

آزمایشگاه سیستم عامل پروژه سوم: زمان بندی پردازها

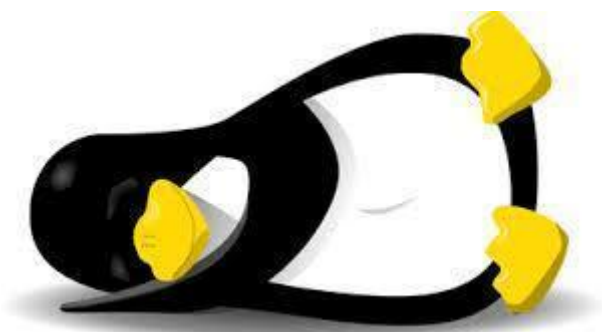


سیستم های عامل - پائیز ۱۴۰۲

دانشکده مهندسی برق و کامپیوتر

مسئولان تمرین:
امیر فراهانی، نسا عباسی

استاد:
دکتر مهدی کارگهی



در این پروژه با زمان بندی در سیستم عامل ها آشنا خواهید شد. در این راستا در ابتدا الگوریتم زمان بندی سیستم عامل xv6 بررسی شده و با ایجاد تغییراتی در آن الگوریتم، زمان بندی صف بازخوردی چند سطحی¹ (MFQ) پیاده سازی می گردد. هم چنین نحوه استفاده از فاکتور زمان در این سیستم عامل بررسی می گردد. در انتهای پروژه، با 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 اول اشاره شد، یکی از مهم ترین وظایف سیستم عامل ها، تخصیص منابع سخت افزاری به برنامه های سطح کاربر است. در این امر، پردازنده مهم ترین منبع از این منابع بوده که توسط زمان بند² سیستم عامل به پردازها تخصیص داده می شود. این بخش از سیستم عامل، در سطح هسته اجرا شده و به بیان دقیق تر، زمان بند ریشه های هسته³ را زمان بندی می کند.⁴

دقت شود وظیفه زمان بند، زمان بندی پردازها (و نه همه کدهای سیستم) از طریق زمان بندی ریشه های هسته متناظر با آنها است. کدهای مربوط به وقفه سخت افزاری، تحت کنترل زمان بند قرار نمی گیرند. اغلب زمان بندهای سیستم عامل ها از نوع کوتاه مدت⁵ 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 گردشی⁶ (RR) است. الگوریتم زمان بندی صف بازخوردی چند سطحی 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 منصف⁷ (CFS) نامیده می شود. در این الگوریتم، پردازها دارای اولویت های مختلف هستند و به طور کلی تلاش می شود که تا جای امکان پردازها با توجه به اولویتشان برای اجرا، سهم متناسبی از پردازنده را در اختیار بگیرند. به طور ساده تر می توان آن را به نوعی زمان بند نوبت گردشی تصور نمود. هر پردازه یک زمان اجرای مجازی⁸ دارد که در هر بار زمان بندی، پردازه دارای کمترین زمان اجرای مجازی، اجرا خواهد شد. هر چه اولویت پردازه بالاتر باشد زمان اجرای مجازی آن به صورت کندتر افزایش می یابد. در جدول زیر الگوریتم های زمان بندی سیستم عامل های مختلف نشان داده شده است [2].

² Scheduler

³ Kernel Threads

⁴ ریشه های هسته در واقع، کدهای قابل زمان بندی در سطح هسته هستند که در نتیجه درخواست برنامه سطح کاربر (در متن پردازه) ایجاد شده و به آن پاسخ داده می شود. در بسیاری از سیستم عامل ها از جمله xv6 تناظر یک به یک میان پردازها و ریشه های هسته وجود دارد.

