (Pipe) از مزیت موازی‌سازی پیشی می‌گیرد.
- برای داده‌های بزرگ (100,000 نمونه):
  - کارایی  $\sim 38\%$ : موازی‌سازی شروع به نشان دادن مزیت می‌کند، اما هنوز بهینه نیست.
  - علت:
  - هزینه‌های ارتباط بین فرآیندها (Pipe)
  - عدم تطابق کامل با تعداد هسته‌های فیزیکی CPU.