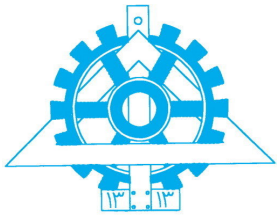
آزمایشگاه سیستم‌عامل
پروژه سوم: زمان‌بندی پردازه‌ها

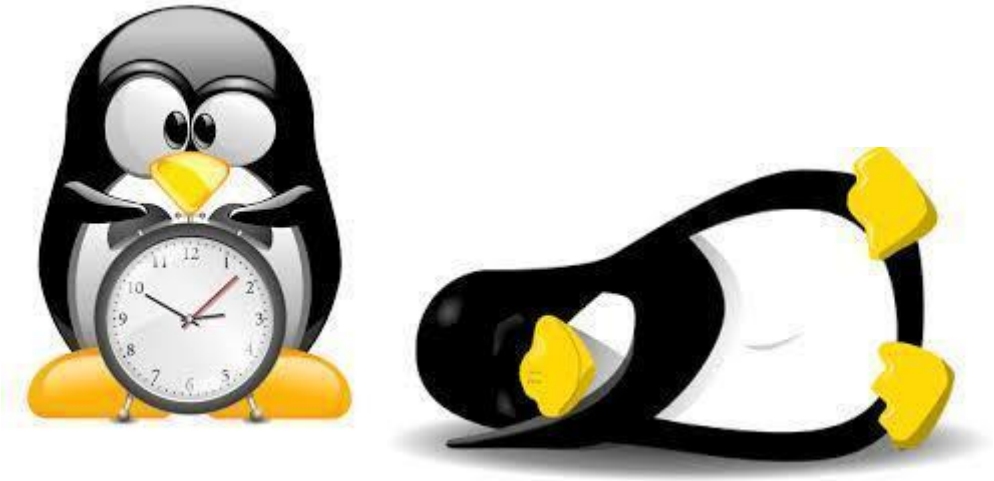


دانشکده مهندسی برق و کامپیوتر

استاد:
دکتر مهدی کارگهی

سیستم‌های عامل - بهار ۱۴۰۱

مسئولان تمرین:
آرمین افشاریان، حمید خدادادی



در این پروژه با زمان‌بندی در سیستم‌عامل‌ها آشنا خواهید شد. در این راستا در ابتدا الگوریتم زمان‌بندی سیستم عامل xv6 بررسی شده و با ایجاد تغییراتی در آن الگوریتم، زمان‌بندی صف بازخوردی چند سطحی[1] (MFQ) پیاده‌سازی می‌گردد. هم‌چنین نحوه استفاده از فاکتور زمان در این سیستم‌عامل بررسی می‌گردد. در انتهای پروژه، با فراخوانی‌های سیستمی پیاده‌سازی شده، از صحت عملکرد زمان‌بند اطمینان حاصل خواهد شد.

[1] Multilevel Feedback Queue Scheduling

مقدمه:

همان‌طور که در پروژه اول اشاره شد، یکی از مهم‌ترین وظایف سیستم‌عامل ها، تخصیص منابع سخت‌افزاری به برنامه‌های سطح کاربر است. در این امر، پردازنده مهم‌ترین منبع از این منابع بوده که توسط زمان‌بند[2] سیستم‌عامل به پردازه‌ها تخصیص داده می‌شود. این بخش از سیستم‌عامل، در سطح هسته اجرا شده و به بیان دقیق‌تر، زمان‌بند ریسه‌های هسته[3] را زمان‌بندی می‌کند.[4]

دقت شود وظیفه زمان‌بند، زمان‌بندی پردازه‌ها (و نه همه کدهای سیستم) از طریق زمان‌بندی ریسه‌های هسته متناظر با آن‌ها است. کدهای مربوط به وقفه سخت‌افزاری، تحت کنترل زمان‌بند قرار نمی‌گیرند. اغلب زمان‌بندهای سیستم‌عامل‌ها از نوع کوتاه‌مدت[5] 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[6] (RR) است. الگوریتم زمان‌بندی صف بازخوردی چند سطحی 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[7] (CFS) نامیده می‌شود. در این الگوریتم، پردازه‌ها دارای اولویت‌های مختلف هستند و به طور کلی تلاش می‌شود که تا جای امکان پردازه‌ها با توجه به اولویتشان برای اجرا، سهم متناسبی از پردازنده را در اختیار بگیرند. به طور ساده تر می‌توان آن را به نوعی زمان‌بند نوبت‌گردشی تصور نمود. هر پردازه یک زمان اجرای مجازی[8] دارد که در هر بار زمان‌بندی، پردازه دارای کمترین زمان اجرای مجازی، اجرا خواهد شد. هر چه اولویت پردازه بالاتر باشد زمان اجرای مجازی آن به صورت کندتر افزایش می‌یابد. در جدول زیر الگوریتم‌های زمان‌بندی سیستم‌عامل‌های مختلف نشان داده شده است [2].

