آزمایش 5

فاطمه راق 40013045
حامد فلاحی 9931043

خلاصه

در این آزمایشگاه، هدف اصلی بررسی مفاهیم برنامه‌نویسی همروند و ارتباط بین فرآیندهای متقابل (IPC) در سیستم‌عامل است. آزمایشگاه شامل پیاده‌سازی برنامه‌های همروند با استفاده از مکانیسم‌های fork و exec در زبان C است، با هدف دستیابی به چندوظیفگی کارآمد از طریق ایجاد چندین فرآیند. علاوه بر این، آزمایشگاه به بررسی استفاده از IPC، به‌ویژه حافظه مشترک، برای تسهیل ارتباط بین این فرآیندهای همروند می‌پردازد.

تمرین ۱:
به منظور انجام این آزمایش یک آرایه به نام hist ایجاد کردیم تا هیستوگرام را ذخیره کند. این آرایه دارای 25 باکس متناظر با مقادیر ممکن شمارنده (-12 تا +12) می‌باشد.از تابع ()randبرای تولید اعداد تصادفی میان 0 تا 100 استفاده کردیم.
اجرای آزمایش:
حلقه بیرونی آزمایش را برای تعداد مختلف نمونه‌ها اجرا می‌کند.
.حلقه داخلی آزمایش را برای هر نمونه شبیه‌سازی می‌کند
برای هر نمونه، 12 عدد تصادفی تولید می‌شود و شمارنده بر اساس شرط (49 =>) به‌روزرسانی می‌شود.
به‌روزرسانی هیستوگرام:
هیستوگرام (hist) بر اساس مقدار نهایی شمارنده پس از تولید 12 عدد تصادفی به‌روزرسانی می‌شود.
نتیجه آزمایش:

```
Serialized - Number of samples: 500     Execution time: 0.000096 seconds
Serialized - Number of samples: 5000    Execution time: 0.000913 seconds
Serialized - Number of samples: 50000   Execution time: 0.009480 seconds
```

کد آزمایش به همراه نتیجه:

Users > fatemehragh > C lab5.c > main()

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

int main() {
    // Array 'hist' with 25 elements
    int hist[25] = {0};

    int counter, random_num, i, j;
    int num_samples[] = {500, 5000, 50000};

    // Solve the problem for each number of samples
    for (j = 0; j < 3; j++) {
        int num_samples_current = num_samples[j];

        // Start measuring execution time
        clock_t start_time = clock();

        // Solve the problem for the current number of samples
        for (i = 0; i < num_samples_current; i++) {
            counter = 0;

            for (int k = 0; k < 12; k++) {
                random_num = rand() % 101;

                if (random_num >= 49) {
                    counter++;
                }
            }

            // Increase the corresponding value in 'hist'
            hist[counter + 12]++;
        }

        // Stop measuring execution time
        clock_t end_time = clock();

        // Calculate the execution time in seconds
        double execution_time = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;

        printf("Serialized - Number of samples: %d\tExecution time: %f seconds\n", num_samples_current, execution_time);
    }

    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Loaded '/usr/lib/system/libsystem_symptoms.dylib'. Symbols loaded.
Loaded '/usr/lib/system/libsystem_trace.dylib'. Symbols loaded.
Loaded '/usr/lib/system/libunwind.dylib'. Symbols loaded.
Loaded '/usr/lib/system/libxpc.dylib'. Symbols loaded.
Loaded '/usr/lib/libc++abi.dylib'. Symbols loaded.
Loaded '/usr/lib/libobjc.A.dylib'. Symbols loaded.
Loaded '/usr/lib/liboah.dylib'. Symbols loaded.
Loaded '/usr/lib/libc++.1.dylib'. Symbols loaded.
Serialized - Number of samples: 500     Execution time: 0.000096 seconds
Serialized - Number of samples: 5000    Execution time: 0.000913 seconds
Serialized - Number of samples: 50000   Execution time: 0.009480 seconds
```

تمرین ۲:

در این برنامه‌ی ، یک نسخه concurrent از آزمایش را با استفاده از مکانیسم fork پیاده‌سازی کردیم. این آزمایش شامل شبیه‌سازی تولید اعداد تصادفی و تجزیه و تحلیل توزیع آن‌هاست. کد از حافظه مشترک برای ذخیره نتایج در آرایه 'hist' استفاده می‌کند که امکان ارتباط بین فرآیندهای والد و فرزند را فراهم می کند

1** .تنظیم حافظه مشترک**:

- از `shmget` برای تخصیص حافظه مشترک برای آرایه 'hist' استفاده شد.
- حافظه مشترک به فرآیند با استفاده از `shmat` متصل شد.

2** .شروع فرآیندها**:

- از `fork` برای ایجاد یک فرآیند فرزند استفاده شد.
- در فرآیند فرزند، همان آزمایش آماری که در نسخه سریال انجام شده بود، انجام شد اما آرایه 'hist' مشترک به‌روزرسانی شد.

3 **IPC و اشتراک داده**:
- از حافظه مشترک ('hist') آرایه) برای اشتراک داده بین فرآیندهای والد و فرزند استفاده شد.
- فرآیند فرزند پس از انجام وظایف خود از حافظه مشترک با `shmdt` جدا شد.

4** .اندازه‌گیری زمان اجرا**:
- زمان اجرای آزمایش همروند با استفاده از `clock()` قبل و بعد از fork اندازه‌گیری شد.
- زمان اجرا برای هر اندازه نمونه در فرآیند والد چاپ شد.

5** .پاکسازی**:
- پس از اتمام آزمایش، منابع حافظه مشترک با `shmctl` پاکسازی شد.

این کد نحوه استفاده از fork و حافظه مشترک را برای همروندسازی آزمایش را نشان می‌دهد و اجازه می‌دهد تا چندین فرآیند به صورت همزمان کار کنند. هدف از این کار، مقایسه عملکرد نسخه concurrent با نسخه سریالی است، به‌ویژه از نظر زمان اجرا برای اندازه‌های مختلف نمونه‌ها.

