**باسمه تعالی**

**گزارش فاز دوم پروژه درس سیستم عامل – دکتر جوادی**

**گروه ۱۵**

**علیرضا شرکت معصوم ۹۷۲۳۰۴۵**

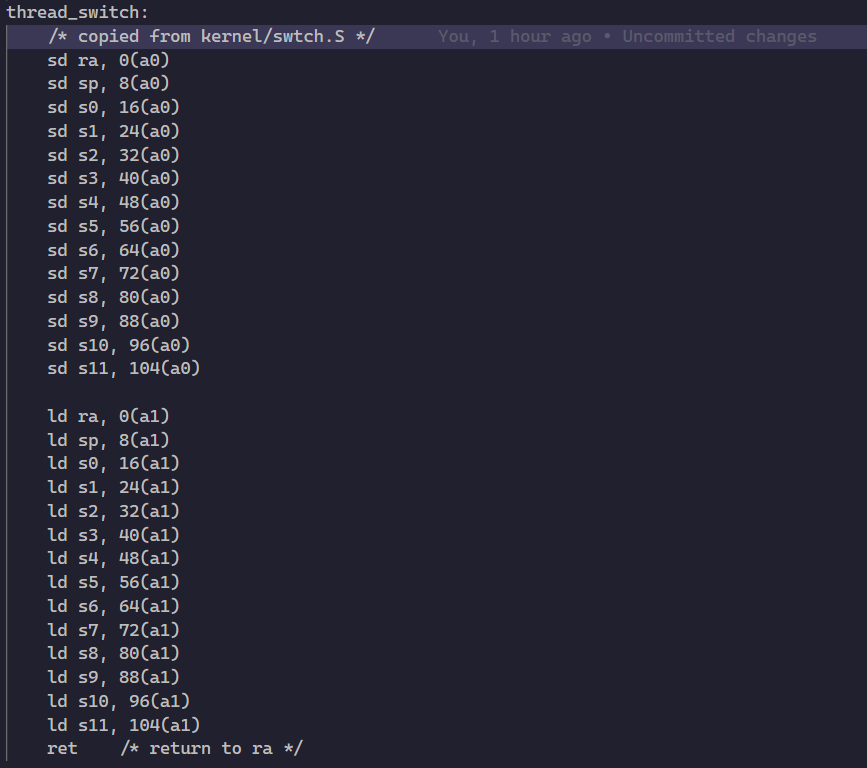
**سید امیر مهدی میر شریفی ۹۸۳۱۱۰۵**

# بخش اول (User Level Multithreading)

## تغییرات فایل user/uthread\_switch.S

ابتدا میخواهیم تابع اسمبلی thread\_switch گفته شده در صورت سوال را پیاده سازی کنیم.

برای اینکار طبق صحبت های گفته شده در صورت سوال و فیلم آموزشی باید به پیاده سازی این تابع در kernel level نگاه کنیم. پس به سراغ فایل kernel/swtch.S رفته و بخش مورد نیاز را در فایل خودمان کپی میکنیم.



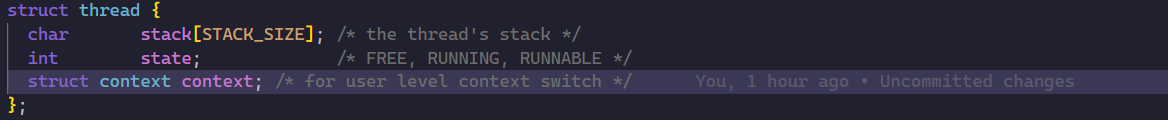
## تغییرات فایل user/uthread.c

برای پیاده سازی context switch اول از همه نیاز به ساختار context داریم که باز هم میتوانیم از پیاده سازی مربوطه در kernel level استفاده کنیم. در این ساختار قرار است آدرس بازگشت و استک پوینتر و باقی رجیستر های مورد نیاز یک نخ تعریف شوند.

کد مربوط به ساختار context از فایل kernel/proc.h کپی شده و در فایل قرار میگیرد.



سپس باید به هر نخ یک محتوا(context) بدهیم که از طریق اضافه کردن context به ساختار نخ انجام میشود. پس تا الان همه ی نخ های ما یک ساختار محتوا دارند که با استفاده از آن میتوانیم context switch کنیم.



