Homework 3- Solution

Lectures 9, 10

Operating Systems

Dr. Javadi

Spring 2023

۱- با فرض اینکه pid های واقعی پردازه پدر و فرزند به ترتیب ۲۶۰۰ و ۲۶۰۳ باشد، خروجی خط های A,B,C,D برنامه زیر را با ذکر دلیل عنوان کنید.

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
  pid_t pid, pid1;

    /* fork a child process */

    if (pid <0) { /* error occured */
      fprintf(stderr, "Fork Failed");
    }
  else if (pid ==0) { /* child process */
    pid1 = getpid();
    printf("child: pid = %d", pid); /* A */
    print("child: pid1 = %d", pid1); /* B */
  }
  else { /* parent process */
    pid1 = getpid();
    printf("parent: pid = %d", pid); /* C */
    printf("parent: pid1 = %d", pid1); /* D */
    wait(NULL);
  }

  return 0;
}
```

میدانیم که مقدار خروجی که تابع fork بر می گرداند، در پردازه پدر برابر pid اصلی پردازه فرزند و در پردازه فرزند برابر صفر است. به همین دلیل در خط A که درون پردازه فرزند قرار دارد، مقدار صفر چاپ می شود و در خط B مقدار pid اصلی آن یعنی ۲۶۰۳ چاپ می شود. در خط C که درون پردازه پدر است، مقدار pid برابر pid اصلی فرزند یعنی ۲۶۰۳ چاپ می شود و در خط D مقدار pid اصلی پردازه پدر یعنی ۲۶۰۰ چاپ می شود.

۲- فرض کنید کد زیر در یک ماشین لینوکس کامپایل و اجرا شده است. همچنین فرض کنید تمامی فراخوانی های سیستمی با موفقیت اجرا می شوند.

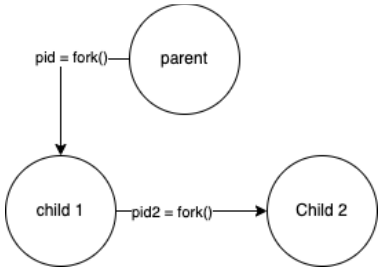
```c
int main() {
  int count = 0;
  int pid=0, pid2=0;

  if ( (pid = fork()) ) {
    count = count + 2;
    printf("%d", count);
  }
  if (count == 0)
  {
    count++;
    pid2=fork();
    printf("%d", count);
  }

