**آزمایشگاه سیستم عامل**

پروژه شماره 3

علی پادیاو – کسری حاجی حیدری – اولدوز نیساری

بهار 1402

Repository Link: <https://github.com/alumpish/OS-Lab-Projects>

Latest Commit Hash: d1ee5c59c94aff422997b7bf955563a2fd43ea85

**مقدمه**

1-

اگر به بدنه تابع sched در فایل proc.c دقت کنیم، مشاهده می‌کنیم که درون این تابع ، تابع swtch صدا شده است. این تابع در حقیقت تابعی است که برای ما عمل context switch را انجام میدهد. برای فراخوانی swtch کانتکست فعلی پردازه به همراه scheduler برنامه فعلی به تابع داده می‌شود تا بتواند اطلاعاتش را با context در آن ذخیره کند تا بتواند بعد از بازگشت بتواند پردازه را بازیابی کند و با scheduler عمل سوییچ را انجام دهد.

پس از این کار وضعیت پردازه از RUNNABLE به RUNNING تبدیل می‌شود.

لازم به ذکر است که تابع sched درون تابع‌های exit(), yield(), sleep() در زمان‌هایی که وضعیت پردازه به RUNNABLE تغییر پیدا میکند صدا زده میشود.

در نهایت در تابع sched، دوباره context switch صورت می‌گیرد و به کانتکست پردازنده‌ای که ذخیره کرده بودیم برمی‌گردیم. در واقع پردازه هیچ گاه از تابع scheduler خارج نمی‌شود و تنها با تعویض متن از پردازنده خارج می‌شود و با فراخوانی swtch دوباره شروع به کار می‌کند.

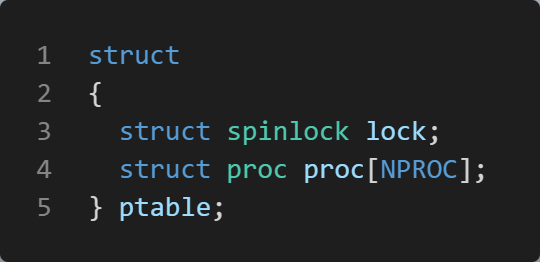
2-

در حقیقت صف اجرا در لینوکس دارای ساختار red black tree است . هر وقت وضعیت یک پردازه RUNNABLE می‌شود به این درخت اضافه می‌شود و هر پردازه‌ای هم که وضعیت آن از RUNNABLE تغییر می‌کند از درخت حذف می‌شود.

بنا به طراحی این نوع درخت، نودهایی که در سمت چپ قرار می‌گیرند زمان پردازش کمتری دارند و عملا چپ ترین نود دارای کم‌ترین زمان است. و در اصل دارای بیشترین اولویت هم هست. در حقیقت در طراحی این صف اجرا vruntime به عنوان کلید استفاده شده است.

3-

در لینوکس هر پردازنده صف جداگانه ای دارد در حالی که در xv6 یک صف مشترک (در اصل همان صف پردازه‌ها) را داریم.



اگر به ساختار استراکت ptable توجه کنیم متوجه می‌شویم که دارای یک آرایه و یک قفل است. آرایه برای ذخیره سازی پردازه‌ها و قفل برای مدیریت دسترسی‌ها است. در واقع قفل به این صورت عمل می‌کند که هنگام استفاده lock فعال می‌شود و پس از آن قفل آزاد می‌شود.

مزیت صف مشترک این است که نیازی به برقراری توازن بین صفوف مختلف نیست در حالی که وقتی چند صف داریم ، وقتی و انرژی زیادی صرف تنظیم کردن صف‌های مختلف می‌شود. در مقابل مزیت داشتن چند صف این است که هر پردازنده فقط به صف خودش دسترسی دارد و نیازی به قفل کردن در هنگام استفاده نیست. در حالی که وقتی از صف مشترک استفاده می‌کنیم در زمان استفاده یک پردازنده از صف، صف را برای بقیه پردازنده‌ها باید قفل کنیم و آن‌ها باید منتظر بمانند تا کار آن پردازنده با صف تمام شود.

4-

اگر در وضعیتی باشیم که هیچ پردازه ای در حالت آماده اجرا (RUNNABLE) نباشد و همه پردازه‌ها در حال اجرا برنامه دیگری باشند، مثلا در حال عملیات I/O باشند. در این زمان اگر وقفه برای مدتی فعال شود، پردازه‌ها زمان می‌یابند تا فرآیند I/O خاتمه یابد و به حالت RUNNABLE در بیایند و روند ادامه یابد. در نتیجه این ساز و کار برای سیستم‌های تک‌هسته‌ای هم نیاز است.

