OS_LAB_CA4

سجاد تقی زاده ۸۱۰۱۰۲۴۲۵ محمد حسین شفیعی ۸۱۰۱۰۲۴۷۰ دنیز مصطفی زادگان ۸۱۰۱۰۲۶۰۳

1. مفهوم مسیر کنترلی (Path Control) را تعریف کنید. تفاوت مسیرهای کنترلی Interrupt، Syscall و Exception چیست؟

مسیر کنترلی به مسیر اجرای CPU گفته می‌شود که مشخص می‌کند کنترل برنامه چگونه بین فضای کاربر و هسته جابه‌جا می‌شود.

- Syscall: مسیر کنترلی ارادی است؛ برنامهٔ کاربر برای دریافت سرویس از کرنل آن را فراخوانی می‌کند.

- Interrupt: مسیر کنترلی غیرارادی و ناهمزمان است؛ توسط سخت‌افزار (مثل تایمر یا دیسک) ایجاد می‌شود.

- Exception: مسیر کنترلی ناشی از خطا یا شرایط خاص هنگام اجرای دستور (مثل تقسیم بر صفر) است.

2. Kernel Reentrant چیست؟ سناریوی ورود مجدد کرنل در xv6 و خطر عدم همگام‌سازی را توضیح دهید.

Kernel Reentrant یعنی کرنل بتواند قبل از خروج کامل از یک مسیر اجرایی، دوباره وارد کد خود شود (مثلاً به‌علت وقفه) بدون اینکه داده‌های داخلی آن خراب شوند.

- یک پروسه در حالت کاربر، یک Syscall (مثلاً read) اجرا می‌کند.
- CPU از طریق دستور ecall وارد کرنل می‌شود.

- کرنل شروع به تغییر ساختارهای مشترک مثل جدول فایل‌ها یا صف دیسک می‌کند.
- در همین لحظه، یک وقفه دیسک رخ می‌دهد.
- CPU اجرای Syscall را موقتاً متوقف کرده و وارد Interrupt Handler کرنل می‌شود.
- حالا کرنل دوبار به‌طور هم‌زمان در حال اجراست:
    - یک‌بار در مسیر Syscall
    - یک‌بار در مسیر Interrupt

<u>خطر در نبود همگام‌سازی:</u>

اگر داده‌های مشترک قفل نشده باشند:

هر دو مسیر ممکن است هم‌زمان یک ساختار کرنلی را تغییر دهند

این باعث Race Condition می‌شود

نتیجه می‌تواند:

a. خرابی داده‌ها
b. از دست رفتن درخواست‌ها
c. کرش سیستم عامل

3. تفاوت Context Process و Context Interrupt چیست؟

Context یعنی مجموعهٔ اطلاعاتی که CPU برای ادامهٔ اجرای یک کد به آن نیاز دارد (مثل رجیسترها)

Context Process :

- مربوط به اجرای یک پروسه مشخص است
- شامل رجیسترها، stack ،program counter و وضعیت پروسه می‌شود
- هنگام Context Switch بین پروسه‌ها ذخیره و بازیابی می‌شود
- می‌تواند Sleep کند یا Block شود

Context Interrupt :

- هنگام رخ دادن یک وقفه سخت‌افزاری ایجاد می‌شود
- روی همان کانتکست پروسهٔ در حال اجرا سوار می‌شود
- هندلر وقفه اجرا می‌شود، نه یک پروسه جدید
- نباید Sleep یا Block شود
- بعد از پایان وقفه، CPU به اجرای همان پروسه برمی‌گردد

4. چرا کد Context Interrupt نباید Block یا Sleep شود؟

کدی که در Interrupt Context اجرا می‌شود، روی Kernel Stack پروسهٔ جاری و با Interrupts Disabled اجرا می‌گردد و فاقد Process Context مستقل است.

اگر این کد:

- Block یا Sleep شود
- زمان‌بند (Scheduler) قادر به تعویض پروسه (Context Switch) نخواهد بود

- زیرا هندلر وقفه پروسه نیست که دوباره زمان‌بندی شود.

در نتیجه:

- CPU در حالت نامشخص باقی می‌ماند،
- وقفه‌های بعدی از دست می‌روند،
- ممکن است سیستم‌عامل دچار Deadlock یا Kernel Panic شود.

5. چرا در Context Interrupt از Spinlock همراه با Disable Interrupts استفاده می‌شود؟

در Interrupt Context سه محدودیت مهم داریم:

Sleep ممنوع است.

کد باید خیلی کوتاه اجرا شود.

