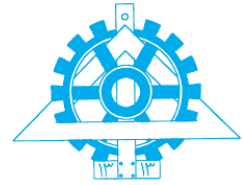




به نام خدا

آزمایشگاه سیستم‌عامل



پروژه چهارم: زمان‌بندی پردازش‌ها

طراح: روزبه بستان‌دوست



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^۱ (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.

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

^۱ Multilevel Feedback Queue Scheduling

^۲ Scheduler

سیستم‌عامل به پردازش‌ها تخصیص داده می‌شود. این جزء سیستم‌عامل، در سطح هسته اجرا شده و به بیان دقیق‌تر، زمان‌بند، ریشه‌های هسته^۳ را زمان‌بندی می‌کند.^۴ دقت شود وظیفه زمان‌بند، زمان‌بندی پردازش‌ها (نه همه کدهای سیستم) از طریق زمان‌بندی ریشه‌های 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 پردازش‌ها دارای اولویت‌های مختلف بوده و به طور کلی تلاش می‌شود تا جای امکان پردازش‌ها با توجه به اولویتشان سهم متناسبی از پردازنده را در اختیار بگیرند. به طور ساده می‌توان آن را به نوعی نوبت‌گردشی تصور نمود. هر پردازش یک زمان اجرای مجازی^۸ داشته که در هر بار زمان‌بندی، پردازش دارای کمترین زمان اجرای مجازی، اجرا خواهد شد. هر چه اولویت پردازش بالاتر باشد زمان اجرای مجازی آن کندتر افزایش می‌یابد. در جدول زیر الگوریتم‌های زمان‌بندی سیستم‌عامل‌های مختلف نشان داده شده است [Pinkston 2014].

³ Kernel Threads

^۴ ریشه‌های هسته کدهای قابل زمان‌بندی سطح هسته هستند که در نتیجه درخواست برنامه سطح کاربر (در متن پردازش) ایجاد شده و به آن پاسخ می‌دهند. در بسیاری از سیستم‌عامل‌ها از جمله xv6 متناظر یک‌به‌یک میان پردازش‌ها و ریشه‌های هسته وجود دارد.

⁵ Short Term

⁶ Round Robin

⁷ Completely Fair Scheduler

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	-
$2.4 \leq \text{Linux} < 2.6$	EPOCH-based	سربرار بالا
$2.6 \leq \text{Linux} < 2.6.23$	O(1) Scheduler	پیچیده و سربرار پایین
$2.6.23 \leq \text{Linux}$	CFS	-
xv6	RR	-

زمان‌بندی در xv6

هسته xv6 از نوع با ورود مجدد^۹ و غیرقبضه‌ای^{۱۰} 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^{۱۱} 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 پردازش می‌تواند به طور داوطلبانه از طریق فراخوانی سیستمی `sys_exit()` تابع `exit()` را فراخوانی نماید. هم‌چنین پردازش بدرفتار، هنگام مدیریت تله به طور داوطلبانه! مجبور به فراخوانی `exit()` خواهد شد (خط ۳۶۴۹). 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 پردازش دارای یک برش زمانی^{۱۲} بوده که حداکثر زمانی است که قادر به نگه‌داری پردازنده در یک اجرای پیوسته می‌باشد. این زمان برابر یک تیک تایمر (حدود ۱۰ میلی‌ثانیه) می‌باشد.^{۱۳} با وقوع وقفه تایمر که در هر

^۹ Reentrant

^{۱۰} Nonpreemptive

^{۱۱} Maskable Interrupts

^{۱۲} Time Slice

^{۱۳} تنظیمات تایمر هنگام بوت صورت می‌پذیرد.

تیک رخ می‌دهد تابع `yield()` فراخوانی شده (خط ۳۴۷۵) و از اتمام برش زمانی پردازش 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 پردازش‌ها یکی از آن‌ها که قابل اجرا است را انتخاب نموده و پردازنده را جهت اجرا در اختیار آن قرار می‌دهد (خط ۲۷۸۱).

