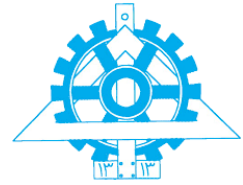




به نام خدا



آزمایشگاه سیستم عامل

پروژه سوم: زمان بندی پردازها

طراحان: بهاران خاتمی، فاطمه حقیقی



در این پروژه با زمان بندی در سیستم عامل ها آشنا خواهید شد. در این راستا الگوریتم زمان بندی xv6 بررسی شده و با ایجاد تغییرهایی در آن الگوریتم زمان بندی صف بازخوردی چندسطحی¹ (MFQ) پیاده سازی می گردد. هم چنین استفاده از فاکتور زمان در این سیستم عامل بررسی می گردد. در انتها توسط فراخوانی های سیستمی پیاده سازی شده، از صحت عملکرد زمان بند اطمینان حاصل خواهد شد.

¹ Multilevel Feedback Queue Scheduling

مقدمه

همان طور که در پروژه یک اشاره شد، یکی از مهم ترین وظایف سیستم عامل، تخصیص منابع سخت افزاری به برنامه های سطح کاربر است. پردازنده مهم ترین این منابع بوده که توسط زمان بند^۲ سیستم عامل به پردازنده ها تخصیص داده می شود. این جزء سیستم عامل، در سطح هسته اجرا شده و به بیان دقیق تر، زمان بند، ریشه های هسته^۳ را زمان بندی می کند.^۴ دقت شود وظیفه زمان بند، زمان بندی پردازنده ها (نه همه کدهای سیستم) از طریق زمان بندی ریشه های هسته متناظر آنها است. کدهای مربوط به وقفه سخت افزاری، تحت کنترل زمان بند قرار نمی گیرند. اغلب زمان بندهای سیستم عامل ها از نوع کوتاه مدت^۵ هستند. زمان بندی بر اساس الگوریتم های متنوعی صورت می پذیرد که در درس با آنها آشنا شده اید. یکی از ساده ترین الگوریتم های زمان بندی که در xv6 به کار می رود، الگوریتم زمان بندی نوبت گردشی^۶ (RR) است. الگوریتم زمان بندی صف بازخوردی چندسطحی با توجه به انعطاف پذیری بالا در بسیاری از سیستم عامل ها مورد استفاده قرار می گیرد. این الگوریتم در هسته لینوکس نیز تا مدتی مورد استفاده بود. زمان بند کنونی لینوکس، زمان بند کاملاً منصف^۷ (CFS) نامیده می شود. در این الگوریتم پردازنده ها دارای اولویت های مختلف بوده و به طور کلی تلاش می شود تا جای امکان پردازنده ها با توجه به اولویتشان سهم متناسبی از پردازنده را در اختیار بگیرند. به طور ساده

² Scheduler

³ Kernel Threads

^۴ ریشه های هسته کدهای قابل زمان بندی سطح هسته هستند که در نتیجه درخواست برنامه سطح کاربر (در متن پردازنده) ایجاد شده و به آن پاسخ می دهند. در بسیاری از سیستم عامل ها از جمله xv6 تناظر یک به یک میان پردازنده ها و ریشه های هسته وجود دارد.

⁵ Short Term

⁶ Round Robin

⁷ Completely Fair Scheduler

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 [۱].

^۸ 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 پردازنده می‌تواند به طور داوطلبانه از طریق فراخوانی سیستمی `sys_exit()`، تابع `exit()` را فراخوانی نماید. هم‌چنین پردازنده 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.)

⁹ Reentrant

¹⁰ Nonpreemptive

¹¹ Maskable Interrupts

زمان‌بندی

همان‌طور که پیش‌تر ذکر شد، زمان‌بند xv6 از نوع نوبت‌گردشی است. به عبارت دیگر هر پردازنده 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 تیک رخ می‌دهد تابع yield() فراخوانی شده (خط ۳۴۷۵) و از اتمام برش زمانی پردازنده 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 پردازنده‌ها یکی از آن‌ها که قابل اجرا است را انتخاب نموده و پردازنده را جهت اجرا در اختیار آن قرار می‌دهد (خط ۲۷۸۱).

(۲) صف پردازنده‌هایی 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 پردازنده‌ها بدین منظور استفاده می‌گردد. در زمان‌بند کاملاً منصف در لینوکس، صف اجرا چه ساختاری دارد؟

(۴) زمان‌بندهای بی‌درنگ^{۱۶} لینوکس را نام برده (۳ مورد) و در خصوص اولویت پردازنده‌های بی‌درنگ نسبت به پردازنده‌های موجود در صف CFS توضیح دهید. آیا همواره اولویت پردازنده‌های بی‌درنگ بیشتر است؟ چرا؟

¹² Time Slice

¹³ تنظیمات تایمر هنگام بوت صورت می‌پذیرد.

