

آزمایشگاه سیستم عامل پروژه سوم: زمان بندی پردازها

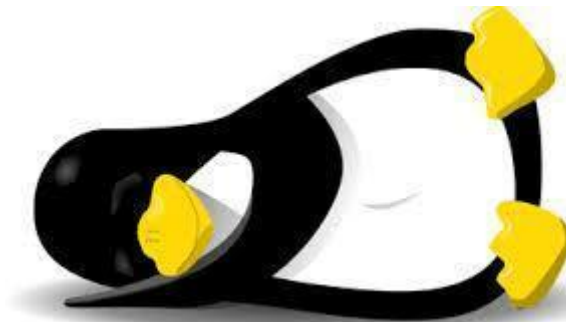


سیستم های عامل - بهار ۱۴۰۲

دانشکده مهندسی برق و کامپیوتر

مسئولان تمرین:
امیر فراهانی، نگین باغبانزاده

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 پردازنده ها تخصیص داده می شود. این بخش از سیستم عامل، در سطح هسته اجرا شده و به بیان دقیق تر، زمان بند ریسه های هسته³ را زمان بندی می کند.⁴

دقت شود وظیفه زمان بند، زمان بندی پردازنده ها (و نه همه کدهای سیستم) از طریق زمان بندی ریسه های هسته متناظر با آن ها 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 آن ها 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، پردازنده ها دارای اولویت های مختلف هستند و به طور کلی تلاش می شود که تا جای امکان پردازنده ها 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 پردازنده یک زمان اجرای مجازی⁸ دارد که در هر بار زمان بندی، پردازنده دارای کمترین زمان اجرای مجازی، اجرا خواهد شد. هر چه اولویت پردازنده 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 ریسه های هسته در واقع، کدهای قابل زمان بندی در سطح هسته هستند که در نتیجه درخواست برنامه سطح کاربر (در متن پردازنده) ایجاد شده و به آن پاسخ داده می شود. در بسیاری از سیستم عامل ها از جمله xv6 تناظر یک به یک میان پردازنده ها و ریسه 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	سربار بالا
$2.6 \leq \text{Linux} < 2.6.23$	Scheduler O(1)	پیچیده و سربار 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 نگهداری 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 پردازنده‌ها چگونه است؟

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

تعویض متن:

پس از انتخاب یک پردازنده جهت اجرا، توابع `switchvm()` و `switchkvm()` حالت حافظه پردازنده 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 ریسک هسته بدون پردازنده متناظر در سطح کاربر است) به حالت پردازنده 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" ریس هسته قبل در old، متن ریس هسته قبل در پنج ثبات بالای پشته سمت چپ و "اشاره گر به متن" ریس هسته جدید در new قرار دارد. اشاره گر به "اشاره گر به متن" ریس هسته جدید در 'old، متن ریس هسته جدید در پنج ثبات بالای پشته سمت راست و اشاره گر به متن ریس 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 بخت آزمایی¹⁹ و در سطح سوم اولین ورود-اولین 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:

همانطور که در کلاس درس فرا گرفتید، برای جلوگیری از گرسنگی²¹، می توان از مکانیزم افزایش سن²² بهره برد. بدین صورت که اولویت پردازش هایی که مدت زیادی صبر کردند و پردازنده به آن ها اختصاص نیافته، به مرور

¹⁷ Context Switch

¹⁸ Round Robin

¹⁹ Lottery Process Scheduling

²⁰ First Come First Serve

²¹ Starvation

²² Aging

افزایش می یابد. در زمان بندی که پیاده سازی می کنید پدازه ها را به طور پیش فرض در صف دوم قرار دهید و در صورتی که پدازه ای ۸۰۰۰ سیکل منتظر مانده باشد، آن را به صف اول منتقل کنید. در صورت بازانتقال این پدازه به صف های دیگر، رصد کردن تعداد سیکل اجرا نشده پدازه 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 پدازه های آماده اجرا را به صورت یک صف حلقوی در نظر می گیریم. بر اساس این زمان بند، به صورت چرخشی پدازنده به پدازه ها برای بازه زمانی حداکثر یک کوانتوم زمانی اختصاص می یابد. به عبارت دیگر زمان بند، پدازه موجود در ابتدای صف را انتخاب نموده و یک تایمر برای پدازنده تنظیم می کند که پس از یک کوانتوم زمانی، پدازنده در اختیار پدازه دیگر قرار گیرد. پدازه ها در این نوع زمان بند به دو صورت عمل می کنند:

● حالت اول زمانی است که زمان مورد نیاز پدازه کمتر یا مساوی یک کوانتوم زمانی است؛ در این حالت پدازه به صورت داوطلبانه پدازنده را رها می کند. پس از آن پدازنده، پدازه بعدی که در ابتدای صف قرار دارد را انتخاب می نماید.