5 -

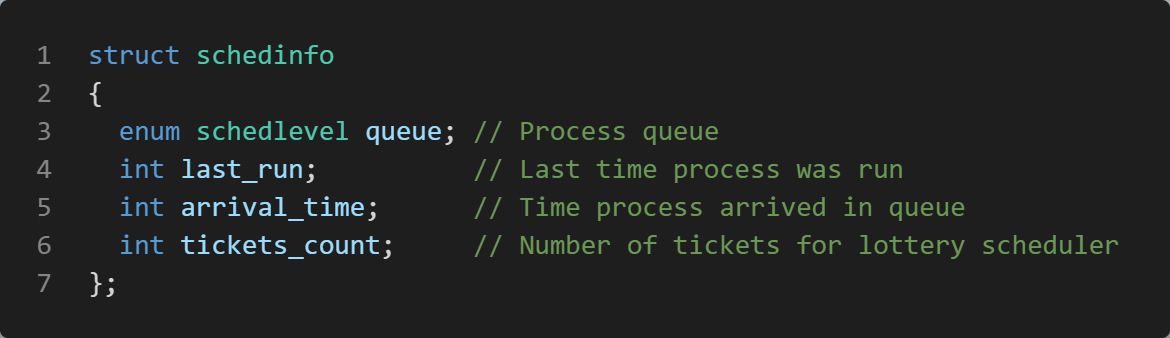
این دو سطح top-half و bottom-half نام دارند.

top-half روالی است که به وقفه‌ها پاسخ می‌دهد و bottom-half روالی است که در اصل زمان‌بندی آن بر عهده top-half است. top-half زمانی که پردازنده scheduler را دریافت می‌کند اجرا می‌شود. در این بخش وقفه و زمان بند هر دو غیرفعال هستند. در این بخش در اصل تنها بخشی که ضروری است اجرا می‌شود. bottom-half برای اجرا کارهای باقی مانده و به تعویق افتاده است. در این بخش وقفه‌ها فعال هستند ولی scheduler غیر فعال است.

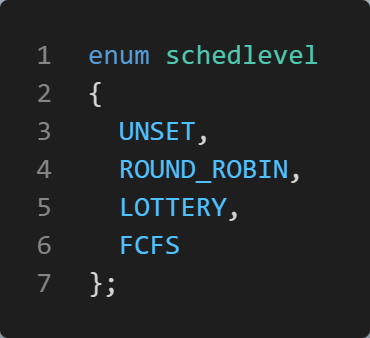
مشکل گرسنگی توسط الگوریتم aging هندل می‌شود. این کار به این صورت انجام می‌شود که با مرور زمان تسک‌های کم اولویت، اولویتشان یک پله افزایش می‌یابد و پله پله به اولین سطح اولویت سیستم می‌رسند و اجرا می‌شوند.