  if (pid2 || pid) {
    wait(NULL);
    count = count+4;
  }
  printf("%d", count);
}
```

درخت اجرای پردازه ها را برای این برنامه رسم کنید. همچنین مشخص کنید که هر پردازه هنگام رسیدن به توابع printf چه مقادیری را چاپ می کنند.

درخت پردازه های این برنامه به شکل زیر خواهد بود. در مجموع، ۳ پردازه در این برنامه به وجود می آید.



ابتدا فقط parent در حال اجراست. در اولین fork پردازه‌ی C۱ ایجاد می‌شود. مقدار pid در parent غیرصفر و در C۱ صفر خواهد بود. Parent وارد بدنه‌ی شرط اول می‌شود و مقدار ۲ را چاپ می‌کند. وارد شرط دوم نمی‌شود و به دلیل مثبت بودن مقدار pid وارد شرط سوم می‌شود و منتظر اتمام اجرای C۱ خواهد شد. پردازه‌ی C۱ وارد بدنه‌ی شرط دوم می‌شود چون مقدار count در لحظه‌ای که fork اول اجرا شد برابر ۰ بود و در پردازه‌ی C۱ هم صفر خواهد بود. در اینجا

پردازه‌ی C۲ با فراخوانی fork دوم ایجاد می‌شود. مقدار count در هر دو پردازه‌ی C۱ و C۲ برابر ۱ است و چاپ می‌شود. پردازه‌ی C۱ نیز به دلیل مثبت بودن مقدار pid۲ از شرط سوم عبور می‌کند. منتظر می‌ماند تا اجرای C۲ با چاپ کردن مقدار count که همچنان برابر ۱ است، پایان یابد. سپس در C۱، مقدار count با افزایشی ۴ واحدی به ۵ می‌رسد، ۵ را چاپ می‌کند و اجرایش پایان می‌یابند. بعد از پایان یافتن اجرای C۱، پردازه‌ی Parent از حالت wait خارج می‌شود. Count را به ۶ افزایش می‌دهد. این مقدار را چاپ کرده و اجرایش به پایان می‌رسد. خروجی نهایی "۲,۱,۱,۱,۵,۶" که ممکن است ترتیب ۲۱۱ بسته به ترتیب اجرای هر پردازه متفاوت باشد اما رشته خروجی حتما با ۱,۵,۶ پایان می‌یابد.

۳- کد C زیر را در نظر بگیرید، هنگامی که زمان اجرای خط B برسد، چه تعداد رشته (thread) در سیستم واسطه این برنامه فعال خواهد بود؟

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

int value = 5;

void *print_message1(void *arg) {
    value += 2;

    char *message = "Hello from thread 1!"
    printf("%s\n", message); // LINE B
    pthread_exit(NULL);
}

void *print_message1(void *arg) {
    value += 2;

    char *message = "Hello from thread 2!
    printf("%s\n", message); // LINE C
    pthread_exit(NULL);
}

int main() {
    pthread_t threads[2];

    pthread_create(&threads[0], NULL, print_message1, NULL);
    pthread_create(&threads[1], NULL, print_message2, NULL);

    pthread_join(threads[0], NULL);
    pthread_join(threads[1], NULL);

    printf("Both threads have completed. Value = %d\n", &value);

    return 0;
}
```

در زمان اجرای خط B بسته به نوع زمان‌بندی سیستم‌عامل، ۲ یا ۳ نخ فعال خواهد بود. یک نخ که متعلق به پردازه‌ی پدر است. با رسیدن برنامه به اولین دستور pthread_create، نخ دوم نیز ایجاد می‌شود. در اینجا بسته به نوع زمان‌بندی سیستم‌عامل و با توجه به اولویت‌های تعریف شده در آن، ممکن است که تابع print_message۱ اجرا شود یا اینکه pthread_create دوم فراخوانی شود و یا در صورتی که سیستم چند هسته‌ای باشد، هر دو به صورت همزمان اجرا شوند. با توجه به موارد گفته شده، ممکن است که در لحظه‌ی اجرای خط B، دو یا سه رشته در سیستم فعال باشند.

۴- کد C زیر را در نظر بگیرید:

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

#define NUM_THREADS 2

int shared_value = 0;

void *increment_value(void *arg) {
    int *my_id = (int*) arg;
    int i;

    for (i = 0; i < 1000000; i++) {
        shared_value++;
    }

    printf("Thread %d has finished.\n", *my_id);
    pthread_exit(NULL);
}

int main() {
    int i, thread_ids[NUM_THREADS];
    pthread_t threads[NUM_THREADS];

    for (i = 0; i < NUM_THREADS; i++) {
        thread_ids[i] = i;
        pthread_create(&threads[i], NULL, increment_value, (void*) &thread_ids[i]);
    }

    for (i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }

    printf("Final shared value: %d\n", shared_value);