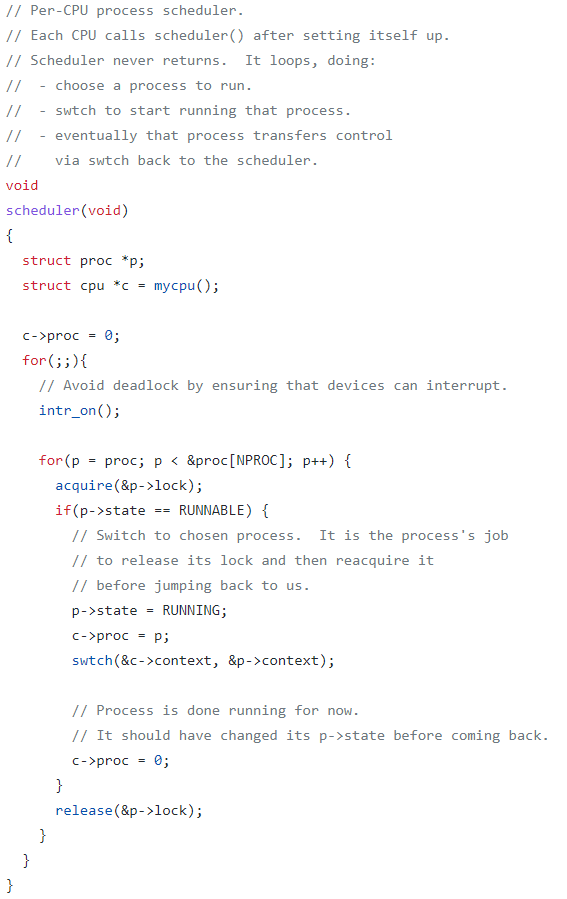
سپس در تابع thread\_create در بخش مشخص شده برای کدمان (بعد از تغییر وضعیت نخ به قابل اجرا) سه خط کد زیر را اضافه میکنیم که باعث می شوند یک context برای نخ فعلی ساخته شده (خط اول) و آدرس برگشت محتوای نخ برابر تابع ورودی بشود(خط دوم) شود و مقدار stack pointer به آخر پشته اشاره کند(خط سوم).



حال به تابع thread\_schedule می رسیم که وظیفه زمان بندی و جا به جایی بین نخ ها را بر عهده دارد و تنها کاری که ما باید بکنیم این است که با استفاده از تابع thread\_switch ای که در فایل قبلی نوشتیم محتوای نخ قبلی را در جایی از حافظه ذخیره کرده و محتوای نخ جدید را لود کنیم. پس تابع thread\_switch را صدا زده و محتوا های ترد فعلی و بعدی را به آن میدهیم.

**بخش دوم**

در ابتدای بخش دوم به سراغ عملکرد پردازشگر برای زمان بندی و روند انتخاب یک پردازه جهت پردازش و عملیات طی شده برای انتخاب و تغییر آن خواهیم رفت

.این کد ، کد تابع زمان بندی است که در فایل proc.c

است و مسئولیت زمان بندی را به عهده دارد. همانطور

که ملاحظه میکنید در ابتدا یک ساختار پردازه و یک

ساختار پردازشگر تعریف کرده ایم. سپس با یک حلقه

بی نهایت و در هر دور آن از اولین پردازه شروع میکنیم

و تمام پردازه ها را چک خواهیم کرد و اگر قابلیت

اجرا داشتند اختیار اجرا را به آنها میدهیم

این اختیار در ابتدا با تغییر وضعیت آنها اتفاق می افتد.

برای مثال state آنها را به running و همچنین

c->proc را برابر پردازشگر قرار میدهیم.

سپس نوبت عملیات context switch است که

به کمک تابع swtch انجام میشود

محتویات تابع swtch به زبان اسمبلی است که

محتوای رجیستر های پردازشگر را تغییر میدهد.

در نهایت پردازشگر را از پردازه میگیریم و دوباره حلقه

جدید را آغاز میکنیم.