**زمان‌بندی بازخوردی چندسطحی**

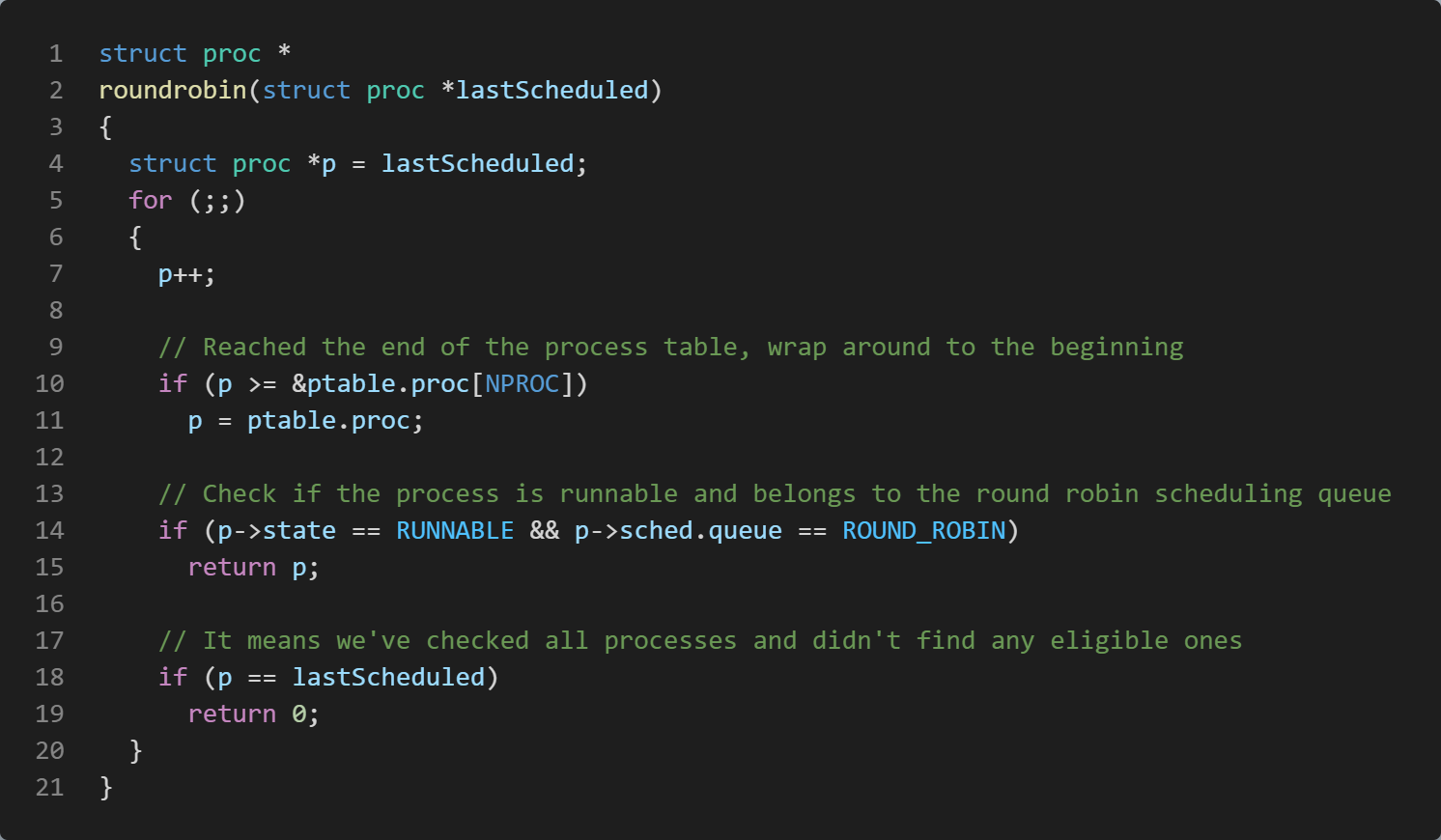
به دلیل مشکل در lock کردن، از همان یک صف xv6 استفاده می‌کنیم. منتها با labelدار کردن processها به نوعی سطح‌ها را ایجاد می‌کنیم. برای این کار، یک استراکت را به ساختار پردازه اضافه می‌کنیم.



درواقع هر پردازه می‌تواند در یکی از سطح‌های زیر باشد:

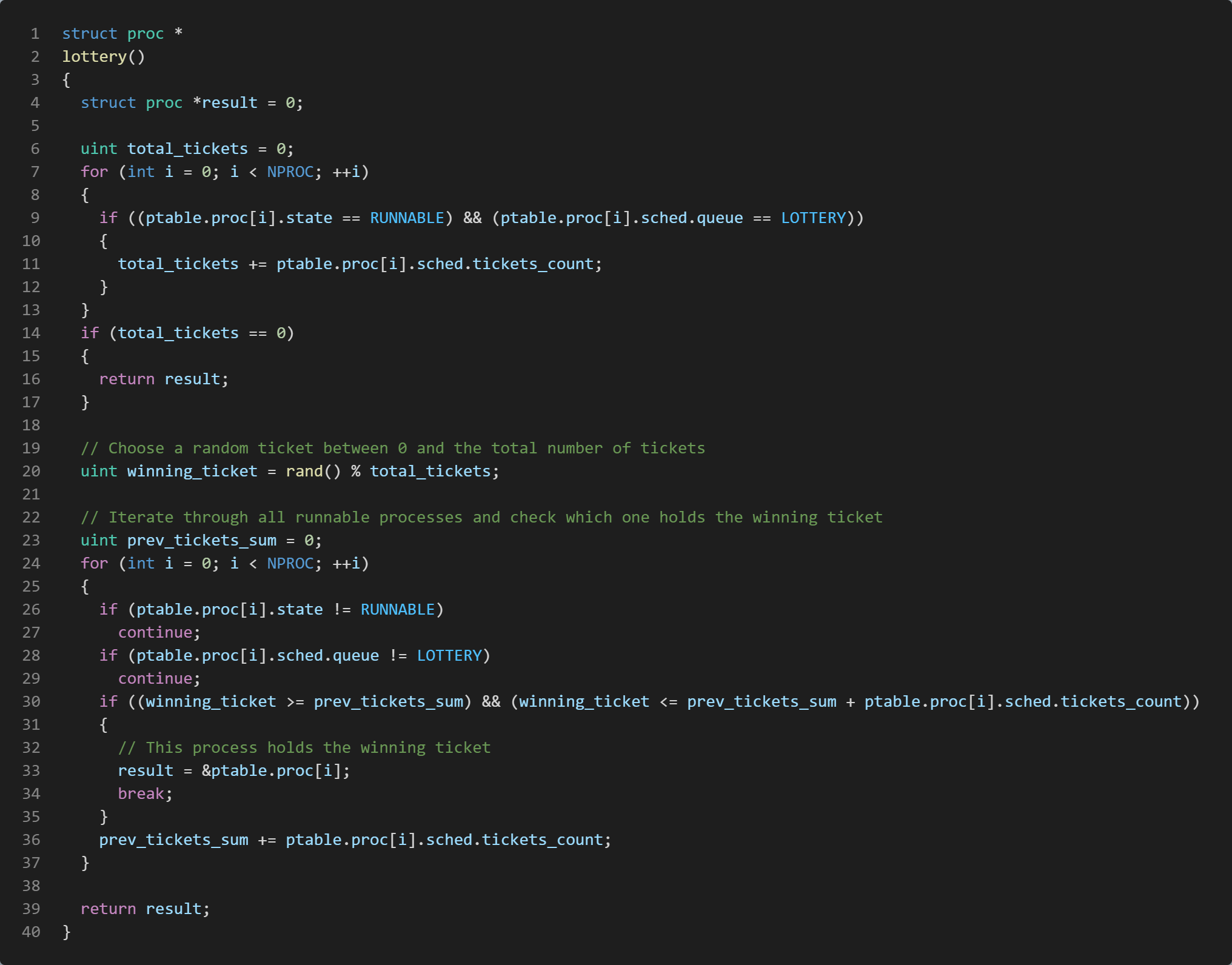


1. **زمان‌بند نوبت گردشی**



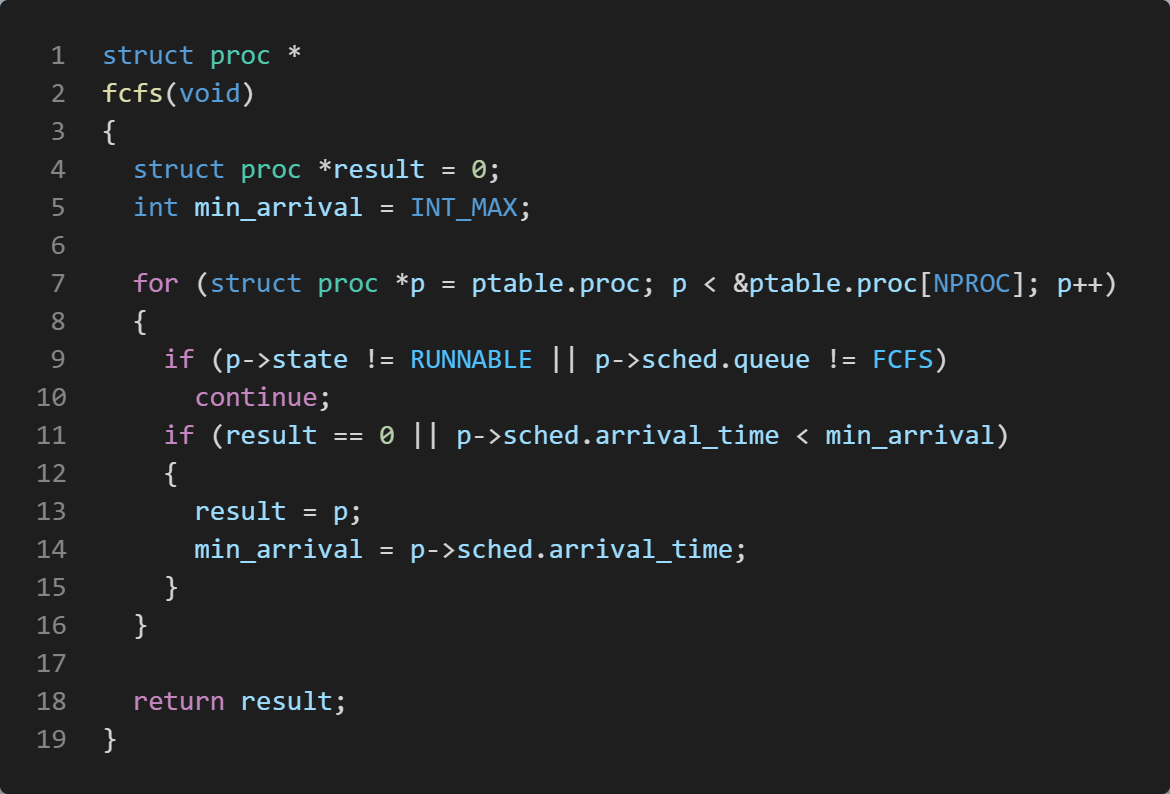
برای این الگوریتم، نیاز داریم که بدانیم آخرین پردازه‌ای که در این سطح زمان‌بندی شده کدام است، تا از بعد آن شروع به اختصاص دادن پردازنده کنیم. برای همین آن پردازه را به تابع roundrobin پاس می‌دهیم. اگر پردازه‌ای پیدا نشود، 0 را برمی‌گردانیم تا در scheduler سراغ سطح بعدی برویم.

