مبانی یونیکس و اینترنت

Eric S. Raymond

مترجم: مهدی فتاحی

بازبینی نهایی و صفحه‌آرایی: علی موسوی

ویراست: ۲/۹

۲۰۰۴/۳/۳

فهرست مطالب

۱. درآمد

۱-۱. هدف این نوشتار

مقصود این نوشتار کمک به کاربرانی است که لینوکس و اینترنت را از راه کار کردن با آن‌ها می‌آموزند. گرچه این راه بسیار خوبی برای کسب مهارت‌های جدید است، گاه باعث می‌شود خلأهای خاصی در آگاهی شخص از مفاهیم پایه‌ای بر جای بماند، این خلأها می‌توانند اندیشیدن خلاق یا برخورد کارآمد با اشکالات را مشکل کنند، چرا که شخص الگوی ذهنی درستی از آنچه واقعاً اتفاق می‌افتد ندارد.

من خواهم کوشید به زبانی روشن و ساده، ساز و کار لینوکس و اینترنت را توضیح دهم. شیوه‌ی ارائه توضیحات، مناسب کسانی است که یونیکس یا لینوکس را بر سخت‌افزارهای رده‌ی PC (رایانه‌های شخصی) به کار می‌برند.

با این حال، غالبا در اینجا فقط از «یونیکس» نام خواهم برد، زیرا آنچه توصیف خواهم کرد درسکوهای (Platforms) مختلف و همه‌ی گونه‌های یونیکس ثابت است.

من فرض خواهم کرد که شما در حال حاضر از یک رایانه‌ی شخصی اینتل استفاده می‌کنید. اگر از Alpha یا PowerPC یا رایانه‌ی دیگری استفاده می‌کنید جزئیات، اندکی متفاوت خواهند بود، اما مفاهیم پایه‌ای همین‌ها هستند.

من چیزی را تکرار نمی‌کنم، بنابراین بایستی به مطالب دقت کنید، اما این بدین معنی هم هست که از تک تک کلماتی که می‌خوانید چیزهایی خواهید آموخت. خوب است در اولین خواندن نگاهی گذرا به مطالب بیندازید؛ بعد از اینکه آنچه را که آموخته‌اید هضم کردید، می‌بایست برگردید و چند بار دیگر دوباره بخوانید.

این نوشتار پیوسته در حال تکامل است. قصد دارم در پاسخ به نظرهایی که از کاربران می‌رسد بخش‌های جدیدی به آن اضافه کنم، بنابراین بهتر است هرازگاه دوباره سری به این نوشته بزنید.

۱-۲. نسخه‌های جدید این نوشته:

نسخه‌های جدید راهنمای مبانی یونیکس و اینترنت هر چند وقت یکبار به comp.os.linux.help و news.answers ارسال خواهد شد. همچنین این نسخه‌های جدید بر روی سایت‌های گوناگون WWW و FTP، از جمله صفحه‌ی اصلی LDP قرار خواهند گرفت.

از طریق نشانی زیر می‌توانید آخرین نسخه‌ی این نوشته را روی شبکه‌ی جهانی وب مشاهده کنید.

http://www.tldp.org/HOWTO/Unix-and-Internet-Fundamentals-HOWTO/index.html

۱-۳. نظرات و اصلاحات

اگر سؤال یا نظری راجع به این نوشته دارید، می‌توانید نامه‌ی الکترونیکی خود را برای اریک اس.ریموند به نشانی esr@thyrsus.com ارسال کنید. من از هر پیشنهاد یا انتقادی استقبال می‌کنم. به ویژه از لینک‌هایی که مرا به توضیحات مفصل‌تر هر مفهوم ارجاع می‌دهند استقبال خواهم کرد. اگر اشتباهی در این نوشتار یافتید لطفاً به من اطلاع دهید تا در نسخه‌ی بعدی، آن را تصحیح کنم. با تشکر.

مترجم: پیشنهاد یا نظر اصلاحی خود را می‌توانید به نشانی mah.fat@gmail.com برای من ارسال کنید.

۱-۴. منابع مرتبط

اگر این نوشته را برای یادگیری هک کردن می‌خوانید، می‌بایست How To Become A Hacker FAQ را هم مطالعه کنید. نوشته‌ی مذکور لینک‌هایی به دیگر منابع مفید دارد.

۲. ساختار بنیادی رایانه‌ی شما

رایانه‌ی شما یک تراشه یا چیپ پردازنده (processor) در خود دارد که محاسبه یا computing حقیقی را انجام می‌دهد. این تراشه حافظه‌ای درونی دارد (همان که اهالی DOS و Windows به آن RAM و یونیکسی‌ها غالباً آن را core (به معنی مغز یا هسته) می‌نامند؛ این اصطلاح خاص یونیکس از زمانی که RAM از ferrite core یعنی حلقه‌های فریتی تشکیل می‌شد در خاطره‌ها باقی مانده است). پردازنده و حافظه روی برد اصلی یا motherboard که قلب کامپیوتر است، قرار دارند.

رایانه‌تان یک نمایشگر و یک صفحه کلید، دیسک سخت، سی‌دی رام (CD-ROM) و احتمالاً یک فلاپی دیسک دارد. دسته‌ای از این افزارها با کارت‌های کنترل‌کننده یا controller cards کار می‌کنند. این کارت‌ها به برد اصلی وصل می‌شوند و به کامپیوتر کمک می‌کنند که آن ابزارها را هدایت کند. سخت افزارهای دسته دیگر،توسط چیپ‌ست (chipset)هایی اختصاصی کار می‌کنند، که مستقیماً روی برد اصلی قرار دارند و همان وظیفه کارت‌های کنترل‌کننده را انجام می‌دهند. صفحه‌کلید شما آنقدر ساده است که نیازی به یک کارت جداگانه ندارد؛ کنترل‌کننده‌ی آن در بدنه‌ی خودش جا داده شده است.

بعداً به برخی جزئیات طرز کار این وسایل خواهیم پرداخت. فعلاً چند مسأله‌ی اساسی را درباره نحوه‌ی کار سخت‌افزارها با هم به خاطر بسپارید:

همه‌ی اجزای داخل جعبه‌ی (case) کامپیوتر شما با یک باس (bus) به یکدیگر وصل می‌شوند. باس در واقع همان چیزی است که کارت‌های کنترل‌کننده (کارت‌های ویدیو، کنترل‌کننده دیسک، و کارت صدا) را به آن وصل می‌کنید. باس شاهراه عبور اطلاعات بین پردازنده، نمایشگر، دیسک و هر چیز دیگری است که در کامپیوترتان دارید.

(اگر درباره کامپیوترهای شخصی به نام‌های «ISA» و «PCI» و «PCMCIA» برخورده‌اید و معنی آن‌ها را نفهمیده‌اید، باید بدانید این‌ها انواعی از باس هستند. ISA ، به استثنای بعضی جزئیات مختصر، همان باس استفاده شده در کامپیوترهای شخصی اولیه IBM در ۱۹۸۰ است؛ این نوع باس اکنون در حال منسوخ شدن است. PCI ، که مخفف Peripheral Component Interconnectio (یعنی ارتباط داخلی اجزای جانبی) است، همان باسی است که در بیشتر PCهای امروزی و همچنین مکینتاش‌های(Macintosh) امروزی به کار می‌رود. PCMCIA گونه‌ای ISA با اتصالات فیزیکی کوچکتر است که در کامپیوترهای قابل حمل یا laptop استفاده می‌شود.)

پردازنده، که باعث کار کردن همه‌ی چیزهای دیگر است، عملاً نمی‌تواند هیچ یک از قطعات دیگر را مستقیماً ببیند، و مجبور است با آن‌ها از طریق باس حرف بزند. تنها جزء سیستم که پردازنده به آن دسترسی واقعاً سریع و بدون واسطه دارد حافظه (core) است. به همین خاطر، برنامه‌ها برای اجرا شدن باید در core (حافظه) باشند.

وقتی کامپیوترتان برنامه یا داده‌ای را از روی دیسک می‌خواند، در واقع آنچه روی می‌دهد این است که پردازنده با استفاده از باس یک درخواست خواندن دیسک به کنترل‌کننده‌ی دیسک شما می‌فرستد. سپس کنترل‌کننده‌ی دیسک از طریق باس به پردازنده علامت می‌دهد که اطلاعات مورد نظر را خوانده و در مکان خاصی در حافظه قرار داده است. در این لحظه پردازنده می‌تواند با استفاده از باس آن داده را ببیند.

صفحه کلید و نمایشگر هم از راه باس با پردازنده ارتباط برقرار می‌کنند، اما به صورت‌های ساده‌تر. بعداً به آن‌ها خواهیم پرداخت. فعلاً آن‌قدر می‌دانید تا بتوانید بفهمید بعد از اینکه کامپیوترتان را روشن می‌کنید چه اتفاقی می‌افتد.

۳. وقتی یک کامپیوتر را روشن می‌کنید چه اتفاقی می‌افتد؟

یک کامپیوتر، بدون برنامه‌ای که در آن اجرا شود تنها توده‌ای بی‌خاصیت از قطعات الکترونیکی است. اولین کاری که یک کامپیوتر پس از روشن شدن باید انجام دهد آغاز کردن برنامه‌ای خاص به نام سیستم عامل است. کار سیستم عامل این است که با پرداختن به کار پیچیده‌ی مدیریت سخت‌افزارها به دیگر برنامه‌ها امکان اجرا شدن بدهد.

فرایند بالا آوردن سیستم عامل، booting یا راه‌اندازی نامیده می‌شود (این کلمه در اصل bootstrapping بوده و اشاره دارد به مثل عامیانه «pulling yourself up by your bootstraps») کامپیوتر شما می‌داند چگونه boot شود زیرا دستورالعمل این کار در یکی از تراشه‌های آن به نام چیپ BIOS گنجانده شده است. BIOS مخفف Basic Input/Output System است (یعنی سیستم ابتدایی ورودی/خروجی).

تراشه‌ی BIOS به کامپیوتر می‌گوید که در یک محل ثابت، معمولاً روی دیسک سختِ دارای پایین‌ترین شماره (دیسک بوت یا دیسک راه اندازی)، دنبال یک برنامه‌ی مخصوص به نام مدیر راه‌اندازی یا bootloader بگردد. (در لینوکس مدیر راه اندازی GRUB یا LILO نام دارد.) برنامه‌ی bootloader به درون حافظه کشیده شده و آغاز می‌شود. کار این برنامه، راه انداختنِ سیستم عامل واقعی است.

مدیر راه اندازی، این کار را با جستجوی یک هسته یا kernel، بارگذاری (load) آن در حافظه و آغاز کردن آن انجام می‌دهد. وقتی لینوکس را بوت می‌کنید و روی نمایشگر LILO و بعد از آن تعدادی نقطه می‌بینید، راه‌انداز در حال بارگذاری هسته است. (هر نقطه به این معنی است که LILO یکی دیگر از disk blockها یا واحد دیسک‌های کد هسته را بارگذاری کرده است.

(شاید از خود بپرسید چرا خود BIOS مستقیماً هسته را بار نمی‌کند ــ چرا کار در دو مرحله و به وسیله‌ی مدیر راه اندازی انجام می‌شود؟ خوب، چون BIOS خیلی هوشمند نیست. در واقع خیلی هم کودن است و لینوکس بعد از مرحله‌ی بوت اصلاً از آن استفاده نمی‌کند. BIOS در اصل برای کامپیوترهای ابتدایی هشت بیتی با دیسک‌های کوچک نوشته شده و در واقع نمی‌تواند به آن مقدار دیسکی که برای راه‌اندازی مستقیم هسته لازم است دسترسی پیدا کند. به علاوه مرحله‌ی مدیر راه‌اندازی به شما این امکان را می‌دهد که یکی از چند سیستم عامل پراکنده در جاهای مختلف دیسک را آغاز کنید، البته در حالت نامحتملی که یونیکس برای شما به اندازه کافی خوب نباشد.)

زمانی که هسته آغاز می‌شود، باید بگردد و بقیه سخت‌افزارها را پیدا کند، و برای اجرای برنامه‌ها آماده شود. هسته این کار را نه با سرکشی به جاهای معمولی حافظه، بلکه با وارسی درگاه‌های ورودی/خروجی یا I/O ports (نشانی‌هایی خاص در باس که احتمال دارد کارت‌های کنترل‌کننده‌ای در آنها منتظر و گوش به فرمان باشند) انجام

می‌دهد. هسته جایی را به طور اتفاقی جستجو نمی‌کند، بلکه از قبل آگاهی زیادی از اینکه چه چیزی را کجا ممکن است پیدا کند و اینکه کنترل‌کننده‌ها در صورت وجود چگونه جواب خواهند داد، در خود دارد. این عملیات را کاوش خودکار یا autoprobing می‌گویند.