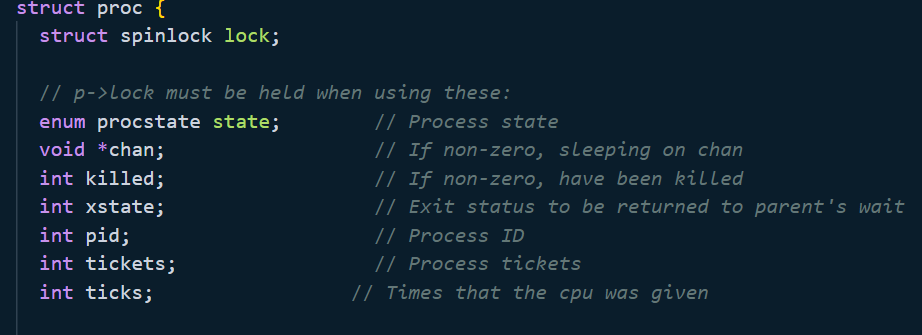
این تابع در تابع اصلی فراخوانی میشود پس از انجام

تنظیمات اولیه در آن و ساخت اولین پردازه کاربر.

پیاده سازی سیستم کال settickets :

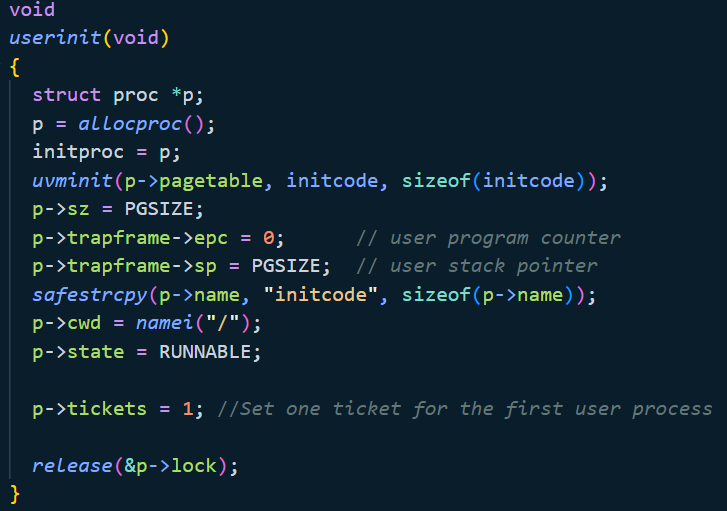
در این بخش که اولین قدم به سوی پیاده سازی الگوریتم زمان بندی لاتری است می بایست تغییراتی در ساختار پردازه ها و بعضی از توابعی که مربوط به عملکرد آن ها است بدهیم زیرا ما در این الگوریتم ، با فیلدهایی از پردازه ها کار خواهیم کرد که تا الان وجود نداشته اند و صرفا اضافه کردن این فیلد ها به ساختار پردازه کافی نیست و در دیگر بخش ها هم نیاز به تغییر است.

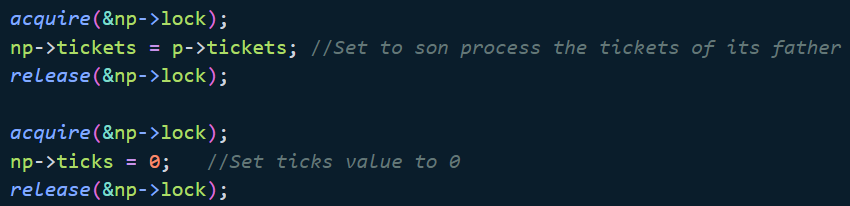
برای اولین قدم می باست در فایل proc.h که ساختار پردازه مشخص شده اند فیلد های tickets & ticks رو اضافه بکنیم . فیلد tickets بیانگر میزان بلیطی است که به پردازه اختصاص داده ایم و فیلد ticks تعداد دفعاتی است که پردازه حین انتخاب پردازه ها برای اجرا پردازه مدنظر را انتخاب کرده است.



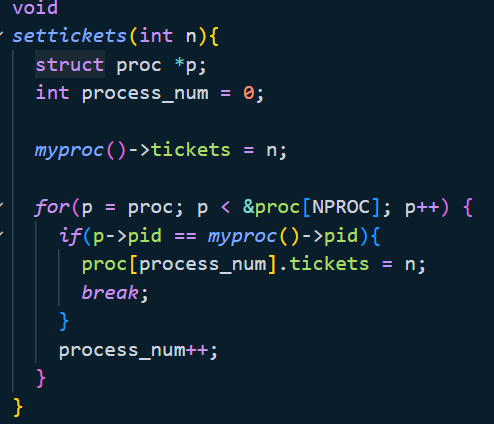
سپس به سراغ فایل proc.c خواهیم رفت که تابع هایی اولیه برای پردازه ها را تعریف میکند.

