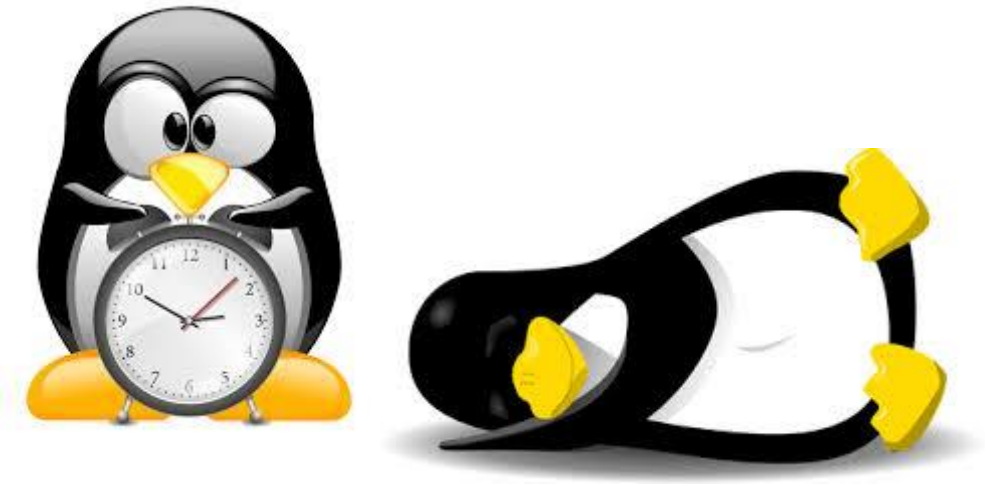
به نام خدا

آزمایشگاه سیستم‌عامل

پروژه سوم: زمان‌بندی پردازه‌ها

طراح: روزبه بستان‌دوست، بهاران خاتمی



در این پروژه با زمان‌بندی در سیستم‌عامل‌ها آشنا خواهید شد. در این راستا الگوریتم زمان‌بندی xv6 بررسی شده و با ایجاد تغییرهایی در آن الگوریتم زمان‌بندی صف بازخوردی چندسطحی[1] (MFQ) پیاده‌سازی می‌گردد. هم‌چنین استفاده از فاکتور زمان در این سیستم‌عامل بررسی می‌گردد. در انتها توسط فراخوانی‌های سیستمی پیاده‌سازی شده، از صحت عملکرد زمان‌بند اطمینان حاصل خواهد شد.

[1] Multilevel Feedback Queue Scheduling

مقدمه

همان‌طور که در پروژه یک اشاره شد، یکی از مهم‌ترین وظایف سیستم‌عامل، تخصیص منابع سخت‌افزاری به برنامه‌های سطح کاربر است. پردازنده مهم‌ترین این منابع بوده که توسط زمان‌بند[۲] سیستم‌عامل به پردازه‌ها تخصیص داده می‌شود. این جزء سیستم‌عامل، در سطح هسته اجرا شده و به بیان دقیق‌تر، زمان‌بند، ریسه‌های هسته[۳] را زمان‌بندی می‌کند.[۴] دقت شود وظیفه زمان‌بند، زمان‌بندی پردازه‌ها (نه همه کدهای سیستم) از طریق زمان‌بندی ریسه‌های هسته متناظر آن‌ها است. کدهای مربوط به وقفه سخت‌افزاری، تحت کنترل زمان‌بند قرار نمی‌گیرند. اغلب زمان‌بندهای سیستم‌عامل‌ها از نوع کوتاه‌مدت[۵] هستند. زمان‌بندی بر اساس الگوریتم‌های متنوعی صورت می‌پذیرد که در درس با آن‌ها آشنا شده‌اید. یکی از ساده‌ترین الگوریتم‌های زمان‌بندی که در xv6 به کار می‌رود، الگوریتم زمان‌بندی نوبت‌گردشی[۶] (RR) است. الگوریتم زمان‌بندی صف بازخوردی چندسطحی با توجه به انعطاف‌پذیری بالا در بسیاری از سیستم‌عامل‌ها مورد استفاده قرار می‌گیرد. این الگوریتم در هسته لینوکس نیز تا مدتی مورد استفاده بود. زمان‌بند کنونی لینوکس، زمان‌بند کاملاً منصف[۷] (CFS) نامیده می‌شود. در این الگوریتم پردازه‌ها دارای اولویت‌های مختلف بوده و به طور کلی تلاش می‌شود تا جای امکان پردازه‌ها با توجه به اولویتشان سهم متناسبی از پردازنده را در اختیار بگیرند. به طور ساده

