دانشگاه اصفهان
دانشکده علوم ریاضی و کامپیوتر

Operating Systems

اصول سیستم‌های عامل

سری دوم تمرینات

نیم‌سال اول ۱۴۰۳-۱۴۰۴

اعضای گروه

محمد ملائی : ۴۰۱۴۰۲۳۰۴۲

علیرضا احمدی وشکی : ۴۰۱۴۰۱۳۰۰۷

یگانه رستگاری : ۴۰۱۴۰۱۳۰۴۰

نام استاد درس

مجتبی رفیعی

تمرین ۱

صحیح یا غلط بودن گزاره‌های زیر را مشخص کنید و دلیل خود را نیز بیان کنید.

(آ) پشته که یکی از بخش های حافظه فرآیند است در واقع حافظه ای دائمی است که به هنگام فراخوانی توابع مورد استفاده قرار می گیرد و حاوی آدرس بازگشت است.
تنها بخشی از گزاره درست است. پشته بخشی از حافظه فرایند است که با فراخوانی تابع، یک رکورد شامل پارامترهای تابع، متغیرهای محلی و آدرس بازگشت به آن اضافه می‌شود. با بازگشت تابع، رکورد از پشته حذف می‌شود. پس اندازه‌ی آن در حین اجرای فرایند می‌تواند تغییر کند. این پشته حافظه موقت محسوب می‌شود نه حافظه‌ی دائمی چرا که اطلاعات تابع تنها تا زمانی که تابع درحال اجرا باشد نگهداری میشوند.

(ب) زمان بند CPU وظیفهٔ انتخاب یک فرآیند از بین همه فرآیندهای موجود در سیستم و تخصیص هستهٔ پردازشی به آن را بر عهده دارد.
گزاره غلط است. زمان بند CPU فرآیند را از میان فرآیندهای آماده انتخاب می‌کند و نه همه‌ی فرآیند های سیستم.

(ج) یک نخ عادی در طول حیات خود ممکن است در LWP های متفاوتی، بخش هایی از اجرای خود را بگذراند.
درستی این گزاره به طراحی سیستم و کرنل بستگی دارد. LWP یک رابط بین نخ‌های سطح کاربرو نخ‌های سطح کرنل محسوب می‌شود که توسط کرنل سیستم عامل مدیریت می‌شود. بدین صورت که از دید نخ کاربر LWP یک پردازندهٔ مجازی است که توسعه دهنده به هنگام توسعه‌ی برنامه‌ی کاربری می تواند آن را برای اجرای نخ های سطح کاربر زمان بندی نماید. از سوی دیگر، هر LWP متصل به یک نخ کرنل است و در واقع این نخ های کرنل هستند که توسط سیستم عامل برای اجرا روی پردازندهٔ فیزیکی زمان بندی می شوند. در مدل چند به چند، نخ‌های سطح کاربر به طور مستقیم به یک نخ سطح کرنل متصل نیستند و به تعداد کوچکتر یا مساوی نخ های سطح کرنل تعمیم می شوند. در مدل دو سطحی، یک نخ سطح کاربر می‌تواند به یک نخ سطح کرنل متصل شود و اگر به نخ خاصی محدود نباشد، می‌تواند در هایLWP متفاوتی اجرا شود. پس در سیستمی که از مدل چند به چند یا مدل دو سطحی استفاده می‌شود نخ سطح کاربر می‌تواند در هایLWP متفاوتی بخش‌هایی از اجرای خود را بگذراند.

(د) فرآیند فرزند می تواند برنامه و دادهٔ یکسان با فرآیند پدر خود داشته باشد
این گزاره درست است. وقتی یک فرایند فرزند ایجاد می‌شود، متن برنامه‌اش را به صورت کامل از پدر خود گرفته‌است پس برنامه آن‌ها یکسان است. چون PCB فرزند با تغییرات کمی با PCB فرآیند پدر خود یکسان است پس می‌تواند داده‌های آن را نیز داشته باشد. البته علاوه بر این‌ها می‌تواند به منابع پدر خود که از سیستم عامل گرفته‌است نیز دسترسی داشته باشد.

(ه) برای برقراری یک پیوند بین هر زوج فرآیند، فرآیندها نیازی به اطلاع از شناسهٔ یکدیگر ندارند.
این گزاره درست است اما بستگی به نوع پیاده‌سازی روش تبادل پیام دارد. یکی از راه‌های برقراری ارتباط بین هر زوج فرایند، تبادل پیام است. در تبادل پیام غیرمستقیم، فرایندها با استفاده از صندوق پستی برای یکدیگر پیام می‌فرستند و چون این صندوق پستی را سیستم‌عامل مدیریت می‌کند تنها به شناسه‌ای مشترک برای این صندوق احتیاج دارند و نه PID یکدیگر.
اما در تبادل پیام مستقیم لازمه یک پیوند بین هر زوج فرایند، اطلاع داشتن از شناسه‌ی یکدیگر است. برخلاف پیاده‌سازی صندوق پستی، این پیوند تنها مختص یک زوج فرایند است و نمی‌توان از آن برای برقراری ارتباط با فرایندهای دیگر استفاده کرد. با در نظر گرفتن این روش، گزاره نادرست است.

تمرین ۲

به سوالات زیر پاسخ کامل دهید.

(آ) مدل اشتراک گذاری حافظه و مدل ارتباطی تبادل پیام را تعریف و با یکدیگر مقایسه کنید.
برای برقراری ارتباط درون فرایندی دو روش وجود دارد: مدل اشتراک گذاری حافظه و مدل ارتباطی تبادل پیام.

