به نام خدا

پروژه طراحی و پیاده‌سازی الگوریتم درخت‌مبانی ریموند در فضای هسته در سیستم عامل FreeBSD

Design and implementation of the Raymond tree mutual exclusion algorithm in kernel-space in FreeBSD

محمد طه جهانگیر Mohammad Taha Jahangir

بهنام مومنی Behnam Momeni

تیرماه ۱۳۹۰

June 2011

فهرست

مقدمه

در این گزارش، در مورد پیاده‌سازی الگوریتم درخت ریموند و بالاخص پیاده‌سازی‌ای که ما انجام داده‌ایم توضیح داده خواهد شد.

این پروژه به عنوان پروژه درس سیستم‌های عامل پیشرفته در دانشکده‌ی مهندسی کامپیوتر دانشگاه صنعتی شریف ارائه شده است و برای کمک به علاقمندان و دانش‌پژوهان طبق مجوز انتشار BSD[1] منتشر می‌گردد.

برخی از واژگان اصلی که در این گزارش استفاده شده‌اند به شرح زیرند:

- سایت: منظور یک سیستم یا رایانه است که الگوریتم را اجرا کرده. آدرس IP مخصوص به خود دارد و پردازه‌های مختلفی در آن اجرا می‌شوند.
- پردازه: (یا process) یک برنامه‌ی اجرا شده روی یک سایت است که می‌تواند فراخوانی سیستمی ورود به ناحیه‌ی بحرانی را صدا کند.

سامانه: منظور کلیت اجرای الگوریتم در چند سایت مختلف است.

---

۱ متن این مجوز به صورت زیر است

معرفی کار به صورت خلاصه

به طور خلاصه باید گفت که پیاده‌سازی ما دارای خصوصیات زیر است:

- پیاده‌سازی کامل در سطح هسته
- امکان درخواست چند پردازنده در یک سیستم
- رعایت بی‌طرفی نسبی (با صرف نظر از تأخیر رسیدن پیام درخواست به سایت دارنده مهره، بین دو درخواست اجابت شده برای یک سایت، امکان اجابت درخواست برای تمامی سایت‌ها فراهم است)
- عدم امکان ورود دو نفر به ناحیه‌ی بحرانی در وجود خطای غیر عمدی
- پاسخگو بودن سایت‌ها (عدم بلوکه شدن) در وجود هر گونه خطا

غیر از گسترش‌هایی که در این پیاده‌سازی می‌توان داشت (که در قسمت کارهای آتی مطرح شده است) مشکلات زیر در این پیاده‌سازی موجود است:

- امکان گم شدن مهره در صورت رخ دادن برخی خطاها (که البته تقریباً غیر قابل اجتناب است)
- برخورد نامناسب و ناکافی در مواجهه با بعضی از خطاها (در قسمت مربوطه، شرح داده شده است)
- کارایی نه چندان خوب در ارتباط بین یک پردازه و سایتی که روی آن اجرا شده است.

کلیات پیاده‌سازی

ما روند پیاده‌سازی را به مراحلی کوچکتر تقسیم کردیم، تا مشکلات مربوط به هر قسمت در همان قسمت مرتفع شوند.

- طراحی منطقی الگوریتم درخت ریموند
- پیاده‌سازی الگوریتم درخت ریموند به صورت کامل در فضای کاربر (user-space)
- پیاده‌سازی اجزای استفاده شده در مرحله‌ی بالا (ایجاد ریسه و ارتباط TCP و Unix domain) در سطح هسته.
- تلفیق دو مرحه‌های بالا و پیاده‌سازی کامل الگوریتم در سطح هسته

لازم به ذکر است که این تفکیک مراحل انجام کل پروژه را بسیار راحت‌تر کرد. همچنین یک کد کامل و تست شده از الگوریتم درخت ریموند در فضای کاربر نیز در مرحله دوم تولید شده است.

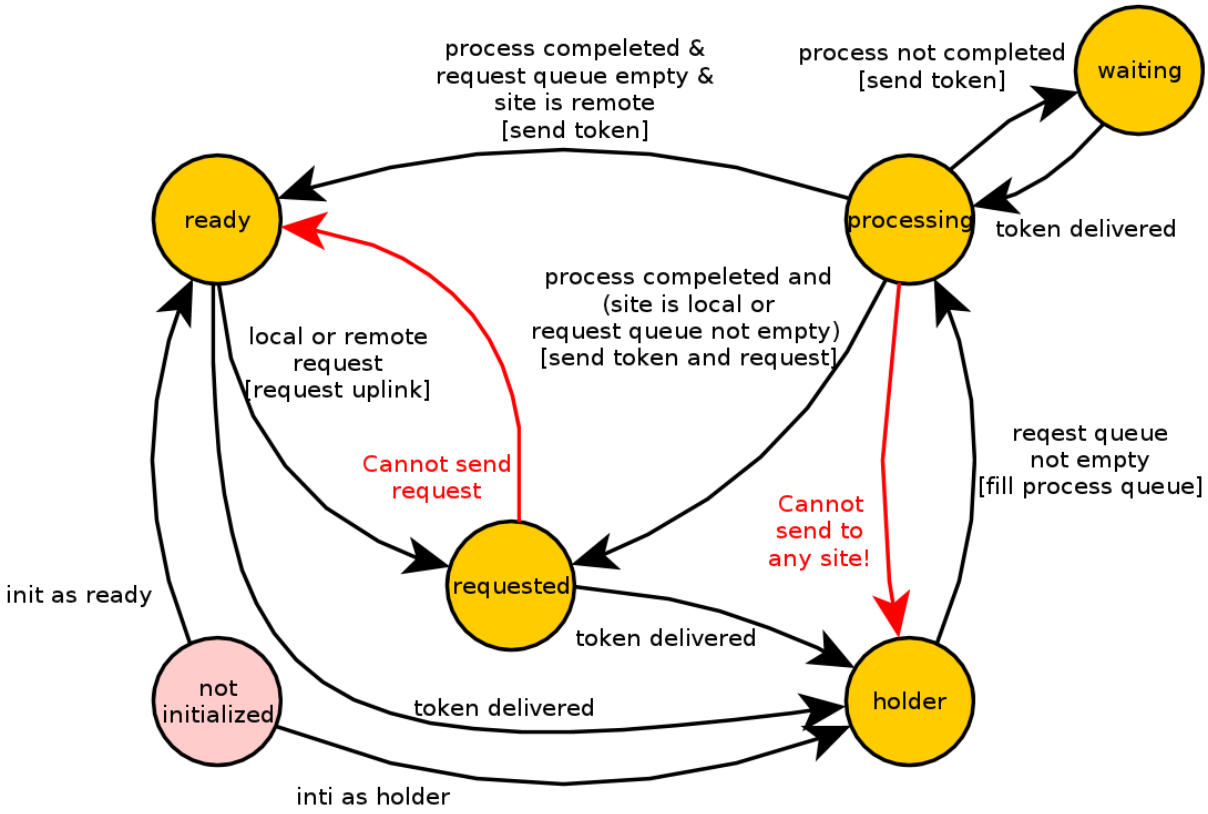
برای رفع ابهام، ذکر این نکته لازم است که اصل سامانه بین سایت‌ها اجرا می‌شود. یک پردازه‌ی محلی برای درخواست و دریافت مهره نیازمند ارتباط با سایت محلی خویش است، سایت محلی با دریافت درخواست محلی شروع به عملیات می‌کند.