ممکن است وقفه‌ها باعث ورود مجدد کرنل شوند.

چرا Spinlock؟

- چون در Interrupt Context نمی‌توان Sleep کرد
- Spinlock باعث انتظار فعال (Busy Waiting) می‌شود
- برای بخش‌های بحرانی کوتاه مناسب است

Spinlock تنها نوع قفلی است که در Interrupt Context قابل استفاده است

چرا Disable Interrupts؟

فرض کن:

- CPU یک Spinlock را گرفته
- قبل از آزاد کردن قفل، یک وقفه رخ دهد
- هندلر وقفه بخواهد همان قفل را بگیرد

نتیجه:

هندلر وقفه در Spinlock گیر می‌کند

CPU قفل را آزاد نمی‌کند

در Interrupt Context امکان Sleep نیست، بنابراین Spinlock استفاده می‌شود و برای جلوگیری از Deadlock ناشی از وقفهٔ تودرتو، وقفه‌ها قبل از گرفتن قفل غیرفعال می‌شوند.

6. سه روش همگام‌سازی را تعریف کرده و مزایا/معایب را بیان کنید.

Spinlock :

قفل با انتظار فعال (Busy Waiting) که Thread تا آزاد شدن قفل، روی CPU می‌چرخد.

مزایا:

- مناسب برای ناحیه‌های بحرانی کوتاه
- قابل استفاده در Interrupt Context

معایب:

- مصرف بالای CPU

- نامناسب برای انتظارهای طولانی

Sleep Lock:

در صورت در دسترس نبودن قفل، پروسه Sleep می‌شود تا قفل آزاد گردد.

مزایا:

- مصرف کم CPU
- مناسب برای عملیات طولانی مثل I/O

معایب:

- غیرقابل استفاده در Interrupt Context
- سربار Context Switch

Lock-Free Programming:

همگام‌سازی بدون قفل با استفاده از دستورات اتمیک.

مزایا:

- عدم بروز Deadlock
- مقیاس‌پذیری بالا در چندهسته‌ای

معایب:

- پیاده‌سازی بسیار پیچیده
- اشکال‌زدایی دشوار

7. چرا در acquire ابتدا cli اجرا می‌شود؟ سناریوی Deadlock در سیستم تک‌هسته‌ای را توضیح دهید.

برای اینکه:

CPU هنگام داشتن قفل، وارد Interrupt Handler نشود

همان CPU دوباره همان قفل را درخواست نکند

از Deadlock روی یک هسته جلوگیری شود

سناریو:

CPU وارد کرنل می‌شود

(acquirelock) اجرا می‌شود و قفل گرفته می‌شود

قبل از (releaselock)، یک Interrupt رخ می‌دهد

CPU وارد Interrupt Handler می‌شود

هندلر وقفه می‌خواهد همان lock را بگیرد

چون قفل گرفته شده:

a. Sleep ممکن نیست
b. CPU در Spinlock می‌چرخد

کدی که باید قفل را آزاد کند، دیگر اجرا نمی‌شود

نتیجه: Deadlock کامل سیستم

8. چرا xv6 به‌جای cli/sti از pushcli و popcli استفاده می‌کند؟

در کرنل xv6، به‌دلیل وجود نواحی بحرانی تودرتو، استفادهٔ مستقیم از دستورات cli و sti ایمن نیست، زیرا ممکن است وقفه‌ها پیش از خروج کامل از تمام نواحی بحرانی مجدداً فعال شوند.

تابع pushcli با غیرفعال‌سازی وقفه‌ها، وضعیت قبلی وقفه‌ها را ذخیره کرده و یک شمارندهٔ تودرتویی (Nesting Counter) را افزایش می‌دهد تا ورود به ناحیهٔ بحرانی ثبت شود.

تابع popcli این شمارنده را کاهش می‌دهد و تنها در صورتی که شمارنده به صفر برسد، وقفه‌ها را مجدداً فعال می‌کند؛ به این معنا که تمام نواحی بحرانی خاتمه یافته‌اند.

به این ترتیب، pushcli/popcli از فعال‌سازی زودهنگام وقفه‌ها جلوگیری کرده و اجرای ایمن نواحی بحرانی تودرتو در کرنل را تضمین می‌کنند.

بخش دوم: مقیاس پذیری، داده های Per-CPU و پروفایلینگ قفلها در xv6

سوال ۱: چرا زمانی که یک پردازنده یک spinlockرا در اختیار دارد، حتماً باید وقفه ها روی آن پردازنده غیرفعال باشند؟

دلیل اصلی غیرفعال کردن وقفه‌ها (Interrupts) هنگام نگه داشتن یک `spinlock`، جلوگیری از بروز **بن‌بست (Deadlock)** روی همان پردازنده است.

