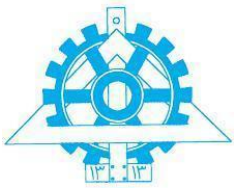


به نام خدا

آزمایشگاه سیستم عامل - بهار ۱۴۰۴

استاد: دکتر مهدی کارگهی

پروژه سوم: زمان‌بندی در xv6

طراحان: بهراد علمی - محمد امانلو

اهداف پروژه

در این پروژه به طور کلی با مفهوم زمان‌بندی[1] و به شکل عملی در سیستم عاملِ xv6 آشنا خواهید شد. در این راستا، ابتدا الگوریتم زمان‌بندی سیستم عامل xv6 بررسی خواهد شد. سپس به پیاده‌سازی یک زمان‌بندِ چند لایه[2] می‌پردازید. در نهایت با استفاده از فراخوانی‌های سیستمی و برنامهٔ‌های سطح کاربر، از صحت پیاده‌سازی خود اطمینان حاصل خواهید کرد.

مقدمه

یکی از مهمترین وظایف سیستم عامل‌ها، تخصیص منابع سخت‌افزاری به برنامه‌های سطح کاربر است. در این امر، پردازنده که یکی از منابع فعال[3] می‌باشد، نیازمند یک زمان‌بند مناسب به منظور اجرا شدنِ هر چه بهترِ پردازه‌ها می‌باشد. از آنجایی که زمان‌بند، نیازمندِ دانستنِ اطلاعاتِ سیستم و همچنین وضعیت هرکدام از پردازه‌ها[4] می‌باشد، غالبا در سطح کرنل اجرا می‌شود که در xv6 نیز همین گونه است.

---

[1] Scheduling

[2] Multi-Level Scheduler

[3] Active

[4] Processes

xv6 سیستم عاملی است که از مدل چند پردازشی متقارن[5] پشتیبانی می‌کند. با این قابلیت، هر پردازنده، یک کپی از کدِ اجراییِ زمان‌بند (تابع زمان‌بند) و همچنین متنِ[6] زمان‌بندِ خود را دارد و پس از انتخاب پردازه از صفِ پردازه‌هایِ آماده به اجرا[7]، توسط زمان‌بند (طبق الگوریتم زمان‌بندی)، با انجام عملیاتِ تعویض متن[8] از متنِ زمان‌بند به متن پردازه، اجرا خواهد شد.

یکی از ساده‌ترین و در عین حال کاربردی‌ترین الگوریتم‌های زمان‌بندی، زمان‌بندیِ نوبت گردشی[9] است که xv6 نیز از همین الگوریتم استفاده می‌کند. در سیستم‌های امروزه اما، به دلیلِ رشدِ نیاز به نشان دادنِ واکنشِ مناسب به اتفاقات متنوع در سیستم، نیاز به استفاده از الگوریتم‌هایِ زمان‌بندیِ پیشرفته‌تر نیز رشد کرده است. در جدول ۱، چند مورد از الگوریتم‌های مورد استفاده در سیستم عامل‌های مختلف آورده شده است.

جدول ۱. الگوریتم‌های زمان‌بندی استفاده شده در سیستم عامل‌های مختلف.

سیستم عامل	الگوریتم زمان‌بندی	توضیحات
Windows NT/Vista/7	MLFQ[10]	دارای ۳۲ سطح اولویت و ۲ صف است. اولویت‌های: ۰ تا ۱۵ برای تسک‌های عادی و ۱۶ تا ۳۱ برای تسک‌های بی‌درنگ[11].
Mac OS X	MLFQ	دارای ۱۲۷ سطح اولویت و ۴ صف اصلی است. اولویت‌های: ۰ تا ۵۱ تسک‌های عادی، ۵۲ تا ۷۹ تسک‌های با اهمیت‌تر، ۸۰ تا ۹۵ رزرو شده برای ریسه‌های کرنل[12] و ۹۶ تا ۱۲۷ ریسه‌هایی که اولویتشان بر مبنای نسبتی از سیکل‌های کلاک پردازنده که به آن‌ها تعلق می‌گیرد، تعریف می‌شود.

[5] Symmetric Multiprocessing (SMP)
[6] Context
[7] Ready Queue
[8] Context Switch
[9] Round Robin
[10] Multilevel Feedback Queue
[11] Real-time
[12] Kernel Threads

FreeBSD	ULE	دارای ۴ کلاس اولویت‌بندی اصلی است. که به ترتیب تسک‌های بی‌درنگ، تسک‌های تعاملی[13]، تسک‌های عادی و تسک‌های بی‌کار[14] هستند. این روش از پیش‌بینی میزان مصرف CPU استفاده می‌کند.
NetBSD	M2	دارای ۲۲۴ سطح اولویت و ۵ کلاس اولویت‌بندی اصلی است که به ترتیب تسک‌های بی‌درنگ، سیستمی، تعاملی با کاربر[15]، عادی با کاربر و تسک‌های بی‌کار است. زمان‌بندی آن مبتنی بر وزن و اولویت ایستا/پویا است.
Solaris	MLFQ	دارای ۵ کلاس که بازه اولویت‌بندی برای هر کلاس متفاوت است. کلاس‌ها به ترتیب تسک‌های بی‌درنگ، سیستمی، تعاملی، اشتراک‌زمانی[16]، اشتراک عادلانه[17] هستند.
2.5 < Linux < 2.6.23	O(1)	مرتبه اجرای ثابت از O(1) است. دو صف اولویت فعال و منقضی شده است. اولویتهای ایستا (۰–۱۴۰) با تعدیل پویا بر اساس رفتار تسک (مثلاً I/O-bound vs CPU-bound).
Linux > 2.6.23	CFS[18]	به هر پردازه، بر اساس سطح اولویت و میزان استفاده، زمان مناسبی برای پردازنده اختصاص دهد و اولویت‌های ثابت در آن کمتر دخیل هستند. زمان‌بندی بر اساس زمان مجازی[19] انجام می‌شود. اولویت‌ها به‌صورت پویا با وزن (nice)