یکی از این توابع userinit است که مقداردهی های اولیه پردازه اولیه کاربر را انجام میدهد و می بایست مقدار بلیط اولیه پردازه را برابر یک که حداقل تعداد بلیط برای اجرا است را مقداردهی بکنیم:



سپس به سراغ تابع fork میرویم ، از آنجایی که برای ما اهمیت دارد که هنگام فرک تعداد بلیط های والد به فرزند ارث برسد پس تنظیمات زیر را انجام میدهیم. همچنین بدهی است که مقدار ticks هم میبایست صفر تنظیم شود برای فرزند:

این کد بخشی از تابع فرک است که مقادیر تیکت و تیک را مقدار دهی میکند.

حال به سراغ تابعی می رویم که مقدار تیکتی که میخواهیم ست کنیم را به فیلد تیکت پردازه بدهد.

همانطورکه در تصویر رو برو مشاهده میکنید این تابع ابتدا مقدار

n که مقدار بلیطی هست که میخواهیم به پردازه اختصاص بدهیم

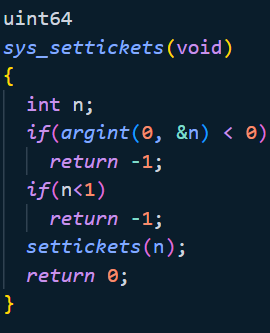
را میگیرد و سپس از طریق تابع myproc() بلیط را به پردازه ای

که در حال اجرای این تابع است اختصاص میدهد.

سپس از میان تمام پردازه های موجود میگردد و پردازه در حال

اجرا را پیدا و مقدار اساین شده به آن را در لیست قرار میدهد تا

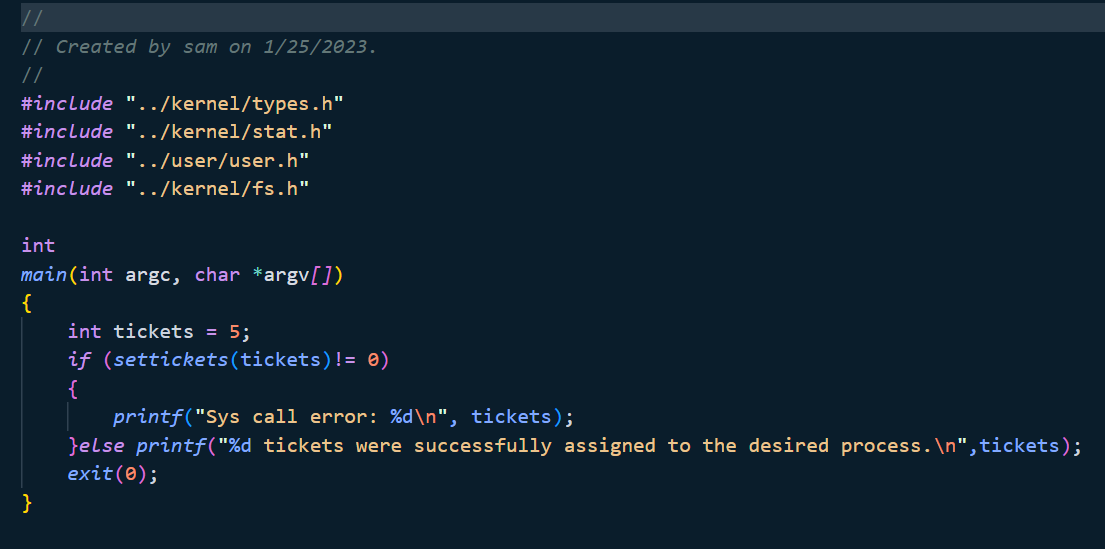
بعدا اگر خواهان اطلاعات پردازه ها به صورت یک لیست بودیم بتوانیم این کار را انجام دهیم.