طراحی دقیق الگوریتم درخت ریموند

گرچه الگوریتم ریموند، الگوریتم ساده‌ای است، طراحی و پیاده‌سازی آن به گونه‌ای که عاری از هر گونه نقطه‌ی ابهام باشد و ضمناً در برابر خطا نیز تحمل‌پذیر باشد. چندان ساده نیست.

در ابتدا کار برای مدل سازی الگوریتم یک مدل حالت به شکل زیر طراحی شد:

(یال‌های قرمز، مخصوص وقوع خطا هستند، عبارت بین [و] عملی است که در حین انتقال حالت انجام می‌شود و عبارت روی یال واقعه یا شرطی است که در صورت برآورده شدن انتقال حالت انجام می‌شود.) پردازش



شکل ۱: مدل حالت الگوریتم پیاده‌سازی شده

در این مدل حالات زیر برای الگوریتم در نظر گرفته شده:

- شروع نشده (not initialized) وقتی است که هنوز الگوریتم شروع به کار نکرده (و سایت بالاتر برای درخواست مهره مشخص نیست)
- دارنده مهره (holder): حالتی را نشان می‌دهد که مهره آزاد در دست سایت است.
- آماده (ready): حالتی است که سایت مهره را در اختیار ندارد، درخواستی هم برای مهره وجود ندارد.
- متقاضی مهره (requested): حالتی را نشان می‌دهد که این سایت تقاضای مهره را به سایت بالادست فرستاده است.
- در حال پردازش (processing): حالتی است که مهره در دست این سایت است و سایت نوبت به نوبت در حال ارسال مهره به صف مربوطه است.
- انتظار (waiting) : در هر مرحله از پردازش، مهره به یکی از سایت‌ها ارسال می‌شود و این سایت در حالت انتظار، منتظر مهره می‌ماند تا مهره برگردد و ادامه پردازش انجام شود.

همچنین هر سایت دو صف از سایت‌ها را نگهداری می‌کند:

- صف درخواست‌کنندگان مهره از این سایت
- صف پردازش: سایت‌هایی که به نوبت مهره به آن‌ها ارسال می‌شود

پیام‌هایی که در این الگوریتم ارسال می‌شوند، پیام‌های زیرند:

- درخواست مهره
- مهره
- مهره و درخواست آن به صورت همزمان (تلفیق دو پیام بالا)، یعنی مهره را بگیر ولی من خودم هم متقاضی هستم. این پیام در مدیریت بی‌طرفی کاربرد دارد و البته تعداد پیام‌های رد و بدل شده را نیز کاهش می‌دهد.
- (و البته پیام مدیریتی درخواست خاتمه‌ی یک سایت)

نحوه مدیریت صف‌ها و بی‌طرفی

در هر حالتی، هرگاه درخواست برای مهره از یک سایت (یا پردازه) دریافت شود، آدرس آن (که در متن پیام موجود است) به صف *درخواست‌کنندگان* اضافه می‌شود. (توجه داریم که از هر سایت حداکثر یک نسخه در صف قرار می‌گیرد و یک سایت دوبار درخواست نمی‌دهد)

وقتی که مهره به یک سایت برسد، تمام کسانی که در صف درخواست کنندگان قرار دارند (که شاید کسی که مهره را هم فرستاده در بین آن‌ها باشد![1]) به صف پردازش منتقل می‌شوند. پر شدن صف پردازش *فقط* به این روش انجام می‌شود. پس از این عمل باز هم درخواست کنندگان جدید به صف درخواست‌کنندگان می‌روند نه صف پردازش.

از طرف دیگر در آخرین عملیات پردازش در سایت الف (که سایت ب آخرین عضو صف پردازش است)، در صورتی که صف درخواست‌کنندگان خالی نباشد، به جای ارسال مهره پیام «مهره و درخواست همزمان» ارسال می‌شود. به این ترتیب سایت ب، قبل از اجابت درخواست سایت الف، هیچ‌گاه مهره را دو بار به یک نفر نخواهد فرستاد. به این ترتیب بی‌طرفی در سامانه تضمین می‌شود.