بیشتر پیام‌هایی که در زمان راه‌اندازی می‌بینید مربوط به هسته است که در درگاه‌ها یا پورت‌های ورودی/خروجی جستجوی خودکار انجام می‌دهد و در می‌یابد که چه چیزهایی در دسترس دارد تا خود را با ماشین شما تطبیق دهد. هسته‌ی لینوکس این کار را بی‌نهایت خوب انجام می‌دهد، بهتر از بیشتر یونیکس‌های دیگر و بسیار بهتر از DOS یا Windows. در واقع، بسیاری از لینوکس‌کارهای قدیمی عقیده دارند مهارت لینوکس در کاوش‌ها یا probeهای هنگام بوت (که همچنین سبب شد نصب آن نسبتاً آسان شود) یکی از دلایل اصلیِ این بود که لینوکس از چارچوب آزمایش‌های آزاد یونیکسی خارج شد و انبوهی از کاربران را جذب خود کرد.

اما بار کردن کامل هسته و راه انداختن آن پایان فرایند راه اندازی نیست؛ این فقط مرحله اول است (که گاهی به آن run level 1 یعنی سطح راه‌اندازی یک می‌گویند). بعد از این مرحله، هسته اداره کارها را به یک جریان (process) مخصوص به نام init می‌سپارد که خود چندین جریان روزمره را آغاز می‌کند.

اولین کار جریان init وارسی سلامت دیسک‌های شماست. فایل‌سیستم‌های دیسک ظریف و آسیب پذیرند؛ اگر این فایل‌سیستم‌ها به خاطر یک اشکال سخت افزاری یا قطع ناگهانی برق صدمه دیده باشند منطقی است که قبل از اینکه یونیکس شما کاملاً راه انداخته شود اقداماتی برای ترمیم آنها صورت گیرد. بعداً در هنگامی که راجع به چگونگی ایجاد اختلال در فایل‌سیستم‌ها صحبت خواهیم کرد به بخشی از این موضوع خواهیم پرداخت.

گام بعدی init شروع کردن چندین برنامه پس‌زمینه یا daemon (به معنی جن، دیو، با تلفظ /dee'mn/ - م) است. یک daemon برنامه‌ای است مانند یک print spooler (برنامه‌ای که دسته ای از فایل‌ها را برای چاپ به صف می‌کند -م)، یک mail listener (برنامه ای که منتظر رسیدن نامه‌های الکترونیکی است - م)، یا یک WWW server که در پس‌زمینه کمین می‌کند و منتظر کارهایی می‌ماند تا انجام دهد. این برنامه‌های خاص غالباً باید چند در خواست یا request را که ممکن است با هم ناسازگار باشند، هماهنگ کنند. دلیل اینکه این برنامه‌ها به شکل daemon هستند این است که در اغلب موارد آسان‌تر است که یک برنامه بنویسیم که همیشه جاری و با همه‌ی درخواست‌ها (request) آشنا باشد تا این که نسخه‌های زیادی از یک برنامه داشته باشیم (که هر کدام یک درخواست را پردازش می‌کنند و همه همزمان در حال اجرا هستند) و بخواهیم از تداخل نکردن آن‌ها مطمئن شویم. مجموعه‌ی خاص daemonهایی که کامپیوتر شما شروع می‌کند ممکن است متغیر باشد، اما تقریباً همیشه شامل یک print spooler (یک daemon که نقش دربان چاپگر شما را دارد) است.

گام بعدی آماده شدن برای کاربرها است. init یک مورد از برنامه‌ای به نام getty را برای زیر نظر گرفتن میز فرمان یا کنسول شما (و احتمالاً موارد بیشتری برای پاییدن پورت‌های سریال dial-in) آغاز می‌کند.

این برنامه همان چیزی است که اعلان ورود یا login prompt را روی کنسول صادر می‌کند. هنگامی که همه daemonها و جریان‌های getty برای یک‌یکِ ترمینال‌ها آغاز شدند، ما در سطح اجرایی ۲ یا run level 2 خواهیم بود. در این مرحله شما می‌توانید وارد سیستم شده (log in) و برنامه‌ها را اجرا کنید.

اما هنوز کار ما تمام نشده است. گام بعدی شروع کردن daemonهای مختلفی است که از شبکه (networking) و دیگر سرویس‌ها پشتیبانی می‌کنند. وقتی این کار هم انجام شده باشد، در سطح اجرایی یا run level سوم هستیم و سیستم کاملاً آماده استفاده است.

۴. وقتی وارد سیستم می‌شوید (log in) چه اتفاقی می‌افتد؟

هنگامی که log in می‌کنید (یک اسم به getty می‌دهید)، هویت خود را به کامپیوتر اعلام می‌کنید. کامپیوتر هم برنامه‌ای را که (به طور منطقی) login یا ثبت ورود، نام دارد اجرا می‌کند، که این برنامه گذرواژه یا password شما را می‌گیرد و وارسی می‌کند که آیا شما اجازه‌ی استفاده از این کامپیوتر را دارید یا خیر. اگر این اجازه را نداشته باشید، به تلاش شما برای ورود به سیستم جواب رد داده می‌شود؛ و اگر داشته باشید، login پس از انجام چند کار روزمره یک مفر فرمان یا command interpreter یعنی همان shell (=پوسته) را آغاز می‌کند. (بله، getty و login می‌توانستند یک برنامه باشند. به دلایل تاریخی که ذکر آنها در اینجا ارزشی ندارد این دو از یکدیگر جدا هستند.) در اینجا کمی بیشتر راجع به آنچه سیستم، قبل از دادن shell به شما انجام می‌دهد صحبت می‌کنیم (بعداً وقتی راجع به مجوز فایل‌ها صحبت خواهیم کرد به دانستن این موارد نیاز خواهید داشت). شما با یک نام کاربری و گذرواژه خود را به سیستم معرفی می‌کنید. این نام کاربری در فایلی به نام /etc/passwd جستجو می‌شود. فایل مذکور مجموعه‌ای از سطرهای متوالی است که هر سطر یک حساب کاربری را توصیف می‌کند.

در قسمتی از هر سطر، شکل رمزی شده‌ی (encrypted) گذرواژه‌ی حسابِ مربوطه وجود دارد (گاهی این قسمت‌های رمزی شده در واقع در فایل دومی به نام /etc/shadow با مجوزهای سخت‌تر نگهداری می‌شوند، که شکستن (cracking) این گذرواژه‌ها را مشکل تر می‌کند). آنچه به عنوان گذرواژه‌ی یک حساب وارد می‌کنید دقیقا به همان صورت رمزی می‌شود، و برنامه‌ی login یکسان بودن آن‌ها را وارسی می‌کند. ایمنی این روش مبتنی بر این نکته است که گرچه رسیدن از گذرواژه‌ی اصلی به گونه‌ی رمزی آن آسان است، عکس این حالت بسیار مشکل می‌باشد. به این صورت، حتی اگر کسی بتواند نسخه رمزی شده‌ی گذرواژه‌ی شما را ببیند نخواهد توانست از حساب کاربری شما استفاده کند. (این بدین معنی هم هست که اگر شما گذرواژه خود را فراموش کنید هیچ راهی برای بازیابی آن وجود ندارد، جز اینکه آن را تغییر دهید و گذرواژه‌ی دیگری را انتخاب کنید.)

وقتی با موفقیت وارد سیستم شدید، از تمام امتیازهایی که به حساب مورد استفاده‌تان نسبت داده شده است، بهره‌مند می‌شوید. شما همچنین ممکن است به عنوان عضوی از یک گروه شناخته شوید. گروه مجموعه‌ای نامگذاری شده از کاربران است که مدیر سیستم ایجاد می‌کند. گروه‌ها می‌توانند امتیازاتی مستقل از امتیازات اعضای خود داشته باشند. هر کاربر می‌تواند عضو چندین گروه باشد. (برای جزئیات درباره‌ی نحوه کار کردن این امتیازها در یونیکس، بخش مربوط به مجوزها را ببینید.)

(توجه کنید که گرچه شما معمولاً به کاربرها و گروه‌ها با نام اشاره می‌کنید، در حقیقت آن‌ها در درون سیستم به شکل شناسه‌های عددی ذخیره می‌شوند. فایل password که به آن اشاره کردیم نام کابری شما را با یک شناسه کابری یا user ID عددی؛ و فایل /etc/group نام گروه‌ها را با شناسه‌های گروهی(group ID) عددی مطابقت

می‌دهد. فرمان‌هایی که با حساب‌ها و گروه‌ها سر و کار دارند این ترجمه را به طور خودکار انجام می‌دهند.)

سطر مربوط به حساب شما شامل فهرست خانگی‌تان یا home directory نیز هست؛ در فایل‌سیستم یونیکس این جایی است که فایل‌های شخصی کاربر قرار دارند. و بالاخره، پوسته‌ی (shell) مورد استفاده شما، یعنی مفسر فرمانی که برنامه‌ی login برای دریافت فرمان‌هایتان شروع می‌کند نیز در این سطر مشخص می‌شود.

۵. وقتی برنامه‌ها را از درون shell اجرا می‌کنید چه اتفاقی می‌افتد؟

shell (=صدف، پوسته) در یونیکس مفسر فرمان‌هایی است که شما وارد می‌کنید. آن را shell می‌نامند چون هسته‌ی سیستم عامل را در دل خود جای می‌دهد و مخفی می‌کند. یکی از ویژگی‌های مهم یونیکس این است که پوسته و هسته برنامه‌هایی جدا از هم هستند که از طریق دسته کوچکی از نداها یا پیام‌های سیستمی (system calls) با هم ارتباط برقرار می‌کنند. این ویژگی وجود چندین پوسته مختلف را برای کسانی که رابط‌های (interfaces) متفاوتی را می‌پسندند میسر می‌کند.

پوسته یا shell معمولی، prompt یا اعلان '$' را به شما می‌دهد که آن را پس از ورود به سیستم می‌بینید (مگر اینکه شما خود علامت دیگری خواسته باشید). ما درباره ترکیب کلمات در shell و چیزهای ساده‌ای که در آن روی نمایشگر می‌بینید صحبت نمی‌کنیم؛ در عوض به آنچه که در پشت صحنه از دید کامپیوتر روی می‌دهد نگاهی می‌اندازیم.

بعد از مرحله راه‌اندازی و قبل از آنکه برنامه‌ای اجرا کنید، می‌توانید درون رایانه خود باغ وحشی از جریان‌ها (processes) را تصور کنید که همه منتظرند کاری انجام دهند. همه‌ی آن‌ها منتظر رویدادها یا events هستند. یک رویداد می‌تواند فشردن یک کلید یا حرکت دادن ماوس از سوی شما باشد. و یا اگر سیستم شما به یک شبکه وصل است، آمدن یک بسته داده (data packet) از شبکه می‌تواند یک رویداد باشد.

هسته یکی از این جریان‌ها است، البته یک جریان استثنایی، چون هسته تعیین می‌کند چه مدت دیگر جریان‌های مربوط به کاربر (user processes) می‌توانند اجرا شوند، و در شرایط عادی تنها جریانی است که به سخت افزارها دسترسی مستقیم دارد. در واقع جریان‌های مربوط به کاربر مجبورند برای دریافت ورودی از صفحه‌کلید، نوشتن چیزی روی نمایشگر، خواندن از (یا نوشتن روی) دیسک و تقریباً هر کار دیگری به جز پردازش سریع بیت‌های موجود در حافظه درخواست خود را به هسته بفرستند. این درخواست‌ها را نداهای سیستمی (system calls) می‌نامند.

معمولاً همه ورودی و خروجی‌ها از طریق هسته منتقل می‌شوند تا هسته بتواند کارها را زمان‌بندی کند و جلوی تداخل جریان‌ها با یکدیگر را بگیرد. چند جریان خاص مربوط به کاربر اجازه دارند هسته را دور بزنند، که این کار معمولاً با دادن دسترسی مستقیم به درگاه‌های ورودی/خروجی انجام می‌شود. X serverها یا خدمات‌رسان‌های X (برنامه‌هایی که در بیشتر انواع یونیکس درخواست‌های دیگر برنامه‌ها را برای انجام کارهای گرافیکی روی نمایشگر مدیریت می‌کنند) مرسوم ترین نمونه چنین جریان‌هایی هستند. ولی ما هنوز به X server نرسیده ایم؛ در حال حاضر شما دارید به یک اعلان shell در یک میز فرمان متنی نگاه می‌کنید.

خود Shell صرفاً یک جریان (process) مربوط به کاربر است، و به طور خاص یک جریان ویژه‌ی استثنایی به

شمار نمی‌آید. Shell با گوش دادن به درگاه ورودی/خروجی صفحه‌کلید (از طریق هسته)، منتظر فشرده شدن کلیدها از سوی شما می‌ماند. وقتی هسته این فشارها را دریافت می‌کند، آنها را به روی نمایشگر منعکس می‌نماید. وقتی هسته یک «Enter» دریافت می‌کند سطری را که وارد کرده‌اید به shell تحویل می‌دهد. Shell سعی می‌کند مجموعه‌ی آن کلیدها را به عنوان فرمان تفسیر کند.

بیایید فرض کنیم شما «ls» را تایپ می‌کنید و کلید Enter را می‌زنید تا برنامه‌ی فهرست‌کننده‌ی یونیکس را فرا بخوانید. Shell قوانینی را که در خود دارد اعمال می‌کند و در می‌یابد که شما می‌خواهید فرمان اجرایی موجود در فایل /bin/ls را اجرا کنید. سپس یک ندای سیستمی (system call) به هسته می‌فرستد و از او می‌خواهد /bin/ls را به عنوان یک جریان خُرد یا child process شروع کند و به آن اجازه دهد از طریق هسته به نمایشگر و صفحه‌کلید دسترسی داشته باشد. سپس shell به خواب می‌رود و منتظر پایان کار ls می‌ماند.