بعد ازاعمال این تنظیمات به سراغ اضافه کردن سیستم کال خواهیم رفت که مراحل آن را در فاز قبلی پروژه بررسی کرده ایم و دیگر در اینجا به آن اشاره نمی کنیم به جز تابع sys\_settickets که در فایل sysproc.c آن را اضافه میکنیم که به بررسی کد آن می پردازیم:

در این تابع مقداری که اساین شده است را میتوانیم با تابع argint() دریافت کنیم. ما توانایی پاس دادن مقدار به سیستم کال ها را به صورت مستقیم نداریم و سیستم xv6

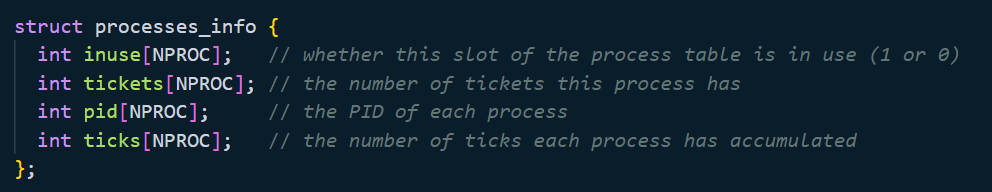
به واسطه ساختار داخلی مخصوص خود و به واسطه تابع argint میتواند آرگومان مد نظر را دریافت کند. بعد از دریافت آرگومان تابعی که در بالا تعریف کردیم را اجرا خواهیم کرد.

حال جهت بررسی درستی عملکرد سیستم کال یک برنامه سطح کاربر مینوسیم و هنگام بالا آمدن سیستم عامل میتوان با تایپ دستور settickets نتیجه را با موقیت مشاهده کرد.



پیاده سازی سیستم کال getprocessesInfo :

در قدم ابتدایی می بایست یک ساختار تعریف کنیم که اطلاعات پردازه ها را در خود نگهداری می کند:



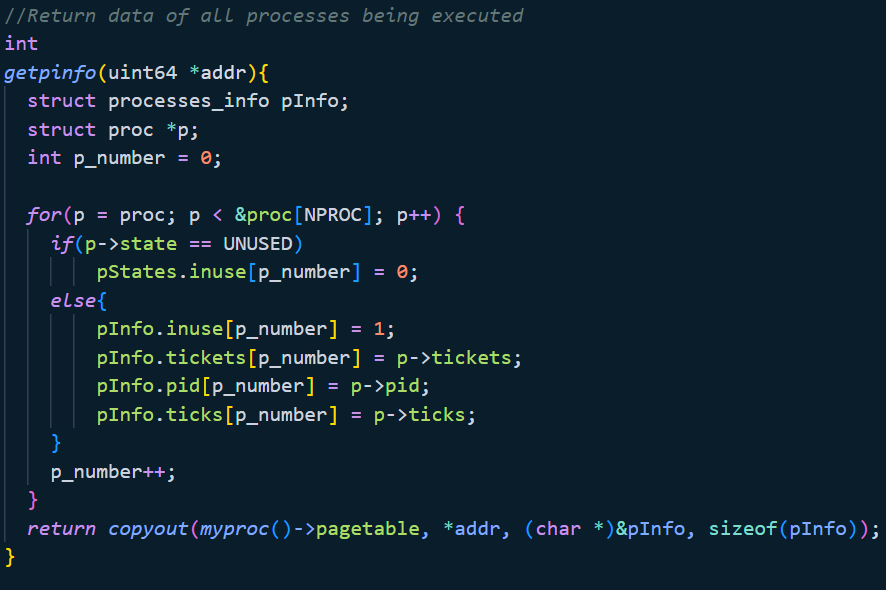
فیلد inuse بیانگر این هست که آیا پردازه در حال استفاده است یا خیر. که مقدار آن به این صورت چک میشود که اگر وضعیت پردازه :

p->state == *UNUSED*

به این صورت باشد آنگاه جزو قابل استفاده ها تعریف نمیشود.

فیلد تیکت ها و تیک ها و آیدی پردازه ها هم آرایه هایی برای نگهداری این اطلاعات خاص پردازه ها هستند.

