-1

- علت غیرفعال کردن وقفه چیست؟ توابع pushcli و popcli به چه منظور استفاده شده و چه تفاوتی با cli و sti دارند؟

غیرفعال کردن وقفه‌ها برای اطمینان از اجرای صحیح و بی‌وقفه کدهایی که نیازمند کنترل کامل CPU هستند، انجام می‌شود. این فرآیند به سیستم اجازه می‌دهد که هنگام اجرای یک بخش حساس از کد، وقفه‌ای توسط سخت‌افزار یا نرم‌افزار رخ ندهد و باعث اختلال در اجرای آن بخش نشود.

توابع popcli و pushcli برای مدیریت دقیق‌تر وقفه‌ها استفاده می‌شوند. این توابع برخلاف دستورهای استاندارد cli و sti که فقط وقفه‌ها را به ترتیب غیرفعال یا فعال می‌کنند، وضعیت فعلی وقفه‌ها را ذخیره و بازیابی می‌کنند. این ذخیره‌سازی و بازیابی به سیستم اجازه می‌دهد که با اطمینان بیشتری وقفه‌ها را برای مدت کوتاهی غیرفعال کند و سپس به حالت قبلی بازگرداند.

تابع pushcli وضیعت وقفه را در فلگ وقفه ذخیره می کند و سپس در استک پوش می کند.این فرایند کمک می کند تا تعداد دفعاتی که وقفه ها غیر فعال شده اند را داشته باشیم

تابع popcli وضیعت فلگ وقفه را از استک می خواند و اگر غیر فعال بود آن را می خواند.

توابع popcli و puchcli برخلاف sti و cli یک سیستم شمارشی دارند که مشخص می کند وقفه ها چند بار غیر فعال شده اند و در صورتی که این شمارنده به 0 برسد وقفه ها فعال می شوند که روش ایمن تری در سناریو های پیچیده مانند spinlock است

-2

- حالات مختلف پردازه ها در xv6 را توضیح دهید. تابع sched چه وظیفه ای دارد؟

حالات مختلف پردازه‌ها درxv6 :

1. **UNUSED** :پردازه‌ای که هنوز تخصیص داده نشده یا قبلاً اجرا شده و منابع آن بازپس‌گیری شده‌اند.
2. **EMBRYO**:حالتی که پردازه تازه ایجاد شده است اما هنوز آماده اجرا نیست. این حالت زمانی رخ می‌دهد که یک پردازه جدید با استفاده از سیستم‌کال fork ساخته می‌شود.
3. **SLEEPING** :پردازه‌ای که منتظر وقوع یک رویداد خاص است .پردازه در این حالت به CPU نیازی ندارد و در صف انتظار قرار می‌گیرد.
4. **RUNNABLE** :پردازه آماده اجرا است و منتظر تخصیص CPU می‌باشد.این حالت به معنای آمادگی کامل برای اجرا است، اما ممکن است به دلیل وجود پردازه‌های دیگر در صف، اجرا نشده باشد.
5. **RUNNING**:پردازه‌ای که در حال حاضر توسط CPU در حال اجرا است.
6. **ZOMBIE** :پردازه‌ای که اجرای آن به پایان رسیده است اما اطلاعات مربوط به آن هنوز توسط پردازه والد بازیابی نشده است.

وظیفه تابع sched

تابع sched مسئول تعویض پردازه‌ها (Context Switching) در **xv6** است. این تابع بخشی از مکانیزم برنامه‌ریزی (Scheduler) است و وظایف زیر را برعهده دارد:

1. **ذخیره وضعیت پردازه فعلی:**
   - زمانی که یک پردازه نیاز به انتظار یا متوقف شدن دارد، schedوضعیت فعلی آن پردازه (مانند رجیسترها و شمارنده برنامه) را ذخیره می‌کند.
2. **انتخاب پردازه جدید برای اجرا:**
   - تابع sched به برنامه‌ریز (Scheduler) این امکان را می‌دهد که یک پردازه جدید از لیست پردازه‌های قابل اجرا (RUNNABLE) انتخاب کند.
3. **تغییر Context به پردازه جدید:**
   - پس از انتخاب پردازه جدید، تابع schedوضعیت آن را بارگذاری کرده و اجرای آن را آغاز می‌کند.

-3

- یکی از روش های سینک کردن این حافظه های نهان با یکدیگر روش Modified-Shared-Invalid است. آن را به اختصار توضیح دهید. (اسلاید های موجود در منبع اول کمک کننده شما خواهند بود)

ابتدا توضیح می دهیم خط کش (cache line) چیست:

تعریف:
خط کش، یک واحد پایه‌ای برای ذخیره داده‌ها در حافظه نهان است. کش‌ها داده‌ها را به صورت قطعه‌های کوچک با اندازه ثابت (معمولاً 32، 64، یا 128 بایت) ذخیره می‌کنند که به آن‌ها خط کش می‌گویند.

