آزمایشگاه سیستم عامل

پروژه سوم: زمان بندی پردازه ها

تاریخ تحویل:22 آذر

علی کرامتی

علی پاکدل صمدی

محمد هنرجو

زمان بندی در xv6

1. **چرا فراخوانی تابع sched() ،منجر به فراخوانی تابع scheduler() میشود؟ (منظور توضیح شیوه اجرای فرایند است).**

یک پردازه که می خواهد پردازنده را رها کند باید process table lock را به دست آورد و lockهای دیگر را رها کند و state خود را آپدیت کند (proc->state) سپس sched() را فرا می خواند. sched() دوباره شروط را چک می کند و در نهایت swtch() را فرا می خواند تا context switch را انجام دهد. (context حال حاضر را در proc->context ذخیره کند و context زمان بند در cpu->scheduler را اجرا کند.)

2. **صف پردازه هایی که تنها منبعی که برای اجرا کم دارند پردازنده است، صف آماده یا صف اجرا نام دارد. در xv6 صف آماده مجزا وجود نداشته و از صف پردازه ها بدین منظور استفاده میگردد.در زمانبند کاملاً منصف در لینوکس، صف اجرا چه ساختاری دارد؟**

زمان بند کاملا منصف در لینوکس بر اساس صف اجرای پردازه ساخته شده که با ساختار درخت های قرمز-سیاه پیاده سازی و مرتب شده اند.

3. **همانطور که در پروژه اول مشاهده شد، هر هسته پردازنده در xv6 یک زمانبند دارد. در لینوکس نیز به همینگونه است. این دو سیستم عامل را از منظر مشترک یا مجزا بودن صف های زمانبندی بررسی نمایید. و یک مزیت و یک نقص صف مشترک نسبت به صف مجزا را بیان کنید.**

لینوکس از صف های زمان بندی مجزا استفاده می کند ولیxv6 ازیک صف مشترک استفاده می کند.

مزیت: در صف های مجزا نیاز به مکانیزم load balancing هستیم که در صف مشترک نیازی به این مکانیزم نیست.

نقض: در صف مشترک نیاز به این داریم که بتوانیم دسترسی های همزمان به صف را کنترل کنیم ولی در صف مجزا چنین کاری نیاز نیست.

4. در هر اجرای حلقه، ابتدا برای مدتی وقفه فعال میگردد. علت چیست؟

چون که ممکن است که پردازه ی قابل اجرایی وجود نداشته نباشد زیرا که پردازه ها در انتظار عملیات I/O هستند و اگر زمان بند وقفه های رو همیشه غیرفعال می گذاشت عملیات I/O هیچ وقت انجام نمی شد.

آیا در سیستم تک هسته ای به آن نیاز است؟

بله باز هم به علت احتمال وجود سناریو بالا به این عمل نیاز داریم.

5. وقفه ها اولویت بالاتری نسبت به پردازه ها دارند. به طور کلی مدیریت وقفه ها در لینوکس در دو سطح صورت میگیرد. آنها را نام برده و به اختصار توضیح دهید.
اولویت این دو سطح مدیریت نسبت به هم و نسبت به پردازه ها چگونه است؟

لینوکس فرایند مدیریت وقفه را به دو سطح تقسیم کرده : نیمه بالایی و نیمه پایینی. نیمه ی بالایی سرویس استاندارد مدیریت وقفه هاست که در آن رخداد وقفه های دیگربا همان شماره غیرفعال است. مدیریت وقفه در نیمه ی پایینی به گونه ای است که وقفه ها غیرفعال نیستند ولی یک زمانبندی وجود دارد که باعث می شود که هیچ وقفه ی نیمه پایینی باعث وقفه در خود نشود.

این تقسیم بندی باعث می شود که هر محاسبه پیچیده ای را در جواب به یک وقفه انجام بدهد بدون اینکه نگران مختل شدن خودش توسط وقفه ی دیگری باشد.

این ساختار نیمه ی پایینی و نیمه ی بالایی باعث می شود که وقتی وقفه ها در نیمه پایینی هستند و هسته در حال عملیاتی بحرانی است می تواند تمام نیمه ی پایینی را غیرفعال کند که باعث جلوگیری از وقفه در عملیات شود و بعد از آن عملیات می تواند دوباره آن ها را فعال کند و وقفه های در نیمه ی پایینی اجرا شوند.

مدیریت وقفه ها در صورتی که بیش از حد زمانبر شود، میتواند منجر به گرسنگی پردازه ها گردد .این میتواند به خصوص در سیستمهای بی درنگ مشکل ساز باشد. چگونه این مشکل حل شده است؟

برا جلو گیری از گرسنگی وقفه ها ان وقفه ی زمان بر را به نیمه ی پایینی انتقال می دهند.

خروجی تست :foo

```
exec shrnn failed
$ shrrn 9 4
$ pi
name      pid      state      queue_level      cycle      arrival      HRNN      MHRRN
.....................................................................................
init      1      SLEEPING        1             10          3           7          313
sh      2      SLEEPING        1             26          9           7          122
foo      8      RUNNING        3             5752          429           7          4
foo      7      SLEEPING        1             1          430           7          2883
foo      6      SLEEPING        1             2          429           2          1441
foo      9      RUNNABLE        2             5749          430           4          3
pi      19      RUNNING        1             2          6187           7          4
$ shrrn 8 9
$ pi
name      pid      state      queue_level      cycle      arrival      HRNN      MHRRN
.....................................................................................
init      1      SLEEPING        1             10          3           7          371
sh      2      SLEEPING        1             30          9           7          126
foo      8      RUNNABLE        3             6907          429           9          5
foo      7      SLEEPING        1             1          430           7          3461
foo      6      SLEEPING        1             2          429           2          1730
foo      9      RUNNING        2             6904          430           4          3
pi      21      RUNNING        1             1          7346           7          4
$
```

```
.....................................................................................
init      1      SLEEPING        1             10          3           7          371
sh      2      SLEEPING        1             30          9           7          126
foo      8      RUNNABLE        3             6907          429           9          5
foo      7      SLEEPING        1             1          430           7          3461
foo      6      SLEEPING        1             2          429           2          1730
foo      9      RUNNING        2             6904          430           4          3
pi      21      RUNNING        1             1          7346           7          4
$ foo &; foo &
$ pi
name      pid      state      queue_level      cycle      arrival      HRNN      MHRRN
.....................................................................................
init      1      SLEEPING        1             10          3           7          425
sh      2      SLEEPING        1             35          9           7          139
foo      8      RUNNING        1             8215          429           9          5
foo      7      SLEEPING        1             1          430           7          4009
foo      6      SLEEPING        1             2          429           2          2004
foo      9      RUNNABLE        1             8206          430           4          3
foo      26      RUNNABLE        2             39          8578           7          186
foo      25      RUNNABLE        1             1          8580           7          0
foo      24      SLEEPING        1             1          8578           7          0
     27              2             1          8580           7          0
pi      28      RUNNING        1             1          8672           7          0
```