Department of
Computer Engineering

Homework 2-Solution

Operating Systems

Fall 2023

Dr. Javadi

پاسخ سوال ۱:

الف) خیر قابل مشاهده نیست زیرا هر نخ ، پشته جداگانه ای برای خود در اختصاص دارد.

ب) بله درست است زیرا در هنگام استفاده از دستور malloc در واقع از فضای heap حافظه تخصیص میشود و چون این فضا بین همه نخ ها یکسان می باشد، پس ممکن است این اتفاق بیفتد.

پ) بله درست است ، هسته فقط از وضعیت نخ های هسته اطلاع دارد و از نخ های سطح کاربر و عملیات آنها اطلاعی ندارد.

ج) خیر ، در یک نخ سطح کاربر سریعتر از یک نخ سطح هسته است زیرا در یک نخ سطح کاربر، سیستم عامل نیازی به انجام عملیات به همان اندازه که در یک نخ سطح هسته انجام می دهد ندارد. در یک نخ سطح کاربر، سیستم عامل فقط باید مقادیر ثبات و شمارنده برنامه را تغییر دهد. این یک عملیات نسبتا ساده است که می تواند به سرعت انجام شود. در مقابل، در یک نخ سطح هسته، سیستم عامل نیاز به انجام عملیات های بسیار بیشتری دارد. سیستم عامل باید وضعیت کل فرآیند، از جمله مقادیر ثبت، شمارنده برنامه و نقشه حافظه (memory map) را ذخیره کند. این یک عملیات بسیار پیچیده تر است که انجام آن بسیار طولانی تر است. علاوه بر این، نخ های سطح هسته سربار بیشتری دارند زیرا نیاز به تعامل با هسته و منابع سیستم دارند، که منجر به context switch کندتر می‌شود.

د) درست ، هنگامی که یک برنامه با نخ های سطح کاربر بر روی یک سیستم چند هسته ای اجرا می شود، تمام نخ های آن توسط یک نخ در سطح هسته مدیریت می شوند. سیستم عامل از وجود نخ های سطح کاربر در فرآیند بی اطلاع است. فقط نخ های سطح هسته را می بیند و زمان بندی می کند. بنابراین، حتی اگر برنامه دارای چندین نخ در سطح کاربر باشد، همه آنها بر روی یک نخ در سطح هسته نگاشت می شوند.

پاسخ سوال ۲:

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

#define NUM_THREADS 3

int sharedValue = 10;

void *increaseValue(void *threadID) {
    long tid = (long)threadID;

    printf("Thread %ld: Shared Value before increment: %d\n", tid, sharedValue);
    sharedValue += 5; // Increase the sharedValue by 5
    printf("Thread %ld: Shared Value after increment: %d\n", tid, sharedValue);

    pthread_exit(NULL);
}

int main() {
    pthread_t threads[NUM_THREADS];
    int rc;
    long t;

    for(t = 0; t < NUM_THREADS; t++) {
        printf("Creating thread %ld\n", t);
        rc = pthread_create(&threads[t], NULL, increaseValue, (void *)t);
        if (rc) {
            printf("Error: unable to create thread, %d\n", rc);
            return 1;
        }
    }