توضیح دقیق سناریوی بن‌بست: اگر وقفه‌ها فعال باشند، ممکن است سناریوی زیر رخ دهد:

1. **گرفتن قفل:** یک کد در کرنل (Kernel Thread) اجرا می‌شود و یک `spinlock` را می‌گیرد (Lock A).
2. **وقوع وقفه:** در حالی که قفل A هنوز در اختیار این کد است، یک وقفه (مثلاً وقفه تایمر یا دیسک) رخ می‌دهد.
3. **توقف اجرا:** پردازنده اجرای کد فعلی را متوقف کرده و به سراغ اجرای "هندلر وقفه" (Interrupt Handler) می‌رود.
4. **درخواست مجدد قفل:** اگر هندلر وقفه برای انجام کارش نیاز به همان قفل (Lock A) داشته باشد و تلاش کند آن را بگیرد (Acquire)، وارد حلقه انتظار (Spin) می‌شود.
5. **بن‌بست (Deadlock):**
    - هندلر وقفه منتظر است تا Lock A آزاد شود.
    - اما Lock A در اختیار کدی است که توسط همین هندلر وقفه متوقف (Preempt) شده است و تا زمانی که هندلر وقفه تمام نشود، آن کد اجرا نخواهد شد تا قفل را آزاد کند.
    - نتیجه این است که پردازنده تا ابد در هندلر وقفه می‌چرخد و سیستم قفل می‌کند.

به همین دلیل، در xv6 (و اکثر سیستم‌عامل‌ها) توابع `acquire` قبل از گرفتن قفل، وقفه‌ها را روی پردازنده غیرفعال می‌کنند (`cli`) و تنها زمانی که قفل آزاد شد (`release`)، وضعیت وقفه‌ها را به حالت قبل برمی‌گردانند.

سوال ۲: اگر وقفه فعال بماند و هندلر وقفه (وخب)Interrupt Handlerاهد همان قفل را بگیرد، چه نوع بن بستی ( )Deadlockرخ میدهد؟ (با رسم شکل یا مثال توضیح دهید).
این وضعیت منجر به یک **بن‌بست تک‌پردازنده‌ای (Single-CPU Deadlock)** می‌شود. در این حالت پردازنده در یک حلقه بی‌پایان گرفتار می‌شود زیرا منتظر رخدادی است (آزاد شدن قفل) که خودش جلوی انجام آن را گرفته است.

شرح سناریو (مثال):

فرض کنید یک متغیر مشترک به نام `Data` داریم که با قفل `Lock` محافظت می‌شود.

1. **گام اول (Process Context):** هسته سیستم‌عامل (Kernel) در حال اجرای یک کد عادی است و قفل `Lock` را می‌گیرد (Acquire) تا `Data` را ویرایش کند.
   - *وضعیت:* `Lock` = **Held (گرفته شده)** توسط کرنل.
2. **گام دوم (Interrupt):** ناگهان یک وقفه (مثلاً وقفه شبکه) روی **همان پردازنده** رخ می‌دهد. چون وقفه‌ها غیرفعال نشده‌اند، پردازنده کار فعلی را رها کرده و بلافاصله به سراغ اجرای **هندلر وقفه (Interrupt Handler)** می‌رود.
3. **گام سوم (Interrupt Handler):** هندلر وقفه برای پردازش داده‌های رسیده، نیاز دارد به همان `Data` دسترسی داشته باشد، بنابراین تلاش می‌کند قفل `Lock` را بگیرد (Acquire).
4. **گام چهارم (Deadlock):** هندلر وقفه می‌بیند که قفل اشغال است، پس وارد حلقه انتظار (Spin) می‌شود تا قفل آزاد شود.
   - **مشکل:** قفل دست کیست؟ دست کدی که همین هندلر وقفه اجرای آن را متوقف کرده است.
   - **نتیجه:** کد اصلی تا وقتی هندلر وقفه تمام نشود، اجرا نخواهد شد (پس نمی‌تواند قفل را آزاد کند). هندلر وقفه هم تا قفل آزاد نشود، تمام نخواهد شد.