البته باید گفت که لزوماً این روش، بهترین روش نیست. چون به احتمال زیاد بین اجابت دو درخواست یک پردازه، مهره بین تمامی سایت‌های درخواست کننده می‌چرخد، هر چند آن‌ها خیلی دور باشند. در این روش تعداد پیام‌های ارسالی بیشتر می‌شود. شاید مناسب باشد که ما دو درخواست یک پردازه‌ی محلی را پشت سر هم اجرا کنیم تا سربار ارسال پیام کاهش یابد.

نکات مربوط به پیاده‌سازی الگوریتم

فایل‌های مهم موجود در پیاده‌سازی ما این فایل‌ها هستند:

- core.c : شامل منطق الگوریتم، مدل حالت و جابجایی بین حالت‌ها
- messaging.c : مسئول ارسال و دریافت پیام بین سایت‌ها
- local.c : مسئول ارتباط با پردازه‌های محلی و رسیدگی به دو تابع enter_critical و exit_critical
- my_kern.c : پیاده‌سازی برخی عملیات که در هسته به طور مستقیم در دسترس نیستند
- syscall.c : تعریف فراخوانی‌های سیستمی، در عمل هر فراخوانی سیستمی یک تابع از core.c یا local.c را صدا می‌زند.
- log.c : چاپ و گزارش پیام‌ها برای مشکل یابی و گزارش خطا

---

۱ البته اگر باشد حتماً آخرین سایت است چون بلافاصله قبل از وصول مهره، درخواست او وصول شده است.

برای دید بهتر نسبت به ساختار نهایی الگوریتم پیاده‌سازی شده در سطح هسته بد نیست نگاهی بـه شکل ۲ در صـفحه ۱۰ بیندازید.

نحوه‌ی ارتباط بین سایت‌ها

هر سایت روی یک درگاه TCP منتظر پیام می‌ماند. این منتظر ماندن منـوط بـه ایجـاد یـک ریسـه بـرای ایـن کـار اسـت. (طبیعیست کسی که سامانه را در یک سایت شروع می‌کند (initiate می‌کند) باید به زندگی برگردد!)
ارتباط بین سایت‌ها با TCP انجام می‌شود. هر کس – چه سایت دیگر و چه یک پردازه محلی – برای درخواست مهـره بایـد یک پیام TCP به سایت محلی بفرستد. بفرستد. همچنین انتقال مهره بین سایت‌ها از طریـق TCP انجـام می‌شـود. در هـر ارتباط دو داده منتقل می‌شود: نوع پیام که یک کاراکتر است، و آدرس فرستنده که برای پاسخ به پیـام مـورد اسـتفاده قـرار می‌گیرد.
اما عملیات انتقال مهره از سایت به یک پردازه با استفاده از Unix domain socket انجام می‌شود. انتخاب این روش سـه دلیل عمده داشت: ۱- روش کار کامل شبیه به TCP است[1] ۲- سریعتر اسـت ۳- محـدودیت تعـداد درگـاه وجـود نـدارد ۴- امکان تمایز بین یک سایت دیگر و یک پردازه را فراهم می‌کند.

مسأله همگامی

مسأله‌ی همگامی یکی از نکاتی است که همواره باید به آن توجه کرد. در پیاده‌سازی ما، استفاده از TCP برای تبادل هر نوع پیام در سطح سایت، باعث می‌شود که مسأله همگامی به کلی مرتفع شود، چون هر سایت در هر لحظه در حال رسیدگی بـه یک درخواست می‌باشد و اصولاً دو ریسه‌ی همزمان در یک سایت وجود ندارند. (ارتباط پردازه‌ها با سایت محلی نیز از طریـق TCP است)
البته در این بین باید توجه داشت که یک سایت، نباید زمانی زیادی را در یک پردازش پیام مصـرف کنـد، و بایـد سـریعاً بـه چرخه‌ی گوش دادن به پیام‌ها بازگردد.

مشکلات سطح هسته

در مرحله‌ی بعد کار، برای پیاده‌سازی قسمت‌های لازم در سطح هسته تلاش شد. قسمت‌هایی که برای استفاده در هسته باید تغییر کنند، شامل موارد زیر است:

- ارسال و دریافت پیام، چه به صورت TCP و چه Unix domain
- ایجاد ریسه در سطح هسته
- توقف و sleep در سطح هسته

---

۱ در حقیقت صف‌های موجود در الگوریتم، هر عضوشان یک sockaddr است که آدرس متقاضی مهره است. ما از امکان چنـد ریخـتی بیـن sockaddr_un (که یک آدرس Unix domain است) و sockaddr_in (که یک آدرس IP است) استفاده کرده‌ایم و هر دوی این‌ها را در یک صف قرار داده‌ایم. این کـار یک مشکل کوچک داشت: اندازه sockaddr_un چون محتوی یک اسم فایل است بسیار بزرگ‌تر (در حدود ۱۲۰ بایت) است. ما برای غلبه بر این مشکل از اسم فایل‌هایی با حداکثر اندازه ۱۳ استفاده کردیم تا اندازه‌ی این سه struct کاملاً با هم یکی باشند. به این ترتیب تمـامی پیام‌هـای ارسـالی بیـن سـایت‌ها اندازه‌ی یکسان دارند و آن اندازه sockaddr به اضافه یک (نوع پیام) است!