1. **زمان‌بند بخت‌آزمایی**



ما در هر پردازه که در سطح lottery است، یک عدد به tickets\_count از 1 تا 10 اختصاص دادیم. در این تابع همه این مقادیر را جمع می‌کنیم و با ایجاد یک عدد رندوم می‌بینیم که در بازه کدام پردازه قرار می‌گیرد. اگر پردازه‌ای انتخاب شد، آن را به scheduler ریترن می‌کنیم. در غیر این صورت، 0 را برمی‌گردانیم تا جستجو در سطح بعدی ادامه پیدا کند.

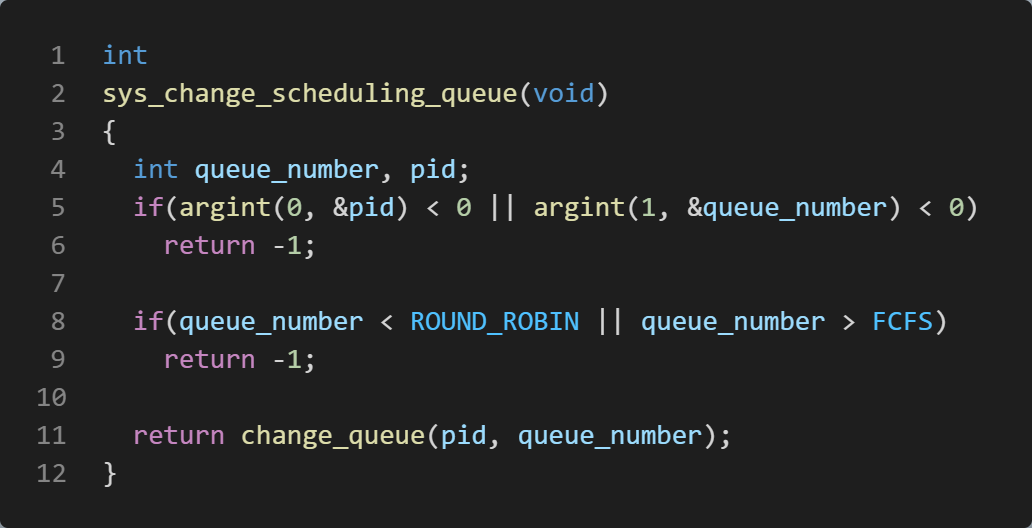
1. **زمان‌بند اولین ورود-اولین رسیدگی (FCFS)**

