به نام خدا

دانشگاه شهید بهشتی - رشته مهندسی کامپیوتر

درس آزمایشگاه سیستم عامل
استاد آهوز

دستور کار شماره یک :

پیاده سازی مسئله تولید کننده-مصرف کننده
با دو روش و مقایسه آنها

اعضای گروه :

متین ارجمند **امیرحسین جعفرزاده**
محدثه صفاری **علی نصرالله پور**

فهرست :

مقدمه :

این پروژه به پیاده سازی و مقایسه دو مدل مختلف برای حل مسئله کلاسیک «تولیدکننده-مصرف‌کننده» می‌پردازد.

- روش مبتنی بر فرایندها که با استفاده از سیستم فراخوان fork و کانال ارتباطی pipe انجام شده است.
- روش مبتنی بر نخ‌ها که با استفاده از pthreads ، حافظه مشترک (buffer) و ابزارهای همگام‌سازی mutex و condition variable پیاده‌سازی شده است.

هدف این گزارش درک تفاوت اشتراک حافظه در فرایند ها و نخ ها و بررسی و مقایسه کارایی دو روش از منظر زمان اجرا و مصرف حافظه است.

پیاده سازی مبتنی بر فرایند ها (Process-Based) :

طبق تصویر ۱.۱ یک فرایند والد به عنوان تولیدکننده (producer)، اعداد صحیح را تولید می‌کند و با استفاده از pipe که کانال یکطرفه‌ای است، این داده‌ها را به فرایند فرزند (مصرف‌کننده) ارسال می‌کند. طبق تصویر ۱.۲ مصرف‌کننده (consumer) این اعداد را می‌خواند و مجموع آنها را محاسبه می‌نماید (تصویر ۱.۳). ارتباط بین فرایندها با عملیات write و read روی pipe انجام می‌شود که مستلزم انتقال داده بین فضای کاربر و فضای کرنل است.

- استفاده از ()fork برای ایجاد فرایند فرزند
- ارتباط از طریق pipe به عنوان کانال یک طرفه
- استفاده از ()wait برای هماهنگی اجرا

:producer

```c
if (pid > 0) {
    // producer
    printf("producer started\n");
    close(pipefd[0]); // close reading
    for (int i = 0; i < NUMS; i++) {
        write(pipefd[1], &i, sizeof(i));
        printf("produced %d\n", i);
    }
    close(pipefd[1]);
    wait(NULL);
```

(۱.۱)

:Consumer

```c
else {    // pid = 0
  // consumer
  printf("consumer started\n");
  close(pipefd[1]); // close writing

  int value;
  while (read(pipefd[0], &value, sizeof(value)) > 0) {
      sum += value;
      printf("consumed %d\n", value);
  }
  printf("consumer result: sum = %d\n", sum);
  close(pipefd[0]);
  exit(0);
```

(۱.۲)

خروجی process.c:

```
consumed 999996
consumed 999997
consumed 999998
consumed 999999
consumer result: sum = 1783293664
```

(۱.۳)

پیاده سازی مبتنی بر نخ ها (Thread-Based) :

در این روش، دو نخ در یک فرایند واحد با استفاده از pthread_create ایجاد و اجرا می‌شوند.

یک نخ تولیدکننده (producer) وظیفه تولید داده‌ها را دارد (تصویر ۲.۱).

طبق تصویر ۲.۲ نخ دیگر مصرف‌کننده (consumer) داده‌ها را از بافر حلقوی مشترک دریافت و پردازش کرده و مجموع آنها را محاسبه می‌نماید (تصویر ۲.۳).

برای مدیریت همزمانی و جلوگیری از شرایط رقابتی (race condition)، از mutex و دو condition variable به نام cond_full و cond_empty استفاده شده است تا دسترسی به بافر کنترل شود و نخ‌ها به طور بهینه منتظر بمانند.

mutex : برای تضمین اینکه در هر لحظه فقط یک نخ به بافر دسترسی دارد.

cond_empty : برای اطلاع از خالی بودن بافر استفاده می‌شود. تولیدکننده روی آن wait را صدا می‌زند و هرگاه بافر خالی شود، مصرف کننده آن را signal می‌کند.

cond_full : برای اطلاع از پر بودن بافر استفاده می‌شود. مصرف کننده روی آن wait را صدا می‌زند و هرگاه بافر پر شود، تولید کننده آن را signal می‌کند.