وقتی /bin/ls کار خود را تمام می‌کند، با فرستادن یک پیام سیستمی به نام exit به هسته می‌گوید که کارش را انجام داده است. سپس هسته shell را بیدار می‌کند و به او می‌گوید که می‌تواند دوباره جریان پیدا کند. shell هم یک اعلان جدید صادر می‌کند و منتظر ورودی بعدی می‌ماند.

با این حال در زمان اجرای فرمان «ls» ممکن است چیزهای دیگری هم در حال اتفاق افتادن باشند (مجبوریم فرض کنیم که شما در حال فهرست گیری از یک شاخه (directory) بسیار طولانی هستید). شما ممکن است به یک میز فرمان یا کنسول مجازی دیگر بروید، در آنجا login کنید و مثلاً یک بازی Quake را شروع کنید. یا فرض کنید به اینترنت وصل هستید. هنگامی که /bin/ls در حال اجراست کامپیوتر شما ممکن است در حال دریافت یا فرستادن نامه‌های الکترونیکی باشد.

۶. دستگاه‌های ورودی و وقفه‌ها (interrupts) چگونه کار می‌کنند؟

صفحه‌کلید رایانه‌ی شما یک دستگاه ورودی بسیار ساده است، ساده است چون حجم کمی داده را با سرعت بسیار کم تولید می‌کند (البته در مقیاس‌های کامپیوتری). هنگامی که یک کلید را فشار می‌دهید یا رها می‌کنید، این رویداد از طریق کابل صفحه‌کلید منتقل می‌شود تا یک وقفه سخت‌افزاری یا hardware interrupt ایجاد کند.

کار سیستم عامل این است که مترصد چنین وقفه‌هایی باشد. برای هر نوع وقفه‌ی ممکن، یک مأمور وقفه یا interrupt handler وجود دارد، یعنی قسمتی از سیستم عامل که تمام داده‌های مربوط به خود (مانند ارزش کلیدهایی که شما می‌زنید و رها می‌کنید) را تا زمانی که امکان پردازش آن‌ها فراهم شود مخفی و نگهداری می‌کند. در واقع کاری که interrupt handler برای صفحه کلید شما انجام می‌دهد این است که ارزش کلیدهای زده شده را به ناحیه‌ای از سیستم در نزدیکی قسمت پایین حافظه می‌فرستد. در آنجا این داده برای زمانی که سیستم عامل به برنامه‌ای که قرار است درحال خواندن صفحه‌کلید باشد اختیار عمل می‌دهد، آماده بررسی خواهد بود.

دستگاه‌های ورودی پیچیده‌تر مثل کارت‌های دیسک یا شبکه هم به شیوه‌ای مشابهِ این کار می‌کنند. پیش‌تر، به یک کنترل‌کننده‌ی دیسک اشاره کردم که از طریق باس علامت می‌داد که یک درخواست (request) مربوط به دیسک دریافت و انجام شده است. آنچه که واقعاً اتفاق می‌افتد این است که دیسک یک وقفه (interrupt) ایجاد می‌کند. سپس مأمور وقفه‌های دیسک (disk interrupt handler) داده بازیابی شده را برای استفاده برنامه‌ای که درخواست مذکور را ارائه کرده است، در حافظه کپی می‌کند.

هر یک از انواع وقفه‌ها سطح اولویت یا priority level مربوط به خود را دارد. وقفه‌های دارای اولویت پایین (مانند رویدادهای مربوط به صفحه کلید) مجبورند پشت سر وقفه‌های با اولویت بالا (مانند تیک‌های ساعت clock ticks یا رویدادهای دیسک) منتظر بمانند. یونیکس طوری طراحی شده تا اولویت بالا را به رویدادهایی بدهد که پردازش سریع آن‌ها برای روان کار کردن سیستم ضروری است.

در پیغام‌های سیستم عامل در هنگام راه‌اندازی ممکن است چیزهایی راجع به شماره‌های IRQ ببینید. احتمالاً می‌دانید که یکی از حالت‌های معمول پیکربندی غلط سخت‌افزار این است که بخواهیم دو افزار مختلف از یک IRQ استفاده کنند، بدون اینکه دقیقاً بدانید چرا.

جواب این است. IRQ کوتاه شده «Interrupt Request» است. سیستم عامل در هنگام شروع نیاز دارد که بداند هر سخت‌افزار از چه وقفه‌های شماره‌گذاری شده‌ای استفاده خواهد کرد، تا بتواند مأمور وقفه (interrupt handler) مناسب را به هر یک از آنها نسبت دهد. اگر دو سخت افزار مختلف بخواهند از یک IRQ استفاده کنند، وقفه‌ها گاه به مأمورهای غلط فرستاده خواهند شد. در بیشتر موارد این حداقل باعث قفل شدن آن افزار می‌شود، و گاهی هم ممکن است آنقدر سیستم عامل را گیج کند که سبب بیهوشی سیستم یا خرابی ناگهانی (crash) آن شود.

۷. چگونه کامپیوتر من چند کار را همزمان انجام می‌دهد؟

در واقع کامپیوتر چنین کاری نمی‌کند.کامپیوترها تنها می‌توانند یک کار (یا جریان) را در آن واحد اجرا کنند. اما یک کامپیوتر می‌تواند کاری را که انجام می‌دهد خیلی سریع عوض کند، و انسان‌های کند را طوری فریب دهد که گمان کنند او چند کار را همزمان انجام می‌دهد. این کار را timesharing (=استفاده نوبتی، سهمیه بندی زمان) می‌گویند.

یکی از وظایف هسته، مدیریتِ سهمیه‌بندی زمان است. هسته قسمتی به نام زمان‌بند یا scheduler دارد که اطلاعاتی راجع به همه‌ی جریان‌های (غیر از هسته) موجود در «باغ وحش» شما را در خود نگهداری می‌کند. در هر یک شصتم ثانیه، یک زمان‌سنج در هسته به سر وقت تعیین شده خود می‌رسد و یک وقفه‌ی ساعتی(clock interrupt) ایجاد می‌کند. زمان‌بند هر روندی را که در جریان است متوقف و در سر جای خود معلق می‌کند، و اختیار عمل را به یک جریان دیگر می‌دهد.

یک شصتم ثانیه ممکن است زمان زیادی به نظر نرسد. اما در ریزپردازنده‌های امروزی همین زمان برای اجرای ده‌ها هزار دستور ماشین، که می‌توانند حجم زیادی کار انجام دهند، کافی است. بنابراین حتی اگر جریان‌های فراوانی وجود داشته باشند، هر کدام می‌تواند در هر نوبت کوتاهِ خود، میزان قابل توجهی از کارش را انجام دهد.

درعمل ممکن است یک برنامه تمام جیره‌ی زمانی خود را دریافت نکند. اگر یک وقفه از یک وسیله ورودی/خروجی بیاید، هسته به طور کارآمدی روند جاری را متوقف می‌کند، interrupt handler یا مأمور وقفه‌ی مربوطه را راه می‌اندازد، و بعد به سر کار قبلی برمی‌گردد. سیل وقفه‌های دارای اولویت بالا، می‌تواند جریان عادی پردازش را از بین ببرد؛ این ناهنجاری در هم کوفتگی یا thrashing نامیده می‌شود و خوشبختانه در یونیکس‌های مدرن ایجاد کردن آن بسیار سخت است.

در حقیقت سهمی که برنامه‌ها از زمان ماشین می‌توانند بگیرند بسیار به ندرت بر سرعت آن‌ها مؤثر است (چند مورد از این قاعده مستثنا هستند از قبیل صدا یا نسل گرافیک‌های سه بعدی). در اغلب موارد، تأخیر وقتی به وجود می‌آید که برنامه مجبور است منتظر داده‌ای که از یک دیسک یا اتصال شبکه می‌آید بماند.

سیستم عاملی که بتواند همواره تعداد زیادی جریان همزمان را پشتیبانی کند چند وظیفه ای یا multitasking نامیده می‌شود. خانواده‌ی سیستم عامل‌های یونیکسی از همان ابتدا برای چندوظیفه‌ای بودن طراحی شده و این کار را بسیار خوب انجام می‌دهد ــ بسیار کاراتر از Windows یا سیستم عامل Mac قدیمی، که multitasking بعداً در آن‌ها کار گذاشته شد و در این کار نسبتاً ضعیف هستند. چند وظیفه‌ای بودنِ کارآمد و قابل اطمینان، بخش عمده‌ای از آن چیزی است که لینوکس را برای کارهای شبکه، ارتباطات و سرویس‌های وب، ممتاز می‌کند.

۸. چگونه کامپیوتر من از تداخل جریان‌ها (processes) جلوگیری می‌کند؟

زمان‌بند یا scheduler هسته مسئول تقسیم‌بندی زمانیِ جریان‌هاست. علاوه بر این، سیستم عامل باید فضا را هم بین جریان‌ها تقسیم کند تا آنها نتوانند در حافظه‌ی مورد استفاده‌ی یکدیگر، تداخل کنند. حتی اگر فرض کنیم که همه‌ی برنامه‌ها سعی دارند با هم همکاری کنند، باز هم نمی‌خواهیم که یک اشکال (bug) در یکی از آن‌ها بتواند دیگر برنامه‌ها را هم دچار اشکال کند. کارهایی که سیستم عامل شما برای حل این مسأله انجام می‌دهد مدیریت حافظه یا memory management نامیده می‌شود.

هر یک از جریان‌ها در «باغ وحش» سیستم شما به فضای مخصوص به خود در حافظه نیاز دارد، تا از آنجا بتواند کد خود را اجرا کند و نیز متغیرها (variables) و نتایج خود را در آنجا نگهداری نماید. می‌توانید این فضا را ترکیبی از یک بخش کد (code segment) فقط خواندنی (حاوی دستورالعمل‌های آن جریان) و یک بخش داده (data segment) قابل نوشتن (حاوی همه متغیرهای جریان مذکور) در نظر بگیرید. بخش داده (data segment) برای هر جریان کاملاً یگانه و منحصر به فرد است، اما اگر دو جریان یک کد واحد را اجرا کنند یونیکس برای افزایش بهره‌وری به طور خودکار ترتیبی اتخاذ می‌کند که هر دو مشترکاً از یک بخش کد (code segment) استفاده نمایند.

۸-۱ . حافظه‌ی مجازی: به بیان ساده

بهره‌وری اهمیت دارد، چون حافظه گران است. بعضی وقت‌ها شما آن‌قدر حافظه در اختیار ندارید که بتواند همه‌ی برنامه‌های در حال اجرا را به طور کامل در خود نگه دارد، به خصوص اگر از یک برنامه بزرگ مثل یک X server استفاده می‌کنید. برای حل این مشکل، یونیکس از روشی به نام حافظه‌ی مجازی یا virtual memory استفاده می‌کند. یونیکس سعی نمی‌کند همه‌ی کدها و داده‌های یک جریان را در حافظه نگهداری کند. در عوض تنها یک گروه کاری (working set) نسبتاً کوچک را حفظ می‌کند، و بقیه اجزای جریان مورد نظر در یک فضای معاوضه یا swap space روی دیسک سخت باقی می‌ماند.

توجه کنید که در گذشته، «بعضی وقت‌ها»ی پاراگراف قبل «تقریباً همیشه» بود، یعنی اندازه‌ی حافظه به نسبت اندازه‌ی برنامه‌های جاری معمولاً کوچک بود و بنابراین معاوضه (swapping) مکرراً انجام می‌شد. امروزه حافظه به مراتب ارزانتر است و حتی ماشین‌های مدل پایین هم مقدار نسبتاً زیادی حافظه دارند. بر ماشین‌های تک کاربره‌ی امروزی با بیش از ۶۴ مگابایت حافظه می‌توان X و ترکیبی از دیگر برنامه‌های معمول را پس از بارگذاری اولیه‌ی آن‌ها در حافظه، بدون هیچ گونه معاوضه، اجرا کرد.

۸-۲. حافظه‌ی مجازی: به بیان مفصل

راستش را بخواهید در بخش قبلی مطالب، بیش از حد ساده بیان شدند. بله، برنامه‌ها بیشتر حافظه‌ی کامپیوتر شما را به صورت یک بانک بزرگ و گسترده از آدرس‌ها که بزرگ‌تر از حافظه‌ی فیزیکی (واقعی) است می‌بینند، و معاوضه یا swapping به این دلیل انجام می‌شود که این تصور غلط حفظ شود. اما سخت افزار شما دست کم ۵ نوع حافظه مختلف در خود دارد و وقتی لازم باشد که برنامه‌ها برای حداکثر سرعت تنظیم شوند، تفاوت‌های بین این انواع حافظه می‌تواند بسیار مهم باشد. برای آنکه واقعاً بفهمید در درون ماشین شما چه می‌گذرد باید طرز کار همه‌ی آن‌ها را بدانید.