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

    printf("All threads have completed. Final shared value: %d\n", sharedValue);
    pthread_exit(NULL);
}
```

خروجی کد بالا ۴ حالت خواهد داشت:

۱ اینکه هر سه رشته به ترتیب اجرا بشوند که در این صورت خروجی رشته اول ۱۵ و خروجی رشته دوم ۲۰ و در نهایت خروجی رشته سوم برابر ۲۵ خواهد بود

۲ قبل از اینکه رشته اول مقدار value_shared را با ۵ جمع کند رشته دوم آن را بخواند که در این صورت خروجی هر دو برابر ۱۵ خواهد بود و خروجی رشته اخر برابر ۲۰ خواهد شد

۳ بعد از اتمام رشته اول ، قبل از اینکه رشته دوم مقدار value_shared را افزایش دهد رشته سوم آن را بخواند که در این صورت خروجی هردو رشته برابر ۲۰ خواهد بود

۴ قبل از اینکه رشته اول مقدار value_shared را افزایش دهد دو رشته دیگر همان مقدار ۱۰ را بخوانند که در این صورت خروجی هر سه رشته برابر ۱۵ خواهد بود.

الف) در صورتی که دو تابع join_pthread و create_pthread در یک حلقه صدا زده شوند برنامه حالت sequential خواهد گرفت به این شکل که قبل از اینکه رشته جدیدی ایجاد شود رشته قبلی کار خود را تمام کرده و خروجی به این شکل خواهد بود که ابتدا رشته اول مقدار value_Shared را برابر ۱۵ می کند بعد رشته دوم ایجاد شده و مقدار ان را برابر ۲۰ میکند و در انتها رشته سوم ایجاد شده مقدار نهایی ان برابر ۲۵ خواهد بود.

ب) فرآیندها و رشته ها دو رویکرد متفاوت برای دستیابی به همزمانی در یک برنامه هستند. در زیرتفاوتهای اصلی بین آنها آورده شده است:

۱ پردازه در مقابل رشته:

در زمینه پردازه ها، ایجاد یک فرآیند جدید کپی از فرآیند موجود میسازد. هر فرآیند فضای حافظه جداگانهای دارد و به صورت مستقل اجرا میشود. به عنوان مقابل، رشته ها در یک پردازه از یک فضای حافظه مشترک استفاده میکنند. آنها سبکتر از پردازه ها هستند و برای برخی از انواع همروندی بهترین گزینه هستند.

۲ اشتراک حافظه:

فرآیندهای ایجاد شده توسط فورک فضای حافظه جداگانه ای دارند. اگر یک فرآیند حافظه خود را تغییر دهد، تغییرات بر رو ی حافظه فرآیندهای دیگر تأثیر نمیگذارد. ولی رشته ها از یک فضای حافظه مشترک استفاده میکنند، بنابراین تغییرات انجام شده توسط یک رشته به صورت فوری در رشته های دیگر در همان پردازه قابل مشاهده هستند. این فضای حافظه مشترک میتواند ارتباط بین رشته ها را ساده تر کند، اما نیازمند مکانیسم های هماهنگسازی برای جلوگیری از تداخلهای داده ها است.

۳ هزینه ایجاد:

در فورک ایجاد یک فرآیند جدید با استفاده از فورک ممکن است نسبت به لحاظ زمانی و منابع نسبت به سایر روشها گران باشد، زیرا شامل کپی کردن کل فرآیند، شامل فضای حافظه آن میشود. ولی در رشته ها ایجاد یک رشته به طور معمول سریعتر و با مصرف منابع کمتری همراه است نسبت به ایجاد یک فرآیند.