مدل اشتراک گذاری حافظه در این مدل، فرایندها با استفاده از یک ناحیه اشتراکی حافظه با یکدیگر ارتباط برقرار می‌کنند. این ارتباط نیازمند وجود فرآیندهای جدیدی به نام فرآیندهای ارتباطی است که برای فراهم کردن یک ناحیه برای حافظهٔ اشتراکی ایجاد شده اند. با یک فراخوان سیستمی، سیستم عامل حافظه اشتراکی را در فضای آدرس فرآیند ارتباطی ایجاد می‌کند. پس از ایجاد حافظه اشتراکی، فرآیندها ناحیه‌ی حافظه‌ی اشتراکی را به فضای آدرسشان اضافه می‌کنند. فرایندها می‌توانند به طور مستقیم به حافظه دسترسی داشته باشند و با عمل read یا write ، بدون دخالت کرنل، ارتباط مستقیم داشته باشند. در این روش احتمال تداخل عمل نوشتن و خواندن داده توسط فرایندها وجود دارد.

مدل تبادل پیام در این مدل، فرآیندهای همکار فعالیت هایشان را بدون اشتراک گذاری فضای مشترک هماهنگ کرده و با رد و بدل پیام با یکدیگر ارتباط برقرار می کنند. یک فرایند فرستنده پیام و دیگری گیرنده پیام است. برای شکل گیری پیوند ارتباطی و ارسال پیام، فرایندها برای ارسال و دریافت پیام نیازمند فراخوان سیستمی و دخالت کرنل هستند. نوع پیوند ارتباطی می‌تواند مستقیم یا غیرمستقیم باشد.
در ارتباط مستقیم، یک پیوند ارتباطی دقیقا برای دو فرآیند در نظر گرفته می‌شود و بین هر زوج فرآیند، دقیقا یک پیوند وجود دارد. فرایندها از شناسه‌ی یکدیگر مطلع هستند و نام دریافت‌کننده و ارسال‌کننده باید به صورت صریح مشخص باشد.
در ارتباط غیرمستقیم، برای پیاده‌سازی این روش از صندوق پستی استفاده می‌شود و فرایندها پیام خود را در صندوق گذاشته یا پیام را حذف می‌کنند. یک پیوند می تواند به بیش از دو فرآیند اختصاص داشته باشد و بین هر زوج فرآیند می تواند بیش از یک پیوند وجود داشته باشد. هر پیوند منطبق با یک صندوق پستی است و هر صندوق پستی شناسه یکتا دارد.

تفاوت‌های دو روش

۱. مدل حافظه اشتراکی سرعت بیشتری نسبت به تبادل پیام دارد زیرا تنها با یک فراخوان سیستمی حافظه ایجاد می‌شود اما در تبادل پیام، برای ارسال هر پیام به استفاده از فراخوان سیستمی نیاز دارند و این مورد باعث کند بودن ارتباط می‌شود.
۲. برای ایجاد هماهنگی بین فرایندها و حفاظت از داده، مدل حافظه اشتراکی نیازمند روش‌هایی مانند سمافور است. اما در تبادل پیام، در نوع پیاده‌سازی هماهنگی در نظر گرفته شده
۳. پیاده سازی مدل تبادل پیام در سیستم های توزیع شده به مراتب ساده تر از مدل اشتراک حافظه است چرا که در سیستم های توزیع شده، فرآیندها بر روی سیستم های مختلفی مستقر هستند.

(ب) حالت هایی را بیان کنید که باعث پایان یافتن فرآیند می شود.

- اجرای فرایند به پایان رسیده باشد.
- اجرای فرایند بیشتر از زمان تعیین شده طول بکشد.
- فرایند به منابع مورد نیاز خود دسترسی ندارد و یا می‌خواهد به منابع غیرمجاز دسترسی داشته باشد.
- فرایند نیازمند حافظه‌ی بیشتر از حافظه موجود باشد.
- سیستم عامل به دلایلی مانند deadlock فرایند را متوقف کند.
- کاربر درخواست پایان یافتن فرآیند را بدهد.
- در برنامه خطایی رخ دهد که توسط برنامه مدیریت نشده باشد مانند خطای تقسیم بر صفر.
- اجرای آن توسط فرآیند پدر متوقف شده باشد.

(ج) برنامه نویسی چندهسته ای و چند نخی را تعریف و با یکدیگر مقایسه کنید. در چه حالت هایی استفاده از سیستم های تک هسته ای مناسب تر از سیستم های چند هسته ای است؟ در برنامه نویسی فرآیندهای چند نخی، چند نخ درون یک فرایند ایجاد می‌شوند و هم روندی را ممکن می‌سازد. نخ های مربوط به فرآیند از منابع، کد و داده‌ی مشترک استفاده می‌کنند و در فضای آدرس یکسانی فعال هستند پس برقراری ارتباط بین آنها راحت‌تر است. با استفاده از نخ‌ها، اگر بخشی از فرآیند بلاک شود یا در حال انجام محاسبات طولانی باشد، بخش های دیگر اجرا شده و قابل استفاده هستند. در این راستا، مزایای این نوع برنامه نویسی را می توان در چهار ردهٔ کلی زیر تقسیم بندی کرد: پاسخگویی سریع، اشتراک گذاری منابع، صرفه اقتصادی، مقیاس پذیری.
در برنامه نویسی چند هسته‌ای هدف تامین مکانیزمی است که با هم روندی و به کارگیری هم زمان چند هسته پردازشی، سبب استفاده‌ی بهتر از CPU و در نهایت تسریع محاسبات شود.
یک طراح سیستم عامل می بایست الگوریتم های زمان بندی پیچیده‌تری را برای استفادهٔ حداکثری از پردازنده طرح ریزی کند به طوری که ابتدا تشخیص دهد کدام وظایف می‌توانند به صورت موازی اجرا شوند، برنامه را به این وظایف تقسیم کند، از وجود تعادل بین وظایف اطمینان حاصل کند و داده‌ها را مدیریت کند. چالش های پیش روی برنامه نویسان در مواجهه با چنین سیستم‌هایی را می‌توان در پنج رده تقسیم بندی نمود: شناسایی وظایف، تعادل، جداسازی داده، وابستگی داده، تست و اشکال زدایی.
تفاوت‌های برنامه نویسی چند نخی و چند هسته‌ای:

۱. اشتراک منابع: در برنامه چند نخی، برقراری ارتباط راحت‌تر از برنامه چندهسته‌ای است زیرا آدرس حافظه، منابع و داده‌های مشترک دارند و نیازی به استفاده از روش‌های IPC ندارند.
۲. اجرای موازی وظایف: در برنامه چند هسته‌ای اجرای موازی از طریق اجرای وظایف بر چندین هسته پردازشی صورت می‌گیرد اما در برنامه چند نخی، هم روندی با اجرای همزمان چند نخ درون یک فرایند امکان پذیر است.
۳. پیچیدگی: برنامه چند هسته‌ای به دلیل تقسیم کار و مدیریت ارتباط بین فرایندها پیچیدگی بیشتری دارد.

در سیستم تک هسته‌ای نخ‌ها صرفاً در بین هم اجرا می شوند یعنی در یک زمان تنها یک نخ اجرا می شود. در سیستم چند هسته‌ای نخ های مجزا می توانند بر روی هسته های مختلف در یک زمان و به صورت موازی اجرا شوند.
در سیستم‌هایی که مصرف انرژی و اندازه واحد پردازشی اهمیت بیشتری نسبت سرعت پردازشی دارد، استفاده از یک پردازنده تک هسته‌ای گزینه‌ای مناسب‌تر و معقول‌تر است. به علاوه در سیستم‌هایی که برنامه‌ها معمولا ساده هستند و تضمین کارکرد درست آنها اولویت بیشتری دارد، استفاده از پردازنده تک هسته‌ای می‌تواند گزینه‌ی بهتری باشد چرا که پیچیدگی‌های سیستم چند پردازنده‌ای را ندارد.

(د) تشریح کنید همکاری بین فرآیندها چه سودی دارد و چه نوع پیچیدگی هایی را در سیستم ایجاد می کند.

سود همکاری بین فرایندها

- **اشتراک گذاری اطلاعات:** ممکن است در چندین برنامهٔ کاربردی، نیاز به دسترسی همزمان به یک قطعه اطلاعاتی مشترک باشد.
- **تسریع محاسبات :** اگر بخواهیم یک تسک را سریع تر اجرا کنیم، می بایست آن را به تسک‌های کوچکتر بشکنیم و هرکدام را به طور موازی با یکدیگر اجرا کنیم. برای یکپارچگی و عملکرد صحیح سیستم در چنین شرایطی نیازمند محیط کاری مشترک برای کنترل و مدیریت زیرتسک‌ها و در نهایت تسک اصلی هستیم.
- **ماژولار یا پیمانه ای بودن:** ممکن است در سیستمی تمایل داشته باشیم که توابع سیستم را در چندین دسته از فرآیندها یا نخ های مجزا تقسیم بندی کنیم. در چنین حالتی برای داشتن عملکرد صحیح و یکپارچه، داشتن محیط همکاری مشترک ضروری است.

پیچیدگی‌های همکاری فرایندها

- **همگام سازی:** ممکن است در چندین برنامهٔ کاربردی، نیاز به دسترسی همزمان به یک قطعه اطلاعاتی مشترک باشد.
- **احتمال رخ دادن deadlock یا گرسنگی:** فرایندها منابعی که دارند را رها نمی‌کنند و منتظر دسترسی به منبع مورد نیاز می‌مانند یا یک فرایند پس از دسترسی به منابع، آن را نگه میدارد و اجازه دسترسی دیگر فرایندها به آن را نمی‌دهد.

(ه) تشریح کنید در تعویض متن نخ ها، چه فعالیت هایی توسط هسته باید انجام شود و با تعویض متن فرآیندها چه تفاوت هایی دارد. در تعویض متن نخ‌ها، پردازنده متن نخ را ذخیره می‌کند و با بارگذاری متن نخ جدید، آن را اجرا می‌کند. از آنجایی که نخ‌ها در یک فرایند هستند، فضای آدرس یکسان دارند و از منابع و داده‌ی مشترک استفاده می‌کنند، تعویض متن سریع‌تر و راحت‌تر انجام می‌شود. پردازنده فقط باید حالت فعلی نخ (شمارنده برنامه و رجیسترها) را باید تغییر دهد.

تفاوت‌ها

۱. تعویض متن نخ وقتی رخ می‌دهد که CPU حالت کنونی نخ را ذخیره می‌کند و اجرای نخ دیگری از همان فرایند را شروع می‌کند. تعویض متن فرایندها زمانی است که زمان‌بند سیستم عامل حالت کنونی برنامه در حال اجرا را ذخیره می‌کند و سراغ برنامه‌ای دیگر می‌رود.