¹⁴ Ready Queue

¹⁵ Run Queue

¹⁶ Realtime Schedulers

(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 پردازها، از قفل‌گذاری استفاده می‌شود. این قفل در لینوکس چه نام دارد؟

(۷) با توجه به مجزا بودن صف‌های زمان‌بندی هر هسته، در خصوص سازوکار توازن بار^{۱۷} 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 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 `switch()` 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 ریشه هسته بدون پردازش متناظر در سطح کاربر است) به حالت پردازش برگزیده، تغییر می‌کند. تابع `switch()` (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

¹⁷ Load Balancing

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.

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 تعویض‌متن کرده بود، در **new'** قرار دارد. متن ریشه هسته جدید از پشته سمت راست به پردازنده منتقل شده (خطوط ۳۰۷۴ تا ۳۰۷۸) و نهایتاً پردازنده سطح کاربر اجرا خواهد شد.

زمان بندی بازخوردی چندسطحی

در این زمان بند، پدازه ها با توجه به اولییتی که دارند در سطوح مختلف قرار می گیرند که در این پروژه فرض شده است که سه سطح و متعاقباً سه اولویت وجود دارد. شما برای آزمودن زمان بند خود باید فراخوانی سیستمی را پیاده سازی کنید که بتواند پدازه را بین سطوح مختلف جابجا کرده تا قادر به اعمال الگوریتم های مختلف هر صف باشید. همانطور که گفته شد زمان بندی که توسط شما پیاده سازی می شود دارای سه سطح می باشد که لازم است در سطح یک الگوریتم زمان بندی بخت آزمایی^{۱۸}، در سطح دوم الگوریتم زمان بندی نوبت گردشی و در سطح سوم الگوریتم زمان بندی بالاترین نسبت پاسخ^{۱۹} (HRRN) را اعمال کنید. لازم به ذکر است که میان سطوح، اولویت وجود دارد. به این صورت که ابتدا تمام پدازه های سطح اول، سپس در صورت خالی بودن سطح اول، تمام پدازه های سطح دوم و در آخر در صورت خالی بودن هر دو سطح قبل، تمام پدازه های سطح سوم اجرا خواهند شد و شما با فراخوانی سیستمی که پیاده سازی می کنید می توانید سطح پدازه ها را تغییر دهید.

همچنین زمان بند پیاده سازی شده توسط شما باید دارای قابلیت افزایش سن^{۲۰} بوده و اگر پدازه ای بیشتر از زمانی معین اجرا نشود، آن پدازه را به سطح اول منتقل نماید.

زمان بند بخت آزمایی

این زمان بند بر پایه تخصیص منابع به پدازه ها به صورت تصادفی می باشد. ولی هر پدازه با توجه به تعداد بلیت شانس که دارد احتمال انتخاب شدنش به عنوان پدازه بعدی برای اجرا مشخص می شود. انتخاب پدازه برای اجرا توسط زمان بند پدازنده به این صورت می باشد که هر پدازه تعدادی بلیت شانس دارد و پدازنده به صورت تصادفی یک بلیت را انتخاب نموده و پدازه صاحب آن بلیت، اجرا خواهد شد. هنگامی که اجرای این پدازه توسط عواملی چون اتمام برش زمانی، مسدود شدن جهت عملیات ورودی/خروجی و ... به پایان رسید، روند مذکور تکرار خواهد شد.

¹⁸ Lottery

¹⁹ Highest Response Ratio Next

²⁰ Aging

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 نوبت گردشی

در این زمان‌بند یک واحد زمانی کوچکی به نام برش یا کوانتوم زمانی^{۲۱} تعریف می‌شود. در این زمان‌بند صف پردازش‌های آماده اجرا را به صورت یک صف حلقوی در نظر می‌گیریم. پردازش‌ها به صورت چرخشی، پردازنده را برای بازه حداکثر، یک کوانتوم زمانی در اختیار می‌گیرند.