۲) صف پردازش‌هایی 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 پردازش‌ها 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 پردازش‌ها، از قفل‌گذاری استفاده می‌شود. این قفل در لینوکس چه نام دارد؟

تعویض متن

پس از انتخاب یک پردازش جهت اجرا، توابع `switchvm()` و `switchkvm()` حالت حافظه پردازش 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)

¹⁴ Ready Queue

¹⁵ Run Queue

مربوط به زمان‌بند (کد مدیریت‌کننده سیستم در آزمایش یک که خود به نوعی ریشه هسته بدون پردازش متناظر در سطح کاربر است) به حالت پردازش برگزیده، تغییر می‌کند. تابع `swtch()` (خط ۳۰۵۷) 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 ۳۰۷۰) به ترتیب در نیمه چپ و راست شکل زیر نشان داده شده است.

esp	new	esp'	new'
	old		old'
	ret addr		ret addr'
	ebp		ebp'
	ebx		ebx'
	esi		esi'
	edi		edi'

اشاره‌گر به اشاره‌گر به متن ریشه هسته قبلی در `old`، متن ریشه هسته قبلی در پنج ثبات بالای پشته سمت چپ و اشاره‌گر به متن ریشه هسته جدید در `new` قرار دارد. اشاره‌گر به اشاره‌گر به متن ریشه هسته جدید در `old'`، متن ریشه هسته جدید در پنج ثبات بالای پشته سمت راست و اشاره‌گر به متن ریشه هسته‌ای که قبلاً این ریشه هسته جدید به آن تعویض متن کرده بود، در `new'` قرار دارد. متن

ریسه هسته جدید از پشته سمت راست به پردازنده منتقل شده (خطوط ۳۰۷۳ تا ۳۰۷۷) و نهایتاً پردازنده سطح کاربر اجرا خواهد شد.

زمان‌بندی بازخوردی چندسطحی

در این زمان‌بند، پردازنده‌ها با توجه به اولییتی که دارند در سطوح مختلف قرار می‌گیرند که در این پروژه فرض شده است که سه سطح و متعاقباً سه اولویت وجود دارد. پردازنده‌هایی مثلاً معادل با پردازنده‌های 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 پردازنده‌ها را تغییر دهد تا قادر به جابه‌جا کردن پردازنده‌ها در سطوح مختلف و اعمال الگوریتم‌های مختلف زمان‌بندی در هر سطح باشید. همان‌طور که گفته شد زمان‌بندی که توسط شما پیاده سازی می‌شود دارای سه سطح می‌باشد که در سطح یک الگوریتم زمان‌بندی بخت‌آزمایی^{۱۶}، در سطح دو الگوریتم زمان‌بندی ابتدا کوتاه‌ترین وظیفه^{۱۷} (SJF) و در سطح سه الگوریتم زمان‌بندی مبتنی بر اولویت^{۱۸} را باید اعمال کنید. لازم به ذکر است که میان سطوح، اولویت وجود دارد. به این صورت که ابتدا تمام پردازنده‌های سطح یک، سپس پردازنده‌های سطح دو و در انتها پردازنده‌های سطح سه اجرا خواهند شد و شما با فراخوانی سیستمی که پیاده‌سازی می‌کنید می‌توانید سطح پردازنده‌ها را تغییر دهید.

زمان‌بند بخت‌آزمایی

این زمان‌بند بر پایه تخصیص منابع به پردازنده‌ها به صورت تصادفی می‌باشد. ولی هر پردازنده با توجه به تعداد بلیت شانس که دارد احتمال انتخاب شدنش به عنوان پردازنده بعدی برای اجرا مشخص می‌شود. انتخاب پردازنده برای اجرا توسط زمان‌بند پردازنده به این صورت می‌باشد که هر پردازنده تعدادی بلیت شانس دارد و پردازنده به صورت تصادفی یک بلیت را انتخاب نموده و پردازنده صاحب آن بلیت، اجرا خواهد شد. هنگامی که اجرای

¹⁶ Lottery

¹⁷ Shortest Job First

¹⁸ Priority Scheduling

این پردازش 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 پردازش می‌تواند بازه‌ای از اعداد را به عنوان بلیت‌های شانس خود داشته باشد. زمان‌بند پردازش‌ها با تولید عددی تصادفی در بازه کل این اعداد، یک بلیت و متناظر با آن یک پردازش را برای اجرا انتخاب می‌کند. به عنوان مثال دو پردازش 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 هر مرحله، عددی تصادفی بین ۱ تا ۱۰۰ را انتخاب نموده و اگر عدد انتخاب شده بین ۱ تا ۶۰ باشد، پردازش A و در غیر این صورت پردازش 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 SJF

این الگوریتم تنها با مشخصه Burst Time (که شما با پیاده‌سازی برنامه سطح کاربر مناسب آن را مقداردهی می‌کنید) فعالیت نموده و همواره پردازش دارای کوچکترین Burst Time در صف دوم اجرا خواهد شد.