۲. تعویض متن نخ نیازی به تعویض آدرس حافظه ندارد. تعویض متن فرایند نیاز به تعویض آدرس حافظه دارد. به همین دلیل تعویض متن نخ کاراتر است.

۳. تعویض متن نخ نسبتا سریع‌تر و ارزان‌تر است.

تمرین ۳

غلط های موجود در کد زیر را پیدا کرده و نخست علت غلط بودن آنها را بیان کرده و سپس آنها را اصلاح کنید.

```
child_1 = fork();
if (child_1 == 0)
{
    shmID = shmget(key, 2795, 1024, 0666);
    memory = shmat(shmID, null, 0);
    child_2 = fork();
    if (child_2 == 0)
    {
        shmID = shmget(key, 2395, 1024, 0666);
        memory2 = shmat(shmID, value, 0);
        write_to_memory(memory2, "child_2 message");
    }
    else
    {
        text = read_from_memory(memory);
        if (text == "kill child_2") {
            kill(child_2);
            write_to_memory(memory, "killed child_2");
        }
        else
        {
            shmID2 = shmget(key, 2395, 1024, 0666 | IPC_CREAT);
            memory2 = shmat(shmID2, value, 0);
            text = read_from_memory(memory2);
            print("child_2 wrote:", text);
            text = read_from_memory(memory);
            print("child_1 wrote:", text);
            write_to_memory(memory, "kill child_2");
            text = read_from_memory(memory);
            print("child_1 wrote:", text);
        }
    }
}
```

- اولین مشکل در استفاده از تابع shmget است. این تابع تنها ۳ ورودی دارد که اولین ورودی آن، کلید حافظه اشتراکی است، سپس اندازه این حافظه و پس از آن permission آخرین ورودی این تابع است. برای اصلاح این مشکل باید key را حذف کنیم و از و از خود کلیدها که پس از آن و به صورت رشته عددی آمده‌اند استفاده کنیم.
- در استفاده از تابع shmat برای گرفتن حاضه اشتراکی memory2، به ورودی دوم، متغیر value داده شده است که یک متغیر تعریف نشده است. ورودی دوم این تابع آدرس حافظه‌ای را میگیرد که فرآیند مربوط به حافظه اشتراکی attach میشود. برای اصلاح میتوانیم از null به جای value استفاده کنیم.
- اگر child_2 زودتر از child_1 اجرا شود برنامه هنگام اجرای shmget با خطا روبه‌رو میشود چرا که حافظه‌ای اشتراکی memory2 ساخته نشده است پس نمیتوان به آن دسترسی پیدا کرد. برای حل این مشکل کافی است به جای 666 قرار دهیم IPC_CREATE | 666.

- فرآیند پدر زودتر از child_1 تمام میشود بنابراین این فرآیند یتیم میشود که این موضوع میتواند باعث ایجاد مشکل در اجرای این فرآیند شود به همین دلیل از wait استفاده میکنیم. در childe_1 نیز باید از wait استفاده کنیم.

پس از اصلاح مشکلاتی که در بالا ذکر شد، برنامه به صورت زیر خواهد بود:

```
int main()
{
    pid_t child_1 = fork();
    if (child_1 == 0)
    {
        int shmID = shmget(2795, 1024, IPC_CREAT | 0666);
        void *memory = shmat(shmID, NULL, 0);
        pid_t child_2 = fork();
        if (child_2 == 0)
        {
            int shmID2 = shmget(2395, 1024, IPC_CREAT | 0666);
            void *memory2 = shmat(shmID2, NULL, 0);
            write_to_memory(memory2, "child_2 message");
        }
        else
        {
            char *text = read_from_memory(memory);
            if (strcmp(text, "kill child_2") == 0)
            {
                kill(child_2, SIGKILL);
                write_to_memory(memory, "killed child_2");
            }
            else
            {
                int shmID2 = shmget(2395, 1024, IPC_CREAT | 0666);
                void *memory2 = shmat(shmID2, NULL, 0);
                text = read_from_memory(memory2);
                printf("child_2 wrote: %s\n", text);
                text = read_from_memory(memory);
                printf("child_1 wrote: %s\n", text);
                write_to_memory(memory, "kill child_2");
                char *text = read_from_memory(memory);
                printf("child_1 wrote: %s\n", text);
                shmdt(memory2);
            }
            wait(NULL);
            shmdt(memory);
        }
    }
    else
    {
        wait(NULL);
    }
    return 0;
}
```

تمرین ۴

به پرسش های زیر با ذکر تمام جزئیات خواسته شده پاسخ کامل بدهید.

(آ) سیستم عاملی را فرض کنید که در آن تنها یک فرآیند در کرنل وظیفه ی نظارت بر امنیت و صحت اجرای نخ های سطح کرنل را بر عهده دارد. حال فرض کنید در کنار چند نخ دیگر که از پیش در این سیستم موجود بودند، یک نخ جدید ایجاد می شود و کرنل برای ساخت نخ ها از معماری یک به یک تبعیت می کند. اکنون اگر نخ جدید نیاز به فراخواندن یک فراخوان سیستمی پیدا کند که نیاز است فرآیند نظارتی نام برده شده در بالا بر روی آن نظارت کند، به پرسش های کار پاسخ دهید:

i. سیستم برای نظارت بر نخ جدید باید به چه شکل عمل کند؟ معماری تولید نخ یک‌به‌یک است. این یعنی به ازای هر نخ تولید شده در فضای کاربر، یک نخ متناظر در فضای کرنلی تولید می‌شود. پس فرآیند نظارتی باید بتواند روی تک‌تک نخ‌های کرنل نظارت داشته باشد. برای نظارت داشتن و رصد کردن فراخوان‌های سیستمی باید یک الگو برای مدیریت انتخاب کنیم. سناریوهای گوناگونی می توان متصور شد.