---

[2] Scheduler

[3] Kernel Threads

[۴] ریسه‌های هسته کدهای قابل زمان‌بندی سطح هسته هستند که در نتیجه درخواست برنامه سطح کاربر (در متن پردازه) ایجاد شده و به آن پاسخ می‌دهند. در بسیاری از سیستم‌عامل‌ها از جمله xv6 تناظر یک‌به‌یک میان پردازه‌ها و ریسه‌های هسته وجود دارد.

[5] Short Term

[6] Round Robin

[7] Completely Fair Scheduler

می‌توان آن را به نوعی نوبت‌گردشی تصور نمود. هر پردازه یک زمان اجرای مجازی[8] داشته که در هر بار زمان‌بندی، پردازه دارای کمترین زمان اجرای مجازی، اجرا خواهد شد. هر چه اولویت پردازه بالاتر باشد زمان اجرای مجازی آن کندتر افزایش می‌یابد. در جدول زیر الگوریتم‌های زمان‌بندی سیستم‌عامل‌های مختلف نشان داده شده است [۱].

[8] Virtual Runtime

سیستم‌عامل	الگوریتم زمان‌بندی	توضیحات
Windows NT/Vista/7	MFQ	۳۲ صف ۰ تا ۱۵ اولویت عادی ۱۶ تا ۳۱ اولویت بی‌درنگ نرم
Mac OS X	MFQ	چندین صف با ۴ اولویت عادی، پراولویت سیستمی، فقط مد هسته، ریسه‌های بی‌درنگ
FreeBSD/NetBSD	MFQ	بیش از ۲۰۰ صف
Solaris	MFQ	۱۷۰ صف
Linux < 2.4	MFQ	-
2.4 ≤ Linux < 2.6	EPOCH-based	سربار بالا
2.6 ≤ Linux < 2.6.23	O(1) Scheduler	پیچیده و سربار پایین
2.6.23 ≤ Linux	CFS	-
xv6	RR	-

زمان‌بندی در xv6

هسته xv6 از نوع با ورود مجدد[9] و غیرقبضه‌ای[10] است. به این ترتیب اجرای زمان‌بند تنها در نقاط محدودی از اجرا صورت می‌گیرد. به عنوان مثال، چنان‌چه در آزمایش دوم مشاهده شد وقفه‌های قابل چشم‌پوشی[11] قادر به وقفه دادن به یکدیگر نبوده و تنها امکان توقف تله‌های غیروقفه را دارند. هم‌چنین تله‌های غیروقفه نیز قادر به توقف یکدیگر نیستند. به طور دقیق‌تر زمان‌بندی تنها در زمان‌های محدودی ممکن است: ۱) هنگام وقفه تایمر و ۲) هنگام رهاسازی داوطلبانه شامل به خواب رفتن یا خروج توسط فراخوانی exit(). به خواب رفتن و فراخواندن exit() می‌تواند دلایل مختلفی داشته باشد. مثلاً یک پردازه می‌تواند به طور داوطلبانه از طریق فراخوانی سیستمی sys_exit()، تابع exit() را فراخوانی نماید. هم‌چنین پردازه بدرفتار، هنگام مدیریت تله به طور داوطلبانه! مجبور به فراخوانی exit() خواهد شد (خط ۳۴۶۹). همه این حالات در نهایت منجر به فراخوانی تابع sched() (۲۸۰۷) و به دنبال آن اجرای تابع زمان‌بندی یا scheduler() می‌گردند (خط ۲۷۵۷).