ارسال و دریافت پیام در سطح هسته

ارسال و دریافت پیام در سطح هسته، آری یا نه؟

در مورد ارسال و دریافت پیام در هسته بحث و گفتگو و مشکل فراوان است.
اولاً در تمام منابع – بلا استثناء – این کار منع شده است! در معروفترین مقاله[1] در مورد ارتباط TCP در لینوکس (و نه FreeBSD!) در ابتدای کار این سؤال را مطرح می‌کند که چرا می‌خواهید در سطح هسته این کار را انجام دهید!
در بسیاری از موارد نیز افراد مختلف در جواب افرادی که خواستار این کار شده بودند به مقاله‌ای معروف در مورد نبایدهای سطح هسته[2] اشاره می‌کردند.

روش‌های مختلف انجام در FreeBSD

گذشته از اینکه چرا آری و چرا نه، دو روش عمده برای انجام این کار در هسته یافت شد:
۱- استفاده از netgraph
netgraph یک زیر سامانه سطح هسته است که نگاهی ماژولار و یکسان به شبکه و توابع آن بدست می‌دهد. تقریباً هر کاری با این زیر سامانه امکان‌پذیر است (مثلاً ساخت یک سخت‌افزار مجازی برای Ethernet over VPN).
این زیر سامانه یک سری از توابع مثل ng_ksocket را برای ارتباط در سطح هسته در اختیار قرار می‌دهد.
۲- دوباره نویسی فراخوانی‌های سیستمی!
فرخوانی‌های سیستمی برای استفاده در فضای کاربر ساخته شده‌اند (مثلاً به هر سوکت یک شماره فایل نسبت می‌دهند که در سطح هسته نیازی به آن نیست). به همین علت به صورت مستقیم اصلاً قابل استفاده نیستند. ولی می‌توان از منطق و روال آن‌ها استفاده کرد و معادل آن‌ها را برای فضای هسته دوباره نوشت.

منابع موجود برای این کار

در مورد netgraph باید گفت که گرچه راهنما (man page) مربوط به آن بسیار بزرگ است ولی متأسفانه (قریب به یقین) هیچ مثالی از استفاده از آن در دنیای واقع (حتی در man page آن!) وجود ندارد! برداشت ما از netgraph این بود که استفاده‌کنندگان از آن زیر مجموعه‌ای از نویسندگانش است!
در مورد دوباره نویسی فراخوانی‌های سیستمی نیز منابع زیادی برای انجام این کار (علی الخصوص در FreeBSD) در دست نیست! تقریباً هیچ‌کس مایل به انجام این کار نبوده است، بعد از جستجوی فراوان و خوشحالی بیش از حد از پیدا کردن یک مورد مثال کوچک از قسمتی از این کار، دیدیم که این سؤال نیز از جانب کمک‌استاد درس بود.
البته برای یادگیری دوباره نویسی فراخوانی‌های سیستمی دو منبع زیر بسیار مفید بودند:

- متن فراخوانی‌ها و دیگر متن‌های سطح هسته به همراه ssh و grep[3] و البته man!

1 P. Padala and R. Parim, “Kernel Korner - Network Programming in the Kernel”, Aug 2005, available at http://www.linuxjournal.com/article/7660
2 G. Kroah-Hartman, “Driving Me Nuts - Things You Never Should Do in the Kernel”, Apr 2005, available at http://www.linuxjournal.com/article/8110
۳ مثلاً اجرای دستور زیر برای اطلاع در مورد ساختار proces:

```
cd /usr/src/sys;  grep 'struct proc {' . -R | grep '\.h'
```

- کتابی به نام TCP/IP Illustrated: The implementation که البته جلد دوم یک مجموعه است و پیاده‌سازی فراخوانی‌های سیستمی مربوط به TCP را نیز شرح می‌دهد.

روش انجام کار

توابع موجود در هسته برای کار با سوکت، توابع solisten، sobind، soaccept، soconnect، socreate و soclose هستند. از بین این توابع دو تابع soaccept و soconnect برای اتصال بلوکه نمی‌شوند. باید کد بخصوصی (که در متن فراخوانی‌های سیستمی مربوط به آن‌ها هست) درج شود.
همچنین مقدار زیادی کد برای بررسی حالات خطا و همینطور حالت سوکت لازم است که برخی از آن‌ها (مثل حالت خاص برای پردازنده mac و همینطور سوکت non-blocking) از متن حذف شده، چون مورد استفاده نبوده‌اند.
همینطور برای دریافت و ارسال اطلاعات از دو تابع sosend و soreceive استفاده شده که این دو تابع داده ساختار مربوط به خود را نیاز دارند. جالب است که داده ساختار مورد قبول برای این دو تابع امکان خواندن و نوشتن در روی چند بافر مختلف به صورت ترتیبی را فراهم می‌کند.
توابع پوششی که ما برای توابع سطح هسته ایجاد کرده‌ایم در فایل my_kern.c قرار گرفته‌اند.

ایجاد ریسه در سطح هسته

در این مورد نیز منبعی که یافت شد خود متن هسته و البته man pageهای مربوط به آن بود.
در FreeBSD 8 دو مفهوم «پردازه‌ی هسته» و «ریسه‌ی هسته» از هم تفکیک شده‌اند. پردازه‌ی هسته با دستوران kproc_start یا kproc_create ایجاد می‌شود، شناسه‌ی هسته‌ی مستقل به خود دارد. در خروجی ps قابل مشاهده است و امکان ارسال signal به آن نیز وجود دارد.
ریسه‌ی هسته با دستورات kthread_start یا kthread_create ایجاد می‌شود و سبکتر از یک پردازه‌ی هسته است.
ما برای پیاده‌سازی استفاده از «ریسه‌ی هسته» استفاده کرده‌ایم.