زمان‌بند مبتنی بر اولویت

در صف سوم، پردازش‌ها بر اساس اولویت اجرا خواهند شد. هرچه عدد اولویت پردازش کمتر باشد، اولویت آن بیشتر بوده و باید در این صف زودتر از سایر پردازش‌ها اجرا شود (به عنوان مثال پردازش‌ای با اولویت پنج از پردازش با اولویت شش زودتر اجرا خواهد شد).

فراخوانی‌های سیستمی مورد نیاز

(۱) پس از ایجاد پردازنده‌ها (به تعداد لازم) باید با نوشتن یک فراخوانی سیستمی مناسب مشخص کنید که هر پردازنده مربوط به کدام صف از سه صفی که پیاده‌سازی کرده‌اید تعلق دارد. همچنین باید بتوان یک پردازنده را از یک صف به صف دیگری انتقال داد.

(۲) باید به هر کدام از پردازنده‌هایی که در صف اول قرار دارند تعدادی بلیت اختصاص دهید تا الگوریتم بخت‌آزمایی قابل اجرا باشد. بنابراین باید یک فراخوان سیستمی پیاده‌سازی کنید که به پردازنده‌های صف اول، بلیت مربوطه را تخصیص دهد.

(۳) باید امکان تعیین **Burst Time** پردازنده‌های موجود در صف دوم وجود داشته باشد. شما باید این فراخوانی سیستمی را پیاده‌سازی کنید.

(۴) پردازنده‌هایی که در صف سوم قرار دارند، باید دارای یک بخش به نام اولویت هم باشند تا ترتیب اجرای پردازنده‌ها در این صف مشخص گردد. بنابراین باید یک فراخوانی سیستمی پیاده‌سازی کنید که به این پردازنده‌ها اولویت مختص خودشان را نسبت دهد.

(۵) برای اینکه برنامه شما قابل تست باشد باید یک فراخوانی سیستمی پیاده‌سازی کنید که لیست تمام پردازنده‌ها را چاپ نموده و در هر سطر این لیست باید نام پردازنده، شماره پردازنده، وضعیت، شماره صف، اولویت (مربوط به پردازنده‌های صف سوم)، تعداد بلیت (مربوط به پردازنده‌های صف اول) و **Burst Time** آن گنجانده شود. یک مثال تقریباً کامل 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

آشنایی با نحوه محاسبه زمان در xv6

شما نیاز به محاسبه زمان ایجاد هر پردازش دارید. در این بخش مستقل از pid پردازش‌های ایجاد شده، یک شمارنده خودافزون پیاده‌سازی کنید که با ایجاد هر پردازش، مقدار فعلی شمارنده به این پردازش تخصیص یابد. سپس مقدار شمارنده یک واحد افزایش یافته و این روند در حین ایجاد هر پردازش تکرار می‌شود. در انتها با توجه به داده‌های ذخیره شده، به کمک ابزاری مانند matplotlib در Python (یا هر ابزار مناسب دیگر) نموداری رسم کنید که محور X آن شماره پردازش و محور Y آن زمان ایجاد پردازش است. این نمودار را در گزارش خود بیاورید و آن را تحلیل نمایید.

(توصیه می‌شود قبل از پیاده‌سازی، این بخش را انجام دهید تا با سازوکار زمان و محاسبه آن در xv6 آشنا شوید).

(۷) متغیر ticks در xv6 چه کاربردی دارد؟

سایر نکات

- فقط فایل‌های تغییر یافته و یا افزوده شده را به صورت zip 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

Donald H. Pinkston. 2014. Caltech Operating Systems Slides.
(2014).