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

زمان‌بندی

همان‌طور که پیش‌تر ذکر شد، زمان‌بند xv6 از نوع نوبت‌گردشی است. به عبارت دیگر هر پردازه دارای یک برش زمانی[12] بوده که حداکثر زمانی است که قادر به نگه‌داری پردازنده در یک اجرای پیوسته

[9] Reentrant

[10] Nonpreemptive

[11] Maskable Interrupts

[12] Time Slice

می‌باشد. این زمان برابر یک تیک تایمر (حدود ۱۰ میلی‌ثانیه) می‌باشد.[۱۳] با وقوع وقفه تایمر که در هر تیک رخ می‌دهد تابع yield() فراخوانی شده (خط ۳۴۷۵) و از اتمام برش زمانی پردازه جاری خبر خواهد داد.

زمان‌بندی توسط تابع scheduler() صورت می‌پذیرد. این تابع از یک حلقه تشکیل شده که در هر بار اجرا با مراجعه به صف پردازه‌ها یکی از آن‌ها که قابل اجرا است را انتخاب نموده و پردازنده را جهت اجرا در اختیار آن قرار می‌دهد (خط ۲۷۸۱).

۲) صف پردازه‌هایی که تنها منبعی که برای اجرا کم دارند پردازنده است، صف آماده[۱۴] یا صف اجرا[۱۵] نام دارد. در xv6 صف آماده مجزا وجود نداشته و از صف پردازه‌ها بدین منظور استفاده می‌گردد. در زمان‌بند کاملاً منصف در لینوکس، صف اجرا چه ساختاری دارد؟

۴) همان‌طور که در پروژه یک مشاهده شد، هر هسته پردازنده در xv6 یک زمان‌بند دارد. در لینوکس نیز به همین‌گونه است. این دو سیستم‌عامل را از منظر مشترک یا مجزا بودن صف‌های زمان‌بندی بررسی نمایید.

۵) در هر اجرای حلقه ابتدا برای مدتی وقفه فعال می‌گردد. علت چیست؟ آیا در سیستم تک‌هسته‌ای به آن نیاز است؟

۶) تابع معادل scheduler() را در هسته لینوکس بیابید. جهت حفظ اعتبار اطلاعات جدول پردازه‌ها، از قفل‌گذاری استفاده می‌شود. این قفل در لینوکس چه نام دارد؟

---

[۱۳] تنظیمات تایمر هنگام بوت صورت می‌پذیرد.

[14] Ready Queue

[15] Run Queue

۷) در خصوص سازوکار توازن بار[۱۶] در زمان‌بند لینوکس به طور مختصر توضیح دهید. این عملیات توسط چه موجودیتی و بر چه اساسی صورت می‌گیرد؟ (راهنمایی: می‌توانید از مقاله این پروژه کمک بگیرید.)