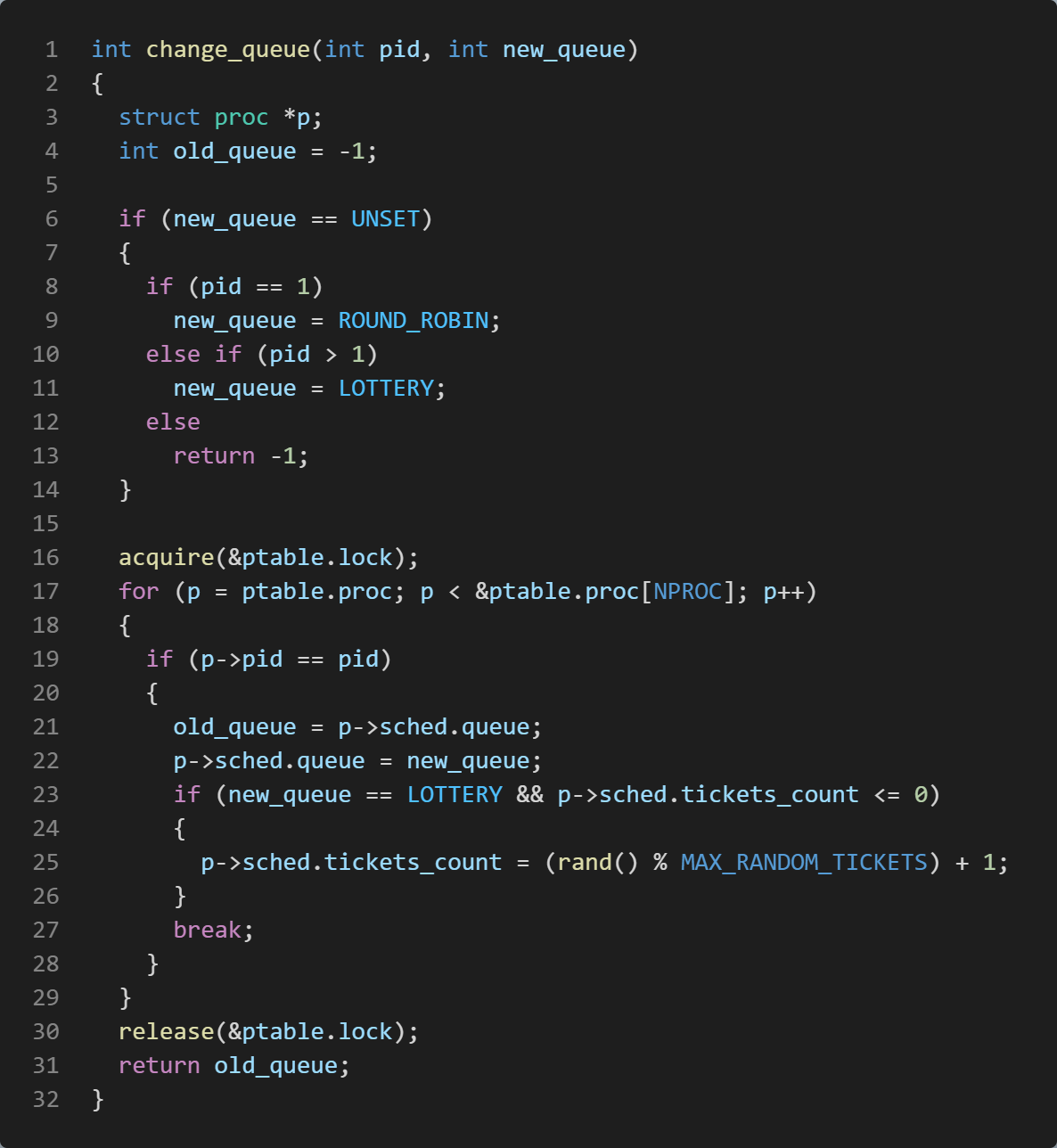


در همه پردازه های این سطح پیمایش می‌کنیم و پردازه‌ای که کمترین arrival\_time را دارد را بر‌می‌گردانیم.

