به نام خدا

پروژه‌ی چهارم آزمایشگاه سیستم‌عامل

تاریخ تحویل: ۹۵/۹/۲۸

اهداف آزمایش:

- آشنایی با زمان‌بند و اولویت وظایف در لینوکس
- آشنایی با نحوه‌ی پیاده‌سازی زمان‌بند
- پیاده‌سازی و تست سیاست زمان‌بندی جدید

چکیده:

در این آزمایش ابتدا به بررسی نحوه‌ی پیاده‌سازی زمان‌بند در هسته‌ی لینوکس 2.6.32 می‌پردازیم و سپس سعی می‌کنیم تا با تغییراتی سیاست fair-share scheduling را پیاده‌سازی کنیم.

شرح آزمایش:

- آشنایی با زمان‌بند لینوکس

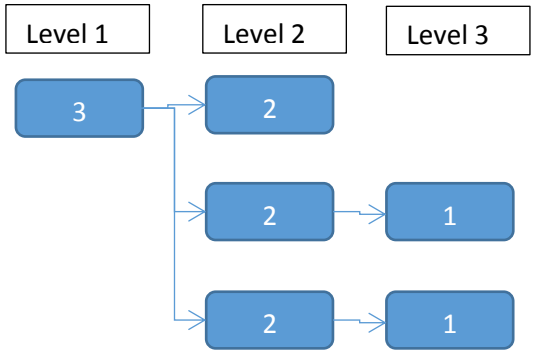
زمان بند پیش‌فرض لینوکس به طور مکرّر میان وظایفِ در حال اجرا Switch می‌کند و به هرکدام سهمی از زمان پردازنده را اختصاص می‌دهد. سیاست پیش‌فرض لینوکس برای زمان‌بندی الگوریتم Completely Fair Scheduler (CFS) است.

یکی از سیاست‌هایی که زمان‌بند می‌تواند طبق آن سهم هر کدام از وظایف را بدهد، سیاست fair-share scheduling است، به این ترتیب که سهم هر پردازه مطابق کاربری که صاحب آن پردازه است، داده می شود. برای مثال اگر 3 کاربر در سیستم داشته باشیم، هر کاربر بدون توجّه به تعداد پردازه‌های در حال اجرایش، $\frac{1}{3}$ از زمان پردازنده را می‌گیرد و آن را میان پردازه‌هایش تقسیم می کند.

همانطور که می‌دانید در لینوکس تمامی فرایندها به جز فرایند init، یک فرایند با عنوان فرایند پدر (parent) دارند. اگر فرض کنیم هرچه فرایند جوان‌تر باشد از اهمیت آن کاسته می‌شود، در این صورت فرایندهایی که نسل طولانی‌تری دارند، باید زمان بیشتری از پردازنده را به خود اختصاص دهند. در لینوکس می‌توانید سلسله مراتب فرایندها را با دستور pstree مشاهده کنید. شکل زیر نمونه‌ای از خروجی این دستور است.

```
init─┬─NetworkManager─┬─dhclient
     │                ├─dnsmasq
     │                └─2*[{NetworkManager}]
     ├─accounts-daemon───{accounts-daemon}
     ├─acpid
     ├─atd
     ├─avahi-daemon───avahi-daemon
     ├─bamfdaemon───3*[{bamfdaemon}]
     ├─bluetoothd
     ├─colord───2*[{colord}]
     ├─console-kit-dae───64*[{console-kit-dae}]
     ├─cron
     ├─cupsd
     ├─2*[dbus-daemon]
     ├─dbus-launch
     ├─gconfd-2
     ├─geoclue-master
     ├─6*[getty]
     ├─gnome-keyring-d───6*[{gnome-keyring-d}]
     ├─gnome-terminal─┬─bash───pstree
     │                ├─gnome-pty-helpe
     │                └─4*[{gnome-terminal}]
     ├─goa-daemon───2*[{goa-daemon}]
     ├─gvfs-afc-volume───{gvfs-afc-volume}
     ├─gvfs-fuse-daemo───3*[{gvfs-fuse-daemo}]
     ├─gvfs-gdu-volume
     ├─gvfs-gphoto2-vo
```

به عنوان مثال سلسله مراتب زیر را در نظر بگیرید:



در این سلسله‌مراتب سه سطح وجود دارد. هرچه به سطوح پایین‌تر (سطح سوم) نزدیک می‌شویم اهمیت فرایندها کم‌تر می‌شود. در این حالت اگر به فرایندهای سطح آخر (یعنی سطح سوم) ۱ بُرِش‌زمانی[1] اختصاص یابد، به پدر این فرایندها (سطح دوم) دو بُرش و به پدربزرگِ آن‌ها (سطح اول) ۳ بُرش زمانی اختصاص می‌یابد.

برای آشنایی با زمان‌بند، از فایل‌های include/linux/sched.h و kernel/sched.c شروع کنید و آن‌ها را بررسی کنید.

[1] Timeslice

- **پس از انجام آزمایش باید بتوانید:**

۱- الگوریتم CFS را همراه با کد آن تشریح کنید.

۲- ساختار داده ی task_struct را که در هدرفایل sched.h تعریف شده است، بررسی کنید. درباره‌ی هریک از اعضایی که داخل آن تعریف شده توضیح مختصری بدهید.

۳- با بررسی الگوریتم CFS خواهید فهمید که در هسته ی لینوکس یک ساختار درخت قرمز و سیاه وجود دارد که برای محاسبه ی سهم پردازه‌های در حال اجرای هر کاربر و تخصیص زمان به آن ها به کار می‌رود. با بررسی تابع‌های اضافه و کم کردن نودها به درخت، روند محاسبات سهم پردازه‌ها را توضیح دهید (برای این کار فایل kernel/sched_fair.c را بررسی کنید.)

- **پیاده سازی سیاست fair-share scheduler:**

در این قسمت شما باید این سیاست جدید را مطابق آن چه که در چکیده توضیح داده شد، پیاده‌سازی و تست کنید. تغییری در این الگوریتم باید بدهید تا فرایندهایی که در سطوح پایین‌تر قرار دارند زمان کمتری از پردازنده را اشغال نمایند. علاوه بر این فرایندهای با اولویت بالاتر (یعنی فرایندهایی که nice value آن‌ها کمتر از 0 است) نباید تحت تاثیر سیاستِ fair-share و سطوح مختلف فرایدها قرار گیرند (یعنی فرقی نمی‌کند فرایندهای با اولویت بالاتر برای کدام کاربر هستند و در چه سطحی قرار دارند، و زمان‌بندی آن‌ها همیشه با CFS انجام می‌شود)

- **تست**

یک سناریو به زبان C بنویسید و نتایج را توضیح دهید. می‌توانید از برنامه‌نویسی bash هم کمک بگیرید.

- **سایر نکات**

تنها بخش‌های تغییریافته‌ی هسته و کدهای مربوط به سناریوی تست را آپلود کنید.