۴ ارتباط و هماهنگسازی:

برای ارتباط بین فرآیندهای مختلف ایجاد شده توسط فورک، نیاز به ارتباط بینفرآیندی (IPC) وجود دارد. این ارتباط با استفاده از مکانیسمهایی مانند لوله ها، صف های پیام یا حافظه مشترک امکانپذیر است. اما رشته ها در یک پردازه بدون نیاز به ارتباط بین فرآیندی به راحتی میتوانند از طریق حافظه مشترک ارتباط برقرار کنند، اما برای جلوگیری از تضاد داده ها نیاز به مکانیسمهای هماهنگسازی است.

۵ قابلیت حمل و نقل:

فورک یک ویژگی از بسیاری از سیستم عامل‌های شباهت دار با یونیکس است. استفاده از رشته ها معمولا قابل حملتر در سیستم عامل‌های مختلف است، زیرا کتابخانه‌های رشته ارتباط یکپارچه‌ای را فراهم میکنند.

۶ همروندی در مقابل همزمانی:

فورک برای همروندی، جایی که وظایف به صورت مستقل اجرا میشوند، مناسبتر است. هر فرآیند بر روی یک وظیفه جداگانه کار می کند و همروندی با اجرای همزمان چندین فرآیند امکانپذیر است. اما رشته‌ها به دلیل اینکه در یک پردازه قرار دارند، برای همزمانی مناسبتر هستند. رشته‌ها در یک پردازه میتوانند به صورت همزمان در زمانهای همپوشانی بر روی جنبه‌های مختلف یک وظیفه کار کنند.

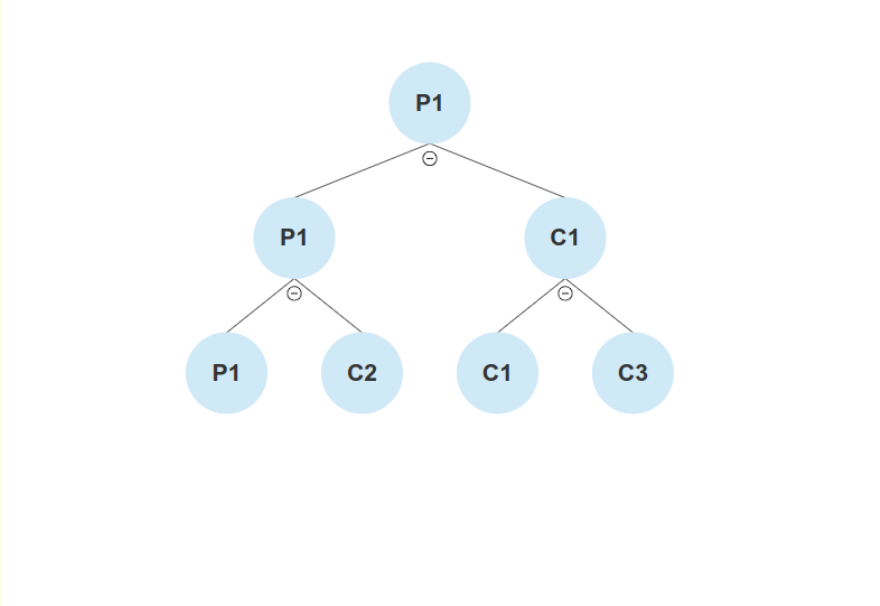
ج) تابع `join_pthread` برای انتظار اتمام یک نخ مشخص و بازیابی منابع آن نخ استفاده میشود. این تابع تا زمانی که نخ مورد نظر اجرا شده و به پایان رسیده باشد، به نخ فراخواننده باز میگردد.

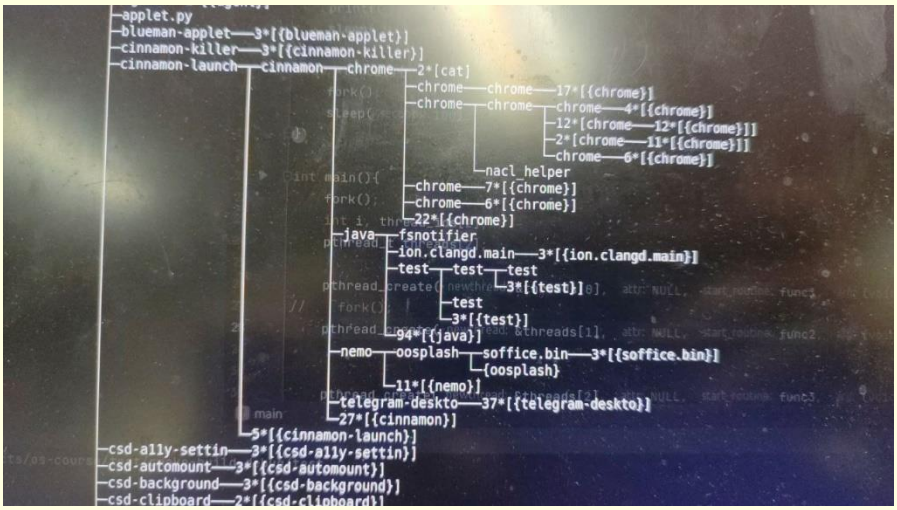
از لحاظ تعداد نخهایی که میتوانند `join_pthread` را بر رو ی یک نخ هدف فراخوانی کنند، هیچ محدودیتی وجود ندارد. بنابراین، تعداد نخهایی که `join_pthread` را بر روی یک نخ هدف صدا میزنند، مستقل از یکدیگر هستند و هیچ محدودیت خاصی وجود ندارد.

این به این معناست که چند نخ میتوانند همزمان روی یک نخ هدف تابع `join_pthread` را صدا بزنند.

پاسخ سوال ۳:

الف) 4
ب) 4
ج) 10
د)





۵) همانطور که در شکل زیر بعد از اجرای کد میتوان مشاهده کرد چون تابع فورک داخل یک نخ صدا زده شده است فقط استک همان نخ کنیم میشود و نخ های فعال در پردازه ی جدید یک عدد خواهد بود. پردازه ی اول دو پردازه ایجاد میکند و ۴ نخ دارد. فورک اول خارج از نخ است پس ۳ نخ پردازه ی مادر هم کپی میشود. فرزند دوم چون توسط فورک داخل نخ ایجاد شده فقط نخ پردازه اصلی را دارد. در ادامه پردازه ی اول نیز تابع فورک را داخل یکی از نخ های خود صدا میزند و یک فرزند با یک نخ ایجاد میشود. پس ۴ پردازه ایجاد میشود. مجموع نخ ها برای پردازه ی p1 برابر با ۴، c1 برابر با ۴، c2 و c3 هم برابر با ۱ است که مجموعا میشود ۱۰ نخ.
از طرفی در p1 و c1 نخ های اول و سوم اجرا شده و در نهایت ۴ ۱ اجرا میشود.