پس از این مرحله حال به سراغ تعریف یک تابع در فایل proc.c میرویم که اطلاعات پردازه ها را در موقع فراخوانی این تابع جمع آوری و به محیط کاربر ارسال خواهد کرد:

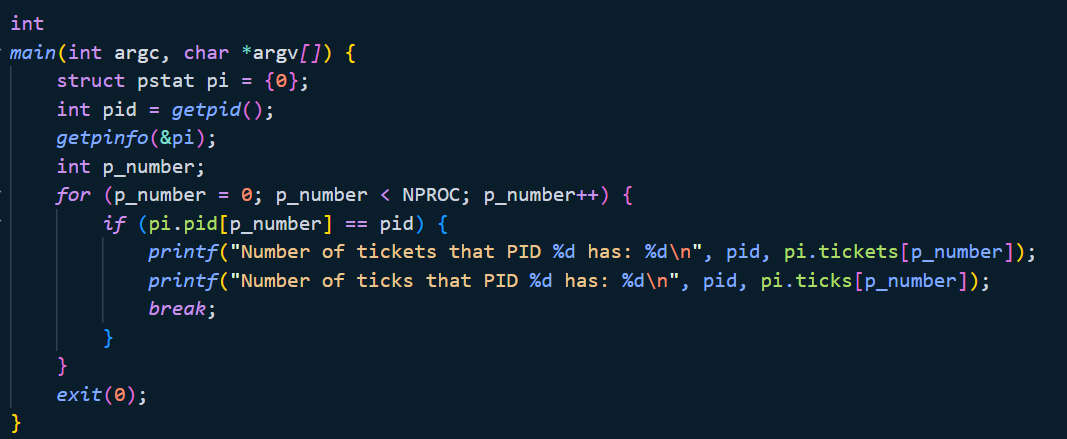


همانطور که مشاهده میکنید تابع getinfo باگرفتن آدرسی که به عنوان آرگومان دریافت میکند اطلاعات پردازه ها را به وسیله تابع copyout از سطح کرنل به سطح کاربر ارسال میکند. به این صورت که با حلقه ای به تعداد پردازه ها ، وضعیت فعال یا فعال نبودن آن ها را بررسی خواهد کرد و همچنین اگر فعال بوده باشند اطلاعاتی همچون تعداد تیکت و تیک ها و آیدی آن را دریافت و به ساختار اطلاعاتی ما اضافه خواهد کرد.

در نهایت تمام اطلاعات پردازه ها را به آدرس خواسته شده در سطح کاربر ارسال میکند.

درنهایت هم مراحل پیاده سازی سیستم کال را انجام خواهیم داد.

حال به سراغ این می رویم که یک برنامه سطح کاربر برای تست آن بنویسیم که این برنامه صرفا جهت تست درستی سیستم کال است.



**پیاده سازی الگوریتم لاتری**

در ابتدا لازم به ذکر است که با توجه به این که در صورت سوال اشاره به این شد برای بلیط ها یک سقف مثلا 1000 را در نظر بگیریم در فایل sysproc.c این بخش به روزرسانی شد و این محدودیت نیز اعمال شد.

همنطور که می دانیم الگوریتم لاتری به این صورت کار میکند که به هر پردازه مقداری بلیط اختصاص پیدا کرده است ، و ما موقع انتخاب پردازه ، در ابتدا یک عدد رندوم بین بازه 0 تا تعداد کل بلیط ها تولید میکنیم.

سپس پردازه ای را انتخاب خواهیم کرد که عدد رندوم در بازه بلیط های آن باشد. در کد به این صورت عمل شده است که از پردازه اول شروع می کنیم. عدد رندومی مثل مثلا 50 و تعداد بلیط ها برای سه پردازه به ترتیب ( برای مثال ) 10و 20 و 30 است. در این صورت اگر عدد رندوم در بازه 0 تا 10 بود پردازه یک انتخاب، اگر عدد رندوم در بازه 11 تا 30 بود پردازه دو انتخاب و اگر عدد رندوم در بازه 31 تا 60 بود پردازه سوم انتخاب میشود. ما این الگوریتم را در کد به این صورت پیاده کرده ایم که از اولین پردازه شروع میکنیم،