صد کردن مستقیم (Direct Monitoring) فرآیند نظارتی با استفاده از system hooks تمامی فراخوانی‌های سیستمی از جانب نخ‌های کرنل را رصد می‌کند. باید ساختار هندلر فراخوانی‌های سیستمی را تغییر داد. به گونه‌ای که به فرآیند نظارتی قبل از اینکه روال مناسب را اجرا کند خبر داده شود. تا بعد از تایید این فرآیند نظارتی اجرای روال صورت گیرد.
مزیت : تاخیر ندارد و به صورت Real-Time رصد کردن را در اختیار دارد.
عیب : سربار زیاد به دلیل اینکه هر فراخوان سیتمی نیاز به دخالت این فرآیند نظارتی را می‌طلبد.

مکانیزم اعلان (Notification Mechanism) در این روش، وظیفه‌ی باخبر کردن فرآیند نظارتی، هنگام رخ دادن فراخوان سیستمی را به کرنل واگذار می‌کنیم. ر این روشی بعضی از فرآیندّا به خود کرنل واگذار می‌شوند. یعنی یک رصد اولیه انجام بدهد و اگر پتانسیل وجود یک خطر امنیتی را حس کرد به فرآیند نظارتی اعلان بفرستد.
مزیت : کرنل برای فرآیندهای مربوط به فرآیند نظارتی آن را باخبر می‌کند. در نتیجه سربار کمتری برای فرآیند نظارتی داریم.
عیب : تاخیر به دلیل این که سیستم اعلان، پتانسیل تأخیر را داراست.

استفاده از صف (Queue-Based Monitoring) فراخوان‌های سیستمی که نیاز به نظارت دارند به ترتیب در یک صف نگهداری می‌شوند. پیاده سازی آن میتواند از طریق یک صف اشتراکی بین کرنل و فرآیند نظارتی صورت گیرد.
مزایا :

- رصد کردن را ساده‌تر می‌کند و پیچیدگی روش‌های قبل را ندارد.
- اجازه می‌دهد که فرآیند نظارتی با سرعت خودش این فراخوان‌های سیستمی را مدیریت کند.

عیب : استفاده از صف نیاز به همگام‌سازی دارد وکرنه باعث وضعیت رقابتی می‌شود.

ii. این سازوکار چه تاثیری بر چندوظیفگی سیستم عامل خواهد گذاشت؟

- منطقا یک سربار اضافه میشود. چون باید نخ‌های اجرایی پیوسته رصد شوند.
- فرآیند نظارتی، یک منبع است. فراخوان‌های سیستمی باید از دروازه‌ی این فرآیند رد شوند. همین رقابت بر سر منبع نیز میتواند یک نقطه ضعف باشد.

- برای جلوگیری از بن‌بست‌ها (Deadlocks) و شرایط رقابتی (Race Conditions) برای وقتی که با استفاده از صف، فرآیند نظارتی را پیاده‌سازی میکنیم، باید بتوانیم همگام‌سازی لازم را صورت بدهیم.
- یک مقوله برای مدیریت به سیستم کامپیوتری اضافه شده است. همین موضوع مقیاس‌پذیری سیستم کامپیوتری را کاهش می‌دهد.

iii. راهکار شما برای حل مشکلات به وجود آمده چیست؟

- به جای بررسی کردن هر فراخوان سیستمی، اطلاعات که مربوط به نخ را کش کنیم و برای بررسی‌های آتی آن‌ها را ملاک قرار بدهیم.
- شدت رصد کردن را تغییر بدهیم. یک سامانه‌ی اولویتی به فراخوان‌های سیستمی بدهیم تا فقط آن دسته از فراخوان‌های سیستمی که احتمال خطای امنیتی زیادی دارند، تماما رصد شوند و این رصد برای بقیه‌ی فرآیندها، با احتمال خطای امنیتی کمتر، کمتر باشد و یا کلاً نباشد(یعنی فرآیند را امن تشخیص دهیم).
- از موازی سازی استفاده کنیم. به گونه‌ای که برای رصد کردن، اجرای نخ‌ها متوقف نشود. و اجرای فرآیند نظارتی در پیش‌زمینه صورت بگیرد.

(ب) فرض کنید داده ای در یک بخش از حافظه (یک فایل) وجود دارد و چند فرآیند نیاز به دسترسی دقیقا همزمان به آن دارند، همچنین همچنین فرض کنید سیستم عامل هیچ تضمینی بر روی کنترل دسترسی همزمان و عدم خرابی داده های این بخش از حافظه (فایل) نمی دهد. سازوکاری طراحی کنید که با استفاده از آن بتوان برنامه ای نوشت که در چنین سیستمی به هنگام نیاز به دسترسی به یک حافظه (فایل) هیچ خرابی رخ ندهد (امکان استفاده از تابع ()print و همچنین ساختارهای mutex و سمافور را نخواهید داشت)
سیستم عامل هیچ تضمینی برای مدیریت و هماهنگی فایل‌ها به ما نمی‌دهد و از ابزارهای mutex و semaphore بعنوان ابزارهای نرم‌افزاری شایع برای این کنترل نمیتوانیم استفاده کنیم. از تابع sleep هم نمی‌توان استفاده کرد. میتوانیم از فایل‌های قفل‌کننده (Lock Files) استفاده کنیم. فرآیندها برای دسترسی به حافظه‌ی اشتراکی ابتدا وجود فایل قفل‌کننده‌ی مربوطه به آن حافظه‌ی اشتراکی را بررسی میکنند. این فایل به عنوان نشانگر یک قفل عمل میکند.