⁵ Short Term

⁶ Round Robin

⁷ Completely Fair Scheduler

⁸ Virtual Runtime

سیستم عامل	الگوریتم زمان بندی	توضیحات
Windows NT/Vista/7	MFQ	۳۲ صف ۰ تا ۱۵ اولویت عادی ۱۶ تا ۳۱ اولویت بی درنگ نرم
Mac OS X	MFQ	چندین صف با ۴ اولویت عادی، پراولویت سیستمی، فقط مد هسته، ریسه های بی درنگ
FreeBSD/NetBSD	MFQ	بیش از ۲۰۰ صف
Solaris	MFQ	۱۷۰ صف
Linux < 2.4	MFQ	-
$2.4 \leq \text{Linux} < 2.6$	EPOCH-based	سر بار بالا
$2.6 \leq \text{Linux} < 2.6.23$	Scheduler O(1)	پیچیده و سر بار پایین
$2.6.23 \leq \text{Linux}$	CFS	-
xv6	RR	-

زمان بندی در xv6:

هسته xv6 از نوع با ورود مجدد⁹ و غیرقبضه ای¹⁰ است. به این ترتیب، اجرای زمان بند تنها در نقاط محدودی از اجرا صورت می گیرد. به عنوان مثال، چنانچه در آزمایش دوم مشاهده شد، وقفه های قابل چشم پوشی¹¹ قادر به وقفه دادن به یکدیگر نبوده و تنها امکان توقف تله های غیر وقفه را دارند. همچنین تله های غیر وقفه نیز قادر به توقف یکدیگر نیستند. به طور دقیق تر زمان بندی، تنها در زمان های محدودی ممکن است:

- هنگام وقفه تایمر
 - هنگام رهاسازی داوطلبانه که شامل به خواب رفتن پردازش یا خروج توسط فراخوانی `exit()` است.
- به خواب رفتن و فراخواندن `exit()` می تواند دلایل مختلفی داشته باشد. مثلاً یک پردازش می تواند به طور داوطلبانه از طریق فراخوانی سیستمی `sys_exit()`، تابع `exit()` را فراخوانی نماید. همچنین پردازش بد رفتار، هنگام مدیریت تله به طور داوطلبانه! مجبور به فراخوانی `exit()` خواهد شد (خط ۳۴۶۹). همه این حالات در نهایت منجر به فراخوانی تابع `sched()`، (خط ۲۸۰۷) و به دنبال آن اجرای تابع زمان بندی یا `scheduler()` می گردند (خط ۲۷۵۷).

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

⁹ Reentrant

¹⁰ Non Preemptive

¹¹ Maskable Interrupts

زمان بندی:

همان طور که پیش تر ذکر شد، زمان بند xv6 از نوع نوبت گردشی است. به عبارت دیگر هر پردازش دارای یک برش زمانی¹² بوده که این برش، حداکثر زمانی است که قادر به نگه داری پردازنده در یک اجرای پیوسته می باشد. این زمان برابر یک تیک تایمر (حدود ۱۰ میلی ثانیه) می باشد.¹³ با وقوع وقفه تایمر که در هر تیک رخ می دهد تابع yield() فراخوانی شده (خط ۳۴۷۵) و از اتمام برش زمانی پردازش جاری خبر خواهد داد.

زمان بندی توسط تابع scheduler() صورت می پذیرد. این تابع از یک حلقه تشکیل شده که در هر بار اجرا با مراجعه به صف پردازش ها یکی از آن ها که قابل اجرا است را انتخاب نموده و پردازنده را جهت اجرا در اختیار آن قرار می دهد (خط ۲۷۸۱).

۲) صف پردازش هایی که تنها منبعی که برای اجرا کم دارند پردازنده است، صف آماده¹⁴ یا صف اجرا¹⁵ نام دارد. در xv6 صف آماده مجزا وجود نداشته و از صف پردازش ها بدین منظور استفاده می گردد. در زمان بند کاملاً منصف در لینوکس، صف اجرا چه ساختاری دارد؟

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.

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

۵) وقفه ها اولویت بالاتری نسبت به پردازش 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.

¹² Time Slice

¹³ تنظیمات تایمر هنگام بوت صورت می پذیرد.

¹⁴ Ready Queue

¹⁵ Run Queue

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

مدیریت وقفه‌ها در صورتی که بیش از حد زمان بر شود، می‌تواند منجر به گرسنگی پردازها گردد. این می‌تواند به خصوص در سیستم‌های بی‌درنگ¹⁶ مشکل ساز باشد. چگونه این مشکل حل شده است؟

تعویض متن:

پس از انتخاب یک پرداز جهت اجرا، توابع `switchvm()` و `switchkvm()` حالت حافظه پرداز 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 زمان بند (کد مدیریت کننده سیستم در آزمایش اول، که خود به نوعی ریس هسته بدون پرداز متناظر در سطح کاربر است) به حالت پرداز 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 با ساختار پشته، قبل و پس از تغییر اشاره گر پشته (خط ۳۰۷۱) به ترتیب در نیمه چپ و راست شکل زیر نشان داده شده است.