- استفاده از pthread برای ایجاد نخ های همزمان
- بافر حلقوی (Circular Buffer) مشترک با اندازه ثابت
- همگام سازی با mutex و condition variables

Producer:

```c
void* producer(void* arg) {
    printf("producer started\n");
    for (int i = 0; i < NUMS; i++) {
        pthread_mutex_lock(&mutex);
        while (count == BUFFER_SIZE) {
            // wait for empty
            pthread_cond_wait(&cond_empty, &mutex);
        }
        buffer[count] = i;
        printf("produced %d\n", i);
        count++;
        pthread_cond_signal(&cond_full);
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

(۲.۱)

Consumer:

```c
void* consumer(void* arg) {
    printf("consumer started\n");
    int sum = 0;
    int consumed_value;
    for (int i = 0; i < NUMS; i++) {
        pthread_mutex_lock(&mutex);
        while (count == 0) {
            // wait for full
            pthread_cond_wait(&cond_full, &mutex);
        }
        consumed_value = buffer[count - 1];
        count--;
        printf("consumed %d\n", consumed_value);
        sum += consumed_value;
        pthread_cond_signal(&cond_empty);
        pthread_mutex_unlock(&mutex);
    }

    printf("consumer result: sum = %d\n", sum);
    return NULL;
}
```

(۲.۲)

خروجی thread.c:

```
consumed 982184
consumed 982183
consumed 982182
consumed 982181
consumer result: sum = 1783293664
```

(۲.۳)

گزارش نتایج :

برای اندازه گیری عملکرد دو مدل پیاده سازی، از دستور usr/bin/time –v در لینوکس استفاده شد تا اطلاعاتی در مورد User Time، System Time، Total CPU Time ، Elapsed Time و Memory Usage بدست آید.

نتیجه process.c :

```
results of process :
        Command being timed: "./process"
        User time (seconds): 0.03
        System time (seconds): 0.39
        Percent of CPU this job got: 173%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.24
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 1248
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 83
        Voluntary context switches: 191
        Involuntary context switches: 113
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```

(۳.۱)

نتیجه thread.c :

```
results of thread :
        Command being timed: "./thread"
        User time (seconds): 0.02
        System time (seconds): 0.04
        Percent of CPU this job got: 103%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.06
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 1608
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 90
        Voluntary context switches: 184
        Involuntary context switches: 59
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```

(۳.۲)

بررسی و تحلیل نتایج :

توضیح پارامتر ها:

- User Time: مدت زمانی که CPU در حالت کاربر بوده است.
- System Time: مدت زمانی که CPU در حالت کرنل بوده است.
- Total CPU Time: کل زمانی که CPU صرف کرده را نشان می‌دهد.
- Elapsed Time: زمان واقعی صرف شده برای اجرای برنامه است.

➢ User Time

- پیاده سازی مبتنی بر فرایند : ۰/۰۳ ثانیه
- پیاده سازی مبتنی بر نخ : ۰/۰۲ ثانیه

❖ تحلیل : نزدیک به هم، زیرا در هر دو روش منطق محاسباتی یکسان است.

➢ System Time

- پیاده سازی مبتنی بر فرایند : ۰/۳۹ ثانیه
- پیاده سازی مبتنی بر نخ : ۰/۰٤ ثانیه

❖ تحلیل : تفاوت زیاد، نشان دهنده هزینه بالای عملیات سیستم عامل در مدل فرایند.
زیرا ارتباط بین دو فرایند نیازمند دخالت مستقیم کرنل است. هربار که producer داده ای را در pipe می‌نویسد، آن داده از فضای کاربر به فضای کرنل کپی می‌شود. و وقتی که consumer داده را می‌خواند، داده از فضای کرنل به فضای کاربر فرایند فرزند کپی می‌شود. این عملیات سربار بالایی دارد و به همین دلیل System Time در مدل فرایند بسیار بالا است.
در مدل نخ، نخ ها در یک فضای حافظه مشترک در فضای کاربر اجرا می‌شوند و داده ها هرگز فضای کاربر را ترک نمی‌کنند فقط برای عملیات سبک همگام سازی ارتباط با کرنل نیاز است.

- Total CPU Time
  - پیاده سازی مبتنی بر فرایند : ۰/۴۲ ثانیه
  - پیاده سازی مبتنی بر نخ : ۰/۰۶ ثانیه
  - ❖ تحلیل : طبق پیش بینی مجموع زمان CPU در مدل فرایند بیشتر است. زیرا Total CPU Time = User Time + System Time

- Elapsed Time
  - پیاده سازی مبتنی بر فرایند : ۰/۲۴ ثانیه
  - پیاده سازی مبتنی بر نخ : ۰/۰۶ ثانیه
  - ❖ تحلیل : در این مثال، پیاده سازی با فرایند ٤ برابر سریع تر از پیاده سازی با فرایند است.

- Memory Usage (بر اساس Maximum resident set size در تصاویر)
  - ❖ تحلیل : طبق خروجی، مدل مبتنی بر نخ حافظه بیشتری از مدل مبتنی بر فرایند مصرف کرده است.
    در مدل نخ، هنگام ایجاد یک نخ جدید، سیستم عامل یک stack مجزا با اندازه پیش فرض بزرگ به آن تخصیص می‌دهد.
    اما در مدل فرایند، فرایند فرزند، بهینه سازی انجام داده و تنها در صورت نیاز، حافظه را کپی می‌کند.
    در این آزمایش به دلیل پیچیده نبودن برنامه ها حافظه مصرفی مدل نخ بیشتر شده اما در برنامه های پیچیده و بزرگ، نخ ها به دلیل اشتراک حافظه، در مصرف حافظه بهینه تر عمل می‌کنند.

تابع ()execlp (که در این پروژه مورد نیاز نبود) :

تابع execlp یکی از توابع خانواده‌ی exec در زبان C است که در سیستم‌عامل‌های شبه یونیکس (مثل Linux و macOS) برای اجرای یک برنامه‌ی جدید درون فرایند فعلی استفاده می‌شود.

وقتی یک فرایند تابع execlp را فراخوانی می‌کند:

- کد فعلی فرایند جایگزین با برنامه‌ی جدید می‌شود.
- حافظه، stack، heap و بخش‌های کد فعلی پاک می‌شوند.
- فقط PID (شناسه فرایند) باقی می‌ماند.

یعنی بعد از اجرای موفق execlp، دستور بعدی در برنامه‌ی فعلی هرگز اجرا نمی‌شود.

شکل کلی تابع:

```
int execlp(const char *file, const char *arg, ..., (char *)NULL);
```

پارامترها:

۱. file : نام برنامه یا مسیر اجرایی که باید اجرا شود (می‌تواند فقط نام باشد چون execlp در PATH جستجو می‌کند).

۲. arg : آرگومان‌های ورودی به برنامه‌ی جدید.

۳. آخرین آرگومان باید NULL باشد تا انتهای آرگومان‌ها مشخص شود.

نتیجه گیری :

در این پروژه، مسئله تولیدکننده-مصرف‌کننده با دو روش مختلف شامل فرایندهای مستقل و نخ‌های همزمان در سیستم عامل لینوکس مورد پیاده‌سازی و بررسی قرار گرفت. ارزیابی هر دو مدل بر اساس سنجه‌های زمان کاربر، زمان سیستم، مجموع زمان CPU ، زمان کل اجرا و میزان مصرف حافظه صورت گرفت.

نتایج آزمایش‌ها نشان داد که روش مبتنی بر نخ به‌واسطه اشتراک کامل فضای حافظه بین نخ‌ها و کم شدن عملیات سیستمی، زمان سیستم بسیار کمتری نسبت به روش فرایند دارد. در مدل فرایند، هر داده برای انتقال باید از فضای کاربر به کرنل کپی شود و بار سیستم عامل بیشتر است، در حالی که نخ‌ها با هم در حافظه مشترک فعالیت می‌کنند و فقط برای همگام‌سازی نیاز به سیستم عامل دارند. همچنین مجموع زمان CPU و زمان اجرا مدل نخ‌ها تا چند برابر کمتر از مدل فرایند است که نشان‌دهنده کارایی بالاتر است.

در زمینه مصرف حافظه مشاهده شد که نخ‌ها به علت تخصیص stack مجزا به هر نخ، در کارهای ساده حافظه بیشتری می‌گیرند، اما در برنامه‌های پیچیده و بزرگ به دلیل اشتراک داده‌ها عملکرد بهینه‌تری خواهند داشت.

در پایان می‌توان نتیجه گرفت برای مسائل هماهنگی و پردازش موازی که نیاز به اشتراک داده و سرعت بالا دارند، استفاده از نخ‌ها بسیار کارآمدتر و بهینه‌تر از فرایندهاست.