توقف در سطح هسته

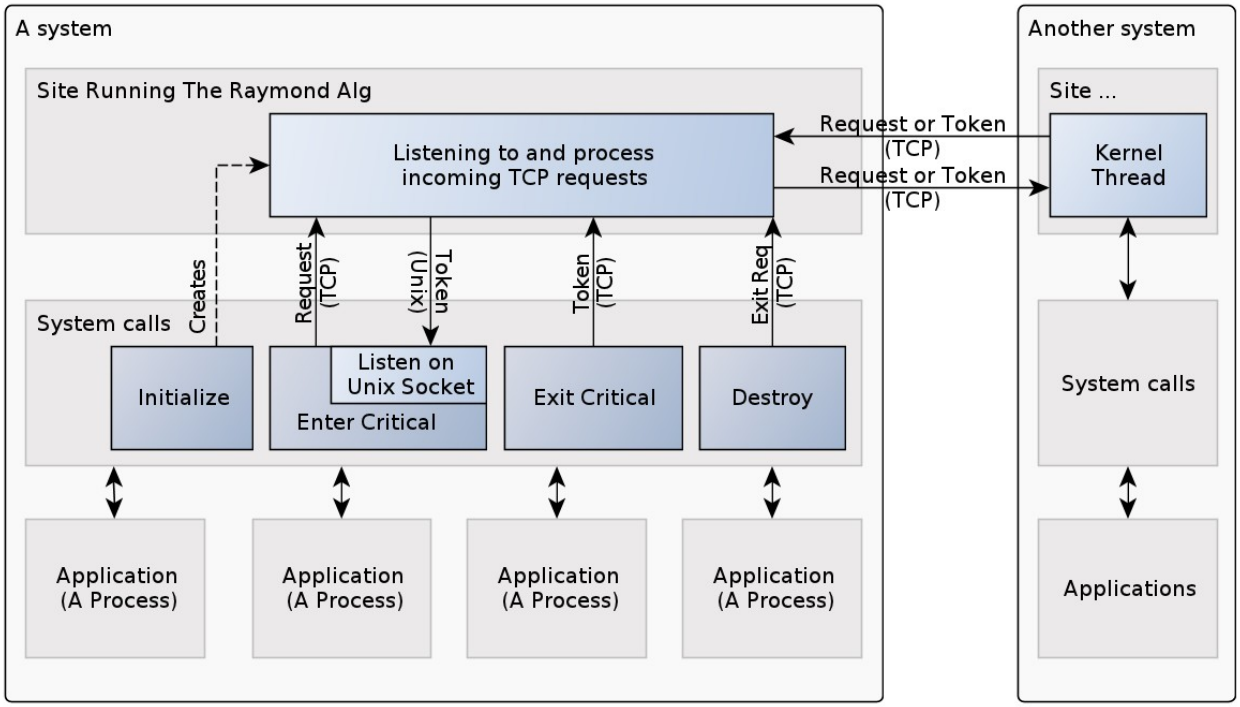
در مورد sleep در سطح هسته که از بقیه ساده‌تر است باید گفت که برای اینکه یک ریسه سطح هسته امکان توقف داشته باشد، حتماً باید option PREEMPTION در هنگام ساخت هسته روشن باشد. این امکان، باعث می‌شود که ریسه‌های سطح هسته نیز امکان توقف داشته باشند.
ما در هنگام کوچک‌سازی هسته این امکان را حذف کرده بودیم که مجبور شدیم دوباره آن‌را به حالت فعال بازگردانیم.

پیاده‌سازی نهایی

تلفیق برنامه‌ی نوشته برای فضای کاربر با فضای هسته، با توجه به مرحله‌ی قبل به راحتی قابل انجام بود. غیر از تغییر includeها و فایل‌های messaging.c و local.c که کاملاً درگیر ارسال و دریافت پیام‌اند بقیه کد (شامل اصل الگوریتم) تغییر بسیار کمی را متحمل شد.
البته با وجود انجام دو مرحله‌ی قبل، باز هم در این مرحله مشکلاتی بروز کرد. دو بار نیز مشکلات حافظه (page fault) در

سطح هسته باعث راه‌اندازی دوباره سیستم (البته VM) شد.

مدل نهایی از روند اجرای الگوریتم در واقعیت به صورت زیر است:



شکل ۲: نگاه کلی به سامانه نهایی

فراخوانی‌های سیستمی

چهار فراخوانی سیستم در سامانه تعبیه شده است:

- فراخوانی init_site که یک سایت را راه‌اندازی می‌کند (با آدرسی که سایت جاری باید در آن درگاه به درخواست‌های گوش دهد و همینطور در صورت وجود آدرس سایت مرحله‌ی بالاتر)
- فراخوانی destroy_site که در حقیقت یک پیام درخواست پایان کار به سایت می‌فرستد. سایت مزبور با دریافت این پایان به کار خود پایان خواهد داد.
- فراخوانی enter_critical برای ورود به ناحیه‌ی بحرانی.

این فراخوانی یک Unix domain socket ایجاد کرده و روی آن منتظر دریافت مهره می‌ماند. پس از ایجاد سوکت قبل از منتظر ماندن پیغام درخواست را به مرحله‌ی بالاتر می‌فرستد.

- فراخوانی exit_critical برای خروج از ناحیه‌ی بحرانی.

آزمون

برای آزمون پیاده‌سازی‌های انجام گرفته و اطمینان اط صحت پیاده‌سازی‌ها و تطابق آن‌ها با طراحی تشریح شده از دو روش آزمون استفاده کردیم. هر یک از این دو روش که در ادامه توضیح داده می‌شود در برنامه‌ای جداگانه نوشته شده‌اند.