پنج نوع حافظه‌ای که گفتیم این‌ها هستند: رجیستر پردازنده (processor registers)، حافظه موقت داخلی (internal (on-chip) cache)، حافظه موقت خارجی (external (off-chip) cache)، حافظه اصلی (main memory)، و دیسک. و دلیل وجود این تعداد انواع حافظه ساده است: سرعت، هزینه دارد. این انواع حافظه را من به ترتیبی فهرست کرده‌ام که از نظر زمان دستیابی (access time)، نزولی و از نظر قیمت، صعودی است. حافظه register سریعترین و گرانترین است و می‌تواند به صورت تصادفی یک میلیارد بار در ثانیه مورد دستیابی قرار گیرد، در حالی که دیسک کندترین و ارزانترین است و می‌تواند حدود ۱۰۰ دستیابی تصادفی یا random access در ثانیه انجام دهد.

در زیر فهرست کاملی می‌بینید که سرعت‌های یک ماشین رومیزی معمولیِ اوایل سال ۲۰۰۰ را نشان می‌دهد. با اینکه سرعت و گنجایش، افزایش و قیمت‌ها کاهش خواهند یافت، می‌توانید توقع داشته باشید که این نسبت‌ها تقریباً ثابت بمانند ــ و همین نسبت‌ها هستند که سلسله مراتب حافظه را شکل می‌دهند.

Disk
    Size: 13000MB Accesses: 100KB/sec

Main memory
    Size: 256MB Accesses: 100M/sec

External cache
    Size: 512KB Accesses: 250M/sec

Internal Cache
    Size: 32KB Accesses: 500M/sec

Processor
    Size: 28 bytes Accesses: 1000M/sec

نمی‌توانیم همه چیز را از سریعترین انواع حافظه بسازیم، چرا که این بیش از حد گران تمام خواهد شد ــ و

اینکه حافظه پرسرعت حتی اگر هم گران تمام نشود، ناپایدار است. یعنی با قطع برق تمام محتویات خود را از دست می‌دهد. به همین دلیل کامپیوترها باید دارای دیسک سخت یا گونه‌های دیگری از ذخیره‌سازهای پایدار باشند که در هنگام قطع برق اطلاعات را حفظ کند. اختلاف زیادی بین سرعت پردازنده و دیسک وجود دارد. وجود سه نوع حافظه‌ای که بین این دو قرار دارند (حافظه موقت داخلی، حافظه موقت خارجی و حافظه اصلی) اساساً برای پر کردن این فاصله است.

لینوکس و یونیکس‌های دیگر خصوصیتی به نام حافظه مجازی دارند. به این معنی که سیستم عامل طوری رفتار می‌کند که گویی حافظه‌ی اصلی آن، بسیار بیشتر از مقدار واقعی است. حافظه‌ی اصلی واقعی کامپیوتر شما به مانند دسته‌ای از پنجره‌ها یا cacheها(حافظه‌های موقت) عمل می‌کند که بر روی یک فضای حافظه‌ی بسیار بزرگتر و «مجازی» قرار می‌گیرد که بیشتر آن فضا در هر زمان، در حقیقت روی دیسک و در یک منطقه‌ی ویژه به نام فضای معاوضه یا swap space واقع است. دور از چشم برنامه‌های کاربر، سیستم عامل مرتباً قطعات بزرگ اطلاعات (که page نامیده می‌شوند) را بین حافظه و دیسک جابجا می‌کند تا این تصور حفظ شود. نتیجه‌ی نهایی این است که حافظه‌ی مجازی شما بسیار بزرگتر اما نه بیش از حد کندتر از حافظه‌ی واقعی است.

اینکه حافظه‌ی مجازی چقدر کندتر از حافظه‌ی واقعی است به این بستگی دارد که الگوریتم‌های معاوضه‌ی سیستم عامل چقدر با شیوه‌ی استفاده‌ی برنامه‌ها ازحافظه‌ی مجازی هماهنگ است. خوشبختانه خواندن‌ها و نوشتن‌های حافظه که از نظر زمانی نزدیک به هم هستند تمایل دارند در فضای حافظه نیز دسته (cluster) شوند. این گرایش را locality (همسایگی) و یا به شکل رسمی‌تر locality of reference (همسایگی ارجاع) می‌نامند، که یک ویژگی خوب است. اگر ارجاع‌های حافظه (references) به طور اتفاقی و بی نظم در فضای حافظه‌ی مجازی پراکنده می‌شدند، آنگاه شما بسیاری از اوقات برای هر ارجاع جدید مجبور بودید یک خواندن و نوشتن دیسک انجام دهید و حافظه‌ی مجازی به کندی دیسک می‌شد. اما از آنجا که برنامه‌ها عملاً همسایگی (locality) قدرتمندی از خود نشان می‌دهند، سیستم عامل می‌تواند برای هر ارجاع معاوضه‌های (swaps) نسبتاً کمی انجام دهد.

به تجربه ثابت شده است که کارآمدترین روش برای گستره‌ی وسیعی از الگوهای مصرفِ حافظه، یک روش بسیار ساده است که الگوریتم LRU یا «Least Recently Used» (= اخیراً کمتر استفاده شده یا کم کاربردترین در زمان اخیر) نامیده می‌شود. سیستم حافظه‌ی مجازی قطعات بزرگ اطلاعات را هنگامی که به آن‌ها نیاز پیدا می‌کند از روی دیسک به گروه کاری (working set) خود می‌برد. وقتی سیستم دچار کمبود حافظه‌ی واقعی برای این گروه کاری می‌شود، قطعه‌ای که اخیراً کمترین استفاده را داشته (Least Recently Used) را تخلیه می‌کند. در همه‌ی یونیکس‌ها و بیشتر سیستم عامل‌های دیگری که از حافظه‌ی مجازی استفاده می‌کنند، اختلاف اندکی درچگونگی استفاده از LRU وجود دارد.

حافظه‌ی مجازی اولین حلقه از زنجیره‌ی پر کردن فاصله بین سرعت پردازنده و سرعت دیسک است. این نوع حافظه آشکارا از سوی سیستم عامل اداره می‌شود. اما هنوز فاصله‌ی زیادی بین سرعت حافظه‌ی اصلیِ واقعی و سرعت دسترسی پردازنده به حافظه‌ی register (=حافظه اختصاصی پردازنده) وجود دارد. حافظه‌های موقت یا cacheهای داخلی و خارجی این فاصله را با روشی مشابه روش حافظه‌ی مجازی که پیشتر توصیف شد، پر می‌کنند. همانطورکه حافظه‌ی فیزیکی اصلی مانند دسته‌ای از پنجره‌ها یا cacheها بر روی فضای معاوضه‌ی دیسک عمل می‌کند، حافظه‌ی موقت خارجی مانند پنجره‌ای روی حافظه‌ی اصلی کار می‌کند. حافظه‌ی موقت خارجی سریعتر (250 میلیون دسترسی در ثانیه، به جای 100 میلیون) و کوچکتر است. سخت‌افزار کامپیوتر (به طور خاص کنترل‌کننده‌ی حافظه) الگوریتم LRU را بر قطعات بزرگ اطلاعات که از حافظه‌ی اصلی آورده می‌شود اعمال می‌کند. به دلایل تاریخی واحد معاوضه‌ی حافظه‌ی موقت، به جای صفحه (page)، خط (line) نامیده می‌شود.

اما این هم برای ما کافی نیست. حافظه‌ی موقت داخلی، با مخفی (cache) کردن قسمت‌هایی از حافظه‌ی موقت خارجی، آخرین گام برای مدیریت کارآمد سرعت را بر می‌دارد. با این حال این حافظه سریعتر و کوچکتر است – در واقع حافظه‌ی موقت داخلی مستقیماً روی چیپ پردازنده قرار دارد.

اگر می‌خواهید برنامه‌های خود را واقعاً سریع کنید، دانستن این جزئیات مفید خواهد بود. برنامه‌های شما وقتی سریعتر می‌شوند که همسایگی (locality) قویتری داشته باشند، زیرا همسایگی قوی باعث می‌شود ذخیره‌سازی موقت (caching) بهتر انجام شود. بنابراین ساده‌ترین راه برای سریعتر کردن برنامه‌ها، کوچک کردن آنهاست. اگر یک برنامه با ورودی/خروجی‌های مکرر دیسک یا انتظارهای طولانی برای رویدادهای شبکه کند نشود، معمولاً با سرعتی معادل کوچکترین حافظه‌ی موقتی که می‌تواند در آن جای گیرد اجرا خواهد شد.

اگر نمی‌توانید کل برنامه را کوچک کنید، تلاش کنید بخش‌هایی از برنامه را که سرعت برای آنها حیاتی است طوری تنظیم کنید که همسایگی قویتری داشته باشند. جزئیات روش‌های انجام چنین کاری فراتر از موضوع این درس‌نامه است؛ زمانی که به این روش‌ها نیاز داشته باشید، آن‌قدر با یک کامپایلر آشنا شده‌اید که بسیاری از آنها را خودتان پیدا کنید.

۸-۳. واحد مدیریت حافظه

حتی هنگامی که آن‌قدر حافظه‌ی فیزیکی در اختیار دارید که نیازی به معاوضه نداشته باشید، آن قسمت از سیتم عامل که مدیر حافظه (memory manager) نامیده می‌شود، باز هم کارهای مهمی برای انجام دادن دارد. مدیر حافظه باید مراقب باشد تا برنامه‌ها فقط بخش داده (data segment) مربوط به خود را دستکاری کنند – یعنی باید از این‌که کدهای غلط یا مخرب در یک برنامه، اطلاعات دیگر برنامه‌ها را هم تخریب کند، جلوگیری نماید. برای این منظور، مدیر حافظه جدولی از بخش‌های داده و بخش‌های کد دارد. هرگاه یک جریان حافظه‌ی بیشتری

درخواست کند یا (معمولاً هنگام خروج) مقداری از حافظه را آزاد کند، این جدول تغییر می‌کند.

کاربرد این جدول فرستادن فرمان‌هایی به یک بخش مخصوص سخت‌افزاری به نام MMU یا واحد مدیریت حافظه است. چیپ‌های پردازنده‌های امروزی، MMU را در درون خود دارند. MMU این توانایی را دارد که حفاظ‌هایی دور برخی بخش‌های حافظه تشکیل دهد، به این ترتیب ارجاع‌های خارج از محدوده‌ی مجاز رد و پس زده می‌شود و سپس یک وقفه (interuupt) خاص ایجاد می‌گردد.

هرگاه یونیکس پیغامی مانند «Segmentation fault» یا «core dumped» یا چیزی شبیه آن نشان دهد، دقیقاً آنچه گفتیم روی داده است. یعنی تلاش برنامه‌ی جاری برای دسترسی به حافظه‌ی (هسته) فراتر از بخش (segment) مخصوص خود، باعث ایجاد یک وقفه‌ی مخرب (fatal interrupt) شده‌است. این نشان‌دهنده‌ی یک اشکال در کد برنامه است، و خروجی‌ای که برجای می‌ماند اطلاعاتی است که برای عیب‌یابی به برنامه‌نویس کمک می‌کند.

به جز جداسازیِ حافظه‌ی مورد استفاده‌ی جریان‌ها، از جنبه‌ی دیگری هم باید از تداخل جریان‌ها جلوگیری کرد. باید بتوانیم دسترسی جریان‌ها به فایل‌ها را هم مدیریت کنیم تا برنامه‌های ناقص یا مخرب نتوانند به قسمت‌های مهم سیستم آسیب برسانند. بدین منظور یونیکس از مجوزهای فایل یا file permissions استفاده می‌کند که به آن خواهیم پرداخت.

۹. چگونه کامپیوتر من چیزها را در حافظه ذخیره می‌کند؟

احتمالاً می‌دانید که همه چیز در کامپیوتر به صورت رشته‌های بیت (=اعداد دوتایی؛ می‌توانید آنها را تعداد زیادی سوییچ خاموش-روشن تصور کنید) ذخیره می‌شود. در این قسمت توضیح خواهم داد که چگونه از این بیت‌ها برای نشان دادن حروف و اعدادی که کامپیوتر پردازش می‌کند استفاده می‌شود.

قبل از ورود به این بحث، شما باید با مفهوم اندازه‌ی کلمه یا word size کامپیوتر خود آشنا باشید. اندازه‌ی کلمه عبارت از اندازه‌ای است که کامپیوتر برای جابجایی واحدهای اطلاعات ترجیح می‌دهد؛ به بیان فنی اندازه‌ی کلمه در واقع پهنای رجیسترهای پردازنده‌ی کامپیوتر است. این رجیسترها، مناطقی هستند که پردازنده، محاسبات منطقی و ریاضی خود را در آنها انجام می‌دهد. وقتی از اندازه‌ی بیت یا bit size کامپیوترها سخن گفته می‌شود(مثلاً می‌گویند کامپیوتر ۳۲بیتی یا ۶۴بیتی) منظور همین است.

بیشتر کامپیوترها (از جمله 386، 486، و کامپیوترهای Pentium) اندازه‌ی کلمه‌ی ۳۲بیتی دارند. ماشینهای 286 قدیمی، ۱۶بیتی بودند. کامپیوترهای mainframe قدیمی اغلب اندازه کلمه ۳۶ بیتی داشتند. پردازنده‌های Opteron شرکت AMD و Itanium شرکت Intel و نیز Alpha محصول شرکت DEC سابق و Compaq فعلی کلمات ۶۴ بیتی دارند.

کامپیوتر حافظه را به شکل رشته‌ای از کلمات می‌بیند که از صفر تا عدد بزرگی (که مقدار آن به اندازه‌ی حافظه‌ی شما بستگی دارد) شماره‌گذاری شده‌اند. این عدد به اندازه‌ی کلمه کامپیوتر شما محدود می‌شود، و به همین دلیل است که برنامه‌ها روی کامپیوترهای قدیمی مانند ۲۸۶ها باید برای نشان کردن مقادیر زیاد حافظه، شکنجه‌ی دردناکی را تحمل می‌کردند. من این مشکلات را در اینجا توصیف نمی‌کنم زیرا هنوز برنامه‌نویس‌های قدیمی‌تر را دچار کابوس می‌کنند.

۹-۱. اعداد

اعداد کامل یا integer را، بسته به اندازه کلمه پردازنده، با کلمات یا با جفت‌های کلمات نشان می‌دهند. معمول‌ترین شکل نمایش یک عدد کامل، یک کلمه‌ی ماشین ۶۴بیتی است.

حساب اعداد کامل، نزدیک به مبنای دو در ریاضی است اما در واقع با آن یکی نیست. پایین‌ترین بیت ۱ است و بعد از آن ۲ و ۴ و ... به صورت دودویی محض قرار می‌گیرند. اما اعداد علامت‌دار (signed numbers) به صورت نشانه‌های متمم ۲ نمایش داده می‌شوند. بالاترین بیت، بیتِ علامت یا sign bit است که مقدار را منفی می‌کند، و هر عدد منفی را می‌توان با وارونه کردن همه‌ی بیت‌ها و بعد اضافه کردن یک بیت، از عدد مثبت متناظر با آن به دست آورد. به همین دلیل است که اعداد کامل در ماشین‌های ۶۴ بیتی از $-۲^{۶۳}$ تا $۲^{۶۳}-۱$ امتداد دارند.

بیت ۶۴ام برای نشانه استفاده می‌شود؛ 0 به معنی یک عدد مثبت یا صفر است، و 1 یک عدد منفی است.

بعضی زبان‌های کامپیوتری به شما اجازه استفاده از حساب بدون علامت یا unsigned arithmetic را هم می‌دهند که مبنای دوی صریح، و فقط شامل صفر و اعداد مثبت است.

بیشتر پردازنده‌ها و بعضی زبان‌ها می‌توانند عملیات خود را با اعداد اعشاری یا floating-point numbers انجام دهند (این قابلیت در تمام چیپ‌های پردازنده‌ی جدید وجود دارد). اعداد اعشاری در مقایسه با اعداد کامل گستره‌ی بسیار وسیعتری از اعداد را در اختیار شما می‌گذارند و بیان کسرها را هم ممکن می‌سازند. ساز و کارهای انجام این عملیات متنوع هستند و پیچیده‌تر از آن‌اند که بتوانیم در اینجا به تفصیل از آنها بحث کنیم، اما به طور کلی این ساز و کارها شبیه «نمادگذاری علمی» یا scientific notation است، مثلاً وقتی می‌گوییم $1.234 * 10^{23}$ رمزگذاری این عدد به یک مانتیس (mantissa) برابر با 1.234 و یک نما برابر با 23 برای ضریب توان ده، تقسیم می‌شود (که به این معنی است که عدد حاصلضرب، 20 صفر خواهد داشت، یعنی 23 منهای 3 مکان دهتایی).

۹-۲. کاراکترها

کاراکترها را معمولاً به صورت رشته‌های ۷ بیتی و با رمزگذاری خاصی به نام ASCII (کد استاندارد آمریکا برای تبادل اطلاعات) نشان می‌دهند. در کامپیوترهای امروزی هر یک از ۱۲۸ کاراکتر ASCII هفت بیت پایینیِ یک اکتت (octet) یا بایت ۸ بیتی است. اکتت‌ها به شکل کلمات حافظه (memory words) بسته‌بندی می‌شوند تا مثلاً یک رشته ۶ کاراکتری فقط دو کلمه‌ی حافظه را اشغال کند. برای دیدن جدول کدهای ASCII در خط فرمان یونیکس فرمان «man 7 ascii» را وارد کنید.

پاراگراف قبلی از دو جنبه گمراه‌کننده بود. جنبه‌ی کم‌اهمیت‌تر این است که اصطلاح اُکتت از نظر فنی صحیح است ولی بسیار به ندرت به کار می‌رود؛ و بیشتر افراد به جای اکتت، بایت را به کار می‌برند و گمان می‌کنند هر بایت ۸ بیت است. اگر بخواهیم قاطعانه صحبت کنیم، اصطلاح «بایت» عام‌تر است؛ در گذشته کامپیوترهای ۳۶ بیتی با بایت‌های ۹ بیتی وجود داشتند (گرچه احتمالاً دیگر هرگز وجود نخواهند داشت).

اما جنبه‌ی مهم‌تر این است که همه‌ی دنیا از استاندارد ASCII استفاده نمی‌کنند. در واقع بسیاری از مردم دنیا نمی‌توانند از آن استفاده کنند چون ASCII گرچه برای انگلیسی آمریکایی مناسب است، بسیاری از کاراکترها و علائم خاص مورد نیاز برای زبانهای دیگر را ندارد. حتی انگلیسی بریتانیایی برای استفاده از ASCII با مشکل نبود علامت واحد پول خود یعنی پوند روبروست.

تلاشهایی چند برای حل این مشکل انجام شده است. همه این روشها از بیت بالایی اضافی (که ASCII استفاده نمی‌کند) استفاده می‌کنند و ASCII را به نیمه پایینی یک character set یا نظام کاراکتری ۲۵۶تایی تبدیل می‌نمایند. پرکاربردترینِ این روشها، همان نظام کاراکتری Latin-1 (یا به اصطلاح رسمی ISO 8859-1) است. این

نظام کاراکتری اصلی لینوکس، HTML و X است. ویندوز مایکروسافت از گونه متغیری از Latin-1 استفاده می‌کند و تعدادی کاراکتر مانند علامت‌های نقل قول چپ و راست را هم در جاهایی که Latin-1 به دلایل تاریخی خالی گذاشته اضافه کرده است (برای خواندن گزارش غم‌انگیزی از مشکلاتی که به این دلیل به وجود می‌آید صفحه demoroniser را مشاهده کنید).

Latin-1 با زبان‌های اروپای غربی شامل انگلیسی، فرانسوی، آلمانی، اسپانیایی، ایتالیایی، هلندی، نروژی، سوئدی و دانمارکی سازگار است. اما این هم کافی نیست و به همین خاطر دسته‌ی دیگری از Latin-2 تا Latin-9 وجود دارد تا زبان‌هایی مانند یونانی، عربی، عبری، اسپرانتو و صربو-کروات را پوشش دهد. برای جزییات بیشتر صفحه‌ی ISO alphabet soup را ببینید.

اما راه حل قطعی این مشکل، استاندارد عظیمی به نام یونیکد Unicode (و دوقلوی همسان آن یعنی SO/IEC 10646-1:1993 است). یونیکد در ۲۵۶ خانه پایینی خود با Latin-1 یکسان است. در بالای این کاراکترها در فضای ۱۶بیتی زبان‌های یونانی، سیریلی (Cyrillic)، ارمنی، عبری، عربی، دوانگری (Devanagari)، بنگالی، گرموکی (Gurmukhi)، گجراتی، اوریا (Oriya)، تامیل، تلوگو، Malayalam، Kannada، Telugu، تایلندی، لائو، گرجی، تبتی، کانای ژاپنی، مجموعه‌ی کامل هانگول مدرن کره‌ای و یک مجموعه‌ی جامع از نشانه‌های چینی-ژاپنی-کره‌ای (CJK) قرار دارند.

۱۰. چگونه کامپیوتر چیزها را روی دیسک ذخیره می‌کند؟

وقتی به دیسک سخت یک کامپیوتر تحت یونیکس نگاه می‌کنید ساختار درخت‌مانندی از فهرست‌ها و فایل‌های نامگذاری شده می‌بینید. معمولاً نیازی نیست که عمیق‌تر به این ساختار نگاه کنید، اما اگر این کار را بکنید در صورت بروز اشکال در دیسک سخت، دانشی که از ساختار درونی دیسک دارید می‌تواند در نجات فایل‌ها به شما کمک کند. متأسفانه راه خوبی برای توصیف سازمان دیسک از سطح فایل به پایین وجود ندارد، بنابراین ناچارم این ساختار را از سطح سخت‌افزار به بالا توصیف کنم.

۱۰-۱. ساختار زیرین دیسک و فایل‌سیستم

رویه‌ی دیسک سخت، یعنی همان جایی که اطلاعات انبار می‌شوند، به شکل یک تخته‌ی دارت بخش‌بندی شده است – یعنی شیارهایی دایره‌ای روی آن قرار دارند و خطوط مستقیمی آنها را به شکل پای (pie) قطع می‌کنند و قطاع‌ها (sector) را می‌سازند. از آنجا که شیارهای نزدیک به لبه‌ی خارجی فضای بیشتری نسبت به شیارهای نزدیک به مرکز دیسک دارند، شیارهای خارجی قطعات یا sectorهای بیشتری دارند. همه قطاع‌ها (یا بلوک‌های دیسک disk blocks) اندازه‌ی یکسانی دارند که در یونیکس‌های امروزی این اندازه معمولا یک K دودویی (یعنی ۱۰۲۴ کلمه‌ی ۸ بیتی) است. هر قطعه‌ی دیسک یک نشانی یا شماره‌ی منحصر به فرد دارد که به آن disk block number می‌گویند.

یونیکس دیسک سخت را به چند اطاقک یا پارتیشن (partition) تقسیم می‌کند. هر پارتیشن گستره‌ای ممتد از قطعه‌های دیسک است که مُجزا از دیگر پارتیشن‌ها، به عنوان یک فایل‌سیستم یا فضای معاوضه (swap space) به کار گرفته می‌شود. دلایل اولیه برای به وجود آمدن پارتیشن‌ها مربوط به بازیابی اطلاعات و رفع خرابی دیسک بود زیرا در آن زمان دیسک‌ها بسیار کندتر و آسیب‌پذیرتر بودند و مرزهای بین پارتیشن‌ها باعث می‌شد بخش کوچکتری از دیسک در هنگام بروز خطا در یک نقطه خراب یا غیر قابل دسترسی شود. امروزه اهمیت پارتیشن‌ها در آن است که می‌توان آنها را فقط خواندنی یا read-only کرد (که امکان دستکاری مزاحمان در فایلهای مهم سیستم را سلب می‌کند) یا از طُرق گوناگون، در شبکه به اشتراک گذاشت (در اینجا به این راه‌ها نمی‌پردازیم). پارتیشن دارای پایین‌ترین شماره غالباً متمایز از بقیه پارتیشن‌هاست زیرا از آن می‌توان به عنوان پارتیشن راه‌اندازی یا boot partition استفاده کرد و یک هسته (kernel) قابل راه‌اندازی در آن جای داد.

هر پارتیشن یا فضای معاوضه (swap space) است (برای ایجاد حافظه مجازی) و یا یک فایل‌سیستم است که فایلهایی را درون خود نگهداری می‌کند. پارتیشن swap صرفاً به عنوان یک توالی خطی از قطعات دیسک به کار گرفته می‌شود. اما فایل سیستم‌ها به روشی برای ارتباط دادن (mapping) نام فایل‌ها با توالی‌های قطعات دیسک

نیاز دارند. از آنجا که فایل ها با گذشت زمان کوچک و بزرگ می‌شوند و تغییر می‌کنند، قطعات داده‌ایِ یک فایل یک توالی خطی نخواهند بود و ممکن است در سراسر پارتیشن پراکنده باشند (یعنی در هر جایی که سیستم عامل در هنگام نیاز بتواند یک قطعه‌ی خالی پیدا کند). این حالت پخش شدن قطعات داده در نقاط مختلف دیسک را پراکندگی یا fragmentation می‌گویند.

۱۰-۲. نام فایل‌ها و دایرکتوری‌ها

در یک فایل‌سیستم ارتباط دادن (mapping) نام‌ها به قطعه‌های دیسک به کمک ساختاری به نام i-node انجام می‌شود. در قعر هر فایل‌سیستم استخری از i-nodeها وجود دارد (پایین‌ترین i-nodeها برای انجام کارهای روزمره و برچسب زدن به کار می‌روند که ما در اینجا به آنها نخواهیم پرداخت). هر i-node معرف یک فایل است. قطعات داده‌ای فایل‌ها (شامل دایرکتوری‌ها) در بالای i-nodeها (یعنی در قطعه‌های دارای شماره‌ی بالاتر) جای دارند.

هر i-node حاوی فهرستی از شماره‌های قطعات دیسک در فایل مربوطه است. (در واقع این نیمی از حقیقت است و فقط در مورد فایل‌های کوچک صدق می‌کند، اما بقیه جزییات در اینجا اهمیت چندانی ندارد). توجه کنید که i-node حاوی نام فایل نیست.

نام‌های فایل‌ها در ساختار دایرکتوری یا directory structure قرار دارند. ساختار دایرکتوری صرفاً نام‌ها را به شماره‌های i-node، مرتبط (map) می‌کند. به همین دلیل در یونیکس یک فایل می‌تواند چند نام واقعی (یا پیوند سخت یا hard link) داشته باشد؛ این نامها صرفاً چند مدخل مختلف در دایرکتوریها هستند که اتفاقاً به یک i-node اشاره می‌کنند.

۱۰-۳. نقاط سوارسازی (mount points)

در ساده‌ترین حالت تمام فایل‌سیستم یونیکس در یک پارتیشن قرار می‌گیرد. گرچه این حالت در بعضی از سیستم‌های یونیکس شخصی و کوچک وجود دارد، حالتی غیر معمول است. معمولاً فایل‌سیستم در چند پارتیشن که ممکن است حتی در دیسک‌های سخت مجزا قرار داشته باشند گسترده می‌شود. بنابراین سیستم شما مثلاً می‌تواند یک پارتیشن کوچک حاوی هسته (kernel)، یک پارتیشن کمی بزرگ‌تر حاوی ابزارهای سیستم عامل، و یک پارتیشن خیلی بزرگ‌تر برای دایرکتوری‌های خانگی کاربران (home directories) داشته باشد.

تنها پارتیشنی که بلافاصله بعد از راه‌اندازی سیستم به آن دسترسی دارید پارتیشن ریشه یا root partition است، که (تقریباً همیشه) همان پارتیشنی است که شما از آن، سیستم را راه‌اندازی (boot) کرده‌اید. در این پارتیشن دایرکتوری ریشه یا اصلیِ فایل‌سیستم قرار دارد. این دایرکتوری بالاترین گره (node) است و همه‌ی چیزهای دیگر

از آن منشعب می‌شوند.

برای آنکه فایل‌سیستم به طور کامل و با همه پارتیشن‌هایش در دسترس شما باشد، بقیه پارتیشن‌ها باید به این دایرکتوری ریشه متصل شوند. تقریباً در میانه‌ی راه‌اندازی سیستم، یونیکس این پارتیشن‌های غیر ریشه را قابل دسترسی می‌کند. یونیکس هر یک از این پارتیشن‌ها را در یک دایرکتوری در پارتیشن ریشه سوار یا mount می‌کند. برای مثال اگر شما یک دایرکتوری یونیکسی به نام usr/ داشته باشید احتمالاً این دایرکتوری یک نقطه سوارسازی یا mount point است که به پارتیشنی حاوی برنامه‌های متعدد اشاره می‌کند. برنامه‌های موجود در usr/ به همراه یونیکس نصب می‌شوند اما در هنگام راه‌اندازی اولیه‌ی سیستم، مورد نیاز نیستند.

۱۰-۴. چگونه یک فایل جستجو می‌شود؟

اکنون می‌توانیم فایل‌سیستم را از بالا به پایین ببینیم. وقتی فایلی (مانند home/esr/WWW/ldp/fundamentals.xml/) را باز می‌کنید آنچه روی می‌دهد این است:

هسته از ریشه‌ی فایل‌سیستم یونیکس شما شروع می‌کند. در آنجا دنبال یک دایرکتوری به نام home می‌گردد. معمولاً home نقطه سوارسازی یک پارتیشن کاربر (user partition) بزرگ و مجزا است، پس هسته به آنجا می‌رود. در بالاترین دایرکتوری آن پارتیشن هسته به دنبال مدخلی به نام esr می‌گردد و یک شماره‌ی i-node استخراج می‌کند. سپس به آن i-node می‌رود، و می‌بیند که قطعات داده مرتبط به آن i-node یک ساختار دایرکتوری تشکیل داده‌اند و سپس www را جستجو می‌کند. بعد از اینکه آن i-node را هم استخراج کرد به زیردایرکتوری متناظر با آن می‌رود و ldp را جستجو می‌کند. این کار هسته را باز هم به یک i-node دایرکتوری دیگر می‌برد. بعد از بازکردن آن i-node هسته یک شماره i-node برای fundamentals.xml می‌یابد. این i-node یک دایرکتوری نیست بلکه حاوی فهرست قطعات دیسک مربوط به فایل fundamentals.xml است.

۱۰-۵. مالکیت، مجوزها و امنیت فایل‌ها

برای جلوگیری از دستکاری تصادفی یا خرابکارانه‌ی داده‌ها توسط برنامه‌ها، یونیکس از مجوزهای دسترسی یا permissions استفاده می‌کند. مجوزهای دسترسی در اصل برای پشتیبانی از اشتراک زمانی چندکاربره یا timesharing طراحی شد تا چند کاربر را که از یک کامپیوتر استفاده می‌کردند، از یکدیگر محافظت کند. این مربوط به زمانی بود که یونیکس عمدتاً در مینی‌کامپیوترهای اشتراکی گران‌قیمت استفاده می‌شد.

برای درک مجوزها شما باید توضیحات مربوط به کاربران و گروهها در بخش «وقتی وارد سیستم می‌شوید (log in) چه اتفاقی می‌افتد؟» را به یاد داشته باشید. هر فایل، یک کاربرِ مالک و یک گروهِ مالک دارد. این کاربر و گروه مالک در ابتدا مربوط به ایجادکننده‌ی فایل هستند و می‌توان آنها را با برنامه‌های chown و chgrp تغییر داد.

مجوزهای اولیه‌ای که می‌توان به یک فایل نسبت داد read (مجوز خواندن اطلاعات از فایل)، write (مجوز تغییردادن فایل) و execute (مجوز اجرای فایل به عنوان یک برنامه) هستند. هر فایل سه گروه مجوز دارد: یک مجوز برای کاربر مالک، یکی برای کاربران موجود در گروه مالک و یکی هم برای دیگران. در هنگام ورود به سیستم شما فقط اجازه خواندن، نوشتن و اجرای فایلهایی را دارید که در آنها بیت‌های مجوز با شناسه‌ی (ID)کاربری شما یا گروهی که شما عضو آن هستید همخوانی داشته باشد. و البته می‌توانید فایل‌هایی را که برای همه قابل دسترس شده‌اند را ببینید.

برای آنکه ببینیم که مجوزها چگونه کار می‌کنند و یونیکس چگونه آنها را نمایش می‌دهد، بیایید به فهرستی از فایلها در یک سیستم یونیکسِ فرضی نگاهی بیندازیم. مثلاً این:

```
snark:~$ ls -l notes
-rw-r--r--    1 esr       users     2993         Jun 17 11:00 notes
```

این یک فایل داده‌ای (data file) ساده است. اطلاعات فوق به ما می‌گوید که این فایل متعلق به کاربری به نام esr است و گروه مالکِ users آن را ایجاد کرده است. احتمالاً دستگاهی که ما با آن کار می‌کنیم همه‌ی کاربران معمولی را به صورت پیش‌فرض در این گروه قرار می‌دهد؛ در کامپیوترهای اشتراکی timesharing شما معمولاً گروههای دیگری مانند staff و admin و wheel می‌بینید (به دلایل واضح گروه‌ها در ایستگاه‌های کاری تک‌کاربره و PCها خیلی مهم نیستند). یونیکس شما ممکن است از گروه دیگری به عنوان پیش‌فرض استفاده کند که احتمالاً هم‌نام با شناسه کاربری شماست.

عبارت -rw-r--r-- نشانگر بیت‌های مربوط به مجوزهای فایل است. اولین خط تیره محل بیت دایرکتوری است، اگر این فایل یک دایرکتوری بود بجای این خط تیره d و اگر فایل یک پیوند نمادین (symbolic link) بود بجای آن l (حرف L) قرار می‌گرفت. بعد از این خط تیره، سه مکان اول مجوزهای کاربر، سه مکان دوم مجوزهای گروه و سه مکان آخر مجوزهای دیگران است (که آنرا غالباً با نام world permissions می‌شناسند). برای این فایل، کاربر مالک یعنی esr اجازه خواندن و نوشتن (=تغییر) فایل را دارد، بقیه افراد گروه users میتوانند آن را بخوانند و بقیه مردم هم اجازه‌ی خواندن آنرا دارند. این یک دسته‌ی مجوز (permission set) معمول برای یک فایل داده‌ای عادی است.

حال بیایید نگاهی به فایلی با مجوزهایی بسیار متفاوت از مثال قبل بیندازیم. این فایل GCC یا کامپایلر C گنو است.

```
snark:~$ ls -l /usr/bin/gcc
-rwxr-xr-x   3 root   bin     64796  Mar 21 16:41  /usr/bin/gcc
```

این فایل به کاربری به نام root و گروهی به نام bin تعلق دارد، تنها root حق نوشتن (=تغییر) آن را دارد اما هر کسی می‌تواند آن را بخواند یا اجرا کند. این حالتی معمول برای وضعیت مالکیت و مجوزهای یک فرمان سیستمیِ از پیش نصب شده است. گروه bin در برخی یونیکس‌ها وجود دارد تا فرمان‌های سیستمی را در خود جای دهد (نام bin یک نام تاریخی و کوتاه‌شده‌ی binary به معنی دوتایی است). یونیکسی که شما از آن استفاده می‌کنید ممکن است به جای گروه bin از گروهی به نام root استفاده کند (که البته با کاربر rootیکسان نیست!).

root (=ریشه) نام سنتی کاربر دارای شناسه‌ی عددی صفر است. root کاربری ویژه و دارای امتیازات زیاد است که می‌تواند همه مجوزها و امتیازات دیگر را نادیده بگیرد. دسترسی به عنوان root، مفید ولی خطرناک است، در هنگامی که به عنوان root وارد سیستم شده‌اید یک اشتباه تایپی ممکن است فایل‌های بسیار مهم سیستمی را دچار اختلال کند در حالی که همان فرمان اگر از جانب یک کاربر معمولی صادر شود نمی‌تواند این فایل‌ها را تغییر دهد.

از آنجا که حساب کاربری root بسیار قدرتمند است دسترسی به آن را باید به دقت محدود کرد. گذرواژه (password) کاربر root مهمترین و حیاتی‌ترین اطلاعات امنیتی مربوط به سیستم شماست و همان چیزی است که نفوذی‌ها (crackers) و مزاحمان سعی دارند به دست بیاورند.

درباره گذرواژه‌ها به یاد داشته باشید: هرگز کلمه عبور خود را روی کاغذ ننویسید، گذرواژه‌هایی که به راحتی می‌شود آنها را حدس زد – مانند اسم کوچک دوست‌دختر، دوست‌پسر یا همسر را انتخاب نکنید. این رویه‌ی غلط، به طرز حیرت‌آوری معمول است و به نفوذگرها کمک بسیار می‌کند. به طور کلی هیچ کلمه‌ای را که در فرهنگ لغت موجود باشد انتخاب نکنید زیرا برنامه‌هایی به نام نفوذگر لغت‌نامه‌ای یا dictionary crackers وجود دارند که با کمک فهرستی از لغت‌های رایج، گذرواژه‌های احتمالی را جستجو می‌کنند. یک ترفند خوب، استفاده از ترکیبی از یک کلمه، یک رقم و پس از آن یک کلمه‌ی دیگر مانند «shark6cider» یا «jump3joy» است، در چنین حالت‌هایی فضای جستجو برای dictionary crackerها بیش از اندازه بزرگ می‌شود. اما از مثال‌هایی که در اینجا زدم استفاده نکنید زیرا نفوذگرها ممکن است پس از خواندن این راهنما آنها را به لغت‌نامه‌های خود اضافه کنند.

حال بیایید به نمونه سومی نگاه کنیم:

```
ark:~$ ls -ld ~
drwxr-xr-x   89  esr       users      9216    Jun 27 11:29  /home2/esr
snark:~$
```

این فایل یک دایرکتوری است (به «d» در اولین خانه مجوزها دقت کنید). همانطور که می‌بینید تنها esr حق

نوشتن (تغییر) دارد اما اجازه‌ی خواندن و اجرا به همه داده شده است.

مجوز خواندن به شما امکان فهرست کردن (لیست کردن) دایرکتوری را می‌دهد یعنی می‌توانید نام فایل‌ها و دایرکتوری‌های درون آن را ببینید. مجوز نوشتن، ایجاد و حذف فایلها را در این دایرکتوری برای شما ممکن می‌سازد. اگر به یاد داشته باشید که در این دایرکتوری فهرستی از نام‌های فایل‌ها و زیردایرکتوریهای موجود در آن وجود دارد این قوانین برای شما معنادار می‌شوند.

داشتن مجوز اجرا در یک دایرکتوری به این معنی است که شما می‌توانید وارد دایرکتوری شوید تا فایل‌ها و دایرکتوری‌های موجود در آن را باز کنید. در عمل این مجوز به شما امکان دسترسی به i-node‌های موجود در دایرکتوری را می‌دهد. یک دایرکتوری که مجوز اجرا برای آن کاملاً مسدود شده باشد بی‌استفاده خواهد شد.

گاهی به دایرکتوری‌هایی برمیخورید که برای همه (world) قابل اجرا است اما همه اجازه خواندن آن را ندارند؛ این یعنی یک کاربر می‌تواند به فایلها و دایرکتوری‌های زیر آن دسترسی پیدا کند فقط در صورتی که نام دقیق آن‌ها را بداند (دایرکتوری قابل لیست کردن نیست).

یک نکته‌ی مهم را باید به یاد داشته باشید: مجوزهای خواندن، نوشتن و اجرا برای یک دایرکتوری، مستقل از مجوزهای فایل‌ها و دایرکتوری‌های درون آن است. به طور خاص، اجازه‌ی نوشتن در یک دایرکتوری به معنی امکان ایجاد فایل‌های جدید و حذف فایل‌های موجود در آن است اما به طور خودکار به شما اجازه‌ی تغییر فایل‌های موجود را نمی‌دهد.

در آخر نگاهی هم به مجوزهای برنامه login می‌اندازیم:

```
snark:~$ ls -l /bin/login
-rwsr-xr-x   1 root    bin        20164   Apr 17 12:57   /bin/login
```

مجوزها همانطور هستند که برای یک فرمان سیستمی انتظار داریم به جز «s» در محلی که باید قاعدتاً مکان مجوز اجرا برای مالک باشد. این شیوه‌ی نمایش یک مجوز ویژه به نام «set-user-id» یا بیت setuid است.

بیت setuid معمولاً به برنامه‌هایی ضمیمه می‌شود که نیاز دارند به کاربران معمولی امتیازات root را، البته به طور محدود، بدهند. وقتی بیت setuid به یک برنامه‌ی اجرایی داده می‌شود، شما امتیازات مالک فایل برنامه را، در هنگامی که برنامه از طرف شما اجرا می‌شود، پیدا می‌کنید، خواه شخصاً این امتیازات را داشته باشید یا خیر.

همانند حساب کاربر ریشه (root)، برنامه‌های setuid مفید اما خطرناک هستند. هرکس که بتواند یک برنامه setuid متعلق به root را دستکاری کند و تحت اختیار خود درآورد می‌تواند با استفاده از آن، یک shell (پوسته فرمان) با امتیازات root باز کند. به همین دلیل، در بیشتر انواع یونیکس، باز کردن یک فایل برای ویرایش و تغییر، به طور خودکار باعث خاموش شدن بیت setuid آن می‌شود. بسیاری از حملات به سیستم‌های یونیکس تلاش

می‌کنند نقاط آسیب‌پذیر در برنامه‌های setuid را هدف قرار داده و آنها را تحت کنترل درآورند. به همین خاطر مدیران سیستم هوشیار، درباره این برنامه‌ها، بسیار محتاط و مراقب هستند و معمولاً برنامه‌های setuid جدید نصب نمی‌کنند.

در مطالب بالا مسأله‌ی مهمی درباره مجوزها را نادیده گرفتیم: چگونه در هنگام ایجاد یک فایل یا دایرکتوری جدید گروه مالک و مجوزهای آن مشخص می‌شود. هر کاربر می‌تواند عضو چندین گروه باشد اما یکی از این گروهها (که در مدخل مربوط به کاربر در فایل/etc/passwd مشخص می‌شود) به عنوان گروه پیشفرض یا default group انتخاب و به طور معمول مالک فایل‌هایی می‌شود که آن کاربر ایجاد می‌کند.

اما درباره‌ی مجوزهای اولیه‌ی فایل‌ها داستان کمی پیچیده‌تر می‌شود. معمولاً برنامه‌ای که فایلی را ایجاد می‌کند، مجوزهای اولیه آن را نیز تعیین می‌کند. اما متغیری در محیط کاربر به نام umask این مجوزها را تغییر می‌دهد. umask مشخص می‌کند که کدام بیت‌های مجوز در هنگام ایجاد فایل غیر فعال شوند. معمول‌ترین مقدار umask که در اکثر سیستم‌ها به صورت پیش‌فرض قرار داده می‌شود -w------- یا ۰۰۲ است که بیت نوشتن (write)برای همه‌ی افراد را خاموش می‌کند. برای جزییات بیشتر راهنمای فرمان umask را در پوسته فرمان مطالعه کنید.

تعیین گروه اولیه برای دایرکتوری‌ها هم قدری پیچیده است. در بعضی یونیکس‌ها دایرکتوری جدید در گروه اصلی کاربر ایجادکننده‌اش قرار می‌گیرد (شیوه‌ی System V) و در دیگر انواع یونیکس، به گروه دایرکتوری والد خود نسبت داده می‌شود (شیوه‌ی BSD). در بعضی یونیکس‌های امروزی، از جمله لینوکس، می‌توان با تعیین set-group-ID برای دایرکتوری مورد نظر، حالت دوم را انتخاب کرد (chmod g+s).

۱۰-۶. چگونه ممکن است اشکال پیش بیاید

قبلاً اشاره کردیم که فایل‌سیستم‌ها چیزهای ظریف و آسیب‌پذیری هستند. اکنون می‌دانیم که برای رسیدن به یک فایل شما باید مسیر پرپیچ و خمی را طی کنید و از میان زنجیره‌ی درازی از نشانی‌های دایرکتوری و i-nodeها بگذرید. حالا فرض کنید که روی دیسک سخت شما یک نقطه‌ی خراب (bad spot) ایجاد شود. حالا چه خواهد شد؟

اگر بخت با شما یار باشد این نقطه‌ی خراب فقط مقداری اطلاعات مربوط به فایلها را از بین خواهد برد. اگر بدشانس باشید، می‌تواند یک ساختار دایرکتوری یا شماره i-node را تخریب کند و یک زیرشاخه‌ی کامل سیستم را بلااستفاده کند، و یا حتی بدتر از این، ساختار را طوری مختل کند که در آن، مسیرهای مختلف به یک قطعه از دیسک یا i-node ختم شود. چنین اختلالی می‌تواند با عملیات عادی مربوط به فایل‌ها به دیگر قسمت‌ها سرایت کند و اطلاعاتی را هم که در اصل در نقطه خرابی نبودند نابود کند.

خوشبختانه، با ایمن‌تر شدن دیسک‌های سخت این نوع سرایت به ندرت اتفاق می‌افتد. با این حال، یونیکس هر چند وقت یک بار تمامیت و سلامت فایل‌سیستم را وارسی می‌کند تا مطمئن شود که اشکالی وجود ندارد. یونیکس‌های امروزی این وارسی‌ها را در هنگام راه‌اندازی سیستم بر روی تک تک پارتیشنها، درست قبل از سوارکردن (mount) آنها، به سرعت اجرا می‌کنند. در هر چند بار راه‌اندازی هم یک وارسی کامل‌تر انجام می‌شود که چند دقیقه‌ای بیشتر طول می‌کشد.

اگر از این مطالب اینگونه برمی‌آید که یونیکس به طرز رعب‌آوری پیچیده و آسیب‌پذیر است، خوب است بدانید که این وارسی‌های هنگام راه‌اندازی غالباً اشکالات معمولی را، قبل از اینکه فاجعه بیافرینند، برطرف می‌کند. دیگر سیستم‌عامل‌ها چنین امکاناتی ندارند، که باعث می‌شود راه‌اندازی آنها قدری سریعتر باشد، اما در هنگام ترمیم دستی، شما را به شدت مأیوس خواهند کرد (البته با این فرض که یک نسخه از Norton Utilities یا امثال آن را هم داشته باشید...).

یکی از شیوه‌های نوین و رایج در طراحی یونیکس، فایل‌سیستم‌های journalling است. این نوع فایل‌سیستمها ترافیک دیسک را طوری تنظیم می‌کنند که تضمین می‌کند دیسک در وضعیتی پایدار قرار خواهد داشت و در هنگام راه‌اندازی مجدد سیستم به این وضعیت بازخواهد گشت. این باعث می‌شود سرعت وارسی‌های هنگام راه‌اندازی بسیار بیشتر شود.

۱۱. زبان‌های کامپیوتری چگونه کار می‌کنند؟

قبلاً به چگونگی اجرای برنامه‌ها پرداختیم. هر برنامه باید در نهایت دسته‌ای از بایت‌ها را، که دستورالعمل‌هایی به زبان ماشین هستند، اجرا کند. اما انسان‌ها زبان ماشین را به خوبی نمی‌دانند، کار با زبان ماشین، حتی برای هکرها، بیشتر به جادو شبیه است و بسیار به ندرت دیده می‌شود.

امروزه تقریباً تمامی کد یونیکس، به جز مقدار اندکی از بخش پشتیبانی مستقیم واسطه‌های سخت‌افزاری (direct hardware-interface support) در هسته، به یک زبان سطح بالا (high-level) نوشته می‌شود. (واژه‌ی high-level واژه‌ای قدیمی و مربوط به زمانی است که باید این نوع زبان‌ها را از زبان‌های سطح پایینِ (low-level) اسمبلرها (assembler) که در واقع لفاف‌های (wrapper) نازکی به دور کدهای ماشین هستند، تمیز می‌دادند).

چندین گونه‌ی مختلف زبان سطح بالا وجود دارد. برای اینکه راجع به این زبان‌ها صحبت کنیم، این نکته را همواره به یاد داشته باشید که کد منبعِ (source code) برنامه (نسخه قابل ویرایش تولید شده توسط انسان) باید به گونه‌ای به کد ماشین ترجمه شود که برای ماشین قابل اجرا باشد.

۱۱-۱. زبان‌های تألیفی (compiled languages)

مرسوم‌ترین نوع زبان‌ها، زبان‌های تألیفی یا compiled languages هستند. زبان‌های تألیفی به وسیله‌ی برنامه‌ی خاصی به نام مؤلف یا compiler به فایل‌های دودویی (binary) حاوی کدِ قابل اجرا برای ماشین ترجمه می‌شوند. وقتی فایل دودویی ایجاد شد، می‌توانید آن را مستقیماً و بدون آنکه دیگر نیازی به دیدن کد منبع باشد، اجرا کنید. (بیشتر نرم‌افزارها به شکل فایل‌های دودویی تألیف‌شده عرضه می‌شوند و کد منبع آن‌ها را نمی‌بینید.) کارکرد (performance) عالی و کامل‌ترین دسترسی به سیستم عامل از ویژگی‌های زبان‌های تألیفی است اما برنامه‌نویسی به این زبان‌ها قدری مشکل است.

زبان C که خود یونیکس با آن نوشته شده، قطعاً مهم‌ترینِ این زبان‌هاست (به همراه گونه دیگرش یعنی C++). زبان FORTRAN زبان تألیفی بسیار قدیمی‌تر و ابتدایی‌تری است که هنوز بعضی دانشمندان و مهندسان از آن استفاده می‌کنند. در دنیای یونیکس زبان تألیفی دیگری رایج نیست. در خارج از دنیای یونیکس COBOL برای برنامه‌های مالی و تجاری بسیار معمول است.

در گذشته بسیاری زبان‌های تألیفی دیگر هم وجود داشت، اما بیشتر آن‌ها یا منسوخ شده‌اند و یا صرفاً مصارف تحقیقاتی دارند. اگر شما یک برنامه‌نویس تازه‌کار یونیکسی هستید زبان تألیفی مورد استفاده شما به احتمال فراوان C یا C++ است.

۱۱-۲. زبان‌های تفسیری (interpreted languages)

زبان‌های تفسیری به یک برنامه‌ی مفسر (interpreter) متکی هستند که این برنامه کد منبع را می‌خواند و هم‌زمان به محاسبات و نداهای سیستمی (system calls) ترجمه می‌کند. برای هر بار اجرا، کد منبع باید دوباره تفسیر شود (و مفسر هم باید وجود داشته باشد).

این زبان‌ها معمولاً از زبان‌های تألیفی کندتر هستند و غالباً دسترسی آن‌ها به سیستم‌عامل و سخت‌افزارها محدود است. با این حال، برنامه‌نویسی به این زبان‌ها آسان‌تر است و بیشتر از زبان‌های تألیفی از خطاهای کدنویسی چشم‌پوشی می‌کنند.

بسیاری از ابزارهای یونیکس، از جمله پوسته فرمان (shell) و bc و sed و awk عملاً زبان‌های تألیفیِ کوچک هستند. زبان‌های BASIC معمولاً تفسیری هستند. Tcl هم همینطور است. از نظر تاریخی مهم‌ترین زبان تفسیری LISP بوده است (LISP نسبت به بیشتر جانشینان خود یک پیشرفت بزرگ به شمار می‌آید). امروزه پوسته‌های (shell) یونیکس و LISP که در درون ویرایشگر Emacs به حیات خود ادامه می‌دهد مهم‌ترین زبان‌های تفسیری خالص هستند.

۱۱-۳. زبان‌های P-code

از سال ۱۹۹۰ نوعی زبان دورگه که از تألیف و تفسیر یکجا استفاده می‌کند اهمیت فزاینده‌ای پیدا کرده‌است. زبان‌های P-code از آن نظر که در آن‌ها کد منبع به فایل دودویی فشرده ترجمه می‌شود شبیه زبان‌های تألیفی هستند، اما محتوای این فایل‌های دودویی کد ماشین نیست. بلکه در واقع کد کاذب یا pseudocode یا P-code است که معمولاً بسیار ساده‌تر اما قدرتمندتر از زبان واقعی ماشین می‌باشد. در هنگامی که برنامه را اجرا می‌کنید، در واقع P-code را تفسیر می‌کنید.

p-code می‌تواند تقریباً با سرعت یک دودویی (binary) تألیف شده (compiled) اجرا شود (مفسرهای p-code را می‌توان نسبتاً ساده، کوچک و سریع کرد). با این حال زبان‌های p-code می‌توانند انعطاف‌پذیری و قدرت یک مفسر خوب را هم داشته باشند.