---

[2] Scheduler

[3] Kernel Threads

[4] ریسه‌های هسته در واقع، کدهای قابل زمان‌بندی 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 سیستم‌عامل‌ها از جمله xv6 تناظر یک‌به‌یک میان پردازه‌ها و ریسه‌های 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	-
$2.4 \leq$ Linux $<$ 2.6	EPOCH-based	سربار بالا
$2.6 \leq$ Linux $<$ 2.6.23	Scheduler O(1)	پیچیده و سربار پایین
$2.6.23 \leq$ 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 غیر وقفه را دارند. هم‌چنین تله‌های غیر وقفه نیز قادر به توقف یکدیگر نیستند. به طور دقیق‌تر زمان‌بندی، تنها در زمان‌های محدودی ممکن است:

- هنگام وقفه تایمر
- هنگام رهاسازی داوطلبانه که شامل به خواب رفتن پردازه یا خروج توسط فراخوانی exit() است.

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 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 زمان‌بندی یا scheduler() می‌گردند (خط ۲۷۵۷).

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

[9] Reentrant

[10] Non Preemptive

[11] Maskable Interrupts

زمان‌بندی:

همان‌طور که پیش‌تر ذکر شد، زمان‌بند xv6 از نوع نوبت‌گردشی است. به عبارت دیگر هر پردازه دارای یک برش زمانی[12] بوده که این برش، حداکثر زمانی است که قادر به نگه‌داری پردازنده در یک اجرای پیوسته می‌باشد. این زمان برابر یک تیک تایمر (حدود ۱۰ میلی‌ثانیه) می‌باشد.[13] 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[14] یا صف اجرا[15] 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؟

۳) همان‌طور که در پروژه اول 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. 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 می‌گردد. علت چیست؟ آیا در سیستم تک‌هسته‌ای به آن نیاز است؟

۵) وقفه‌ها اولویت بالاتری نسبت به پردازه‌ها دارند. به طور کلی مدیریت وقفه‌ها در لینوکس در دو سطح صورت می‌گیرد. آن‌ها را نام برده و به اختصار توضیح دهید.

---

[12] Time Slice

[13] تنظیمات تایمر هنگام بوت صورت می‌پذیرد.

[14] Ready Queue

[15] Run Queue

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

مدیریت وقفه‌ها در صورتی که بیش از حد زمان‌بر شود، می‌تواند منجر به گرسنگی پردازه‌ها گردد. این می‌تواند به خصوص در سیستم‌های بی‌درنگ[16] مشکل‌ساز باشد. چگونه این مشکل حل شده است؟

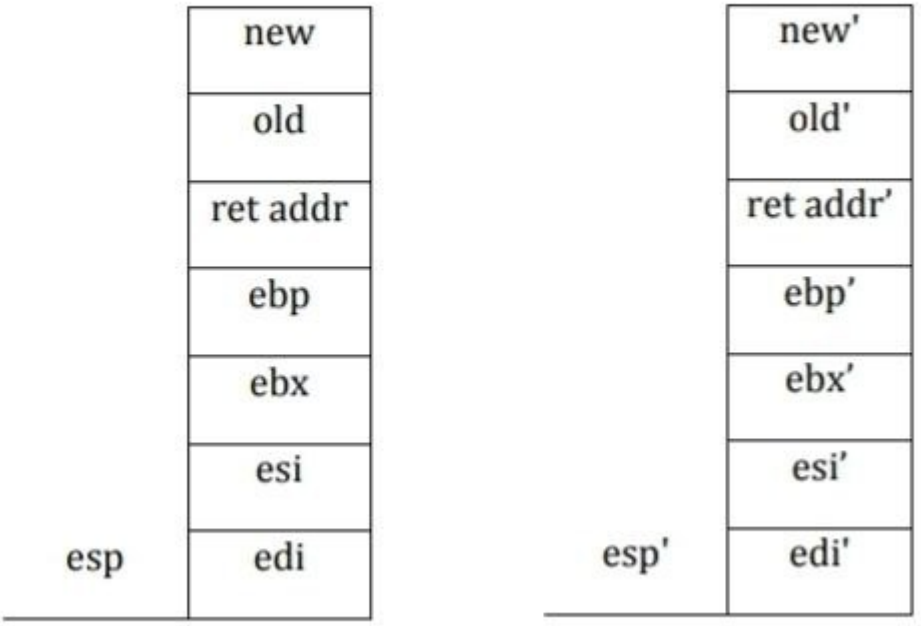
تعویض متن:

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 اول، 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 با ساختار پشته، قبل و پس از تغییر اشاره‌گر پشته (خط ۳۰۷۱) به ترتیب در نیمه چپ و راست شکل زیر نشان داده شده است.

[16] Realtime Systems



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[17] 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.

[17] Context Switch

زمان‌بندی بازخوردی چند سطحی:

در این زمان‌بند، پردازه ها 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. شما برای آزمودن زمان‌بند خود باید فراخوانی سیستمی ای را پیاده‌سازی کنید که بتواند پردازه را بین سطوح مختلف جابجا کرده تا قادر به اعمال الگوریتم های مختلف در هر صف باشید. همانطور که گفته شد زمان‌بندی که توسط شما پیاده‌سازی می‌شود دارای سه سطح می‌باشد که لازم است در سطح یک الگوریتم زمان‌بندی نوبت گردشی[18]، در سطح دوم الگوریتم زمان‌بندی اولین ورود - اولین رسیدگی[19] و در سطح سوم الگوریتم زمان‌بندی اول بهترین کار[20] را اعمال کنید. لازم به ذکر است که میان سطوح، اولویت وجود دارد؛ به این صورت که ابتدا تمام پردازه های سطح اول، سپس در صورت خالی بودن سطح اول، تمام پردازه‌های سطح دوم و در صورت خالی بودن هر دو سطح قبل، تمام پردازه‌های سطح سوم اجرا خواهند شد.

و شما با فراخوانی سیستمی ای که پیاده‌سازی می‌کنید می‌توانید سطح پردازه‌ها را تغییر دهید.

همچنین زمان‌بند پیاده‌سازی شده توسط شما باید دارای قابلیت Aging بوده و اگر پردازه‌ای بیشتر از زمانی معین اجرا نشود، آن پردازه را به سطح اول منتقل کند.

سازوکار افزایش سن:

همانطور که در کلاس درس فرا گرفتید، برای جلوگیری از گرسنگی[21]، می‌توان از مکانیزم افزایش سن[22] بهره برد. بدین صورت که اولویت پردازه‌هایی که مدت زیادی صبر کردند و پردازنده به آن‌ها اختصاص نیافته، به مرور افزایش می‌یابد. در زمان‌بندی که پیاده‌سازی می‌کنید پردازه‌ها را به طور پیش‌فرض در صف دوم قرار دهید و در

[18] Round Robin

[19] First Come First Serve

[20] Best Job First

[21] Starvation

[22] Aging

صورتی که پردازه‌ای ۸۰۰۰ سیکل منتظر مانده باشد، آن را به صف اول منتقل کنید. در صورت باز انتقال این پردازه به صف های دیگر، رصد کردن تعداد سیکل اجرا نشده پردازه را از ابتدا از سر بگیرید.

- **سطح اول: زمان‌بند نوبت گردشی**

در این زمان‌بند یک واحد زمانی کوچک به نام برش زمانی یا کوانتوم زمانی[23] داریم. در این زمان‌بند صف پردازه‌های آماده اجرا را به صورت یک صف حلقوی در نظر می‌گیریم. بر اساس این زمان بند، به صورت چرخشی پردازنده به پردازه‌ها برای بازه زمانی حداکثر یک کوانتوم زمانی اختصاص می‌یابد.

به عبارت دیگر زمان‌بند، پردازه موجود در ابتدای صف را انتخاب نموده و یک تایمر برای پردازنده تنظیم می‌کند که پس از یک کوانتوم زمانی، پردازنده در اختیار پردازه دیگر قرار گیرد. پردازه‌ها در این نوع زمان‌بند به دو صورت عمل می‌کنند:

- حالت اول زمانی است که زمان مورد نیاز پردازه کمتر یا مساوی یک کوانتوم زمانی است؛ در این حالت پردازه به‌صورت داوطلبانه پردازنده را رها می کند. پس از آن پردازنده، پردازه بعدی که در ابتدای صف قرار دارد را انتخاب می‌نماید.
- حالت دوم، حالتی که زمان مورد نیاز پردازه بیشتر از یک کوانتوم زمانی است؛ در این حالت تایمر خاموش شده و منجر به وقفه در اجرا می‌گردد. سپس تعویض‌متن رخ داده و پردازه در انتهای صف اجرا قرار می گیرد. پس از آن، پردازنده، پردازه ابتدای صف اجرا را انتخاب می‌کند.