روش اول: فراخوانی‌های سیستمی

در این روش برنامه‌ای در فضای کاربر (user space) نوشته شده است که امکان فراخوانی هر یک از چهار فراخوانی سیستمی را از خط فرمان (command line) فراهم می‌کند. این برنامه در فایل test.c نوشته شده و بعد از کامپایل کردن یک فایل اجرایی با نام a.out ایجاد می‌نماید. در ذیل چهار فراخوانی سیستمی مذکور را تشریح می‌کنیم:

- فراخوانی سیستمی آغازین‌دهی سایت (init): با مقداردهی اولیه باعث می‌شود که سایت به درخواست‌ها بر روی یک اتصال TCP گوش داده و ۱. صاحب مهره باشد یا اینکه ۲. به سایتی دیگر اشاره کند که به صورت مستقیم یا غیر مستقیم صاحب مهره باشد و بتواند درخواست دریافت مهره را به آن سایت ارسال نماید. این فراخوانی با دستور ذیل قابل اجرا است:

  $ ./a.out init <address> <port> [<uplink_addr> <uplink_port>]

  در این دستور بعد از کلمه‌ی init آدرس IP سایت و پورت آن آورده می‌شود. اگر سایت صاحب مهره باشد به آرگومان دیگری نیاز نیست و در غیر این صورت باید آدرس IP .و پورت سایت بالادستی در دو آرگومان بعدی داده شوند.

- فراخوانی سیستمی خاتمه‌دهی (destroy): با ارسال پیام خاص خاتمه به سایت باعث می‌شود که سایت فرآیند گوش دادن به پیام‌ها (در ریسه‌ای در هسته) را متوقف نماید. پس از اینکار امکان unload کردن ماژول مربوطه از هسته بدون ایجاد مشکل نیز وجود خواهد داشت. برای اینکار می‌توان از دستور زیر استفاده کرد:

  $ ./a.out destroy

- فراخوانی سیستمی ورود به ناحیه‌ی بحرانی (enter critical section): این فراخوانی با اجرای الگوریتم درخت ریموند همان‌طور که در بخش‌های قبل توضیح داده شد مهره را به سایت می‌رساند. این فراخوانی تا زمان دریافت مهره بلاک خواهد شد. برای اینکار می‌توان از دستور زیر استفاده کرد:

  $ ./a.out enter

- فراخوانی سیستمی خروج از ناحیه‌ی بحرانی (exit critical section): این فراخوانی با اجرای الگوریتم درخت ریموند همان‌طور که در بخش‌های قبل توضیح داده شد مهره را به دیگر سایتهای متقاضی ارسال می‌نماید. در صورت عدم تقاضای سایرین مهره به ریسه‌ی هسته‌ی سایت محلی منتقل شده و آنجا باقی می‌ماند (holder) تا درخواستی به سایت برسد. برای اینکار می‌توان از دستور زیر استفاده کرد:

  $ ./a.out exit

با اجرای دستورات فوق بر روی چندین ماشین مجازی و ارسال درخواست ورود به ناحیه‌ی بحرانی به ترتیب‌های مختلف و دستور آزادسازی مهره به ترتیب می‌توان روند اجرای الگوریتم را به دقت تعقیب و از صحت آن اطمینان حاصل کرد.

روش دوم: آزمون خودکار با معماری کاربر-کارگزار

گرچه روش قبل نمایانگر صحت عمل‌کرد پیاده‌سازی انجام گرفته است ولی همواره امکان وجود خطاهایی که تنها اجرای سریع و همروند چندین پردازه آن‌ها را نمایان می‌سازد وجود دارد. برای اطمینان از عدم وجود چنین خطاهایی یک برنامه با معماری کاربر-کارگزار و در فضای کاربر نوشته شده که از فراخوانی‌های سیستمی پیاده‌سازی شده استفاده کرده و به طور همروند تمامی حالات الگوریتم ریموند را به اجرا در می‌آورد.

این برنامه شامل دو فایل می‌باشد.

- client.c: شامل برنامه‌ی سمت کاربر. این برنامه پیغام‌هایی را به برنامه‌ی کارگزار ارسال می‌کند.
- server.c: شامل برنامه‌ی کارگزار. این برنامه پیام‌های برنامه‌های کاربر را دریافت و ثبت می‌کند.

برنامه‌ی کاربر

این برنامه در چند دور عمل می‌کند و در هر دو تعدادی پیغام به کارگزار ارسال می‌کند. به این شکل که در آغاز هر دور با کمک فراخوانی سیستمی (با بدست آوردن مهره) به ناحیه‌ی بحرانی وارد شده و پیام‌های آن دور را ارسال کرده و در نهایت با اعلام پایان پیام‌ها، از کارگزار پیامی مبنی بر دریافت پیام‌های ارسالی (acknowledgment) دریافت می‌کند. سپس از ناحیه‌ی بحرانی خارج می‌شود و دور بعدی را به همین منوال آغاز می‌کند. بنابراین چندین نمونه از این برنامه بر روی چندین سایت در چندین دور سعی می‌کنند پیام‌های خود را به کارگزار برسانند. این برنامه به صورت زیر اجرا می‌شود:

$ ./client.out <site-id> <server-address> <server-port> <rounds#> <each-round-msgs#>

در این دستور آرگومان شناسه‌ی برنامه/سایت است که با چاپ آن در سمت کارگزار می‌توان پیام‌های برنامه‌های کاربر متفاوت را از هم تفکیک نمود. دو آرگومان بعدی آدرس IP و پورت کارگزار برای ارسال پیام‌های را تعیین می‌کند. آرگومان بعدی تعیین می‌کند که سایت باید چند دور پیام ارسال کند و آخرین آرگومان تعیین می‌کند که در هر دور چند پیام باید ارسال شود. در صورتی که ناحیه‌ی بحرانی به درستی کنترل شود هر مجموعه پیام مربوط به هر دور هر برنامه‌ی کاربر باید به صورت قطعه‌ای و غیر interleave شده به کارگزار تحویل شود.