[13] Interactive
[14] Idle
[15] User Interactive
[16] Time Sharing
[17] Fair-share
[18] Completely Fair Scheduler
[19] virtual runtime

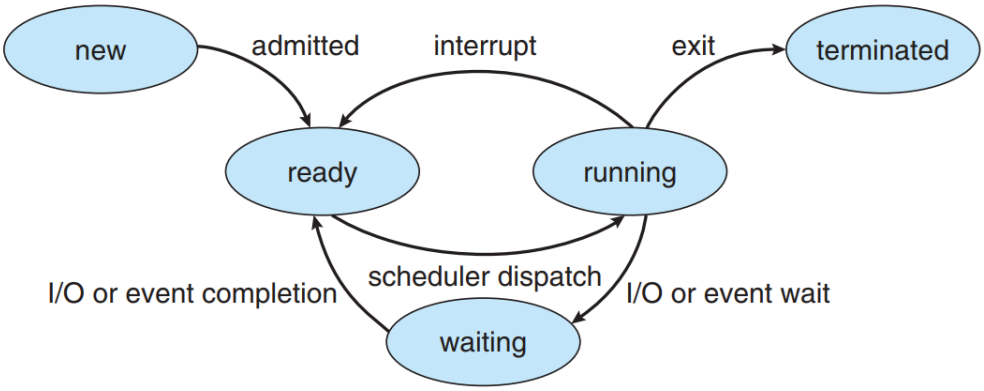
		values از ۲۰- تا ۱۹+) تنظیم می‌شوند. عدم استفاده از صف‌های سنتی با تمرکز بر تقسیم عادلانه CPU بین تمام تسک‌ها.
xv6	RR	همه فرآیندها از اولویت یکسانی برخوردارند و به ترتیب نوبتی اجرا می‌شوند، که باعث می‌شود زمان‌بندی مناسب برای سیستم‌های ساده و بی‌درنگ باشد، اما در مدیریت پیچیده‌تر اولویت‌ها و تسک‌ها دچار چالش است.

همانطور که مشاهده می‌کنید امروزه، زمان‌بندی‌هایی از کلاس‌های زمان‌بندی استفاده می‌کنند بسیار محبوب هستند. در این تمرین نیز با شکلی از این زمان‌بندی آشنا خواهید شد.

شرح پروژه

قبل از پیاده‌سازی زمان‌بندِ مورد نظر در این تمرین، ابتدا نیاز است تا با مفاهیم اولیه و همچنین نمود آن‌ها در xv6 آشنایی داشته باشید. در این قسمت به همین موضوع خواهیم پرداخت.

به طور کلی هر پردازه در چرخهٔ وضعیتِ خود شامل وضعیت‌های مختلفی است. شکل ۱ که مربوط به فصل ۳ منبع درس می‌باشد، این چرخه را به خوبی توصیف کرده است.



شکل ۱. چرخهٔ وضعیت یک پردازه.

xv6 نیز از این قاعده مستثنا نیست و به منظور زمان‌بندی و مدیریت پردازه‌ها، وضعیت هر کدام از پردازه‌ها را در PCB[20] آن پردازه نگه می‌دارد. عمدهٔ عملیات زمان‌بندی و اقدامات مربوط به آن و همچنین داده‌ساختارهای مرتبط، در فایل‌های proc.c و proc.h قابل مشاهده هستند.

۱) ساختار PCB و همچنین وضعیت‌های تعریف شده برای هر پردازه را در xv6 پیدا کرده و گزارش کنید. آیا شباهتی میان داده‌های موجود در این ساختار و ساختار به تصویر کشیده شده در شکل 3.3 منبع درس وجود دارد؟ (ذکر حداقل ۵ مورد و معادل آن‌ها در xv6)

۲) هر کدام از وضعیت‌های تعریف شده معادل کدام وضعیت در شکل ۱ می‌باشند؟

همان‌طور که در شکل ۱ نیز مشاهده می‌کنید، هر پردازه در هر وضعیتی که باشد، توسط رویدادهای مختلف به وضعیت‌های دیگر گذار[21] می‌کند.

تغییر وضعیت در xv6

در این قسمت با گذارهای نمایش داده شده در شکل ۱ و معادل آن‌ها در xv6 آشنا می‌شوید.

Admitted

پس از اجرای سیستم عامل، اولین پردازه (initcode) توسط تابع userinit ساخته خواهد شد (main:36). در این تابع، مقداردهی‌های اولیه انجام می‌شود تا پردازهٔ مربوط به برنامهٔ سطحِ کاربرِ init.c ایجاد شود. در این برنامه، پردازهٔ init، عملیات fork را انجام می‌دهد تا پردازهٔ جدیدی تحت رابطهٔ فرزندی با آن ایجاد شود (init.c:24). این پردازهٔ جدید در واقع بستر اجرای بقیهٔ برنامه‌های سطح کاربر در xv6 یا همان shell می‌باشد که با آن کم و بیش برخورد داشته‌اید. در واقع تمامی برنامه‌های سطح کاربر fork و exec ای از پردازهٔ sh یا همان shell می‌باشند.