¹⁶ Realtime Systems

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

اشاره گر به "اشاره گر به متن" ریسه هسته قبلی در old، متن ریسه هسته قبلی در پنج ثبات بالای پشته سمت چپ و "اشاره گر به متن" ریسه هسته جدید در new قرار دارد. اشاره گر به "اشاره گر به متن" ریسه هسته جدید در 'old، متن ریسه هسته جدید در پنج ثبات بالای پشته سمت راست و اشاره گر به متن ریسه هسته ای که قبلاً

این ریسه هسته جدید به آن تعویض متن¹⁷ کرده بود، در 'new' قرار دارد. متن ریسه هسته جدید از پشته سمت راست به پردازنده منتقل شده (خطوط ۳۰۷۴ تا ۳۰۷۸) و نهایتاً پردازش سطح کاربر اجرا خواهد شد.

زمان بندی بازخوردی چند سطحی:

در این زمان بند، پردازش ها با توجه به اولویتی 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 ای را پیاده سازی کنید که بتواند پردازش را بین سطوح مختلف جابجا کرده تا قادر به اعمال الگوریتم های مختلف در 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 نوبت گردشی¹⁸، در سطح دوم الگوریتم زمان بندی آخرین ورود-اولین کار¹⁹ و در سطح سوم الگوریتم زمان بند اولین بهترین کار²⁰ را اعمال کنید. لازم به ذکر است که میان سطوح، اولویت وجود دارد؛ به این صورت که ابتدا تمام پردازش های سطح اول، سپس در صورت خالی بودن سطح اول، تمام پردازش های سطح دوم و در صورت خالی بودن هر دو سطح قبل، تمام پردازش های سطح سوم اجرا خواهند شد.

و شما با فراخوانی سیستمی ای که پیاده سازی می کنید می توانید سطح پردازش ها را تغییر دهید. همچنین زمان بند پیاده سازی شده توسط شما باید دارای قابلیت Aging بوده و اگر پردازش های بیشتر از زمانی معین اجرا نشود، آن پردازش را به سطح اول منتقل کند.

سازوکار افزایش سن:

¹⁷ Context Switch

¹⁸ Round Robin

¹⁹ Last Come First Serve

²⁰ Best Job First

همانطور که در کلاس درس فرا گرفتید، برای جلوگیری از گرسنگی²¹، می توان از مکانیزم افزایش سن²² بهره برد. بدین صورت که اولویت پردازشهایی که مدت زیادی صبر کردند و پردازنده به آنها اختصاص نیافته، به مرور افزایش می یابد. در زمان بندی که پیاده سازی می کنید پردازشها را به طور پیش فرض در صف دوم قرار دهید و در صورتی که پردازش ۸۰۰۰ سیکل منتظر مانده باشد، آن را به صف اول منتقل کنید. در صورت بازانتقال این پردازش به صف های دیگر، رصد کردن تعداد سیکل اجرا نشده پردازش را از ابتدا از سر بگیرید.

● سطح اول: زمان بند نوبت گردشی

در این زمان بند یک واحد زمانی کوچک به نام برش زمانی یا کوانتوم زمانی²³ داریم. در این زمان بند صف پردازشهای آماده اجرا را به صورت یک صف حلقوی در نظر می گیریم. بر اساس این زمان بند، به صورت چرخشی پردازنده به پردازشها برای بازه زمانی حداکثر یک کوانتوم زمانی اختصاص می یابد. به عبارت دیگر زمان بند، پردازش موجود در ابتدای صف را انتخاب نموده و یک تایمر برای پردازنده تنظیم می کند که پس از یک کوانتوم زمانی، پردازنده در اختیار پردازش دیگر قرار گیرد. پردازشها در این نوع زمان بند به دو صورت عمل می کنند:

● 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 می کند. پس از آن پردازنده، پردازش بعدی که در ابتدای صف قرار دارد را انتخاب می نماید.