- اگر فایل قفل‌کننده وجود داشته باشد، فرآیند صبر می‌کند تا قفل آزاد شود. در مدتی که فرآیند صبر می‌کند تا فایل قفل آزاد شود، باید در busy waiting بماند که بهینه نیست.
- اگر فایل قفل‌کننده وجود نداشته باشد، فرآیند فایل قفل‌کننده را ایجاد می‌کند.
  - در ایجاد این قفل باید از یک عملیات *اتمیک* استفاده کنیم.
  - اتمیک بودن سبب می‌شود تا هنگامی که چند فرآیند بخواهند همزمان یک فایل قفل ایجاد کنند فقط یکی از آنان موفق بشود.
  - مثل تابع open با فلگ O_CREATE | O_EXCL در سیستم‌های POSIX
  - یا با directory creation که در بیشتر فایل سیستم‌ها اتمیک است.
- پس از ایجاد حافظه، فرآیند می‌تواند عملیات مورد نظرش را (خواندن یا نوشتن) در حافظه‌ی اشتراکی پیاده کند.
- پس از اتمام کار فرآیند فایل قفل‌کننده را حذف می‌کند تا بقیه‌ی فرآیندها هم بتوانند به آن حافظه‌ی اشتراکی دسترسی پیدا کنند.

(ج) در مدل تبادل پیام با توجه به اینکه حافظه برای نگهداری پیام محدود است، سازوکاری ارائه دهید که بتوانیم (با تقریب خوبی) تعداد نامحدودی پیام در صندوق پیام بفرستیم. میتوانیم از مفهوم بافر نامحدود بهره ببریم. این بافر پیاده‌سازی‌های گوناگونی دارد. از جمله پیاده‌سازی با لیست پیوندی یا با لیست‌های داینامیک. در تئوری می‌توانیم تا بی‌نهایت در این بافر پیام درج کنیم. اما نکته‌ی مهم اینست که در عمل حافظه محدودیت دارد. لیست پیوندی تا

بینهایت پیش نمی‌رود. برای همین میگوییم با تقریب خوبی این حافظه نامحدود است.
ایده‌ی دیگر برای بهبود حافظه اینست که در این پیاده‌سازی یک بیشینه حافظه تعریف کنیم و پس از اینکه بافر پر شد با یک سیاست مشخص شروع به جایگزینی یک پیام انتخابی با پیام جدید کنیم. مثلا سیاست انتخابی می‌تواند این باشد: پیامی که از همه زودتر به بافر اضافه شده است از همه زودتر هم خارج شود(مانند یک صف سیاست FIFO داشته باشد). میتوانیم از یک صف حلقوی (یا لیست پیوندی حلقوی) برای این پیاده‌سازی استفاده کنیم. راه‌های زیادی برای این پیاده‌سازی پیش روی ماست.
هر کدام ممکن است نکات مثبت و منفی گوناگونی داشته یاشند. مثلا یک پیاده‌سازی ممکن است سرعت دسترسی به پیام بیشتری داشته باشد و در ازای آن مجبور به حذف پیام‌های بیشتری شود.
حتی میتوانیم یک قدم جلوتر برویم و از یک حافظه‌ی جانبی (مانند swap در فرآیندها) برای افزایش گنجایش بافر استفاده کنیم. این فیچر پیچیدگی را منطقا زیاد می‌کند اما در تئوری با این روش می‌تونیم پیام‌های بیشتری را به صورت همزمان ذخیره کنیم.

(د) شرایطی را بیان کنید که در آن موازی سازی به هیچ وجه نمی تواند به تسریع محاسبات کمک کند.

- فرض کنید که میخواهید یک فرآیند بسیار سبک را اجرا کنید. در این حالت سربار اجرای موازی آن از سودی که دارد بیشتر می‌شود.
- فرض کنید چند تسک که میخواهید اجرا کنید به هم وابستگی زیادی داشته باشند. این شرایط باعث محدودیت در اجرای موازی میشود. و مانند مورد بالا در نهایت ممکن است سربار آن بسیار بالا برود.
- الگوریتم‌هایی وجود دارند که مانند یک دنباله و به هم متصل اجرا می‌شوند. در این الگوریتم اجرای موازی و شکستن فرآیند به چند تسک ممکن نیست.

(ه) تحقیق کنید که در کدام سیستم عامل ها مدل های چندنخی دو سطحی و چند به چند پیاده سازی شده است و همچنین آیا هنوز استفاده ای از این مدل ها صورت می گیرد یا خبر؟ علت آن را نیز بیان کنید.

مدل چند به چند (Many-to-Many)

- این مدل در برخی نسخه‌های قدیمی‌تر از Solaris (مانند Solaris 2.x) و IRIX پیاده‌سازی شده است.
- در برخی از سیستم‌های قدیمی‌تر یونیکس از این مدل استفاده می‌شد.
- این مدل به تدریج کنار گذاشته شد و امروز به ندرت از آن استفاده می‌شود

مدل دو سطحی (Two-Level)

- در نسخه‌های قدیمی‌تر FreeBSD از این مدل استفاده میشد.
- امروزه به ندرت استفاده می‌شود.