● حالت دوم، حالتی که زمان مورد نیاز پدازه 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 پدازه در انتهای صف اجرا قرار می گیرد. پس از آن، پدازنده پدازه ابتدای صف اجرا را انتخاب می کند.

نکته ای که باید در پیاده سازی این الگوریتم رعایت شود این است که پدازه ها به ترتیب ورود به صف، اجرا خواهند شد و پدازه جدید، به انتهای زنجیره پدازه های منتظر اجرا افزوده می شود.

²³ Time Quantum

● سطح دوم: زمان بند بخت آزمایی

این زمانبند بر پایه تخصیص منابع به پردازها به صورت تصادفی می باشد. هر پرداز به توجه به تعداد بلیت شانشی که دارد احتمال انتخاب شدنش به عنوان پرداز بعدی برای اجرا مشخص می شود. انتخاب پرداز برای اجرا توسط زمانبند پردازنده به این صورت می باشد که هر پرداز تعدادی بلیت شانس دارد و پردازنده به صورت تصادفی یک بلیت را انتخاب نموده و پرداز صاحب آن بلیت، اجرا خواهد شد. هنگامی که اجرای این پرداز 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 پرداز می تواند بازه ای از اعداد را به عنوان بلیت های شانس خود داشته باشد. زمانبند پردازها با تولید عددی تصادفی در بازه کل این اعداد، یک بلیت و متناظر با آن یک پرداز را برای اجرا انتخاب می کند. به عنوان مثال دو پرداز A و B داریم و A دارای ۶۰ بلیت شانس (بلیت های شماره ۱ تا ۶۰) و B دارای ۴۰ بلیت شانس (بلیت های شماره ۶۱ تا ۱۰۰) می باشد. زمانبند در هر مرحله، عددی تصادفی بین ۱ تا ۱۰۰ را انتخاب نموده و اگر عدد انتخاب شده بین ۱ تا ۶۰ باشد، پرداز A و در غیر این صورت پرداز B انتخاب می گردد. در زیر مثالی از ۱۰ مرحله انتخابی توسط زمانبند پردازنده نشان داده شده است.

عدد تصادفی انتخاب شده توسط زمانبند: 11 55 48 94 88 62 51 78 3 40

پرداز انتخاب شده بر اساس عدد تصادفی: A A B A B B B A A A

● سطح سوم: زمان بند اولین ورود-اولین رسیدگی (FCFS)

با نحوه عملکرد FCFS در کلاس درس آشنا شده اید. در این زمان بند، پردازهای که اول می آید، ابتدا اجرا می شود و پرداز بعدی (به ترتیب ورود به آخر صف) تنها پس از اجرای کامل قبلی شروع می شود.

نکته قابل توجه در این الگوریتم زمان ایجاد هر پرداز می باشد. لازم است تا با تغییر در ساختار فایل های 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 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 پوسته²⁴ ساخته و اجرا می‌شوند به سایر پردازش‌ها که تنها fork می‌شوند و exec نمی‌شوند اولویت داده‌شود 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. مقداردهی بلیت بخت‌آزمایی: باید به هرکدام از پردازش‌هایی که در صف دوم قرار دارند تعدادی بلیت اختصاص دهید تا الگوریتم بخت‌آزمایی قابل اجرا باشد. بنابراین باید یک فراخوانی سیستمی پیاده‌سازی کنید که به پردازش‌های صف دوم بلیت مربوطه را تخصیص دهد. ورودی، PID پردازش مورد نظر و مقدار بلیت آن خواهد بود.
3. چاپ اطلاعات: برای اینکه برنامه شما قابل آزمون 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، زمان ورود و مقدار بلیت بخت‌آزمایی در آن گنجانده شود. یک مثال نیمه کامل در شکل زیر نشان داده شده است. توجه کنید در صورتی که تمامی مقادیر فوق چاپ نشود، نمره کسر می‌گردد.

²⁴ Shell

name	pid	state	queue	arrive time	ticket	cycle
init	1	SLEEPING	0	0	0	1
sh	2	SLEEPING	0	2	8	1
foo	6	RUNNABLE	2	1055	25	10
foo	5	SLEEPING	2	1055	25	1
foo	7	RUNNABLE	2	1055	25	13
foo	8	RUNNABLE	2	1055	25	7
foo	9	RUNNABLE	2	1055	25	12
foo	10	RUNNABLE	2	1055	25	10
foo	11	RUNNABLE	2	1055	25	8
foo	12	RUNNABLE	2	1055	25	11
foo	13	RUNNABLE	2	1055	25	13
foo	14	RUNNABLE	2	1063	47	15
foo	15	RUNNABLE	2	1063	47	16
ps	16	RUNNING	2	1153	27	0
\$						

جهت حصول اطمینان از زمان بند خود، یک برنامه سطح کاربر با نام foo بنویسید که تعدادی پردازش در آن ساخته شده و پردازشها عملیات پردازشی²⁵ 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.

²⁵ 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.