```
Using Fork & IPC - Number of samples: 500       Execution time: 0.000202 seconds
Using Fork & IPC - Number of samples: 5000      Execution time: 0.000125 seconds
Using Fork & IPC - Number of samples: 50000     Execution time: 0.000114 seconds
```

```
// Concurrent version
printf("\nConcurrent Version (Using Fork & IPC):\n");
for (j = 0; j < 3; j++) {
    int num_samples_current = num_samples[j];

    // Start measuring execution time
    clock_t start_time = clock();

    // Fork processes
    pid_t pid = fork();

    if (pid == 0) {  // Child process
        for (i = 0; i < num_samples_current; i++) {
            counter = 0;

            for (int k = 0; k < 12; k++) {
                random_num = rand() % 101;

                if (random_num >= 49) {
                    counter++;
```

```
            }
        }

        // Increase the corresponding value in 'hist'
        hist_concurrent[counter + 12]++;
    }

    // Detach shared memory
    shmdt(hist_concurrent);
    exit(0);
} else if (pid > 0) {  // Parent process
    wait(NULL);
} else {  // Fork failed
    fprintf(stderr, "Fork failed.\n");
    return 1;
}

// Stop measuring execution time
clock_t end_time = clock();

// Calculate the execution time in seconds
double execution_time = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;

printf("Number of samples: %d\tExecution time: %f seconds\n", num_samples_current, execution_time);
}
```

تمرین ۳:

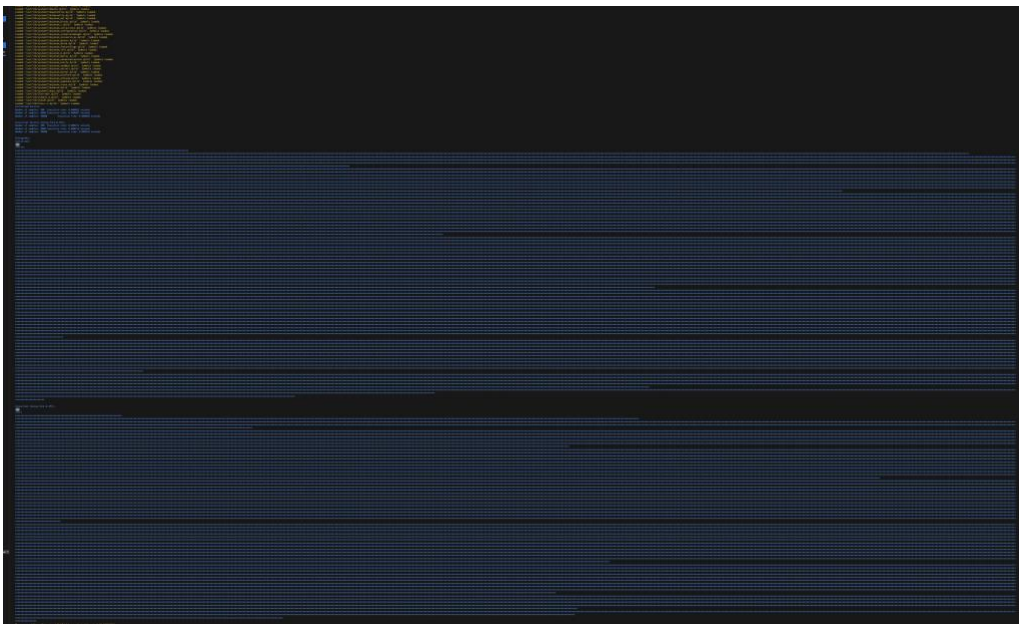
بله، برنامه ارائه شده ممکن است با شرایط مسابقه مواجه شود. شرایط مسابقه در هنگامی رخ می‌دهد که چندین فرآیند یا نخ به صورت همزمان به داده‌های مشترک دسترسی پیدا کنند و نتیجه نهایی به ترتیب اجرا بستگی داشته باشد. در بخش همزمان از کد شما، هر دو فرآیند پدر و فرزند به صورت همزمان آرایه hist_concurrent را به‌روز می‌کنند که می‌تواند به نتایج پیش‌بینی‌ناپذیر منجر شود.

برای حل شرایط مسابقه، باید اطمینان حاصل کرد که تنها یک فرآیند (یا پدر یا فرزند) در هر زمان به داده‌های مشترک دسترسی پیدا کند. یک راه برای این کار استفاده از مکانیسم‌های هماهنگی مانند سمافور یا میوتکس است.

```
        // Use semaphore to protect the critical section
        sem_wait(sem);
        hist_concurrent[counter + 12]++;
        sem_post(sem);
```

تمرین ۴:

نتیجه ترسیم هیستوگرام:



مقایسه نتیجه در حالت serilaized, concurrent:

۵۰۰	۵۰۰۰	۵۰۰۰۰
۰.۰۰۰۱۰۶	-۰.۰۰۰۷۸	-۰.۰۰۹۳۶۶

همانطور که در جدول مقایسه دیده شد برای ۵۰۰ نمونه این آزمایش در حالت سریالی سرعت بیستری داست اما برای نمونه های با سایز ۵۰۰۰ سرعت فرآیند همروند اندکی بیشتر است اما برای نمونه با سایز ۵۰۰۰ این اختلاف بسیار بیستر شد در نتیجه اگر سایز نمونه بسیار زیاد باشد استفاده از روش های concurrent بهینه تر است.