چرا این مدل‌ها دیگر رایج نیستند ؟

۱. پیچیدگی مدیریت: نگاشت نخ‌های کاربر به نخ‌های هسته نیازمند الگوریتم‌های پیچیده است که می‌تواند مشکلاتی مانند گرسنگی منابع (Resource Starvation) ایجاد کند.
۲. بهبود سخت‌افزار: پردازنده‌های چند هسته‌ای مدرن و معماری‌های جدید به مدل‌های ساده‌تر (مانند یک به یک) اجازه می‌دهند به طور کارآمدتری عمل کنند

۳. بهبود سیستم‌های عامل: هسته سیستم‌عامل‌های مدرن قابلیت مدیریت کارآمد نخ‌ها را در مدل یک به یک فراهم کرده است.

(و) سیستم عاملی را فرض کنید که در آن هیچ امکان اشتراک گذاری داده ای میان فرآیند ها وجود نداشته باشد. در این صورت بگویید چگونه می توان داده ای را میان چند فرآیند به اشتراک گذاشت.
اگر نتوان از روش‌های مستقیم برای اشتراک‌گذاری اطلاعات بین فرآیندها استفاده کرد، باید برای انتقال داده از روش‌های غیرمستقیم استفاده کنیم.

- فایل‌های موقتی در دیسک ذخیره شده باشند و فرآیندها بتوانند از طریق آن با هم داده رد و بدل کنند یا میتوانیم سیستم‌عامل را قاطی این مکانیزم کنیم و بخشی از حافظه‌ی اصلی که به یک فایل مرتبط می‌شود در اختیار فرآیندها قرار بگیرد.
- یا مثلا با استفاده از پایگاه‌های داده این عمل صورت گیرد. فرآیندها اطلاعات را در یک پایگاه داده ذخیره و بازیابی کنند.

(ز) فرض کنید در یک سیستم عامل هیچ تضمینی بر روی عدم هم پوشانی بخش داده و هرم نباشد. سازوکاری ارائه کنید که با استفاده از آن در برنامه ی خود بتوانیم از همپوشانی جلوگیری کنیم.
اگر سیستم‌عامل این عدم هم‌پوشانی را مدیریت نمی‌کند، به این معنی است که ما باید به صورت دستی این محدودیت و مرز را مدیریت کنیم. اگر مطمین باشیم که حافظه‌ی هرم به صورت کاملا ایزوله از دیگر بلاک‌های حافظه استفاده می‌شود، ی‌توانیم مطمین باشیم که با بخش داده نیز هم‌پوشانی و تداخلی ندارد. اول از همه یه بخشی از حافظه را به هرم تخصیص می‌دهیم. حال توابع شخصی‌سازی شده‌ای برای تخصیص حافظه از هرم یا آزاد کردن از هرم بنویسیم. با این اقدامات می توانیم مطمین شویم که هرم هیچ تصرفی در بخش داده نخواهد کرد.

- می‌توانیم یک گارد محافظتی نیز اطراف حافظه‌ی هرم قرار دهیم تا مطمین شویم از بخش داده تصرفی در حافظه‌ی هرم صورت نمی‌گرد. . به این گارد صفحات حافظه‌ای نگهبان (Memory Guard Pages) می‌گویند
- تعریف: صفحاتی از حافظه که غیرقابل‌دسترسی‌اند(برای مثال نه می‌توان در آن‌ها نوشت و نه از آن‌ها خواند). اگر برنامه‌ای بخواهد برای ذخیره یا بازیابی به آن‌ها دسترسی بیابد با خطا مواجه خواهد شد
- می‌توانیم قبل از حافظه‌ی هرم و بعد از آن از این گاردهای محافظتی استفاده کنیم. به این طریق می‌توانیم مطمین شویم که حافظه‌ی هرم از محدوده‌ی تعیین شده به آن‌طرف‌تر دسترسی نخواهد داشت.

تمرین ۵

پرسش های زیر دارای نمره ی اضافی هستند و پاسخ به آنها ضروری نیست. در صورت تمایل به دریافت نمره ی اضافی به آنها به صورت کامل پاسخ بدهید.

(آ) فرض کنید در سیستم عامل فراخوانی وجود دارد که پس از فراخوانی آن توسط برنامه فرآیند برای مدتی مسدود شده و پس از آماده شدن داده ی مورد نظر آن داده توسط سیستم عامل با برنامه به اشتراک گذاشته می شود. در این صورت به سوالات زیر پاسخ دهید:

i. از میان دو شیوه ی صندوق پیام و اشتراک حافظه کدام روش برای این منظور مناسب تر است؟ پاسخ خود را توجیه کنید.
اگر سیستم‌عامل بخواهد چنین فراخوانی را داشته باشد از آنجایی که میتواند مسدود کردن و سپس ادامه کار را به راحتی انجام دهد، استفاده از حافظه اشتراکی گزینه مناسب‌تری است چرا که سرعت بیشتری دارد، پیچیدگی خاصی ندارد و احتیاجی به کپی کردن دیتا در فرآیند نیست. سیستم‌عامل پس از اینکه اجرای فرآیند را مسدود کرد، داده را درون حافظه اشتراکی قرار میدهد و آدرس آن را به فرآیند می‌دهد و اجرای آن را از سر میگیرد.