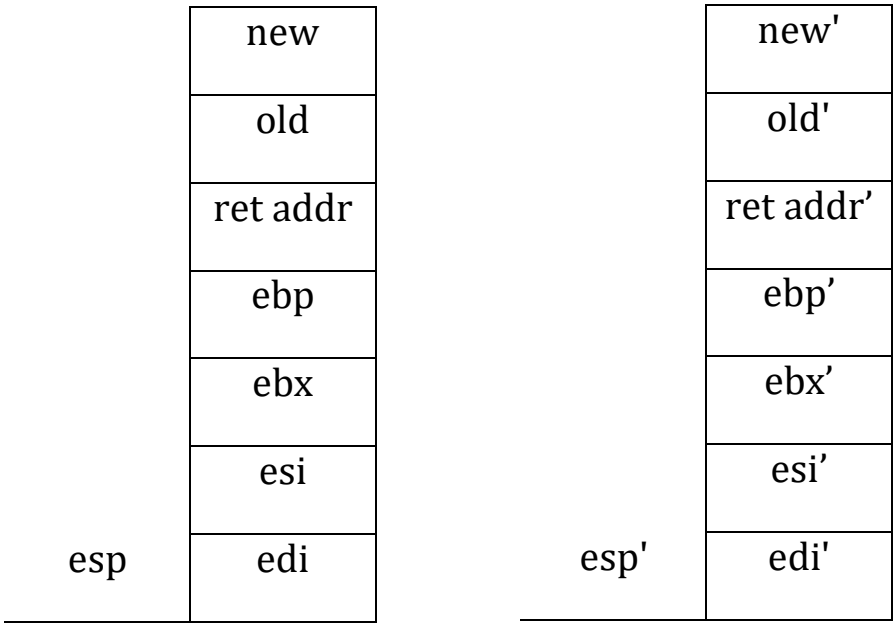
تعویض متن

پس از انتخاب یک پردازه جهت اجرا، توابع switchuvm() و switchkvm() حالت حافظه پردازه را به حالت جاری حافظه سیستم تبدیل می‌کنند. در میان این دو عمل، حالت پردازنده نیز توسط تابع swtch() از حالت (محتوای ساختار context (خط ۲۳۲۶) که ساختار اجرایی در هسته است) مربوط به زمان‌بند (کد مدیریت‌کننده سیستم در آزمایش یک که خود به نوعی ریسه هسته بدون پردازه متناظر در سطح کاربر است) به حالت پردازه برگزیده، تغییر می‌کند. تابع swtch() (خط ۳۰۵۸) دارای دو پارامتر old و new می‌باشد. ساختار بخش مرتبط پشته هنگام فراخوانی این تابع در شکل زیر نشان داده شده است.

esp + 8	new
esp + 4	old
esp	ret addr

بخش مرتبط ساختار پشته پیش و پس از تغییر اشاره‌گر پشته (خط ۳۰۷۱) به ترتیب در نیمه چپ و راست شکل زیر نشان داده شده است.

[16] Load Balancing



اشاره‌گر به اشاره‌گر به متن ریسه هسته قبلی در old، متن ریسه هسته قبلی در پنج ثبات بالای پشته سمت چپ و اشاره‌گر به متن ریسه هسته جدید در new قرار دارد. اشاره‌گر به اشاره‌گر به متن ریسه هسته جدید در old’، متن ریسه هسته جدید در پنج ثبات بالای پشته سمت راست و اشاره‌گر به متن ریسه هسته‌ای که قبلاً این ریسه هسته جدید به آن تعویض‌متن کرده بود، در new’ قرار دارد. متن ریسه هسته جدید از پشته سمت راست به پردازنده منتقل شده (خطوط ۳۰۷۴ تا ۳۰۷۸) و نهایتاً پردازه سطح کاربر اجرا خواهد شد.

زمان‌بندی بازخوردی چندسطحی

در این زمان‌بند، پردازه‌ها با توجه به اولویتی که دارند در سطوح مختلف قرار می‌گیرند که در این پروژه فرض شده است که سه سطح و متعاقباً سه اولویت وجود دارد. پردازه‌هایی مثلاً معادل با پردازه‌های ویرایش متن به طور پیش‌فرض دارای کمترین اولویت (اولویت ۳) هستند. شما برای آزمودن زمان‌بند خود باید فراخوانی سیستمی را پیاده کنید که بتواند اولویت پردازه‌ها را تغییر دهد تا قادر به جابه‌جا کردن پردازه ها در سطوح

مختلف و اعمال الگوریتم‌های مختلف زمان‌بندی در هر سطح باشید. همان‌طور که گفته شد زمان‌بندی که توسط شما پیاده‌سازی می‌شود دارای سه سطح می‌باشد که در سطح یک الگوریتم زمان‌بندی بخت‌آزمایی[۱۷]، در سطح دو الگوریتم بالاترین نسبت پاسخ[۱۸] (HRRN) و در سطح سه الگوریتم کمترین اولویت باقی‌مانده[۱۹] (SRPF) را باید اعمال کنید. لازم به ذکر است که میان سطوح، اولویت وجود دارد. به این صورت که ابتدا تمام پردازه‌های سطح یک، سپس پردازه‌های سطح دو و در انتها پردازه‌های سطح سه اجرا خواهند شد و شما با فراخوانی سیستمی که پیاده‌سازی می‌کنید می‌توانید سطح پردازه‌ها را تغییر دهید. دقت شود زمان‌بند، تک‌هسته‌ای است.

زمان‌بند بخت‌آزمایی

این زمان‌بند بر پایه تخصیص منابع به پردازه‌ها به صورت تصادفی می‌باشد. ولی هر پردازه با توجه به تعداد بلیت شانش که دارد احتمال انتخاب شدنش به عنوان پردازه بعدی برای اجرا مشخص می‌شود. انتخاب پردازه برای اجرا توسط زمان‌بند پردازنده به این صورت می‌باشد که هر پردازه تعدادی بلیت شانس دارد و پردازنده به صورت تصادفی یک بلیت را انتخاب نموده و پردازه صاحب آن بلیت، اجرا خواهد شد. هنگامی که اجرای این پردازه توسط عواملی چون اتمام برش زمانی، مسدود شدن جهت عملیات ورودی/خروجی و ... به پایان رسید، روند مذکور تکرار خواهد شد.