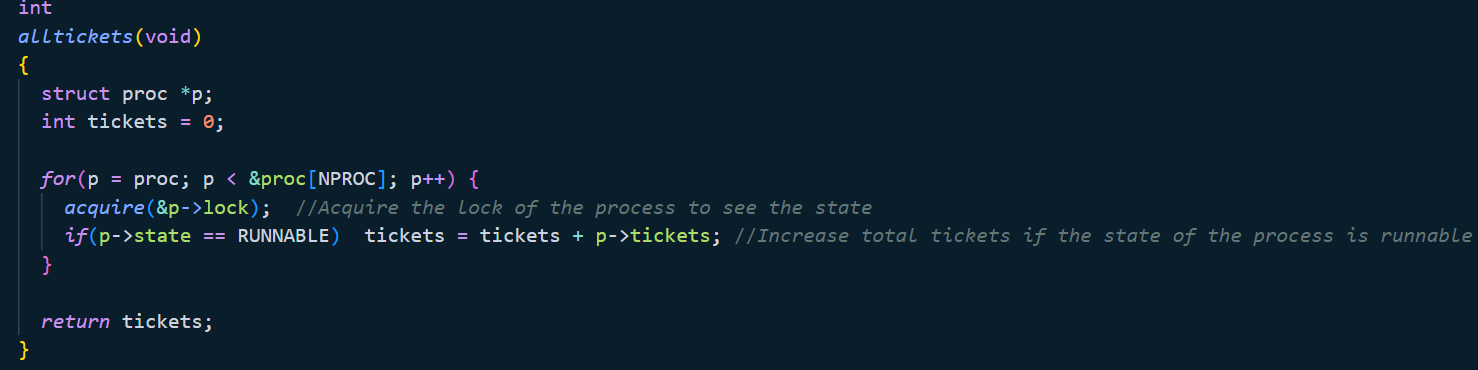
عدد رندوم را منهای تعداد بلیط پردازه ی انتخاب شده میکنیم. اگر عدد منفی شود، یعنی عدد رندوم در بازه پردازه اول بوده است و پردازه انتخاب میشود و الا به سراغ پردازه بعدی با عدد رندوم آپدیت شده خواهیم رفت.

برای مثال در اینجا درمرحله اول 50 را منهای 10 خواهیم کرد ، که میشود 40 و مثبت است. پس باید سراغ پردازه بعدی برویم. 40 را منهای 20 میکنیم که تعداد بلیط پردازه دوم است و میشود 20 که همچنان مثبت است و می بایست به سراغ پردازه بعدی برویم. در مرحله بعد 20 را منهای 30 میکنیم و حاصل منفی است.

بنابراین پردازه سوم انتخاب میشود. در ابتدا بیان شد که برای انتخاب پردازه سوم عدد رندوم میبایست در بازه 31 تا 60 می بود که با این الگوریتم به درستی پردازه سوم انتخاب شد.

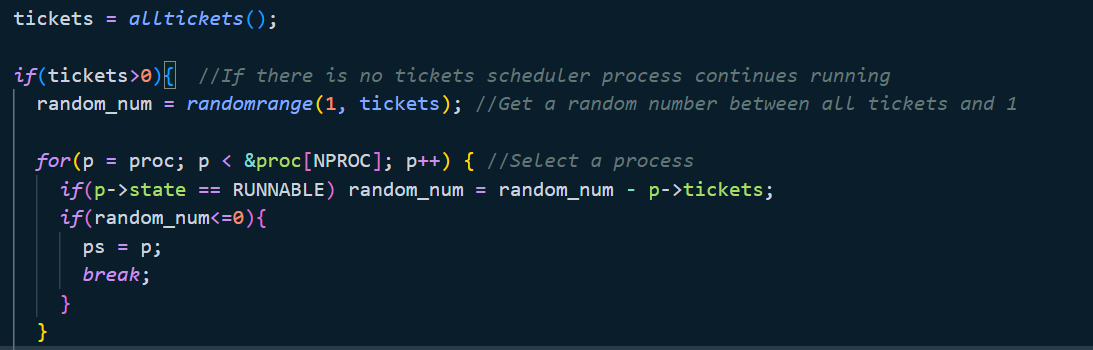
همانطورکه در توضیحات بالا بیان شد درکنار خود پیاده سازی الگوریتم ما نیاز به دو پیاده سازی دیگر داریم.