ii. استفاده از روش دیگر چه معایبی خواهد داشت؟
استفاده از انتقال پیام به دلیل سرباری که روی کرنل به‌دلیل system call ها دارد نمیتواند گزینه خوبی برای دیتاهایی با اندازه زیاد باشد و به دلیل اینکه دیتا باید از صف پیام‌ها به حافظه فرآیند منتقل شود تاخیر بیشتری نیز دارد.

iii. به جز اشتراک پیام و اشتراک گذاری حافظه آیا روش بهتری برای انجام این کار سراغ دارید؟ روش خود را به صورت کامل شرح دهید و بگویید چرا این روش از روش های یاد شده در بالا بهتر است.
روش دیگری که می‌توان برای این کار استفاده کرد، فایل‌های موقت هستند که بدین صورت که سیستم‌عامل یک فایل موقت ایجاد کرده و داده‌ها را در آن ذخیره می‌کند و آدرس آن را به فرآیند برمی‌گرداند. در این روش فرآیند می‌تواند از این فایل بخواند، در آن بنویسد و یا حتی آن را ذخیره کنید تا بعدا از این فایل دوباره استفاده کند. این کار پیچیدگی کمتری از دو روش بالا دارد و سرعت آن نیز مناسب خواهد بود.

(ب) شما یک برنامه چند نخی برای پردازش داده های ورودی طراحی کرده اید که از سه نخ مختلف برای پردازش داده ها استفاده می کند. نخ اول باید داده ها را از فایل ها بخواند، نخ دوم داده ها را پردازش کرده و نخ سوم نتایج را ذخیره کند. پس از اجرا، متوجه می شوید که سرعت پردازش کندتر از حد انتظار است و برخی نخ ها به طور غیرضروری منتظر نخ های دیگر می مانند.

(آ) مشکل اصلی در نحوه زمان بندی نخ ها چیست؟
مشکلی اصلی در این است که زمانبندی نخ‌ها به صورتی است که وقتی داده خوانده نشده است، CPU به نخ پردازشی اختصاص داده میشود درحالی که چیزی برای پردازش وجود ندارد یا به نخ ذخیره سازی اختصاص داده میشود درحالی که چیزی برای ذخیره سازی وجود ندارد پس مدت زمانی را که میتوانست به نخی اختصای یابد که میتواند کار مفیدی انجام دهد به نخی اختصاص می‌یابد که باید منتظر بماند.

(ب) چگونه زمان بندی مناسب نخ ها می تواند به بهبود عملکرد برنامه کمک کند؟
اگر زمانبدی نخ‌ها به درستی انجام شود، نتیجه پردازش‌ها به صورت جریانی از داده‌ها ذخیره می‌شود و احتیاجی نیست که کاربر منتظر پایان تمام برنامه باشد تا از نتایج استفاده کند ولی این اتفاق نیازمند زمان‌بندی درست اجرای نخ‌ها است تا زمان کلی اجرای برنامه بیشتر از چیزی که باید نشود چرا که این موضوع باعث هدر رفتن وقت کاربر میشود.

(ج) در این سناریو، چه روش هایی برای جلوگیری از Context Switching غیرضروری و بهینه سازی زمان بندی نخ ها وجود دارد؟

در این سناریو باید از مکانیزم‌هایی استفاده کنیم که همگام‌سازی این نخ‌ها را به ارمغان می‌آورند. در این مکانیزم‌ها تا زمانی که داده‌ها برای پردازش به اندازه کافی نباشند، نخ پردازشی اجرا نمی‌شود و یا تا زمانی که چیزی پردازش نشده است، نخ ذخیره سازی اجرا نمی‌شود. اگر این مکانیزم‌ها را به علاوه اینکه برای هر نخ زمان اجرای مناسبی را در هر زمانی بین Context Switch در نظر بگیریم، می‌تواند از Context Switch های غیر ضروری جلوگیری کنیم و زمان اجرای برنامه را بهینه کنیم.

(د) اگر برنامه شما روی یک سیستم چند هسته ای اجرا شود، چه تفاوت هایی در نحوه زمان بندی نخ ها وجود خواهد داشت؟ چگونه می توان از هسته ها به طور بهینه استفاده کرد؟

چون برنامه رو یک سیستم چند هسته‌ای اجرا میشود دیگر احتیاج نداریم که بین نخ‌ها جابه‌جا شود اما همچنان به مکانیزم‌های همگام‌سازی نیاز داریم تا بتوانیم نخ‌ها را به صورت همزمان اجرا کنیم.

(ج) بررسی کنید آیا ممکن است که فرآیندی در یک بازه زمانی یتیم و سپس در بازه ای دیگر تبدیل به فرآیند زامبی شود؟ برعکس این سناریو محتمل است؟ (تشریح کنید و مثال بزنید)

بله این اتفاق ممکن است بیفتد برای مثال فرآیندی را درنظر بگیرید که terminate شده است درحالی که فرزندش هنوز اجرا می‌شود، این فرزند یتیم شده است و پس از اتمام اجرا ممکن است فرآیند دیگری که در مدیریت این فرآیند را برعهده گرفته است در حالت مسدود باشد و فرآیند فرزند تبدیل به زامبی شود.

برعکس این اتفاق ممکن نیست چرا که فرآیند زامبی، فرآیندی است که اجرای آن به اتمام رسیده است ولی فرآیند پدر هنوز به آن رسیدگی نکرده است پس حتی اگر پدر این فرآیند نیز terminate شود، سیستم‌عامل به فرآیند فرزند رسیدگی و آن را از لیست فرآیندها حذف می‌کند.