هر بلیت معادل یک عدد طبیعی بوده و هر پردازه می‌تواند بازه‌ای از اعداد را به عنوان بلیت‌های شانس خود داشته باشد. زمان‌بند پردازه‌ها با تولید عددی تصادفی در بازه کل این اعداد، یک بلیت و متناظر با آن یک پردازه را برای اجرا انتخاب می‌کند. به عنوان مثال دو پردازه A و B داریم و A دارای ۶۰ بلیت شانس (بلیت‌های شماره ۱ تا ۶۰) و B دارای ۴۰ بلیت شانس (بلیت‌های شماره ۶۱ تا ۱۰۰) می‌باشد. زمان‌بند در

[17] Lottery

[18] Highest Response Ratio Next

[19] Shortest Remaining Time

هر مرحله، عددی تصادفی بین ۱ تا ۱۰۰ را انتخاب نموده و اگر عدد انتخاب شده بین ۱ تا ۶۰ باشد، پردازه A و در غیر این‌صورت پردازه B انتخاب می‌گردد. در شکل زیر مثالی از ۱۰ مرحله انتخابی توسط زمان‌بند پردازنده نشان داده شده است.

```
Ticket number -  73 82 23 45 32 87 49 39 12 09.
Resulting Schedule -  B  B  A  A  A  B  A  A  A  A.
```

زمان‌بند HRRN

در این بخش، تقریبی از الگوریتم HRRN پیاده‌سازی خواهد شد [۲]. در این حالت شما باید دو ویژگی برای پردازه‌های خود به عنوان زمان ورود[۲۰] و تعداد سیکل اجرا[۲۱] مشخص نمایید. برای محاسبه زمان ورود می‌توانید از زمان سیستم هنگام ایجاد پردازه استفاده نموده و برای محاسبه تعداد سیکل اجرا، باید یک مشخصه برای پردازه خود با همین نام در نظر بگیرید. مقدار پیش فرض تعداد سیکل اجرا را ۱ در نظر بگیرید. اجرای هر پردازه ۱ واحد به تعداد سیکل اجرایی آن می‌افزاید. الگوریتم زمان‌بندی به این صورت است که در ابتدای هر برش زمانی، پردازه دارای بیشترین مقدار HRRN (نسبتی که در معادله زیر ارائه شده است) اجرا خواهد شد.

$$HRRN = \frac{WaitingTime}{ExecutedCycleNumber}$$

$$WaitingTime = CurrentTime - ArrivalTime$$

لازم به ذکر است که زمان انتظار[۲۲] برابر اختلاف زمان جاری با زمان ایجاد پردازه است.

[20] Arrival Time

[21] Execution Cycle Number

[22] Waiting Time

زمان‌بند SRPF

در این قسمت نیز تقریبی از الگوریتم کوتاه‌ترین زمان باقی‌مانده (SRTF) پیاده‌سازی خواهد شد [۲]. هر پردازه مشخصه‌ای تحت عنوان اولویت باقی‌مانده[۲۳] دارد. مقداردهی این مشخصه توسط فراخوانی سیستمی صورت خواهد گرفت. الگوریتم بدین صورت است که در هر برش زمانی، پردازه دارای کمترین مقدار اولویت باقی‌مانده انتخاب و اجرا خواهد شد (در صورت وجود چندین پردازه با اولویت باقی‌مانده یکسان، یکی از آن‌ها به طور تصادفی انتخاب خواهد شد). پس از اجرای برش زمانی هر پردازه، ۰٫۱ واحد از اولویت باقی‌مانده مربوطه، کاسته می‌شود. توجه شود که اولویت باقی‌مانده هیچ‌گاه کمتر از صفر نخواهد شد.