در نهایت برای اعمال خاتمه‌ی آزمون به برنامه‌ی کارگزار و در نتیجه بستن فایلها و ختم برنامه‌ها می‌توان از دستور زیر برای ارسال پیام متناسب استفاده کرد.

$ ./client.out kill <server-address> <server-port>

این دستور با ارسال پیامی به کارگزار باعث ختم آزمون می‌شود.

برنامه‌ی کارگزار

این برنامه با گوش دادن بر روی یک اتصال TCP پیام‌های چندین برنامه‌ی کاربر را همزمان و با کمک ریسه‌های pthread دریافت می‌کند. این پیام‌ها در یک فایل ثبت می‌شوند. برای این منظور می‌توان از دستور زیر استفاده کرد.

$ ./server.out <port-number> <output-file-address>

در این دستور آرگومان اول تعیین کننده‌ی پورتی است که کارگزار روی آن گوش می‌دهد و آرکومان دوم تعیین کننده‌ی آدرس فایل حاوی پیام‌های ثبت شده که باید قطعه‌ای متمایز باشند.

مقابله با خطا

خطاهای زیادی در هنگام اجرای الگوریتم قابل رخ دادن است. این خطاها را به دو دسته عمدی و غیر عمدی تقسیم می‌کنیم.

همچنین این خطاها از نظر قسمتی که با خطا مواجه شده است، به صورت زیر قابل طبقه‌بندی هستند:

- خطای یک سایت
- خطای خط ارتباطی
- خطای پردازه

قابل ذکر است که خطاهایی در این قسمت مطرح شده شامل خطاهای بدیهی نیست و بیشتر مواردی مطرح شده که در باید در طراحی سامانه مورد توجه قرار می‌گرفته‌اند.

خطای عمدی و غیر عمدی

خطای عمدی خطاهایی هستند که ما انتظار نداریم در یک استفاده درست (و غیر خرابکارانه) این خطاها اتفاق بیفتند. مثلاً انتظار نداریم که درختی که ساخته می‌شود شامل دور باشد! این‌گونه خطاها بیشتر به استفاده کنندگان از سامانه برمی‌گردد. از طرف دیگر به خطاهایی که معمولاً غیر قابل اجتناب می‌شوند و همواره (بدون هیچ‌گونه عمدی) احتمال اتفاق افتادن آن‌ها وجود دارد غیرعمدی می‌گوییم.

البته خطاهای عمدی لزوماً خرابکارانه نیستند، مثلاً به اشتباه در وارد کردن آدرس سایت بالاتر نیز خطای عمدی می‌گوییم.

گرچه بهتر است که سامانه در مورد هر دو نوع خطا مقاوم باشد، ولی تأکید ما بیشتر روی خطاهای غیرعمدی است. اگر در پیاده‌سازی ما این خطا کنترل شده است، نحوه‌ی مواجهه با آن نیز درج شده است.

خطای پردازه

خطاهای زیر برای یک پردازه قابل تصور است:

۱. هنگامی که درخواست مهره داده است، در حین انتظار برای درخواست کشته شود! (غیر عمدی)

نحوه مواجهه: اولاً در هر صورت فایل مربوط به unix socket حذف خواهد شد. ثانیاً بعداً هنگام ارسال مهره به این پردازه، سایت محلی خطا را نادیده خواهد گرفت.

۲. پس از درخواست مهره و قبل از آزادسازی آن کشته شود. (غیر عمدی)

یک ایده‌ی پیاده‌سازی نشده این است که مانند دیگر منابع (فایل و سوکت) کاری کنیم که هنگام بسته شدن پردازه، این منبع نیز آزاد شود.

۳. هنگامی که مهره را در اختیار گرفت، آن را آزاد نکند. (ممکن است عمدی و غیر عمدی باشد)

یک ایده این است که هنگام درخواست برای مهره، حداکثر زمان درخواستی را هم مشخص کند.

۴. هنگامی که مهره را در اختیار دارد دوباره درخواست مهره کند. (عمدی)

می‌توانیم ثبت کنیم که الان مهره دست کیست. به این ترتیب درخواست دوباره رد خواهد شد.

۵. هنگامی که مهره را در اختیار ندارد فراخوانی آزادسازی مهره را صدا کند. (عمدی)

با ثبت کردن که اینکه مهره دست کیست، این مشکل نیز حل می‌شود.

خطای سایت

سایت که در حال اجرای سامانه است (در حقیقت هسته‌ی سایت) ممکن است با خطا مواجه شود. البته بیشتر خطاها، مربوط به ارتباط سایت با دیگر سایت‌ها می‌باشد که در قسمت خطاهای ارتباطی مطرح شده است. خطاهایی که به عنوان خطای سایت می‌توان نام برد، موارد زیر است:

۶. خاموش شدن سیستم فیزیکی یا قطع اجرای سامانه بدون هیچ‌گونه اطلاع (تشخیص آن از خطای ارتباطی ناممکن است)

۷. سایت به علت بروز خطا (مثلاً هنگام گوش دادن به درخواست‌های ورودی) مجبور به ترک سامانه شود. (در این حالت امکان اجرای یکسری از کارها هنگام خروج از سامانه فراهم است)
ایده: می‌توانیم پیغام «لغو» به بقیه بفرستیم تا بیهوده منتظر ما نمانند.

خطاهای دیگر مثل عوض شدن آدرس IP سایت یا قطعی کلی از شبکه، یا به خطای ارتباطی منجر می‌شود یا به خطای گوش دادن به درخواست‌های ورودی.

خطای ارتباطی

با توجه به اینکه ارتباط بین ریسه‌ی اصلی سایت (در هسته) و فراخوانی سیستمی پردازه در پیاده‌سازی ما با استفاده از Unix domain socket انجام می‌شود. دو نوع خطای ارتباطی محتمل است: خطای ارتباط سایت با سایت و سایت با فراخوانی سیستمی.

