به نام خدا

آزمایشگاه سیستم عامل

پروژه سوم آزمایشگاه سیستم عامل

(زمان بندی پردازه ها)

گروه ۶

اعضای گروه :

محمد سعادتی	**۸۱۰۱۹۸۴۱۰**
محمد عراقی	**۸۱۰۱۹۸۴۳۶**
سید عماد امامی	**۸۱۰۱۹۸۵۲۷**

زمان بندی در xv6

۱) چرا فراخوانی تابع sched() ،منجر به فراخوانی تابع scheduler() می شود؟ (منظور توضیح شیوه اجرای فرایند است.)

همانطور که در تابع sched() در proc.c دیده میشود (خطوط ۳۶۵ تا ۳۸۱)، این تابع وظیفه دارد وضعیت cpuرا بررسی کند و در صورت وجود مشکلاتی همچون قفل بودن ptable (صف پراسس ها) یا cpu یا حالتی که پراسسی در حال ران شدن باشد یا schedule قابل اینتراپت نباشد ، به ما اطلاع میدهد که چنین مشکلاتی وجود دارد و از panic استفاده میکند .

اما در غیر این صورت پراسس را آماده ی scheduling کرده و با استفاده از تابع swtch() و استفاده از -mycpu()>scheduler منجر به فراخوانی تابع scheduler() میشود .

۲) صف پردازه هایی که تنها منبعی که برای اجرا کم دارند پردازنده است، صف آماده ۱۴ یا صف اجرا نام دارد. در xv6 صف آماده مجزا وجود نداشته و از صف پردازه ها بدین منظور استفاده می گردد. در زمان بند کاملاً منصف در لینوکس، صف اجرا چه ساختاری دارد؟

برخلاف scheduling بر اساس الگوریتم RR که از صف پردازه ها استفاده شده و بر اساس پراسس ها و به طور گردشی عمل می کند ، در الگوریتم CFS که در سیستم عامل لینوکس (از ورژن 2.6.23 به بعد) استفاده میشود ، صف ما به صورت per-CPU run queues است یعنی به جای اینکه بر اساس پراسس ها های فعال و غیر فعال باشد ، بر اساس وضعیت CPU عمل میکند و نود های آن به صورت تایم اردر بوده و توسط ساختار red-black tree سورت شده و بررسی میشوند و در هر مرحله پراسس با کمترین زمان اجرا و بیشترین اولویت که در درخت مان همان چپ ترین نود است ، اول اجرا میشود .

اگر کار پراسس در این مدت تمام شد از درخت حذف می شود ، اما اگر در زمان تخصیص داده شده کامل نشد درخت دوباره بر اساس زمان صرف شده سورت میشود و دوباره چپ ترین نود درخت انتخاب میشود .

در کل به طور ساده CFS را در لینوکس میتوان پیاده سازی به صورت weighted fair queuing (WFQ) خلاصه کرد و CPU بر اساس وزن thread ها بین آن ها پخش میشود .

۳) همان طور که در پروژه اول مشاهده شد، هر هسته پردازنده در xv6 یک زمان بند دارد. در لینوکس نیز به همین گونه است. این دو سیستم عامل را از منظر مشترک یا مجزا بودن صف های زمان بندی بررسی نمایید. و یک مزیت و یک نقص صف مشترک نسبت به صف مجزا را بیان کنید.

درxv6 یک صف مشترک برای همه پرازنده ها داریم که به شکل زیر تعریف شده است:

```
struct {
 struct spinlock lock;
 struct proc proc[NPROC];
} ptable;
```

در این ساختمان یک قفل برای مدیریت دسترسی های همزمان و یک صف از پردازه ها وجود دارد. در هنگام استفاده قفل را اعمال می کنیم سپس ptable را رها می کنیم. ولی در لینوکس برای هر پردازنده یک صف مخصوص به خود در نظر گرفته شده است.

یک مزیت برای صف مشترک نسبت به صف مجزا این است که نیازی به هندل کردن load پردازنده ها نداریم (مثلا نیازی به load balancing نداریم) زیرا همه پردازنده ها در یک صف هستند. یک نقص صف مشترک نسبت به صف مجزا این است که باید دسترسی های همزمان به صف را بررسی کنیم که برای این مسئله از روش های locking استفاده می شود.