نکته: پارامترهای جدیدی که برای مدهای مختلف زمان‌بندی به پردازه اضافه می‌کنید و هنگام ایجاد پردازه، آن‌ها را مقداردهی می‌کنید باید به گونه‌ای مقداردهی شوند که به پردازه هایی که از طریق Shell و با exec() اجرا می‌شوند نسبت به پردازه هایی که با fork() ساخته می‌شوند و از طریق Shell اجرا نمی‌شوند اولویت داده شود.

فراخوانی‌های سیستمی مورد نیاز

۱) **تغییر صف پردازه:** پس از ایجاد پردازه‌ها (به تعداد لازم) باید با نوشتن یک فراخوانی سیستمی مناسب مشخص کنید که هر پردازه به کدام صف از سه صفی که پیاده‌سازی کرده‌اید تعلق دارد. هم‌چنین باید بتوان یک پردازه را از یک صف به صف دیگری انتقال داد. این فراخوانی سیستمی، PID پردازه و شماره صف مقصد را به عنوان ورودی دریافت می‌کند.

۲) **مقداردهی بلیت بخت‌آزمایی:** باید به هرکدام از پردازه‌هایی که در صف اول قرار دارند تعدادی بلیت اختصاص دهید تا الگوریتم بخت‌آزمایی قابل اجرا باشد. بنابراین باید یک فراخوانی سیستمی پیاده‌سازی کنید

[23] Remaining Priority

که به پردازه‌های صف اول، بلیت مربوطه را تخصیص دهد. ورودی، PID پردازه مورد نظر و مقدار بلیت آن خواهد بود.

۳) **مقداردهی اولویت اجرا در صف SRPF**: یک فراخوانی سیستمی پیاده‌سازی نمایید که اولویت جدیدی به باقی‌مانده یک پردازه تخصیص دهد. این فراخوانی سیستمی، PID مربوط به پردازه و اولویت باقی‌مانده جدید را به عنوان ورودی دریافت می‌کند.

۴) **چاپ اطلاعات:** برای اینکه برنامه شما قابل تست باشد باید یک فراخوانی سیستمی پیاده‌سازی کنید که لیست تمام پردازه‌ها را چاپ نموده و در هر سطر این لیست باید نام پردازه، شماره پردازه، وضعیت، شماره صف، اولویت، تعداد بلیت، تعداد سیکل‌های اجرا و نسبت HRRN در آن گنجانده شود. یک مثال نیمه‌کامل در شکل زیر نشان داده شده است.

```
name      pid      state            priority         createTime
-------------------------------------------------------------
init      1        SLEEPING         2                16
 sh       2        SLEEPING         2                56
 ps       48       RUNNING          2                20736
 foo      15       SLEEPING         2                9423
 foo      16       RUNNING          10               9423
```

جهت حصول اطمینان از زمان‌بند خود، یک برنامه سطح کاربر با نام foo بنویسید که تعدادی پردازه در آن ساخته شده و پردازه‌ها عملیات پردازشی[۲۴] انجام دهند تا فرصت بررسی عملکرد زمان‌بند وجود داشته باشد. می‌توان این برنامه را با اجرای دستور foo& در پس‌زمینه اجرا نموده و در این حین، توسط فراخوانی سیستمی چاپ اطلاعات از نحوه عملکرد آن مطلع شد.

[24] CPU Intensive

سایر نکات

- کدهای خود را مشابه پروژه‌های پیشین در Gitlab بارگذاری نموده و آدرس مخزن، شناسه آخرین Commit و گزارش پروژه را در سایت درس بارگذاری نمایید.
- پاسخ تمامی سؤالات را در کوتاه‌ترین اندازه ممکن در گزارش خود بیاورید.
- همه افراد باید به پروژه مسلط باشند و نمره تمامی اعضای گروه لزوماً یکسان نخواهد بود.
- در صورت مشاهده هرگونه شباهت بین کدها یا گزارش دو گروه، به هر دو گروه نمره ۰ تعلق می‌گیرد.
- فصل ۵ کتاب xv6 می‌تواند مفید باشد.
- هر گونه سؤال در مورد پروژه را فقط از طریق فروم درس مطرح نمایید.

موفق باشید

مراجع

[1] Donald H. Pinkston. 2014. Caltech Operating Systems Slides.

[2] William Stallings. 2018. *Operating Systems: Internals and Design Principles, 9/e* (9th ed.). Pearson IT Certification, USA.