*ممکن است با اجرای کد به نتایج دیگری برسید که به دلیل ارث بری output buffer از پردازه ی مادر به فرزند ممکن است چند ۱ اضافه تر چاپ شود اما اگر هر خروجی را در خط مجزا چاپ کنید این مشکل برطرف خواهد شد.

پاسخ سوال ۴:

ابتدا هر یک از پردازه های والد و فرزند شانس اجرا دارند و خطوط ابتدایی خروجی به صورت

Process 1

Process 2

یا بلعکس است،سپس والد برای اجرا فرزند wait می کند و در فرزند execv صدا زده می شود و به جای ادامه اجرا دستورات بدنه if دستور ls -۱ اجرا می گردد که خروجی آن تمام فایل های موجود در دایرکتوری فایل برنامه،هر یک در یک خط جداگانه، است(که این خروجی حتما پس ۱ processخواهد بود و ممکن است پیش یا پس از ۲ process باشد) ،در نهایت عبارت process ۱ terminated مربوط به پردازه والد چاپ می شود.

نکته:توجه کنید process 1 finished مربوط به پردازه فرزند دیگر چاپ نمی شود)به خاطر اجرا دستور exec)

پاسخ سوال ۵:

الف) ۵۵ ، هر یک بار که بدنه producer اجرا میشود، با توجه به تغییری که در مقدار inمیدهد،شرط in==out در consumer ارضا نمیشود،بنابراین هر مقداری را که consumerقرار میدهد، producer همان را در مرحله بعد خواهد خواند و خروجی حاصلجمع مقادیر ۱ تا ۱۰ می باشد(توجه کنید با توجه به آنکه ++پیش از produced_next آمده است و در طرف دیگر در بدنه حلقه هر بار consumed_next آپدیت میشودبه sum هم اضافه میگردد و بعد شرط حلقه مجددا چک میشود، بنابراین مقدار ۱۰ نیز توسط consumer خوانده میشود به جمع اضافه میشود.)

ب)۱۵ یا ،۲۵ هر دو پاسخ قابل قبول است، مرحله به مرحله داریم:

:I

BUFFER: 1, 2

Sum = 1

In = 2

Out =1

:II

BUFFER: 1, 2, 3, 4

Sum = 1+2

In = 4

Out = 2

:III

BUFFER: 6, 2, 3, 4, 5

Sum = 1+2+3

In = 1

Out = 3

:IV

BUFFER: 6, 7, 8, 4, 5

Sum = 1+2+3

In = 3

Out = 3

توجه کنید، پس از آنکه producer در این مرحله مقدار inرا برابر با ۳ کرد و و دو مقدار جدید را در بافر قرار داد، هنگامی که نوبت به اجرای بدنه حلقه consumerمیرسد، مقدار in==outخواهد بود و continueاجرا میشود و این یک بار اجرا بدنه حلقه consumer خواهد بود.

:V

BUFFER: 6, 7, 8, 9, 10

Sum = 1+2+3+9

In = 0

Out = 4

در اینجا مقدار sum برابر با ۱۵ خواهد بود و اگر با فرضی که گفته شده است به ازای هر دو بار producer یک بار consumer اجرا شود و با توجه به آنکه producer دیگر اجرا نمیشود، کار را سمت consumer رها کرده باشید و مقدار را ۱۵ حساب کرده باشید مقدار صحیح است اما اگر کار را ادامه داده باشید:

:VI

BUFFER: 6, 7, 8, 9, 10

Sum = 1+2+3+9+10

In = 0

Out = 0

باید به مقدار ۲۵ رسیده باشید و این پاسخ نیز مورد قبول است.

پ)۸ یا ،۳۵ مطابق با روند قسمت ب که پیش بروید با استدلال اول به مقدار ۸ و با استدلال دوم به مقدار ۳۵ میرسید.