۳) با توجه به توضیحات گفته شده، کدام یک از توابع موجود در proc.c منجر به انجام گذار از حالت new به حالت ready که در شکل ۱ به تصویر کشیده شده، خواهد شد؟ وضعیت یک پردازه در xv6 در این گذار از چه حالتی/حالت‌هایی به چه حالت/حالت‌هایی تغییر می‌کند؟ پاسخ خود را با پاسخ سوال ۲ مقایسه کنید.

[20] Process Control Block

[21] Transition

Scheduler Dispatch

با قرار گرفتن یک پردازه در حالت ready و پس از گرفتن نوبت توسط زمان‌بند، پردازه اجرا خواهد شد. تابع scheduler وظیفهٔ این نوبت‌دهی را بر عهده دارد (proc.c:323).

همانطور که پیش‌تر اشاره شد، هر پردازنده در xv6، یک اجرای مستقل از زمان‌بند خود را دارد. اما نکتهٔ حائز اهمیت این است که در xv6 تمامی اطلاعات پردازه‌های موجود، در یک جدول مشترک قرار دارند (proc.c:10). به منظور جلوگیری از رقابت[22] میانِ دو پردازنده در این جدول و ایجاد تغییرات همزمان؛ صف پردازه‌ها با استفاده از یک قفل محافظت شده است. قفل و مکانیزم قفل‌گذاری در ادامهٔ درس و همچنین آزمایشگاه بعد بررسی خواهد شد.

۴) سقف تعداد پردازه‌های ممکن در xv6 چه عددی است؟ در صورتی که یک پردازه تعداد زیادی پردازهٔ فرزند ایجاد کند و از این سقف عبور کند، کرنل چه واکنشی نشان داده و برنامهٔ سطح کاربر چه بازخوردی دریافت می‌کند؟

تابع scheduler در یک حلقه بی‌نهایت اجرا می‌شود و الگوریتم زمان‌بندی را اجرا می‌کند. ابتدا وقفه‌ها را فعال می‌کند و سپس با قفل کردن ptable، پردازهٔ واجد شرایط اجرا شدن (RUNNABLE) را طبق الگوریتم پیدا می‌کند.

پس از انتخاب پردازه توسط زمان‌بند، به پردازنده‌ای که زمان‌بندِ آن فعال شده است واگذار و سپس عملیات تعویض متن انجام می‌شود و وضعیت پردازه به حالت RUNNING تغییر پیدا می‌کند. با پایان یافتن اجرای پردازه، سیستم به فضای کرنل برمی‌گردد و از ادامهٔ جایی که ابتدا تعویض متن رخ داده بود، اجرای زمان‌بندِ آن پردازنده، ادامه پیدا می‌کند (proc.c:346,347). پس، تابع scheduler یک بار تا انتهای صف می‌رود و هر پردازه را به شیوه‌ای که گفته شد اجرا می‌کند. بعد از بررسی آخرین پردازه در صف، قفل جدول پردازه‌ها را آزاد کرده و سپس حلقهٔ بی‌نهایت تکرار خواهد شد و این فرایند از ابتدای صف مجددا انجام می‌شود.

۵) چرا نیاز است در ابتدای هر حلقه تابع scheduler، جدول پردازه‌ها قفل شود؟ آیا در سیستم‌های تنها یک پردازنده (یک هستهٔ پردازشی) دارند هم نیاز است این کار صورت بگیرد؟

[22] Race/Contention

۶) با فرض اینکه xv6 در حالت تک پردازنده‌ای (تک هسته‌ای) در حال اجراست، اگر یک پردازه به حالت RUNNABLE برود و صف پردازه‌ها در حال طی شدن باشد (proc:335)، در مکانیزم زمان‌بندی xv6 نسبت به موقعیت پردازه در صف، در چه iteration ای امکان schedule پیدا می‌کند؟ (در همان iteration یا در iteration بعدی)

Context Switch

تعویض متن به فرایندی گفته می‌شود که در آن سیستم عامل یک پردازه در حال اجرا را متوقف کرده و یک پردازه جدید را اجرا می‌کند. همچنین در صورت وقوع یک وقفه[23] مثل وقفه تایمر یا یک وقفه از سمت سخت‌افزار، سیستم عامل نیاز دارد تا پردازه جاری را متوقف کرده و به وقفه رسیدگی کند.

در xv6، تعویض متن با استفاده از تابع swtch انجام می‌گیرد که وظیفهٔ ذخیره و بازیابی وضعیت را دارد:

*void swtch(struct context *old, struct context *new);*

این تابع وضعیت پردازه جاری را در old ذخیره می‌کند و وضعیت پردازه جدید را از new بارگذاری می‌کند. ساختار context وظیفهٔ نگهداری آخرین وضعیت پردازه را دارد.

۷) رجیسترهای موجود در ساختار context را نام ببرید.