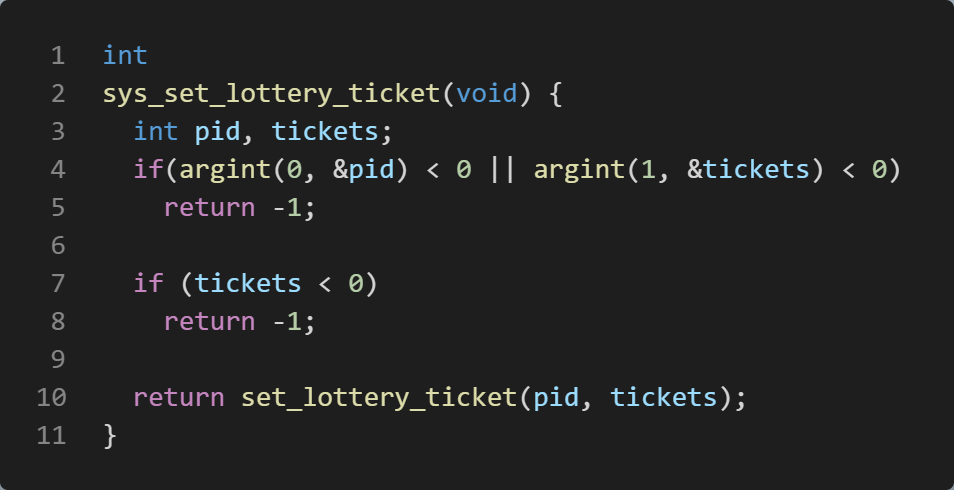
**فراخوانی‌های سیستمی**

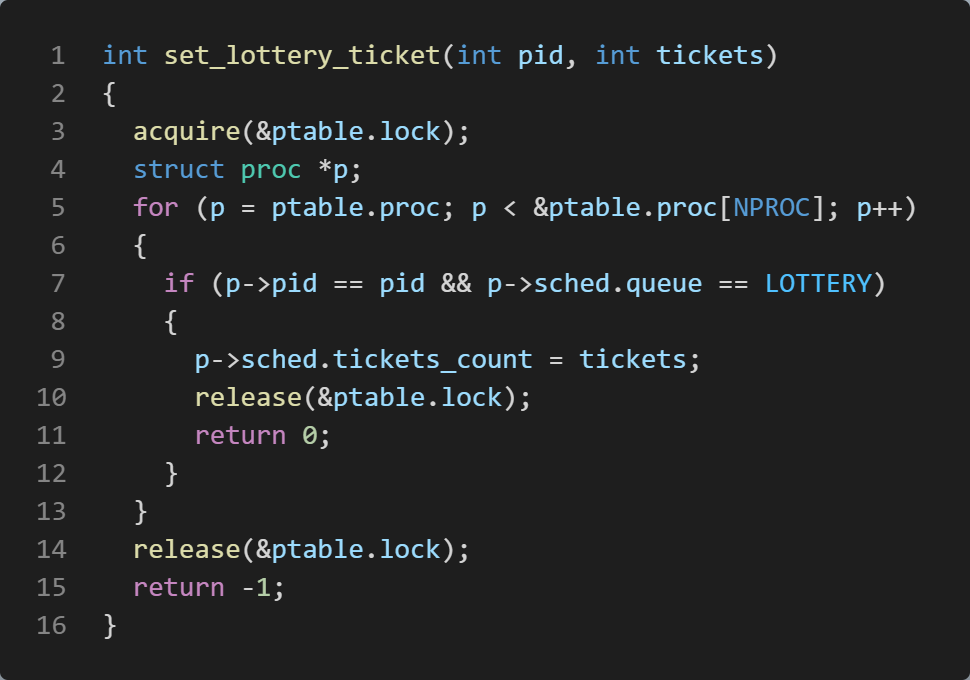
1. **تغییر صف پردازه**



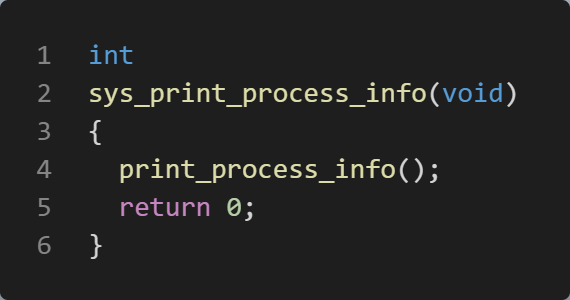


1. **مقداردهی بلیت بخت‌آزمایی**





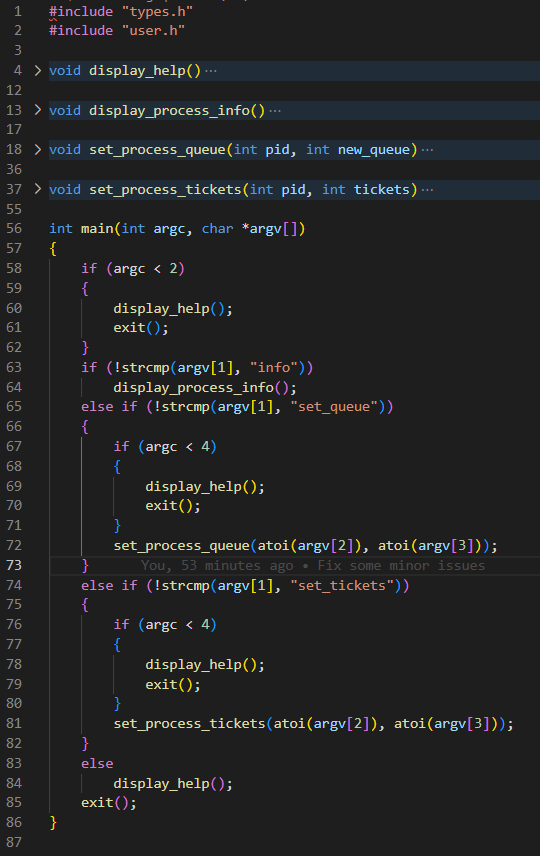
1. **چاپ اطلاعات**