سوال ۳: با توجه به اسلایدهای درس، اگر چندین هسته پردازنده بخواهند به طور مداوم یک متغیر مشترک سراسری (مثلاً)GlobalCounterرا تغییر دهند، پدیدهای رخ میدهد که باعث

کندی سیستم میشود. پروتکلهای همگام سازی حافظه نهان (مانند)MESIدر این شرایط چه میکنند؟

پدیده‌ای که رخ می‌دهد اصطلاحاً **"Cache Line Bouncing"** (پرش خط حافظه نهان) یا **"Cache Ping-Pong"** نامیده می‌شود که منجر به ترافیک شدید روی گذرگاه (Bus) و کندی سیستم می‌شود.

پروتکل **MESI** در این شرایط به صورت زیر عمل می‌کند:

1. **بی‌اعتبارسازی (Invalidation):** طبق پروتکل MESI، وقتی یک هسته (مثلاً Core 1) می‌خواهد روی متغیر مشترک بنویسد، باید مالکیت انحصاری (Modified/Exclusive) آن خط کَش (Cache Line) را به دست آورد. بنابراین، پروتکل پیامی می‌فرستد که باعث می‌شود کپیِ آن متغیر در کَشِ تمام هسته‌های دیگر به وضعیت **Invalid (I)** تغییر کند.

سوال ۴: چرا استفاده از متغیرهای محلی ( )Per-CPUاین مشکل را تا حد زیادی کاهش میدهد؟ به طور خلاصه توضیح دهید چگونه نگه داشتن شمارنده ها بهصورت per-CPUمیتواند تعداد invalidation های کش را کم کند.

استفاده از متغیرهای **Per-CPU** مشکل رقابت بر سر کش را به روش زیر حل می‌کند:

۱. **جداسازی داده‌ها (Data Partitioning):** به جای داشتن یک متغیر سراسری واحد که همه هسته‌ها برای نوشتن روی آن بجنگند، به هر هسته یک نسخه اختصاصی از متغیر داده می‌شود (مثلاً یک آرایه که هر اندیس آن مخصوص یک CPU است).

۲. **حذف بی‌اعتبارسازی (Eliminating Invalidation):** در پروتکل‌های کش (مثل MESI)، وقتی هسته A روی متغیر اختصاصی خودش می‌نویسد، چون آدرس حافظه‌ی آن متفاوت از متغیر هسته B است، نیازی ندارد پیامی بفرستد تا کش هسته B را بی‌اعتبار (Invalidate) کند.

۳. **نتیجه:** هر پردازنده می‌تواند متغیر خود را در کش خودش نگه دارد (Cache Hit بالا) و بدون ایجاد ترافیک روی گذرگاه سیستم (Bus) و بدون مزاحمت برای سایر هسته‌ها، مقدار آن را

تغییر دهد. تنها زمانی که نیاز به مقدار "کل" باشد (که رخدادی نادرتر است)، مقادیر تک‌تک هسته‌ها با هم جمع می‌شوند.

سوال ۵: تفاوت اصلی بین (spinlockدسته اول توابع در)xv6و (sleeplockدسته دوم توابع) در چیست؟ کدام یک باعث «انتظار مشغول» ( )Busy Waiting میشود و کدام یک پردازنده را به پروسس دیگری واگذار میکند؟

۱. قفل جرخشی (Spinlock):
• نوع انتظار: باعث انتظار مشغول (Busy Waiting) می شود.
• عملکرد: زمانی که یك بردازنده به قفل بسته می رسد، در یک حلقه تکرار (Loop) مداوم باقی می ماند و وضعیت ففل را چک می کند تا آزاد شود. در این حالت پردازنده رها نمی شود.
• کاربرد: مناسب برای انتظار های کوتاه است.

۲. قفل خواب (Sleeplock):
• نوع انتظار: پردازنده را به پروسس دیگری واگذار می کند (Yields CPU).
• عملکرد: زمانی که قفل اشغال است، بروسه فعلی به خواب (Sleep) می رود (وضعیت Blocked) و سیستم عامل پردازنده را به پروسه دیگری می دهد تا زمانی که قفل آزاد و پروسه بیدار شود.
• کاربرد: مناسب برای انتظار های طولانی (مانند عملیات ورودی/خروجی O/1) است.

🥲

بخش سوم: قفل های اولویت دار و مسئله گرسنگی

سوال ۱: آیا در طراحی plock امکان گرسنگی برای پردازه‌های با اولویت پایین وجود دارد؟ یک سناریوی دقیق بنویسید.

پاسخ:

بله، امکان گرسنگی وجود دارد.

از آنجا که سیاست این قفل همیشه انتخاب "بالاترین اولویت" است، اگر یک جریان مداوم از پردازه‌های با اولویت بالا وجود داشته باشد، پردازه اولویت پایین هرگز انتخاب نمی‌شود.