حالات پروتکل MSI

1. **Modified**
   - خط کش در این حالت تغییر داده شده است و نسخه آن در حافظه اصلی قدیمی است.
   - این خط در کش پردازنده فقط به صورت محلی موجود است و هیچ کش دیگری نسخه‌ای از آن ندارد.
   - اگر پردازنده دیگری نیاز به این داده داشته باشد، کش باید نسخه اصلاح‌شده را به حافظه اصلی بنویسد تا دیگر پردازنده‌ها بتوانند از آن استفاده کنند.
2. **Shared**
   - خط کش در این حالت با نسخه حافظه اصلی هماهنگ است و توسط چندین پردازنده قابل خواندن است.
   - پردازنده‌ها فقط می‌توانند این داده را بخوانند. عملیات نوشتن نیازمند تغییر حالت به **Modified** یا **Invalid** است.
3. **Invalid**
   - خط کش در این حالت نامعتبر است و حاوی داده‌ای نیست که پردازنده بتواند به آن اعتماد کند.

- اگر پردازنده بخواهد به این خط دسترسی پیدا کند، باید نسخه جدید را از حافظه اصلی یا کش دیگر پردازنده‌ها بارگیری کند.

مکانیزم عملکرد پروتکل MSI :

- پروتکل MSI با استفاده از **بروزرسانی پیام‌ها** بین کش‌ها و حافظه اصلی، وضعیت خطوط کش را بین پردازنده‌ها همگام‌سازی می‌کند.
- این پروتکل بر اساس دو نوع عملیات عمل می‌کند:

1. **عملیات خواندن :**
   - اگر خط مورد نیاز در حالت **Shared** یا **Modified** باشد، کش می‌تواند داده را مستقیماً استفاده کند.
   - اگر خط در حالت **Invalid** باشد، کش باید داده را از حافظه اصلی یا کش دیگر پردازنده‌ها بخواند.
2. **عملیات نوشتن :**
   - اگر خط در حالت **Modified** باشد، داده مستقیماً نوشته می‌شود.
   - اگر خط در حالت **Shared** باشد، ابتدا به حالت **Invalid** یا **Modified** تغییر می‌یابد.
   - اگر خط در حالت **Invalid** باشد، کش باید داده را بارگیری کند و سپس عملیات نوشتن انجام شود.

-4

- یکی از روشهای همگام سازی استفاده از قفل هایی معروف به قفل بلیت[15] است. این قفلها را از منظر مشکل مذکور در بالا بررسی نمایید.

مکانیزم قفل بلیت:

1. **ساختار قفل بلیت:**
   - **Now serving:** شماره پردازه‌ای که در حال حاضر مجاز به دسترسی به منبع است.
   - **Next Ticket:** شماره بعدی که منتظر دریافت اجازه دسترسی است.
2. **نحوه کار:**
   - هر پردازه یک **بلیت** (یا شماره صف) دریافت می‌کند که با افزایش مقدار متغیر **Next Ticket** مشخص می‌شود.
   - پردازه تا زمانی که مقدار **Now Serving** برابر با شماره بلیت آن شود، منتظر می‌ماند.
   - پس از اتمام کار، پردازه مقدار **Now Serving** را افزایش می‌دهد تا پردازه بعدی بتواند دسترسی پیدا کند

- دو مورد از معایب استفاده از قفل با امکان ورود مجدد را بیان نمایید.

دیباگ سختی دارند چون وقتی یک ریسه می تواند چند بار قفل را بگیرد ممکن است مسائلی مثل بن بست رخ دهد و نتوانیم بفهمیم کی قفل گرفته شده است.

سربار زیادی نیاز دارند زیرا یک شمارنده باید داشته باشیم که هر قفل چند بار گرفته شده است

- یکی دیگر از ابزار های همگام سازی قفل Read-Write lock است. نحوه کارکرد این قفل را توضیح دهید. و در چه مواردی این قفل نسبت به قفل با امکان ورود مجدد برتری دارد.

عملکرد قفل: اجازه می دهد چند ریسه همزمان یک منبع را بخوانند.موقع خواندن اجازه نوشتن نداریم.همچنین وقتی یک عملیات نوشتن در حال انجام است همه ی نوشتن ها و خواندن های دیگر بلاک می شود.

نسبت به قفل های با ورود مجدد concurreny بیشتری دارند چون چند خواننده اجازه ی خواندن همزمان دارند.یوتیلیزیشن ریسورس ها نیز در جاهایی که چندین خواننده داریم بهتر است زیرا فقط یک ریسه اجازه دسترسی ندارد.