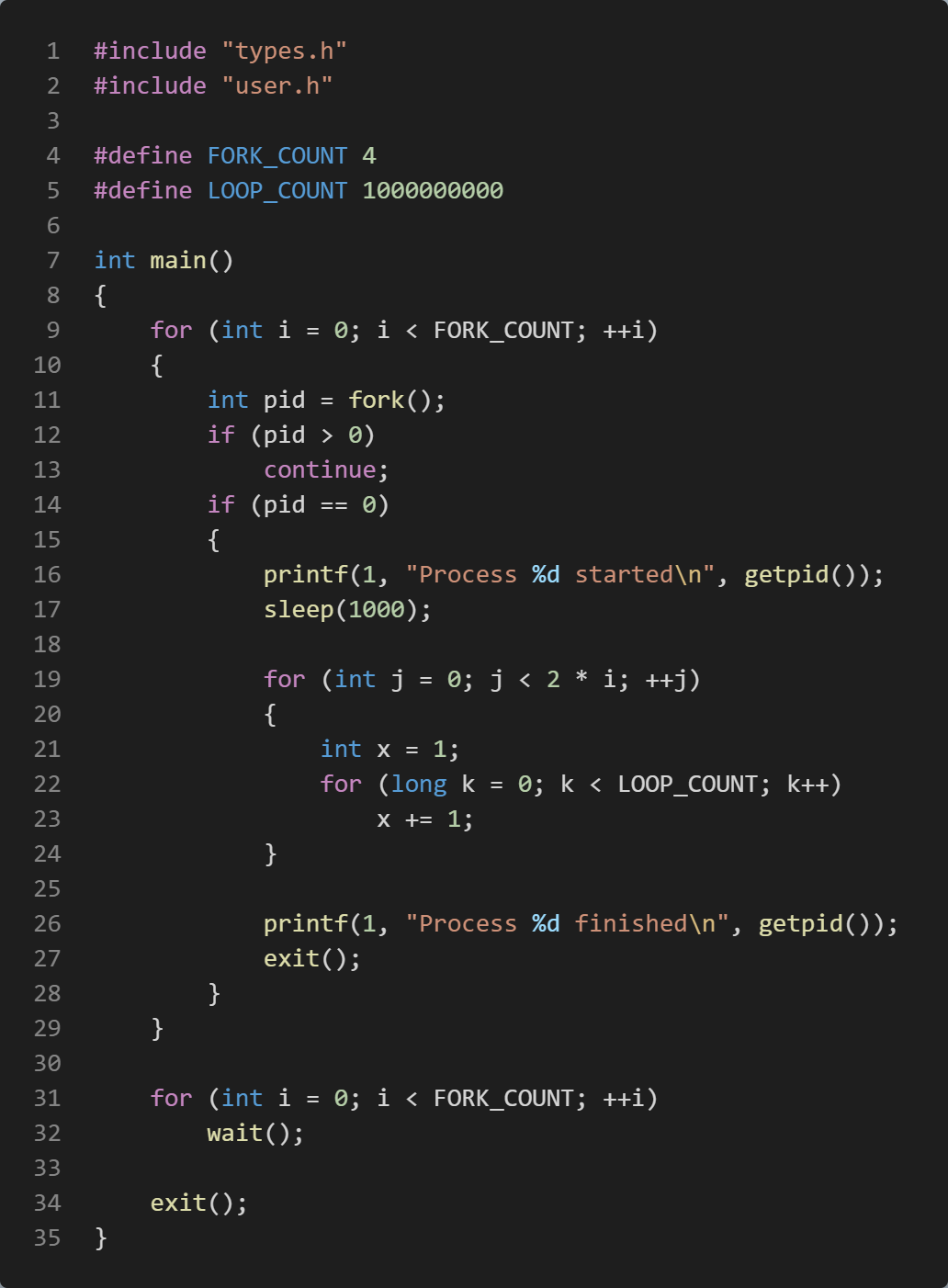


**برنامه سطح کاربر**

برنامه‌ای به نام sched ایجاد کردیم که در آن از 3 سیستم‌کالی که تعریف کردیم استفاده می‌کنیم.



برنامه دیگری به نام foo نیز داریم که 4 پردازه در خودش fork می‌کند تا اطلاعات آن را با سیستم‌کال چاپ اطلاعات مشاهده کنیم.



**اجرای برنامه سطح کاربر**