به عبارت دیگر زمان‌بند، پردازش موجود در ابتدای صف را انتخاب نموده و یک تایمر برای پردازنده تنظیم می‌کند که پس از یک کوانتوم زمانی، پردازنده در اختیار پردازش دیگر قرار گیرد. پردازش‌ها در این نوع زمان‌بند به دو صورت عمل می‌کنند. حالت اول زمانی است که زمان مورد نیاز پردازش کمتر یا مساوی یک کوانتوم زمانی است، در این حالت پردازش به صورت داوطلبانه پردازنده را رها می‌کند. پس از آن پردازنده، پردازش بعدی که در ابتدای صف قرار دارد را انتخاب می‌نماید. حالت دوم، حالتی که زمان مورد نیاز پردازش بیشتر از یک کوانتوم

²¹ Time Quantum

زمانی است. در این حالت تایمر خاموش شده و منجر به وقفه در اجرا می‌گردد. سپس تعویض‌متن رخ داده و پردازش در انتهای صف اجرا قرار می‌گیرد. پس از آن پردازنده، پردازش ابتدای صف اجرا را انتخاب می‌کند. نکته‌ای که باید در پیاده‌سازی این الگوریتم رعایت شود این است که پردازش‌ها به ترتیب ورود به صف، اجرا خواهند شد و پردازش جدید، به انتهای زنجیره پردازش‌های در حال انتظار افزوده می‌شود.

زمان‌بند HRRN

در این بخش، تقریبی از الگوریتم HRRN پیاده‌سازی خواهد شد [۲]. در این حالت شما باید دو ویژگی برای پردازش‌های خود به عنوان زمان ورود^{۲۲} و تعداد سیکل اجرا^{۲۳} مشخص نمایید. برای محاسبه زمان ورود می‌توانید از زمان سیستم هنگام ایجاد پردازش استفاده نموده و برای محاسبه تعداد سیکل اجرا، باید یک مشخصه برای پردازش خود با همین نام در نظر بگیرید. مقدار پیش فرض تعداد سیکل اجرا را ۱ در نظر بگیرید. اجرای هر پردازش ۱ واحد به تعداد سیکل اجرایی آن می‌افزاید. الگوریتم زمان‌بندی به این صورت است که در ابتدای هر برش زمانی، پردازش 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^{۲۴} برابر اختلاف زمان جاری با زمان ایجاد پردازش است.

نکته: پارامترهای جدیدی که برای الگوریتم‌های مختلف زمان‌بندی به پردازش اضافه می‌کنید و هنگام ایجاد پردازش، آن‌ها را مقداردهی می‌کنید باید به گونه‌ای مقداردهی شوند که اجرای پردازش‌هایی که exec

²² Arrival Time

²³ Execution Cycle Number

²⁴ Waiting Time

می‌شوند (مانند پردازنده‌هایی که از طریق Shell اجرا می‌شوند) به سایر پردازنده‌ها تقدم داشته باشد تا قادر به اجرای فراخوانی‌های سیستمی که در ادامه معرفی می‌شوند در حین اجرای برنامه‌ی سطح کاربر باشید و shell قفل نشود و به اجرای این فراخوانی‌های سیستمی اولویت دهد تا به این وسیله بتوانید صحت زمان‌بند خود را بررسی کنید.

مکانیزم افزایش سن

همانطور که در کلاس درس فرا گرفتید، برای جلوگیری از گرسنگی^{۲۵}، اولویت پردازنده‌هایی که مدت زیادی منتظر بوده و پردازنده به آن‌ها اختصاص نیافته به مرور افزایش می‌یابد. در زمان‌بندی که پیاده‌سازی می‌کنید پردازنده‌ها را به طور پیش‌فرض در صف سوم قرار دهید و در صورتی که پردازنده‌ای ۲۵۰۰ سیکل منتظر مانده باشد آن را به صف اول منتقل کنید. در صورت بازانتقال این پردازنده به صف‌های دیگر، رصد کردن تعداد سیکل اجرا نشده‌ی پردازنده را از ابتدا از سر بگیرید.

فراخوانی‌های سیستمی مورد نیاز

(۱) **تغییر صف پردازنده:** 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. این فراخوانی سیستمی، PID پردازنده و شماره صف مقصد را به عنوان ورودی دریافت می‌کند.

(۲) **مقداردهی بلیت بخت‌آزمایی:** 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

²⁵ Starvation

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

(۳) چاپ اطلاعات: برای اینکه برنامه شما قابل تست باشد باید یک فراخوانی سیستمی پیاده‌سازی کنید که لیست تمام پردازش‌ها را چاپ نموده و در هر سطر این لیست باید نام پردازش، شماره پردازش، وضعیت، شماره صف، تعداد بلیت (مربوط به پردازش‌های صف اول)، 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 پردازش در آن ساخته شده و پردازش‌ها عملیات پردازشی^{۲۶} 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.

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.

²⁶ CPU Intensive

- 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.