● حالت دوم، حالتی که زمان مورد نیاز پردازش بیشتر از یک کوانتوم زمانی است؛ در این حالت تایمر خاموش شده و منجر به وقفه در اجرا می گردد. سپس تعویض متن رخ داده و پردازش در انتهای صف اجرا قرار می گیرد. پس از آن، پردازنده پردازش ابتدای صف اجرا را انتخاب می کند.

²¹ Starvation

²² Aging

²³ Time Quantum

نکته‌ای که باید در پیاده‌سازی این الگوریتم رعایت شود این است که پردازها به ترتیب ورود به صف، اجرا خواهند شد و پرداز جدید، به انتهای زنجیره پردازهای منتظر اجرا افزوده می‌شود.

● سطح دوم: زمان‌بند آخرین ورود-اولین رسیدگی (LCFS)

با نحوه عملکرد LCFS در کلاس درس آشنا شده‌اید. در این زمان‌بند، پردازهای که آخر می‌آید، ابتدا اجرا می‌شود و پرداز بعدی (به ترتیب ورود به آخر صف) تنها پس از اجرای کامل قبلی شروع می‌شود.

نکته قابل توجه در این الگوریتم زمان ایجاد هر پرداز می‌باشد. لازم است تا با تغییر در ساختار فایل‌های مربوط به پرداز (proc.c, proc.h) زمان ایجاد هر پرداز را در اختیار داشته باشید.

نکته: پارامترهای جدیدی که برای الگوریتم‌های مختلف زمان‌بندی به پرداز اضافه می‌کنید و هنگام ایجاد پرداز، آن‌ها را مقداردهی می‌کنید، باید به گونه‌ای مقداردهی شوند که به پردازهای که exec می‌شوند، مانند پردازهای که توسط پوسته²⁴ ساخته و اجرا می‌شوند به سایر پردازها که تنها fork می‌شوند و exec نمی‌شوند اولویت داده‌شود تا پوسته شما قفل نشود.

● سطح سوم: زمان‌بند اول بهترین کار (BJF)

در این بخش تقریبی از الگوریتم BJF پیاده‌سازی خواهد شد. در این حالت لازم است برای پرداز زمان ورود و تعداد سیکل اجرا را مشخص نمایید. برای محاسبه زمان ورود می‌توانید از زمان سیستم هنگام ایجاد پرداز استفاده نموده و برای محاسبه تعداد سیکل اجرا، باید یک مشخصه برای پرداز خود با همین نام در نظر بگیرید. مقدار اولیه تعداد سیکل اجرا را هنگام ساخته شدن پرداز برابر صفر در نظر بگیرید. به ازای هر بار اجرای پرداز ۰.۱ واحد به تعداد سیکل اجرایی آن بیفزایید. ابتدا معیاری را تحت عنوان رتبه برای هر پرداز تعریف می‌کنیم. این معیار برای هر پرداز به صورت زیر قابل تعریف است:

$$\text{rank} = (\text{Priority} * \text{PriorityRatio}) + (\text{ArrivalTime} * \text{ArrivalTimeRatio}) + (\text{ExecutedCycle} * \text{ExecutedCycleRatio}) + (\text{ProcessSize} * \text{ProcessSizeRatio})$$

²⁴ Shell

در این فرمول با داشتن اطلاعات در مورد اولویت، زمان ورود پردازش به صف، تعداد سیکل‌های اجرا شده هر برنامه و سایر پردازش‌ها می‌توانیم رتبه هر پردازش را داشته باشیم. عدد پایین‌تر اولویت، معادل اولویت بالاتر است. چهار ضریب فوق توسط فراخوانی‌های سیستمی مربوطه مقداردهی می‌شود. زمانبندی بر اساس رتبه پردازش‌ها صورت می‌گیرد و اولویت اجرا با پردازش‌های است که رتبه کمتری دارد.

❖ 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 به پردازش‌هایی که exec می‌شوند، مانند پردازش‌هایی که توسط پوسته²⁵ ساخته و اجرا می‌شوند به سایر پردازش‌ها که تنها fork می‌شوند و exec نمی‌شوند اولویت داده شود تا پوسته شما قفل نشود.

فراخوانی‌های سیستمی مورد نیاز:

1. **تغییر صف پردازش:** پس از ایجاد پردازش‌ها (به تعداد لازم)، باید با نوشتن یک فراخوانی سیستمی مناسب مشخص کنید که هر پردازش مربوط به کدام صف از سه صفی که پیاده‌سازی کرده‌اید، تعلق دارد. همچنین باید بتوان یک پردازش را از یک صف به صف دیگری انتقال داد. این فراخوانی سیستمی، PID پردازش و شماره صف مقصد را به عنوان ورودی دریافت می‌کند و صف پردازش را تعیین می‌کند یا تغییر می‌دهد.
2. **مقدار دهی پارامتر BJB در سطح پردازش:** طی این فراخوانی سیستمی، باید بتوان پارامترهای موثر در محاسبه مقدار BJB متناظر با یک پردازش را تغییر داد. ورودی این فراخوانی سیستمی، PID پردازش مورد نظر و مقدار برای ضریب در معادله BJB می‌باشد.
3. **مقدار دهی پارامتر BJB در سطح سیستم:** طی این فراخوانی سیستمی، باید بتوان پارامترهای موثر در محاسبه مقدار BJB متناظر با همه پردازش‌ها را تغییر داد. ورودی این فراخوانی سیستمی، مقدار برای ضریب در معادله BJB می‌باشد.

²⁵ Shell

4. چاپ اطلاعات: برای اینکه برنامه شما قابل آزمون باشد، باید یک فراخوانی سیستمی پیاده سازی کنید که لیست تمام پردازها را چاپ نموده و در هر سطر این لیست باید نام پرداز، شماره پرداز، وضعیت، شماره صف، زمان ورود و باقی اطلاعات در آن گنجانده شود. یک مثال در شکل زیر نشان داده شده است. توجه کنید در صورتی که تمامی مقادیر فوق چاپ نشود، نمره کسر می گردد.

Process_Name	PID	State	Queue	Cycle	Arrival	Priority	R_Prt	R_Arvl	R_Exec	R_Size	Rank
init	1	sleeping	1	1	0	3	1	1	1	1	12292
sh	2	sleeping	1	2	5	3	1	1	1	1	16394
foo	5	runnable	2	44	369	3	1	1	1	1	12704
foo	4	sleeping	2	0	368	3	1	1	1	1	12659
foo	6	sleeping	2	44	369	3	1	1	1	1	12704
foo	7	sleeping	2	44	369	3	1	1	1	1	12704
foo	8	sleeping	2	44	369	3	1	1	1	1	12704
foo	9	runnable	2	44	369	3	1	1	1	1	12704
schedule	10	running	2	1	812	3	1	1	1	1	17200

جهت حصول اطمینان از زمان بند خود، یک برنامه سطح کاربر با نام foo بنویسید که تعدادی پرداز در آن ساخته شده و پردازها عملیات پردازشی²⁶ انجام دهند؛ تا فرصت بررسی عملکرد زمان بند وجود داشته باشد. می توان این برنامه را با اجرای دستور زیر در پس زمینه اجرا نموده و در این حین، توسط فراخوانی سیستمی چاپ اطلاعات از نحوه عملکرد آن مطلع شد.

foo&

توجه کنید که در برنامه foo فراخوانی سیستمی صدا نمی شود. فراخوانی های سیستمی فوق را به صورت برنامه سطح کاربری در بیاورید که بتوان آن را به صورت مستقیم از پوسته فراخوانی کرده و آرگومان ها را به آن ارسال نمود.

²⁶ 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 سوالات 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 0 تعلق می‌گیرد.
- فصل پنجم کتاب xv6 می‌تواند مفید باشد.
- بهتر است هرگونه سوال در مورد پروژه را از طریق فروم درس مطرح نمایید.

موفق باشید.

مراجع:

- [1] Mohammed A F Al-Husainy. 2007. Best-job-first CPU scheduling algorithm. *Inf. Technol. J.* 6, 2 (2007), 288–293. Retrieved from <https://scialert.net/fulltext/?doi=itj.2007.288.293>
- [2] Donald H. Pinkston. 2014. Caltech Operating Systems Slides.