۴) در هر اجرای حلقه، ابتدا برای مدتی وقفه فعال می گردد. علت چیست؟ آیا در سیستم تک هسته ای به آن نیاز است؟

در وضعیتی که هیچ پردازه RUNNABLE وجود ندارد و همه پردازه ها در حال ورودی گرفتن یا آماده به خروجی دادن می باشند، اگر وقفه وجود نداشته باشد و غیرفعال باشد، عمل ورودی و خروجی هرگز تمام نمی شود پس برای اینکه عمل ورودی و خروجی به درستی صورت گیرد در هر حلقه برای مدتی وقفه فعال می شود تا این وضعیت رخ ندهد.

بله. زیرا این موضوع وابسته به تعداد هسته ها نیست و ممکن است همین اتفاق مجدد رخ دهد بنابراین باید وقفه داشته باشیم.

۵) وقفه ها اولویت بالاتری نسبت به پردازه ها دارند. به طور کلی مدیریت وقفه ها در لینوکس در دو سطح صورت می گیرد. آن ها را نام برده و به اختصار توضیح دهید.

اولویت این دو سطح مدیریت نسبت به هم و نسبت به پردازه ها چگونه است؟

مدیریت وقفه ها در صورتی که بیش از حد زمان بر شود، می تواند منجر به گرسنگی پردازه ها گردد. این می تواند به خصوص در سیستم های بی درنگ مشکل ساز باشد. چگونه این مشکل حل شده است؟

این سوال از جنبه های مختلفی قابل بررسی است که برای مثال به دو جنبه و معیار آن اشاره میکنیم :

۱) دو سطح مدیریت وقفه در لینوکس به شکل زیر میباشند :

IRQ sharing

در این حالت مدیریت کننده وقفه ، چندین interrupt service routins (ISRs) را اجرا می کند و هر ISR یک تابع مربوط به یک دیوایس است که IRQ line را شیر می کند .

به دلیل اینکه از قبل نمیتوان فهمید که کدام دیوایس سبب اجرای IRQ شده ، هر سرویس روتین اجرا می شود تا مشخص شود که آیا دیوایس مربوط به آن نیاز به بررسی دارد یا خیر . که در این صورت ISR مربوطه تمام عملیات های لازم در صورت اعلام وقفه توسط دیوایس را اجرا می کند .

IRQ dynamic allocation

در این حالت هر IRQ line به یک دیوایس درایور در آخرین لحظه ممکن مربوط است .

مثلا IRQ line مربوط به یک فلاپی دیوایس تنها زمانی که کاربر به آن فلاپی دسترسی پیدا میکند allocate میشود و در این صورت یک IRQ vector میتواند برای چندین دیوایس استفاده شود حتی اگر IRQ line مشترکی نداشته باشند . البته این دیوایس ها نمیتوانند همزمان در حال اجرا باشند .

۲) جنبه ی دیگری که میتوان مدیریت وقفه در لینوکس را از آن بررسی کرد به صورت زیر است :

لینوکس مدیریت وقفه را به دو دسته بندی تقسیم میکند : top half و bottom half .

Top half به صورت IRS استاندارد می باشد که در اجرای آن وقفه های بازگشتی غیر فعال اند . در این حالت وقفه های با خط یکسان غیرفعال اند اما وقفه های دیگر میتوانند اجرا شوند . قسمت bottom half سرویس روتین به صورتی اجرا میشود که تمام وقفه ها در آن فعال اند و یک miniature scheduler بررسی میکند تا نیمه ها باعث

وقفه در خودشان نشوند .bottom-half scheduler هر زمان که یک IRS خارج میشود به صورت اتوماتیک اجرا میشود.

به این صورت هسته می تواند در جواب هر وقفه بدون اینکه خودش دچار وقفه شود پراسسینگ را انجام دهد و اگر در هنگام اجرای یک bottom half یک وقفه رخ دهد ، میتواند درخواست اجرای آن را بکند اما تا زمانی که اجرای آن تمام شود به تاخیر می افتد . هر اجرای bottom half میتواند توسط یک top half دچار وقفه شود اما توسط bottom half مشابه این امکان وجود ندارد .

اینگونه interrupt handler میتواند نیمه های پایینی را اجرا کند و هر زمان وارد یک critical section شدیم ، با غیر فعال کردن bottom half های مربوطه می تواند از ایجاد ثقفه توسط سکشن های مشابه خودداری کند . سپس می توان bottom half هایی که توسط top half در صف قرار گرفته اند را دوباره اجرا کند .

برای حل مشکل گرسنگی پردازه ها از روش aging استفاده میشود و که با ارسال پردازه هایی که مدت زیادی است اجرا نشده اند به صف های بالاتر و با اولویت بیشتر میتواند اولویت اجرای آن ها را بیشتر کرده تا دچار گرسنگی نشده و پردازه انجام شود .