سناریوی گرسنگی:

۱. فرض کنید پردازه A با اولویت ۱۰ درخواست قفل می‌کند. چون قفل اشغال است، به صف می‌رود و می‌خوابد.

۲. در همین حین، پردازه B با اولویت ۱۰۰ می‌آید و در صف قرار می‌گیرد.

۳. قفل آزاد می‌شود. سیستم لیست را می‌گردد و چون 100 از 10 بزرگتر است، قفل را به پردازه B می‌دهد.

۴. قبل از اینکه کار پردازه B تمام شود، پردازه C با اولویت ۱۰۰ وارد صف می‌شود.

۵. پردازه B قفل را آزاد می‌کند. سیستم دوباره می‌گردد و بین A (اولویت ۱۰) و C (اولویت ۱۰۰)، قفل را به C می‌دهد.

۶. اگر همیشه قبل از تمام شدن کار پردازه جاری، یک پردازه جدید با اولویت بالا وارد شود، نوبت به A هرگز نمی‌رسد و تا ابد در صف می‌ماند.

سوال ۲:مکانیزم قفل بلیطی (Ticket Lock) که شبیه سیستم نوبت‌دهی نانوایی است را توضیح دهید.

پاسخ:

قفل بلیطی دقیقاً مثل سیستم نوبت‌دهی در بانک یا نانوایی عمل می‌کند تا نظم (FIFO) را رعایت کند. این قفل از دو شمارنده (Counter) استفاده می‌کند:

۱. شمارنده نوبت‌دهی (Ticket Counter): هر کس که می‌آید، یک شماره می‌گیرد و این عدد یکی زیاد می‌شود (مثلاً مشتری شماره ۵).

۲. شمارنده اعلام نوبت (Now Serving): این عدد نشان می‌دهد الان نوبت کیست (مثلاً باجه می‌گوید: شماره ۴).

نحوه کار:

- وقتی پردازه‌ای قفل می‌خواهد، یک شماره (Ticket) می‌گیرد.
- چک می‌کند که آیا Ticket == Now Serving است؟
- اگر برابر بود، وارد می‌شود. اگر نبود، منتظر می‌ماند تا عدد روی تابلو به عدد بلیط او برسد.
- وقتی کارش تمام شد، عدد Now Serving را یکی زیاد می‌کند تا نفر بعدی وارد شود.

سوال ۳:قفل اولویت‌دار را با قفل بلیطی از نظر انصاف، پیچیدگی و گرسنگی مقایسه کنید.

۱. از نظر انصاف :

- **قفل بلیطی:** این قفل **کاملاً عادلانه** است. رفتار آن دقیقاً مانند صف نانوایی است؛ یعنی هر پردازه‌ای که زودتر درخواست دهد، زودتر هم قفل را می‌گیرد سیاست FIFO. هیچکس نمی‌تواند صف را دور بزند.

- **قفل اولویت‌دار :** این قفل **ناعادلانه** است. در اینجا زمان ورود اهمیتی ندارد و فقط "اولویت" مهم است. ممکن است پردازه‌ای مدت‌ها منتظر باشد، اما یک پردازه جدید با اولویت بالاتر بیاید و زودتر از او وارد شود.

۲. از نظر پیچیدگی پیاده‌سازی و سربار :

- **قفل بلیطی :** این قفل **سربار بسیار کمی** دارد. پیاده‌سازی آن فقط نیاز به دو متغیر عددی ساده و عملیات جمع دارد. حافظه خاصی مصرف نمی‌کند و بسیار سریع است.

- **قفل اولویت‌دار :** این قفل **سربار محاسباتی بالایی** دارد. برای پیاده‌سازی آن باید حافظه پویا اختصاص داد، لیست پیوندی ساخت و برای پیدا کردن نفر بعدی، کل لیست را جستجو کرد که زمان‌بر است.

۳. از نظر گرسنگی :

- **قفل بلیطی :** این قفل در برابر گرسنگی **ایمن است**. چون صف خطی است، نوبت به همه می‌رسد و هیچ پردازه‌ای تا ابد پشت قفل نمی‌ماند.

- **قفل اولویت‌دار :** این قفل در برابر گرسنگی **ایمن نیست**. اگر همیشه پردازه‌های مهم اولویت بالا وارد سیستم شوند، نوبت به پردازه‌های کم‌اهمیت نمی‌رسد و آن‌ها دچار گرسنگی می‌شوند.