در مورد ارتباط سایت با پردازه، خطاهای زیر متصور است: (با توجه به اینکه در پیاده‌سازی ما بعد از موفقیت در گوش دادن به یک Unix socket درخواست ارسال می‌شود، اگر در مرحله‌ی ایجاد Unix socket مشکلی پیش بیاید اصلاً درخواست به مرحله‌ی بالاتر ارسال نمی‌شود.)

۸. پردازه پس از ارسال درخواست، به هر علتی منتظر مهره نمی‌ماند (مثلاً چون کشته می‌شود). سایت در هنگام ارسال مهره به پردازه با خطای ارتباطی مواجه می‌شود. (غیر عمدی)
نحوه مواجهه: سایت اگر هنگام ارسال مهره با خطای ارسال مواجه شود، آن را نادیده می‌گیرد.

۹. در هنگامی که پردازه درخواست داده و منتظر مهره است، Unix socket به صورت عمدی پاک می‌شود، به این ترتیب دیگر امکان دریافت مهره وجود ندارد. (عمدی) (از وقوع این خطا چشم‌پوشی شده)

در مورد خطای ارتباط بین دو سایت که شایع‌ترین نوع خطاست. خطاهای زیر متصور است:

۱۰. هنگامی که درخواست یک سایت (یا پردازه) به سایت الف رسید و سایت الف نیز مهره را در اختیار نداشت، باید یک پیغام درخواست به سایت بالاتر بفرستد. ممکن است موفق به ارسال درخواست مهره نشود! (غیر عمدی)
نحوه‌ی مواجه: سایت به حالت آماده بر می‌گردد. این کار گرچه باعث می‌شود سایت بلوکه نشود ولی ممکن است مهره هیچ‌گاه به کسی که درخواست داده نرسد. (لزوم timeout! یا پیغام abort)

۱۱. هنگام ارسال مهره به سایت دیگر خطا رخ دهد (غیر عمدی).

۱۲. یک سایت (یا حتی پردازه) درخواست را فرستاده و منتظر مهره است، اما به علت خطایی دیگر (مانند قطع شبکه)

هیچ‌گاه مهره دریافت نمی‌شود. (غیر عمدی)

نحوه‌ی مواجهه: مانند حالت بالا، سایت خطا را نادیده می‌گیرد.

اگر آدرس سایت بالادستی اشتباه وارد شده باشد، خطای ۱۰. رخ خواهد داد.

کارهای آینده

برای پیشرفت این پیاده‌سازی ایده‌های موجود و مسائل باز زیادی وجود دارد. برخی از ایده‌ها صرفاً به خاطر کمبود وقت پیاده‌سازی نشده‌اند.

ایده‌ها و مسائل بازی که به ذهن ما رسیده است، به این ترتیب است:

- امکان خروج یک سایت از سامانه:

در حالت کنونی امکان خروج یک سایت به صورت مدیریت شده وجود ندارد. به صورت مدیریت شده یعنی اینکه سایتی که می‌خواهد خارج شود:

  - اگر مهره را در اختیار دارد، آنرا به فرد دیگری بدهد.
  - اگر قبلاً به علت درخواست سایت‌های پایین‌تر درخواستی به مرحله‌ی بالاتر داده است، تکلیف آن را مشخص کند.
  - سایت‌های زیر دست خود را از خروج آگاه کند و سایت دیگری (مانند سایت بالاتر خود را) به عنوان سایت بالاتر به آن‌ها پیشنهاد کند.

پیاده‌سازی این مسأله مستلزم آگاهی یک سایت از سایت‌های زیر دست خود است، به این ترتیب که هر سایت هنگام شروع وجود خود را به مرحله‌ی بالاتر اطلاع دهد.

- بازگشت مهره در صورت کشته‌شدن پردازه حامل آن

همانطور که فایل‌ها و سوکت‌ها هنگام مرگ یک پردازه آزاد می‌شوند، مهره نیز در صورت مرگ پردازه به سایت باز گردد تا از گم‌شدن مهره در پردازه‌ها جلوگیری شود. (البته هنوز امکان گم‌شدن مهره مثلاً در حالت رفتن برق موجود است!)

- مشخص کردن حداکثر زمان انتظار برای مهره

برای اینکه یک سایت یا پردازه در صورت گم‌شدن مهره (به هر علت) یا قطع شدن ارتباط بلوکه نشود.

- وجود پیغام abort برای جواب رد دادن به یک درخواس

مثلاً اگر timeout داشته باشیم. بعد از سپری شدن آن باید به کسانی که به ما امید بسته‌اند پیغام رد درخواست بفرستیم.

- مشخص کردن سقف بالا برای زمان در اختیار داشتن مهره (برای حل خطای ۳.)
- اگر یک سایت نتواند مهره را به سایت دیگر منتقل کند، برای مدتی به این تلاش ادامه دهد، (آن را به صف درخواست کنندگان بازگرداند)
- ایجاد یک حالت جدید requesting برای سایت‌ها

این حالت که نشان می‌دهد سایت سعی دارد به سایت بالاتر درخواست بفرستد. وجود این حالت باعث می‌شود در

صورت بروز خطا سایت به صورت مداوم برای ارسال درخواست به مرحله‌ی بالاتر تلاش کند و البته بلوکه نشود.

- فراخوانی سیستمی یا پیام برای تغییر اشاره‌گر به سایت بالاتر (به منظور تغییر ساختار سامانه)
- بررسی معتبر بودن آدرس سایت و آدرس سایت بالاتر هنگام راه‌اندازی یک سایت (مثلاً آدرس نباید ۰.۰.۰.۰ یا ۱۲۷.۰.۰.۰ باشد یا مثلاً آزمون وجود داشتن سایت)