۸) همانطور که می‌دانید یکی از مهم‌ترین رجیسترها قبل از هر تعویض متن Program Counter است که نشان می‌دهد روند اجرای برنامه تا کجا پیش رفته است. با ذخیره‌سازی این رجیستر می‌توان محل ادامهٔ برنامه را بازیابی کرد. این رجیستر در ساختار context چه نام دارد؟ این رجیستر چگونه قبل از انجام تعویض متن ذخیره می‌شود؟

Interrupt

تابع trap (trap.c:37) در سیستم‌عامل xv6 برای مدیریت وقفه‌ها به‌کار می‌رود. در صورتی که پردازه در حالت RUNNING باشد و وقفه تایمر (یعنی T_IRQ0+IRQ_TIMER) ایجاد شود، منجر به دو رویداد

[23] interrupt

می‌شود. ابتدا زمان سیستم به اندازهٔ یک واحد افزایش پیدا می‌کند. که یکی از پردازنده‌ها مسئولیت این موضوع را بر عهده دارد (trap:50). رویداد دوم در صورتی که پردازه‌ای در حالت اجرا باشد، تابع yield برای آن فراخوانی شده و آن پردازه، پردازنده را رها می‌کند (trap:106).

۹) همانطور که در قسمت قبل مشاهده کردید، ابتدای تابع scheduler، ایجاد وقفه به کمک تابع sti، فعال می‌شود. با توجه به توضیحات این قسمت، اگر وقفه‌ها فعال نمی‌شد چه مشکلی به‌وجود می‌آمد؟

۱۰) به نظر شما وقفهٔ تایمر هر چه مدت یک بار صادر می‌شود؟ (راهنمایی: می‌توانید با اضافه کردن یک cprintf پس از *ticks++* و یا با مراجعه به تابع lapicinit واقع در lapic.c این موضوع را مشاهده کنید.)

تابع yield (proc.c:386) وظیفه تغییر حالت پردازه از RUNNING به RUNNABLE را دارد؛ یعنی پردازه از اجرا خارج شده و پردازنده را رها می‌کند تا زمان‌بند در خصوص اجرای پردازه‌ها مجددا تصمیم بگیرد. فرایند آن به این شکل است:
ابتدا با قفل کردن جدول پردازه‌ها[24]، پردازه فعلی در جدول به حالت RUNNABLE تغییر وضعیت می‌دهد. سپس تابع sched فراخوانی می‌شود تا زمان‌بند، امکان انتخاب پردازه بعدی را داشته باشد. در پایان، قفل جدول پردازه‌ها آزاد می‌شود و کنترل به زمان‌بند برمی‌گردد. تابع sched (proc.c:366) در نهایت، یک تعویض متن از پردازهٔ کنونی، به تابع scheduler مربوط به همین پردازنده انجام می‌دهد.

۱۱) با توجه به توضیحات داده شده، چه تابعی منجر به انجام شدنِ گذار interrupt در شکل ۱ خواهد شد؟

۱۲) با توجه به توضیحات قسمت scheduler dispatch می‌دانیم زمان‌بندی در xv6 به شکل نوبت گردشی است. حال با توجه مشاهدات خود در این قسمت، استدلال کنید، کوانتوم زمانی این پیاده‌سازی از زمان‌بندیِ نوبت گردشی چند میلی ثانیه است؟

[24] Process Table (ptable)

Wait

در صورتی که پردازه نیازمندِ به‌وقوع پیوستنِ اتفاقی برای ادامه انجام پردازش خود باشد، مانند عملیات I/O یا منتظر ماندن برای یک رویداد، برای آن صبر می‌کند. در این حالت، پس از به‌وقوع پیوستن اتفاق مورد نظر، سیستم عامل وظیفه دارد تا پردازه را از این اتفاق آگاه سازد.

یکی از این رویدادها، منتظر ماندن برای اتمام پردازه فرزند از طرف والد می‌باشد که به کمک تابع wait انجام می‌شود. این تابع به والد این امکان را می‌دهد تا منتظرِ خاتمهٔ کارِ یکی از فرزندان خود باشد و زمانی که یکی از آن‌ها به پایان رسید، والد را از اتمام آن آگاه و از سیستم پاک‌سازی کند.

۱۳) تابع wait در نهایت از چه تابعی برای منتظر ماندن برای اتمام کار یک پردازه استفاده می‌کند؟

۱۴) با توجه به پاسخ سوال قبل، استفاده(های) دیگر این تابع چیست؟ (ذکر یک نمونه)

۱۵) با این تفاسیر، چه تابعی در سطح کرنل، منجر به آگاه سازی پردازه از رویدادی است که برای آن منتظر بوده است؟

۱۶) با توجه به پاسخ سوال ۱۵، این تابع منجر به گذار از چه وضعیتی به چه وضعیتی در شکل ۱ خواهد شد؟

۱۷) آیا تابع دیگری وجود دارد که منجر به انجام این گذار شود؟ نام ببرید.

Exit

در انواع حالت‌های پردازه‌ها، حالت ZOMBIE نشان‌دهندهٔ پردازه‌ای است که اجرای آن به پایان رسیده، اما هنوز منابع آن به‌طور کامل آزاد نشده و منتظر است تا والد از طریق wait آن را جمع‌آوری کند. با فراخوانی تابع exit در xv6، وضعیت پردازه به حالت ZOMBIE می‌رود و والدِ پردازه نیز به حالت RUNNABLE تغییر می‌کند تا در فرصت بعدی که اجرا شد، فرزندِ خاتمه‌یافتهٔ خود را جمع‌آوری کند.

۱۸) در بخش ۳.۳.۲ منبع درس با پردازه‌های Orphan آشنا شدید، رویکرد xv6 در رابطه با این گونه پردازه‌ها چیست؟

زمان‌بندی چند کلاسی:

همان‌طور که پیش‌تر اشاره شد، امروزه نیازمند استفاده از مکانیزم‌های زمان‌بندی پیشرفته‌تر هستیم. یکی از این مکانیزم‌های پرکاربرد، زمان‌بندی چند کلاسی می‌باشد. در این قسمت قصد داریم پیاده‌سازی از این الگوریتم در xv6 داشته باشیم.

سیستم از دو کلاس زمان‌بندی با اولویت‌های مختلف تشکیل شده است:

- کلاس اول - پردازه‌های بی‌درنگ (بالاترین اولویت): در کلاس اول، از الگوریتم EDF برای زمان‌بندی پردازه‌های بی‌درنگ استفاده خواهیم کرد.
- کلاس دوم - پردازه‌های عادی (اولویت عادی): در کلاس دوم، از یک زمان‌بندی بازخوردی چند سطحی استفاده خواهیم کرد که پیش‌تر با این مفهوم در درس آشنا شده‌اید:
  - سطح اول (اولویت بالاتر): پردازه‌های تعاملی با الگوریتم نوبت گردشی (RR) و کوانتوم ۳۰ میلی‌ثانیه.
  - سطح دوم (پیشفرض): پردازه‌های عادی با الگوریتم FCFS.

در این مکانیزم اولویت‌ها به صورت ایستا[25] ارزیابی خواهند شد و پس از هر بار وقفه تایمر، زمان‌بند، پردازه‌ای که بالاترین اولویت را داشته باشد، انتخاب خواهد کرد. به تبع، ترتیب بررسی آن، ابتدا از کلاس اول شروع شده و سپس به کلاس دوم و سطح‌های آن خواهد رسید.

در این فرایند، بجز پردازه‌های init و sh که می‌بایست در سطح اول از کلاس دوم قرار بگیرند، پردازه‌هایی که به صورت عادی ایجاد خواهند شد (استفاده از fork) در سطح دوم از کلاس دوم قرار خواهند گرفت.

همچنین پردازه‌هایی که با استفاده از فراخوانیِ سیستمیِ ایجادِ پردازه‌های بی‌درنگ ایجاد می‌شوند در کلاس اول قرار خواهند گرفت.

[25] Fixed Priority

راهنمایی پیاده‌سازی: در بعضی مواقع ممکن است پردازهٔ قابل اجرایی در سیستم حضور نداشته باشد. در این شرایط ممکن است (با توجه به پیاده‌سازی) در صورت عدم مقدار دهیِ struct proc* p واقع در تابع scheduler در هر iteration از حلقه بی‌نهایت ((;;)for)، مقدار این متغیر، مقدار قدیمی خود را نگه دارد. این مسئله منجر به این می‌شود که پردازه‌ای که شرایط اجرا ندارد برای اجرا صادر شود که می‌تواند مشکلات زیادی را به وجود بیاورد. پیشنهاد می‌شود در ابتدای حلقه، مقدار p را به مقداری پیشفرض مانند NULL مقداردهی کنید و در انتهای منطق scheduler خود اگر پردازه این شرایط را نداشت (NULL نبود) برای اجرا صادر شود.

۱۹) فرض کنید در این مکانیزم زمان‌بندی، تمامی پردازنده‌ها مشغول اجرای پردازه‌های کلاس اول باشند. در این صورت برای shell xv6 چه اتفاقی می‌افتد؟ دیدگاه خود را نیز نسبت به این موضوع شرح دهید.

کلاس دوم - سطح اول: زمان‌بند نوبت گردشی با کوانتوم زمانی

برای شروع، ابتدا بدون تغییر فرکانس وقفهٔ تایمر، زمان‌بندی xv6 را به حالتی تغییر دهید که کوانتوم زمانی نوبت گردشی آن ۳۰ میلی ثانیه (بر حسب تیک زمانی سیستم در نظر بگیرید) باشد. برای اطمینان از صحت پیاده‌سازی خود، xv6 را در حالتی اجرا کنید که تعداد پردازنده‌های مورد استفاده 1 عدد باشد. برای این کار می‌توانید در هنگام اجرای دستور make پرچم CPUS را برابر با 1 قرار دهید و یا در Makefile این متغیر را تغییر دهید.

توجه: در صورتی که نسخه QEMU شما از 6.2[26] بالاتر باشد، می‌بایست تغییراتی را در Makefile ایجاد کنید تا تنظیمات تعداد پردازنده‌ها به درستی اعمال شود. برای این کار تغییرات انجام شده در این لینک را اعمال کنید.

در قدم بعدی در فایل trap.c از cprintf استفاده کنید تا pid پردازه در هر تیک از اجرا و یا pid پردازه همراه با مدت زمان اجرای آن بر حسب تیک زمانی سیستم چاپ کنید. در نهایت برنامهٔ سطح کاربری بنویسید که درون خود چند پردازه ایجاد کند و هر پردازه مشغول انجام عملیاتی شود که به اندازه کافی طول می‌کشد. تصویری از خروجی اجرای سیستم عامل در این حالت را در گزارش خود قرار دهید.

[26] برای یافتن نسخه QEMU می‌توانید از دستور qemu-system-x86_64 --version استفاده کنید.

راهنمایی: به دلیل وجود صف‌های دیگر در زمان‌بندی و اجرا از آنان، احتمالا نیاز داشته باشید تا اندیس آخرین پردازهٔ اجرا شده برای صف نوبت‌گردشی را نگه دارید تا در دفعهٔ بعد، از ادامهٔ صف، زمان‌بندی را از پی بگیرید. (به دلیل نیاز به تغییر شرایطی که در قسمت Scheduler Dispatch توضیح داده شد.)

۲۰) مقدار CPUS را مجددا به عدد ۲ برگردانید. آیا همچنان ترتیبی که قبلا مشاهده می‌کردید پا برجاست؟ علت این امر چیست؟

برای اطمینان از پیاده‌سازیِ درستِ دیگر الگوریتم‌های زمان‌بندی نیز می‌توانید از همین روش استفاده کنید و سپس تمامی اجزای زمان‌بند را کنار هم قرار داده و صحت‌سنجی نهایی را انجام دهید.

کلاس دوم - سطح دوم: اولین ورود-اولین رسیدگی[27]

با این الگوریتم در درس آشنا شده‌اید. در این زمان‌بندی، زودترین پردازه‌ای که وارد صف شده باشد، اول انتخاب می‌شود. در این الگوریتم زمان‌بندی، تا اتمام انجام شدن پردازه‌ای که نسبت به دیگر پردازه‌ها زودتر به صف وارد شده، پردازه‌های بعد از آن در صف اجرا نخواهند شد.

نکته: تنها ملاک ترتیب اجرای پردازه‌ها در این الگوریتم زمان‌بندی، زمان ورود آن‌ها به صف می‌باشد.

کلاس اول: اول(،) زودترین موعد[28]

در درس با زمان‌بند EDF آشنا شدید. یکی از نکات مثبت این الگوریتم، بهینه بودن در برابرِ مدت زمانِ تاخیر پردازه‌ها نسبت به موعدشان می‌باشد. در این قسمت قصد داریم تا این الگوریتم را در شرایط Soft Real-Time پیاده‌سازی کنیم.

جدول ۲. نمونه‌ای از چند پردازه، زمان‌های شروع مطلق و موعدهای نسبی‌شان.

Process	Start Time (Absolute)	Deadline (Relative)
p1	10	12
p2	8	10

[27] First Come First Served (FCFS)

[28] Earliest Deadline First

p3	12	9

به منظور بررسی این زمان‌بند، از مثالی روی پردازه‌های تعریف شده در جدول ۲ بهره خواهیم برد (مقادیر جدول بر اساس تیک هستند). در این جدول، ۳ پردازه داریم که هر کدام به ترتیب در زمان‌های ۸، ۱۰ و ۱۲ در سیستم ایجاد می‌شوند. پردازهٔ p1 می‌بایست حداکثر تا زمان ۲۲، پردازهٔ p2 تا زمان ۱۸ و پردازهٔ p3 تا زمان ۲۱ به کار خود را انجام داده تا به موعد تعیین شدهٔ خود برسند. به این ترتیب، به منظور تبعیت از الگوریتم EDF، به ترتیب پردازه‌های p2 سپس p3 و در نهایت p1 را اجرا خواهیم کرد.

همچنین به عنوان فرض پیاده‌سازی در خصوص پردازه‌هایی که به موعد خود نمی‌رسند (آن را از دست می‌دهند)، آن‌ها را همچنان اجرا خواهیم کرد. از طرفی به منظور کمینه کردن مجموع تاخیرها، اولویت اجرا در این صف را به پردازه‌ای می‌دهیم که دارای کمترین میزانِ حاصل تفریقِ زمانِ فعلی با موعدِ پردازه باشد. به عنوان مثال، اگر در مثالی که بررسی کردیم یک پردازهٔ p4 وجود می‌داشت که موعدِ آن در زمان مطلق ۷ به اتمام می‌رسید، اما تا زمان ۸ کار آن به اتمام نرسیده بود، در زمان 8 اولویت اجرا با این پردازه می‌بود.
به منظور انجام این پیاده‌سازی، می‌توانید از فرمول زیر بهره ببرید:

$$argmin_i(CurrentTime - Deadline_i)$$

توجه: پردازه‌هایی که به صورت عادی ایجاد می‌شوند دارای پارامتر ددلاین نیستند. برای مقدار پیشفرض برای این نوع پردازه‌ها می‌توانید از عدد صفر و یا عددی بسیار بالا استفاده کنید.

۲۱) در پیاده‌سازی زمان‌بند EDF به دلیل ماهیت Soft Real-Time بودن اینگونه تسک‌ها از فرض پیاده‌سازی مطرح شده استفاده کردیم. در صورتی که اینگونه تسک‌ها Hard Real-Time بودند، بنظر شما استفاده از چه فرض پیاده‌سازی مناسب‌تر می‌بود؟

سازوکار افزایش سن در کلاس دوم[29]

همانطور که در کلاس درس بیان شد، یکی از روش‌های جلوگیری از قحطی‌زدگی[30]، استفاده از سازوکار افزایش سن است. بدین صورت که اولویت پردازه‌هایی که مدت زیادی صبر کرده‌اند و پردازنده به آنها اختصاص نیافته است، به مرور افزایش می‌یابد. در زمان‌بندی که پیاده‌سازی می‌کنید، بجز اولین پردازه و shell که می‌بایست در سطح اول قرار گیرند، دیگر پردازه‌ها را به طور پیش‌فرض در **سطح دوم** قرار دهید. سازوکار افزایش سن برای پردازه‌ها را به صورت زیر قرار دهید:

مدت زمان انتظار پردازه فقط در حالت RUNNABLE محاسبه می‌شود و پس از ۸۰۰ سیکل انتظار در سطح دوم، پردازه به سطح اول منتقل می‌شود.

توجه: پس از انتقال پردازه به سطح جدید، با cprintf، شماره پردازه به همراه سطح مبدا و مقصد را گزارش کنید (مثال: PID 3: Queue 2 to 1).

۲۲) به چه علت مدت زمانی که پردازه در وضعیت SLEEPING می‌باشد به عنوان زمان انتظار پردازه از منظر زمان‌بندی در نظر گرفته نمی‌شود؟

نکته: توجه کنید که پردازهٔ پوسته (shell) و همچنین پردازه‌هایی که یک fork از این پردازه می‌باشند، می‌بایست در سطح اول قرار گیرند تا پوسته نسبت به دیگر پردازه‌های عادی قفل نشود، همچنین برای درستی برنامه‌های خود می‌بایست پردازه‌هایی که از سمت پوسته exec می‌شوند را به سطح پیش‌فرض انتقال دهید.

دلیل این کار به خاطر این است که پوسته برای اجرای برنامه‌ها ابتدا یک پردازهٔ فرزند ایجاد می‌کند و روی آن wait را صدا می‌زند. این پردازهٔ فرزند بسته به نیاز (مانند اجرای پس‌زمینه) ممکن است فرزندهای دیگری ایجاد کند که روی آن‌ها در ادامه exec انجام دهد. در نهایت، این پردازهٔ فرزند exit را صدا می‌زند و shell از حالت wait خارج می‌شود. در صورتی که این پردازهٔ فرزند در اولویت اجرا (سطح اول) نمی‌بود و به قسمت exit نمی‌رسید، shell در حالت wait می‌ماند و در نتیجه پوسته قفل می‌شد.

در خصوص برنامه‌هایی که در نهایت به وسیله exec ای از fork پوسته (shell) اجرا می‌شوند، اولویتی که پیش‌تر آن را بالا بردیم را به ارث می‌برند. این پردازه‌ها در مقابل پردازه‌هایی که خودشان فرزندانی

[29] Aging
[30] Starvation

ایجاد می‌کنند، نسبت به این فرزندان اولویت پیدا می‌کنند و این رفتار مطلوبی نخواهد بود. برای حل این موضوع می‌بایست در هنگام exec کردن بررسی کنیم اگر پردازه‌ای که در حال انجام عملیات exec می‌باشد پردازهٔ اولیه (initproc) یا پردازهٔ پوسته (sh) نبود، سطح زمان‌بندی این پردازه را به سطح پیش‌فرض انتقال دهیم (برای این کار می‌توانید از فراخوانی سیستمی که برای تغییر سطح نوشتید استفاده کنید).

فراخوانی‌های سیستمی مورد نیاز:

فراخوانی سیستمی	توضیح
ساخت پردازه با موعدِ از پیش تعیین شده[31]	این فراخوانی سیستمی شامل یک ورودی از نوع نوع int خواهد بود که موعد پردازه‌ای که ساخته می‌شود را مشخص خواهد کرد. این فراخوانی سیستمی تنها راه قرار گرفتن یک پردازه در کلاس اول است.
تغییر سطح[32]	این فراخوانی سیستمی شامل دو ورودی از نوع int است که ورودی اول pid پردازه مورد نظر و ورودی دوم نشان دهندهٔ صف مقصد می‌باشد.
چاپ اطلاعات	با صدا زدن این فراخوانی سیستمی تمامی اطلاعات مربوط به پردازه از قبیل نام، شماره پردازه، وضعیت، کلاس زمان‌بندی، الگوریتم زمان‌بندی، مدت زمان انتظار،

[31] در صورتی که موعد تعیین شده عددی غیرمعقول از ساخت این پردازه جلوگیری شود.

[32] توجه داشته باشید که اگر صف مقصد نامعتبر بود، صف مبدا و مقصد یکسان و یا در کلاس‌های متفاوت بودند، از این اقدام جلوگیری شود. همچنین با اجرای این فراخوانی، مدت زمان منتظر ماندن پردازه نباید تغییر کند.

	موعدِ پردازه، تعداد تیک‌هایی که پردازه پشت سر هم اجرا شده است و زمان ورود به صف کنونی[33]، چاپ می‌شود.

```
name            pid  state     class     algorithm   wait time  deadline  consecutive run  arrival
.....................................................................................................
init            1    SLEEPING  normal    mlfq(RR)    0          0         0                0
sh              2    SLEEPING  normal    mlfq(RR)    0          0         0                0
scheduletest    3    RUNNABLE  normal    mlfq(FCFS)  0          0         0                211
scheduletest    4    RUNNABLE  normal    mlfq(FCFS)  2          0         0                212
scheduletest    5    RUNNABLE  normal    mlfq(FCFS)  2          0         0                212
scheduletest    6    RUNNABLE  normal    mlfq(FCFS)  2          0         0                212
scheduletest    7    RUNNABLE  normal    mlfq(FCFS)  2          0         0                212
scheduletest    8    RUNNING   real-time EDF         0          300       0                213
```

شکل ۲. اجرای فراخوانی سیستمی چاپ اطلاعات.