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

[23] Time Quantum

- **سطح دوم: زمان‌بند اولین ورود-اولین رسیدگی (FCFS)**

با نحوه عملکرد FCFS در کلاس درس آشنا شده اید. در این زمان‌بند، پردازه ای که اول می‌آید، ابتدا اجرا می‌شود و پردازه بعدی (به ترتیب ورود به آخر صف) تنها پس از اجرای کامل قبلی شروع می‌شود.

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

- **سطح سوم: زمان‌بند اول بهترین کار (BJF)**

در این بخش تقریبی از الگوریتم BJF پیاده‌سازی خواهد شد. در این حالت لازم است برای پردازه زمان ورود و تعداد سیکل اجرا را 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 اولیه تعداد سیکل اجرا را هنگام ساخته شدن پردازه برابر صفر در نظر بگیرید. به ازای هر بار اجرای پردازه ۰.۱ واحد به تعداد سیکل اجرایی آن بیفزایید. ابتدا معیاری را تحت عنوان رتبه برای هر پردازه تعریف می‌کنیم. این معیار برای هر پردازه به صورت زیر قابل تعریف است:

rank = (Priority * PriorityRatio) + (ArrivalTime * ArrivalTimeRatio) + (ExecutedCycle * ExecutedCycleRatio)

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

❖ **نکته:** پارامترهای جدیدی که برای الگوریتم‌های مختلف زمان‌بندی به پردازه اضافه می‌کنید و هنگام ایجاد پردازه، آن‌ها را مقداردهی می‌کنید، باید به گونه‌ای مقداردهی شوند که به پردازه هایی که exec

می‌شوند، مانند پردازه‌هایی که توسط پوسته[24] ساخته و اجرا می‌شوند به سایر پردازه‌ها که تنها fork می‌شوند و exec نمی‌شوند اولویت داده‌شود تا پوسته شما قفل نشود.

❖

فراخوانی‌های سیستمی مورد نیاز:

1. **تغییر صف پردازه:** پس از ایجاد پردازه‌ها (به تعداد لازم)، باید با نوشتن یک فراخوانی سیستمی مناسب مشخص کنید که هر پردازه مربوط 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 و صف پردازه را تعیین می کند یا تغییر می دهد.
2. **مقدار دهی پارامتر BJF در سطح پردازه:** طی این فراخوانی سیستمی، باید بتوان پارامتر های موثر در محاسبه مقدار BJF متناظر با یک پردازه را تغییر داد. ورودی این فراخوانی سیستمی، PID پردازه مورد نظر و مقدار برای ضریب در معادله BJF می باشد.
3. **مقدار دهی پارامتر BJF در سطح سیستم:** طی این فراخوانی سیستمی، باید بتوان پارامتر های موثر در محاسبه مقدار BJF متناظر با همه پردازه ها را تغییر داد. ورودی این فراخوانی سیستمی، مقدار برای ضریب در معادله BJF می باشد.
4. **چاپ اطلاعات:** برای اینکه برنامه شما قابل آزمودن 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، زمان ورود، مقدار ضریب موثر، مقدار رتبه، تعداد سیکلی که پردازنده به آن پردازه اختصاص یافته است در آن گنجانده شود. یک مثال **نیمه‌کامل** 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_level	cycle	arrivel
init	1	SLEEPING	1	56	0
sh	2	SLEEPING	1	30	82
foo	5	RUNNABLE	2	6	2763
foo	4	SLEEPING	1	45	2752
foo	6	RUNNABLE	2	4	2766
foo	7	RUNNABLE	2	2	2771
foo	8	RUNNABLE	2	9	2774

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[25] انجام دهند؛ تا فرصت بررسی عملکرد زمان‌بند وجود داشته باشد. می‌توان این برنامه را با اجرای دستور زیر در پس‌زمینه 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 نمی‌شود. **فراخوانی‌های 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:

- **آدرس مخزن** و **شناسه آخرین تغییر** خود را در محل بارگذاری 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.