    return 0;
}
```

الف) خروجی این کد را پیش بینی کنید و توضیح دهید که چرا فکر می‌کنید این خروجی را تولید می‌کند.

پیام Thread has finished دو بار چاپ می‌شود اما نمی‌توان مطمئن بود که ترتیب آن به کدام یک از دو صورت زیر خواهد بود:

“Thread 0 has finished\n Thread 1 has finised\n” **یا** “Thread 1 has finished\n Thread 0 has finised\n”

اینکه خط printf توسط کدام رشته زودتر اجرا شود وابسته به زمان‌بندی سیستم عامل است. از آنجایی که پردازه‌ی پدر ابتدا روی رشته‌های فرزندش wait می‌کند و سپس دستور چاپ مقدار shared value را اجرا می‌کند، عبارت "Final shared value: X" حتما بعد از عبارت‌های بالا چاپ خواهد شد. مقدار X به احتمال خیلی زیادی برابر ۲۰۰۰۰۰۰ خواهد بود اما از آن جایی که دو رشته به صورت همزمان اجرا می‌شوند، بر فرض وجود شرایط رقابت (race condition) احتمال دارد که مقدار نهایی X کمتر از ۲۰۰۰۰۰۰ باشد اما این مقدار قطعا از ۱۰۰۰۰۰۰ بیشتر خواهد بود.

ب) در رابطه با کد داده شده، تفاوت بین استفاده از رشته‌ها و استفاده از فورک برای رسیدن به موازی سازی را توضیح دهید. به طور خاص، هنگام استفاده از فورک، آیا پردازش فرزند مقدار shared_value را برای هر دو فرآیند فرزند و والد تغییر می‌دهد؟ پاسخ خود را توضیح دهید. چرا هنگام استفاده از رشته ها متفاوت است؟

زمانی که پردازه‌ی پدر دستور fork را فراخوانی می‌کند، کل اطلاعات و داده‌های مربوط به پردازه‌ی پدر برای ایجاد فرزند duplicate می‌شوند و پردازه‌ی فرزند فضای اختصاصی خودش را بر روی حافظه خواهد داشت که مقادیر آن در اولین لحظه بعد از fork، عینا با مقادیر ذخیره شده در فضای حافظه‌ی مربوط به پردازه‌ی پدر برابر خواهد بود. اما بعد از دستور fork، پردازه‌ی پدر و فرزندش مستقل از هم ادامه می‌یابند و داده‌های موجود در حافظه‌ی آن‌ها به صورت مستقل تغییر خواهد کرد و دو پردازه در حالت عادی به داده‌های یکدیگر دسترسی ندارند. در چنین شرایطی، مقدار نهایی shared_value در پردازه‌ی پدر صفر خواهد بود. (اگر فرض کنیم که تابع increament_value در پردازه پدر اجرا نمی‌شود) اما در ایجاد thread، بخشی از حافظه میان پردازه‌ی پدر و رشته‌های ایجاد شده مشترک است. در واقع code ,data و files بین تمام رشته‌ها مشترک است و فقط thread id ,stack و register به صورت اختصاصی به هر رشته تعلق دارد. پس اگر یکی از رشته‌ها روی مقدار shared_value که به صورت global برای همه‌ی رشته‌ها در دسترس است تغییری ایجاد کند، این مقدار برای تمامی رشته‌ها تغییر خواهد کرد.

۵- فرض کنید یک پردازه دستور `value = value + 1` را اجرا می کند. همچنین یک پردازه دیگر به صورت همروند و مستقل دستور `value = value - 2` را اجرا می کند. اگر `value` یک متغیر مشترک بین این دو پردازه باشد و فقط در این دو دستور استفاده شده باشد و مقدار این ابتدایی این متغیر `value = 5` باشد، همه مقادیر ممکن برای متغیر `value` بعد از اجرای این دو پردازه را بنویسید.

اگر ابتدا پردازه اول مقدار را بخواند و آپدیت کند اما آن را ننویسد، سپس کار پردازه دو کاملا تمام شود و سپس مقدار نوشته شود، مقدار ۶ = value خواهد بود. در صورتی که اتفاق بالا ابتدا برای پردازه دوم بیفتد، مقدار value برابر ۳ خواهد بود. اگر یکی بعد از اتمام کامل دیگری اتفاق بیفتد، مقدار ۴ = value خواهد بود.