اول ، پیاده سازی تابعی است که تعداد کل بلیط های پردازه ها را بشمارد. این پیاده سازی را در فایل proc.c انجام می دهیم:

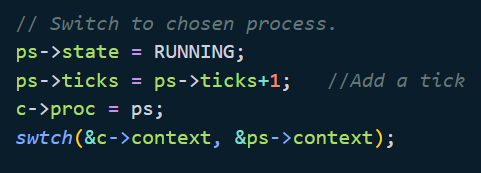


همانطور که مشاهده کردید این تابع بلیط پردازه هایی که قابلیت انتخاب شدن و اجرا را دارد میشمرد.

تابع بعدی تابع رندوم است که می بایست آن را به کتابخانه های سطح کرنل اضافه کنیم که این کار با اضافه کردن فایل random.c اتفاق افتاده است و میتوان از طریق تابع randomrange یک عدد تصادفی بین بازه دلخواه تولید کرد.

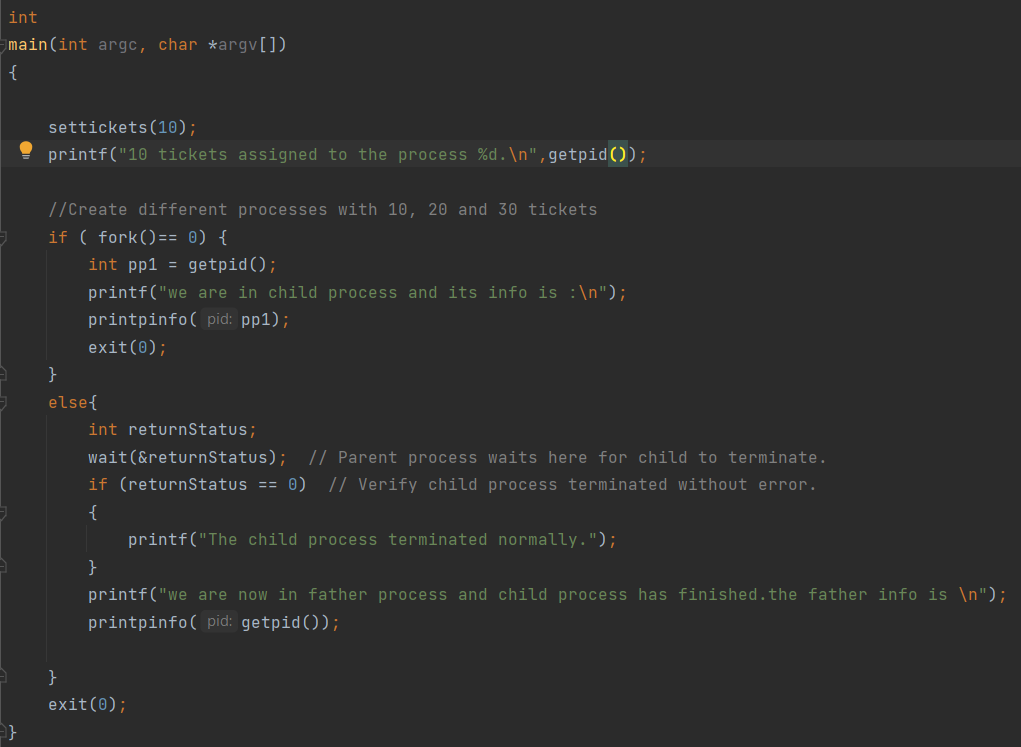
عکس زیر مراحل پیاده سازی الگوریتم است که توضیح داده شد و در تابع scheduler در فایل proc.c پیاده سازی شده است. بقیه اجزای این تابع همان عملکردی را دارند که در ابتدای این بخش درباره شیوه زمان بندی توضیح داده شد.

پس از انتخاب پردازه می بایست مقدار تیک آن را یکی افزایش دهیم که در قطعه کد زیر قابل مشاهده است.



تست یک:

در این تست به یک پردازه ده تیکت میدهیم و پس از فورک اطلاعات فرزندش را چاپ و بعد از پایان فرزند اطلاعات خودش را نیز چاپ میکنیم تا چک کنیم با فورک میزان تیکت داده شده درست است یا خیر



بخش امتیازی:

