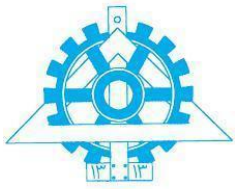


به نام خدا

آزمایشگاه سیستم عامل - پاییز ۱۴۰۳

پروژه سوم: زمان‌بندی در xv6

طراحان: بهراد علمی - محمدرضا نعمتی

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، عموما در سطح کرنل اجرا می‌شود که در xv6 نیز همین گونه است.

xv6 سیستم عاملی است که از پردازنده‌های چند هسته‌ای[5] پشتیبانی می‌کند. با این قابلیت، هر هسته، یک کپی از کدِ اجراییِ زمان‌بند (تابع زمان‌بند) و همچنین متنِ[6] زمان‌بندِ خود را دارد و پس از انتخاب

---

[1] Scheduling

[2] Multi-Level Scheduler

[3] Active

[4] Processes

[5] Multicore Processor

[6] Context

پردازه از صف پردازه‌های 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 این سیستم‌ها 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 ۲ صف است. اولویت‌های: ۰ تا ۱۵ برای تسک‌های عادی 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] ۹۶ تا ۱۲۷ ریسه‌هایی که اولویتشان بر مبنای نسبتی از سیکل‌های کلاک پردازنده که به آن‌ها تعلق می‌گیرد،

[7] Ready Queue
[8] Context Switch
[9] Round Robin
[10] Multilevel Feedback Queue
[11] Real-time
[12] Kernel Threads

		تعریف می‌شود
FreeBSD/NetBSD	MLFQ	دارای ۲۵۵ سطح اولویت و ۵ کلاس اولویت‌بندی اصلی است.
Solaris	MLFQ	دارای ۵ کلاس که بازه اولویت‌بندی برای هر کلاس متفاوت است.
2.5 < Linux < 2.6.23	O(1)	مرتبه اجرای ثابت
Linux > 2.6.23	CFS[13]	به هر پردازه، بر اساس سطح اولویت و میزان استفاده، زمان مناسبی برای پردازنده اختصاص دهد و اولویت‌های سخت و ثابت در آن کمتر دخیل هستند.
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 محدودیت دارد.

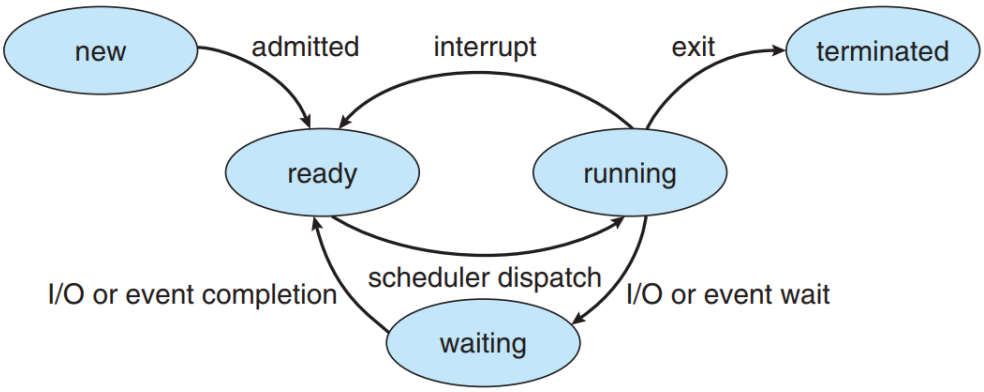
همانطور که مشاهده می‌کنید زمان‌بندی از نوع MLFQ بسیار محبوب است. در این تمرین با شکلی از این زمان‌بندی آشنا خواهید شد.

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.

[13] Completely Fair Scheduler

به طور کلی هر پردازه در چرخهٔ حیاتِ[14] 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[15] 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[16] می‌کند.

[14] Life Cycle
[15] Process Control Block
[16] Transition

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 (initproc) 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 اولین پردازه (initproc)، عملیات fork را انجام می‌دهد تا پردازهٔ جدیدی تحت رابطهٔ فرزندی با initproc 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 در آزمایشگاه اول آشنا شدید. 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.

Scheduler Dispatch

با قرار گرفتن یک پردازه در حالت ready، 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 هسته از 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[17] میانِ دو هسته 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.

[17] Race/Contention

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 هسته‌ای 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 هسته، 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 تک‌پردازه‌ای هم نیاز است این کار صورت بگیرد؟

۶) با فرض اینکه xv6 در حالت 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[18] 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 می‌شود. ابتدا زمان سیستم به اندازهٔ یک واحد افزایش پیدا می‌کند. که یکی از هسته‌های پردازنده مسئولیت این موضوع را بر عهده دارد (trap:50). رویداد دوم در صورتی که پردازه‌ای در حالت اجرا باشد، تابع yield برای آن فراخوانی می‌شود (trap:106).

[18] interrupt

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 به وجود می‌آمد؟

۱۰) بنظر شما وقفهٔ تایمر هر چه مدت یک بار صادر می‌شود؟ (راهنمایی: می‌توانید با اضافه کردن یک cprintf پس از *ticks++* 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[19]، 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 پردازه کنونی، به تابع scheduler مربوط به همین هسته از 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؟

[19] ptable

Wait

در صورتی که پردازه نیازمند به وقوع پیوستن اتفاقی برای ادامه انجام پردازش خود باشد، مانند عملیات I/O یا منتظر ماندن برای یک رویداد، به حالت waiting خواهد رفت. در این حالت، پس از به وقوع پیوستن اتفاق، 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 امکان می‌دهد که منتظر خاتمهٔ کار 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 ۹، 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 حالت پردازه، حالت ZOMBIE نشان‌دهندهٔ پردازه‌ای است که اجرای آن به پایان رسیده، اما هنوز منابع آن به‌طور کامل آزاد نشده است و منتظر است که والد از طریق wait آن را جمع‌آوری کند. با فراخوانی تابع exit در xv6، وضعیت پردازه به حالت ZOMBIE می‌رود و والد پردازه نیز به حالت RUNNABLE تغییر می‌کند تا در فرصت بعدی که اجرا شد، این اتفاق رخ دهد.

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

زمان‌بندی بازخوردی چند سطحی:

همان‌طور که پیش‌تر اشاره شد، امروزه نیازمند استفاده از الگوریتم‌های زمان‌بندی پیشرفته‌تر هستیم. یکی از این الگوریتم‌های پرکاربرد، الگوریتم زمان‌بندی بازخوردی چند سطحی[20] می‌باشد که در درس با آن آشنا شده‌اید. در این قسمت قصد داریم پیاده‌سازی از این الگوریتم در xv6 داشته باشیم.

سطح اول: زمان‌بند نوبت گردشی با کوانتوم زمانی

برای شروع ابتدا زمان‌بندی xv6 را به حالتی تغییر دهید که کوانتوم زمانی نوبت گردشی آن ۵۰ 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 هسته‌های مورد استفاده 1 عدد باشد. برای این کار می‌توانید در هنگام زدن دستور make پرچم CPUS را برابر با 1 قرار دهید یا در Makefile این متغیر را تغییر دهید. 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 را چاپ کنید . در نهایت برنامهٔ سطح کاربری‌ای 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.

نکته: 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.)

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

برای صحت درستی دیگر الگوریتم‌های زمان‌بندی نیز می‌توانید از همین روش استفاده کنید و سپس هر سه سطح را در xv6 کنار هم قرار داده و صحت‌سنجی نهایی را انجام دهید.

[20] Multilevel Feedback Queue

سطح دوم: اول(،) کوتاه‌ترین کار[21] [22]

در درس با زمان‌بند SJF آشنا شدید. یکی از مشکلات این زمان‌بند عدم امکان پیاده‌سازی به دلیل Halting Problem است. در این قسمت قصد داریم تا با استفاده از روش‌های تقریبی، یک پیاده‌سازی از زمان‌بندی داشته باشیم که از ایدهٔ SJF استفاده می‌کند. در این زمان‌بند هر تسک دارای یک عدد پیشنهادی در مورد زمان اجرا[23] و یک عدد بیان‌گر سطح اطمینان[24] از Burst Time می‌باشد. هر بار عددی بین ۰ تا ۹۹ (باقیمانده به ۱۰۰) تولید می‌شود و در صورتی که عدد تولید شده کوچکتر یا مساوی از سطح اطمینان بود، پردازه برای اجرا انتخاب شود و در غیر این صورت، پردازهٔ بعدی طبق سیاست SJF انتخاب شود. با یک مثال عملکرد این زمان‌بند را بررسی خواهیم کرد:

Process	Burst Time	Confidence
p1	10	80
p2	8	60
pe	12	90

بر اساس الگوریتم SJF، پردازه‌ای که کمترین میزان اجرا دارد انتخاب خواهد شد که در اینجا پردازهٔ ۲ می‌باشد. در صورتی که عدد تولید شده بزرگتر از 60 باشد، پردازهٔ ۱ (پردازهٔ کوچکترِ بعدی) اجرا می‌شود و در صورتی که مجددا عدد تولید شده از سطح اطمینان بیشتر شود (80) به سراغ پردازهٔ بعد خواهیم رفت. از آنجایی که پردازهٔ ۳، پردازهٔ آخر می‌باشد، مستقل از سطح اطمینان، آن را اجرا خواهیم کرد.

مقادیر پیش‌فرض برای سطح اطمینان و زمان اجرا به ترتیب **۵۰** و **۲** است.

[21] Shortest Job First (SJF)

[22] این زمان‌بند در حقیقت پیاده‌سازی دقیقی از SJF نیست.

[23] Burst Time

[24] Confidence

سطح سوم: اولین ورود-اولین رسیدگی[25]

با این الگوریتم در درس آشنا شده‌اید. در این زمان‌بندی، اولین پردازه‌ای که در صف آمده باشد، سریع‌تر انتخاب می‌شود. الگوریتم زمان‌بندی تا اتمام این پردازه به سراغ پردازهٔ بعدی نخواهد رفت.

راهنمایی: برای پیاده‌سازی این الگوریتم نیاز است زمان (tick) وارد شدن پردازه به این صف را ذخیره و با استفاده از آن، پردازه‌ای که زودتر وارد شده را پیدا کرده و اولویت انتخاب توسط زمان‌بند را به آن اختصاص دهید. توجه کنید که الگوریتم FCFS بر اساس **آخرین زمان ورود** پردازه به این صف انجام می‌شود، **نه زمان ایجاد** پردازه.

برش‌دهی زمانی[26]

در حالت کلی می‌توان ارتباطات میان صف‌ها را به دو صورت Fixed Priority یا time-slicing (در بعضی کاربردها time-sharing) تعیین کرد. در حالت Fixed Priority، اولویت صف‌ها ثابت هستند و اولویت اجرا همواره با صفی است که دارای بیشترین اولویت است و تنها در صورتِ نبودِ پردازهٔ آماده-به-اجرا در این صف، به صف اولویت پایین‌تر مراجعه خواهیم کرد. اما در روش time-slicing به هر صف قسمتی از زمان پردازنده برای اجرا را اختصاص می‌دهیم و تسک‌ها طی این زمان، با سیاست حاکم بر داخل صف اجرا خواهند شد. در این تمرین قصد داریم تا بجای استفاده از اولویت ثابت میان صف‌ها از روش time-slicing استفاده کنیم تا به دیگر صف‌ها نیز فرصت اجرا داده شود. در این قسمت قصد داریم تا این روش را با استفاده از سیاست WRR انجام دهیم. برای این کار در نظر بگیرید که برش زمانی واحد (Weight=1)، برابر با ۱۰۰ میلی ثانیه است و وزن سطح‌های اول، دوم و سوم به ترتیب برابرند با ۳، ۲ و ۱.

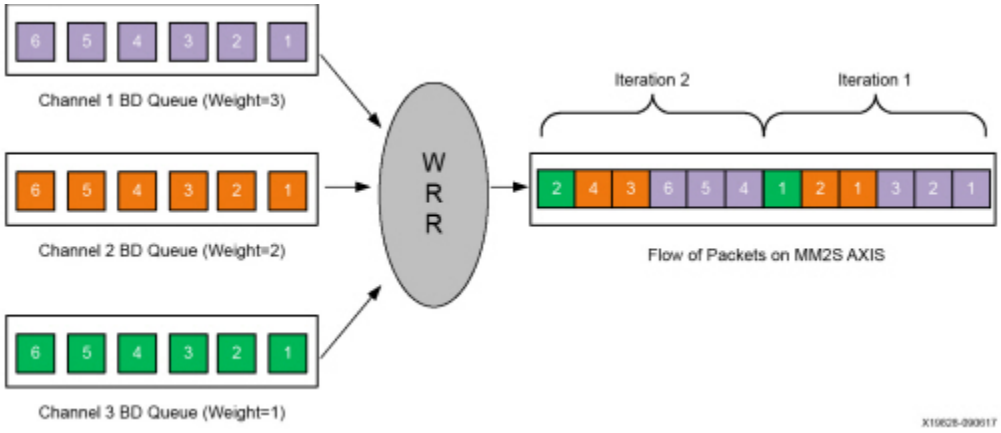
همانطور که پیش‌تر اشاره شد، هر هسته داری زمان‌بند خود می‌باشد. در نتیجه در این نحوهٔ زمان‌بندی می‌بایست هر cpu با توجه به برش زمانی خود زمان‌بندی را انجام دهد. برای این کار می‌توانید با اضافه کردن داده‌های مورد نیاز خود به ساختار cpu در فایل proc.h، این داده‌ها را در توابع scheduler و trap تغییر داده و بر اساس آن‌ها تصمیم‌گیری لازم را انجام دهید.

[25] First Come First Served (FCFS)

[26] Time Slicing

۲۰) در صورت نیاز به مقداردهی اولیه به فیلدهای اضافه شده در ساختار cpu، در چه تابعی از xv6 بهتر است این کار انجام گیرد؟ (راهنمایی: به main.c مراجعه کنید)

برای نحوهٔ کارکرد این سیاست می‌توانید از تصویر زیر کمک بگیرید:



شکل ۲. نحوهٔ عملکرد نوبت گردشی وزن‌دار

در صورتی که پردازه‌ای در یک صف وجود نداشت یا پردازه‌های موجود در صف زودتر از زمانی که سیاست WRR به آن‌ها تخصیص داده است، تمام شوند، صف بعدی اجرا خواهد شد ولی برش زمانی باقیمانده از صف قبل به صف بعدی منتقل نخواهد شد و صف بعدی طبق برش پیش‌فرض خود اجرا خواهد شد.

۲۱) با توجه به سیاست‌های پیاده‌سازی شده در سطح‌های دوم و سوم و همچنین استفاده از روش time-slicing، توجیه کنید چرا همچنان مشکل starvation امکان رخ دادن دارد؟

سازوکار افزایش سن[27]

همانطور که در کلاس درس بیان شد، یکی از روش‌های جلوگیری از قحطی‌زدگی[28]، 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، تمامی پردازه‌ها

[27] Aging
[28] Starvation

بجز اولین پردازه و shell را به طور پیش‌فرض در **صف سوم** قرار دهید و در صورتی که پردازه‌ای ۸۰۰ سیکل منتظر مانده باشد، آن را به صف دوم منتقل کنید. در صورتی که پردازه‌ای در سطح دوم مجددا ۸۰۰ سیکل منتظر ماند، آن را به صف اول منتقل کنید. دقت کنید که مدت زمانی که پردازه منتظر مانده است، به محض رسیدن نوبت اجرا به آن و یا فعال شدن سازوکار افزایش سن، صفر می‌شود و بعد از رها کردن پردازنده، (در صورت خاتمه نیافتن) مجددا به مدت زمان انتظار آن در هر سیکل از اجرای سیستم اضافه می‌شود. در نتیجه شما می‌بایست تنها در حالت‌هایی که پردازه در وضعیت RUNNABLE می‌باشد، به مدت زمان انتظار پردازه اضافه کنید.

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

در صورت انتقال پردازه به صف‌های دیگر، رصد کردن تعداد سیکل اجرا نشده را به صورت مداوم انجام دهید. دقت داشته باشید که با هر بار فعال شدن سازوکار افزایش سن، با استفاده از cprintf شماره pid پردازه به همراه صف مبدا و مقصد آن را گزارش کنید.

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

دلیل این کار به خاطر این است که shell برای اجرای برنامه‌ها ابتدا یک پردازهٔ فرزند ایجاد می‌کند و روی آن wait می‌کند. 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 (صف 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 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، صف زمان‌بندی این پردازه را به صف

پیش‌فرض انتقال دهیم (برای این کار می‌توانید از فراخوانی سیستمی که برای تغییر صف نوشتید استفاده کنید).

فراخوانی‌های سیستمی مورد نیاز:

فراخوانی سیستمی	توضیح
مقداردهی اولیهٔ زمان اجرای تخمینی و سطح اطمینان	این فراخوانی سیستمی شامل سه ورودی خواهد بود که هر سه از نوع int خواهد بود. ورودی اول pid پردازه مورد نظر، ورودی دوم زمان اجرای تخمینی و ورودی سوم سطح اطمینان است که عددی بین ۰ تا ۱۰۰ می‌باشد.
تغییر صف[29]	این فراخوانی سیستمی شامل دو ورودی از نوع int که باشد 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، وضعیت، عدد صف، مدت زمان انتظار، عدد اطمینان، زمان اجرای تخمینی، تعداد تیک‌هایی که پردازه پشت سر هم اجرا شده است و زمان ورود به صف کنونی[30]، چاپ می‌شود.

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

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

name	pid	state	queue	wait time	confidence	burst time	consecutive run	arrival
init	1	SLEEPING	0	0	50	2	2	0
sh	2	SLEEPING	0	0	50	2	0	0
scheduletest	3	RUNNING	2	0	50	2	3	3353
scheduletest	4	RUNNABLE	2	672	50	2	0	3354
scheduletest	5	RUNNABLE	2	672	50	2	0	3354
scheduletest	6	RUNNABLE	2	672	50	2	0	3354
scheduletest	7	RUNNABLE	2	671	50	2	0	3355
scheduletest	8	RUNNABLE	2	671	50	2	0	3355

شکل ۳. نمونه‌ای از خروجی فراخوانی سیستمی چاپ اطلاعات در ترمینال

برنامهٔ سطح کاربری استفاده کنید که درون خود از این فراخوانی‌های سیستمی استفاده کرده و صحت پیاده‌سازی شما را به تصویر بکشد. 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.

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. 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 تا حد ممکن کوتاه بنویسید.
- زمان‌بندی شما می‌بایست هم در حالت تک‌هسته‌ای (CPUS=1) و هم در حالت چند هسته‌ای (CPUS >= 2) معتبر بوده و به درستی کار کند.
- همه افراد می‌بایست به پروژه مسلط باشند و نمره تمامی اعضای گروه لزوما یکسان نیست.

- 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