بهینه‌سازی در یافتن پردازه‌ها

با توجه به اینکه پردازه‌ها در کلاس‌ها و صف‌های مختلف قرار دارند می‌بایست برای هر صف مربوطه، پردازه‌های مربوط به آن صف را شناسایی و اجرا کرد. یکی از راه‌های ابتدایی برای این کار گشتن در تمامی پردازه‌های ptable به دنبال پردازه‌ای است که در صف مربوطه باشد. اما یکی از مشکلاتی که این روش دارد، زمانی است که پردازهٔ قابلِ اجرایی در آن صف قرار ندارد ولی همچنان این عمل انجام خواهد شد که منجر به هدر رفت زمان CPU خواهد شد. در این قسمت به منظور حل این موضوع تعداد پردازه‌هایی که در هر صف و کلاس به شکل RUNNABLE قرار دارند را نگه خواهیم داشت و این تعدادها را هر بار به‌روز خواهیم کرد. سپس از این مقادیر برای تصمیم گیری در خصوص اینکه در صف و یا کلاس مربوطه دنبال پردازه‌ای بگردیم یا خیر استفاده خواهیم کرد.

۲۳) با توجه به دانشی که از پاسخ به سوالات قبل به دست آورده‌اید و پیاده‌سازی که تا به اینجا انجام داده‌اید، در چه توابعی می‌بایست از مقادیر مربوط به تعداد پردازه‌های آماده هر صف کم و یا به آن اضافه کنیم؟ (می‌توانید از شکل ۱ برای مرور فرایند طی شده نیز کمک بگیرید) (نکته: به مکانیزم‌ها و فراخوانی‌های سیستمی که پیاده‌سازی کرده‌اید نیز توجه داشته باشید)

[33] در صورت تغییر یافتن صف پردازه، زمان ورود به صف باید آپدیت شود. همچنین در هنگام ایجاد پردازه، مقدار آن برابر ticks سیستم است.

برنامه‌های سطح کاربر

در این قسمت می‌بایست برنامه‌های سطح کاربری بنویسید تا صحت پیاده‌سازی شما را به تصویر بکشند. توجه کنید حتی الامکان از sleep استفاده نکنید و از حلقه‌های تو در تو و محاسبات متنوع استفاده کنید تا برنامهٔ خود را به میزان چند ثانیه در حالت قابل اجرا و یا در حال اجرا نگه دارید.

در برنامه سطح کاربر شما می‌بایست پردازه‌های مختلف در کلاس‌های مختلف ایجاد شوند. با استفاده از فراخوانی‌های سیستمی چاپ اطلاعات و تغییر سطح میان دو سطح مختلف، سناریوهای مختلفی رقم بخورند و اطلاعات مورد نیاز در هر مقطع نمایش داده شود.

سایر نکات

- پروژه خود را در Github یا Gitlab پیش برده و در نهایت یک نفر از اعضای گروه کدها را به همراه پوشهٔ .git[34] زیپ کرده و در سامانه با فرمت OS-Lab3-<SID1>-<SID2>-<SID3>.zip آپلود نمایید.
- رعایت نکردن مورد فوق کسر نمره را به همراه خواهد داشت.
- بخش خوبی از نمرهٔ شما را پاسخ دهی به سوالات مطرح شده تشکیل می‌دهد که به شما در درک نحوهٔ کارکرد xv6 و پیاده‌سازی قسمت زمان‌بندی کمک می‌کند.
- پاسخ سوال‌ها را مختصر و مفید بنویسید.
- در پیاده‌سازی خود در خصوص مواردی که ذکر نشده‌اند می‌توانید فرض منطقی و دلخواه خود را داشته باشید.
- زمان‌بندی شما می‌بایست هم در حالت تک‌پردازنده‌ای (CPUS=1) و هم در حالت چند پردازنده‌ای (CPUS >= 2) معتبر بوده و به درستی کار کند.

[34] برای اطمینان حاصل کردن از وجود پوشه .git. می‌توانید گزینهٔ نمایش فایل‌های مخفی (Show Hidden Files) را فعال و یا از دستور ls -a استفاده کنید.

- تابع printf در سطح برنامه‌های کاربر و تابع cprintf در سطح کرنل و همچنین ابزار gdb، امکانات خوبی در هنگام عیب‌یابی می‌باشند، در صورت نیاز آن‌ها را به کار بگیرید.
- همه افراد می‌بایست به پروژه مسلط باشند و نمره تمامی اعضای گروه لزوما یکسان نخواهد بود.
- فصل پنجم کتاب xv6 بسیار مفید خواهد بود و مطالعهٔ آن توصیه می‌شود.
- تمامی مواردی که در جلسه توجیهی، گروه اسکایپ و فروم درس مطرح می‌شوند، جزئی از پروژه خواهند بود. در صورت وجود هرگونه سوال یا ابهام میتوانید با ایمیل دستیاران مربوطه یا گروه اسکایپی درس در ارتباط باشید.
- این تمرین صرفا برای یادگیری شما طرح شده است. در صورت محرز شدن تقلب در تمرین، مطابق با قوانین درس برخورد خواهد شد.

موفق باشید!