زبان‌های پایتِن (Python)، پرل (Perl) و جاوا (Java) مهمترین زبان‌های p-code هستند.

۱۲. اینترنت چگونه کار می‌کند؟

برای آن که چگونگی کار کردن اینترنت را بهتر بفهمید، بیایید به اتفاقاتی که در هنگام یک عملیات ساده اینترنتی می‌افتد نگاهی بیندازیم. این عملیات، هدایت مرورگر (browser) از صفحه‌ی اول این راهنما به صفحه‌ی خانگی آن روی وب در سایت Linux Documentation Project است. صفحه اول راهنما این است:

```
http://www.tldp.org/HOWTO/Unix-and-Internet-Fundamentals-HOWTO/index.html
```

این یعنی صفحه‌ی اول راهنما در فایل HOWTO/Unix-and-Internet-Fundamentals-HOWTO/index.html در دایرکتوری صادراتِ World Wide Web در میزبان www.tldp.org قرار دارد.

۱۲-۱. نام‌ها و مکان‌ها

اولین کاری که مرورگر شما باید انجام دهد برقراری تماس شبکه‌ای با کامپوتری است که فایل مورد نظر در آن قرار دارد. برای این کار، مرورگر باید ابتدا موقعیت host یا میزبانِ www.tldp.org را در شبکه پیدا کند (کلمه میزبان (host)، کوتاه شده‌ی ماشین میزبان (host machine) و یا میزبان شبکه (network host) است؛ و www.tldp.org یک نام میزبان (hostname) است). مکان یا موقعیت متناظر با این نام در واقع عددی به نام IP address است (بعداً معنای IP را توضیح خواهیم داد).

بدین منظور، مرورگر به برنامه‌ای به نام name server (کارگزار اسامی) مراجعه می‌کند. name server ممکن است در کامپیوتر شما باشد، اما محتمل‌تر این است که بر روی یک کامپیوتر خدمت‌رسان باشد که دستگاه شما با آن ارتباط برقرار می‌کند.

وقتی در یک شرکت خدمات اینترنت یا ISP ثبت نام می‌کنید، یقیناً در قسمتی از فرایند برپاسازی (setup) باید نشانیِ IP یک name server موجود در شبکه‌ی ISP را به نرم‌افزارهای اینترنتی خود بدهید.

name serverهایی که بر روی کامپیوترهای مختلف هستند با یکدیگر ارتباط برقرار می‌کنند و تمام اطلاعاتی را که برای مشخص (= resolve) کردن نام میزبان‌ها (یعنی ارتباط دادن آن‌ها به نشانی‌های IP متناظر) لازم است مبادله می‌کنند و بروز نگه می‌دارند.

ممکن است name server شما طی فرایند یافتن www.tldp.org از سه یا چهار سایت مختلف در شبکه پرس و جو کند اما این کار معمولاً خیلی سریع انجام می‌شود (شاید در کمتر از یک ثانیه). در بخش بعدی به جزئیات کار

name serverها خواهیم پرداخت.

name server به مرورگر شما می‌گوید که نشانی IP سایت www.tldp.org این است: 152.19.254.81 با دانستن این نشانی، کامپیوتر شما قادر خواهد بود مستقیماً با www.tldp.org تبادل بیت داشته باشد.

۱۲-۲. سیستم نام دامنه (Domain Name System)

مجموعه‌ی برنامه‌ها و پایگاه‌های داده‌ای که با همکاری هم، نام میزبان‌ها (hostnames) را به نشانی‌های IP ترجمه می‌کنند، به طور کلی DNS یا سیستمِ نامِ دامنه نامیده می‌شوند. وقتی در جایی به DNS server اشاره می‌شود منظور همان چیزی است که ما پیشتر nameserver نامیدیم. اکنون توضیح خواهم داد که کل این سیستم چگونه کار می‌کند.

نام میزبان‌ها یا hostnameهای اینترنتی از قسمت‌هایی که با نقطه از هم جدا می‌شوند، ساخته شده‌اند. یک دامنه (domain) مجموعه‌ای از کامپیوترها است که پسوند نام آن‌ها مشترک است. دامنه‌ها می‌توانند در درون دامنه‌های دیگر باشند. برای مثال کامپیوتر www.tldp.org در subdomain یا زیردامنه‌ی tldp.org. در دامنه‌ی org. قرار دارد.

هر دامنه توسط یک name server معتبر که نشانی‌های IP بقیه‌ی کامپیوترهای موجود در دامنه را می‌داند تعریف می‌شود. این name server معتبر یا اولیه (primary) ممکن است چند پشتیبان (backup) داشته باشد زیرا ممکن است روزی از کار بیافتد یا اصطلاحاً down شود. اگر در جایی نامی از secondary name server یا DNS ثانویه شنیدید منظور یکی از همین پشتیبان‌هاست. این سرورهای ثانویه معمولاً هر چند ساعت یک بار اطلاعات خود را با سرور اولیه هماهنگ می‌کنند، تا هر تغییری که در ارتباط نام میزبان به IP در سرور اولیه به وجود می‌آید به طور خودکار در همه‌ی سرورها اعمال شود.

و اما بخش مهم داستان این است: nameserverهای یک دامنه نیازی به دانستن مکان همه کامپیوترهای موجود در دامنه‌های دیگر (از جمله زیردامنه‌های خودشان) ندارند، و تنها باید موقعیت nameserverهای آن دامنه‌ها را بدانند. در مثال ما، nameserver معتبر دامنه‌ی org. نشانی IP مربوط به nameserver دامنه tldp.org. را می‌داند اما نشانی بقیه کامپیوترهای موجود در tldp.org. را نمی‌داند.

دامنه‌ها در سیستم DNS مانند یک درخت بزرگِ وارونه مرتب می‌شوند. در بالا، سرورهای ریشه قرار دارند. همه نشانی IP سرورهای ریشه را می‌دانند؛ این نشانی‌ها به نرم‌افزار DNS شما مخابره می‌شوند. سرورهای ریشه نشانی IP مربوط به nameserverهای دامنه‌های سطح بالا مانند com. و org. را می‌دانند، اما نشانی کامپیوترهای موجود در آن دامنه‌ها را نمی‌دانند. سرورهای دامنه‌های سطح بالا هم موقعیت nameserverهای دامنه‌هایی را که مستقیماً در زیر خودشان قرار دارند را می‌دانند و الی آخر.

DNS به گونه‌ای طراحی شده که هر کامپیوتر بتواند حداقلِ اطلاعاتی را که درباره شکل این درخت لازم دارد به دست بیاورد؛ و تغییرات داخلی در شاخه‌های درخت، به سادگی و با تغییر ارتباط‌های نام به IP یا (name-to-IP mapping) در پایگاه داده‌های یکی از سرورهای معتبر امکان‌پذیر باشد.

وقتی نشانی IP سایت www.tldp.org را پرس و جو (query) می‌کنید، آن چه که اتفاق می‌افتد از این قرار است: ابتدا nameserver شما از یکی از سرورهای ریشه سؤال می‌کند که آیا می‌تواند یک nameserver برای org. پیدا کند یا خیر. بعد از گرفتن جواب، از سرور org. نشانی IP مربوط به nameserver دامنه‌ی tldp.org. را می‌پرسد. وقتی که این نشانی را هم به دست آورد، از nameserver مربوط به tldp.org. نشانی میزبان www.tldp.org را می‌پرسد.

بیشتر اوقات nameserver شما نیازی ندارد که تمام این کارها را انجام دهد. nameserverها نگهداری (cacheing) زیادی انجام می‌دهند، یعنی وقتی nameserver شما یک نام میزبان را می‌یابد (resolve می‌کند) ارتباط IP یافته شده با نام میزبان مربوطه را تا مدتی در حافظه نگه می‌دارد. به همین دلیل وقتی برای اولین بار به یک وب‌سایت می‌روید، معمولاً فقط برای اولین صفحه‌ای که دریافت می‌کنید مرورگر شما پیام می‌دهد که در حال یافتن (look up) میزبان است.

سرانجام زمانی می‌رسد که ارتباط نام-به-نشانی (name-to-address mapping) منقضی می‌شود و DNS مجبور می‌شود که دوباره پرس‌وجو کند– اهمیت این کار در این است که در صورت تغییر نشانی نام میزبان از باقی ماندن همیشگی اطلاعات نامعتبر در nameserver جلوگیری می‌شود. نشانی IP نگهداری شده (cached) برای یک وب سایت نیز، در صورتی که میزبان غیرقابل دسترس شود، دور ریخته می‌شود.

۱۲-۳. بسته‌ها و روترها (Packets and Routers)

کاری که مرورگر شما می‌خواهد انجام دهد ارسال فرمان زیر به سرور وب در www.tldp.org است:

```
GET  /LDP/HOWTO/Fundamentals.html HTTP/1.0
```

این کار بدین صورت انجام می‌شود: فرمان به یک بسته یا packet تبدیل می‌شود، یک بسته قطعه‌ای متشکل از بیت‌هاست و شبیه یک تلگرام است که سه چیز مهم به دور آن پیچیده می‌شود: نشانی مبدأ یا source address (که نشانی IP کامپیوتر شماست)، نشانی مقصد یا destination address که 152.19.254.81 است و یک شماره درگاه (port) یا شماره سرویس (service) (در این مثال 80) که نشان می‌دهد که بسته یک درخواست مربوط به

وب جهانی است.

سپس کامپیوتر شما بسته را از طریق سیم به خط ارتباطی (یعنی ارتباط شما با ISP یا شبکه محلی) ارسال می‌کند تا اینکه بسته به یک دستگاه مخصوص به نام روتر (router) می‌رسد. روتر نقشه‌ای از اینترنت در حافظه خود دارد؛ این نقشه هرچند همیشه کامل نیست، اما کاملاً همسایگی شبکه‌ی شما را توصیف می‌کند و می‌داند که چگونه به روترهای همسایگی‌های دیگرِ موجود در اینترنت متصل شود.

بسته شما ممکن است در راه رسیدن به مقصد، از چندین روتر بگذرد. روترها هوشمند هستند. آن‌ها وارسی می‌کنند که چقدر طول می‌کشد تا روترهای دیگر دریافت هر بسته را اعلام کنند. آن‌ها از این اطلاعات برای هدایت ترافیک به خطوط ارتباطی سریعتر استفاده می‌کنند. همچنین به کمک این اطلاعات از خارج شدن احتمالی روترها (و یا کابل‌ها)ی دیگر از شبکه مطلع می‌شوند و در صورت امکان با یافتن یک مسیر جایگزین این مسئله را جبران می‌کنند.

باور عامیانه‌ای وجود دارد که می‌گوید اینترنت طوری طراحی شده است که در صورت وقوع جنگ هسته‌ای سالم بماند. چنین چیزی حقیقت ندارد، اما طراحی اینترنت از نظر بهره‌برداری مطمئن از سخت‌افزارهای ظریف و شکننده در یک جهان غیر قابل اطمینان، فوق‌العاده خوب است. دلیل اصلی این امر آن است که هوشمندی اینترنت به جای آنکه (مانند شبکه تلفن) در چند سوئیچ بزرگ و آسیب‌پذیر متمرکز باشد، در میان هزاران روتر پخش شده است. بدین ترتیب بیشتر اختلالات به محل وقوع خود، محدود می‌شوند و شبکه می‌تواند آن‌ها را دور زده و از مسیرهای دیگر استفاده کند.

هنگامی که بسته‌ی ارسالی شما به کامپیوتر مقصد می‌رسد، آن کامپیوتر با استفاده از شماره‌ی سرویس، بسته را به سرور وب (web server) مربوطه تحویل می‌دهد. سرور وب با نگاه‌کردن به نشانی منبع یا source address بسته، محلی را که باید پاسخ خود را به آنجا بفرستد پیدا می‌کند. سرور وب برای ارسال صفحه‌ی مورد نظر، آن را به چند بسته تبدیل می‌کند. اندازه‌ی بسته‌ها متغیر است و به رسانه‌ی انتقال مورد استفاده در شبکه و نوع سرویس، بستگی دارد.

۱۲-۴. TCP و IP

برای درک چگونگی انتقال‌های چندبسته‌ای، باید بدانید که اینترنت در واقع از دو پروتکل (= مجموعه قوانین انتقال اطلاعات که به دو کامپیوتر امکان ارتباط می‌دهد - م) استفاده می‌کند، که یکی بر روی دیگری قرار دارد.

پروتکل زیرین، که IP (به معنی Internet Protocol) نام دارد، بر روی هر بسته نشانی مبدأ (source) و نشانی مقصد (destination) دو کامپیوتری را که در شبکه، تبادل اطلاعات می‌کنند، برچسب می‌زند. برای مثال، وقتی به نشانی http://www.tldp.org دسترسی پیدا می‌کنید، بسته‌هایی که می‌فرستید، نشانی IP کامپیوتر شما، مثلاً

192.168.1.101 و نیز نشانی کامپیوتر www.tldp.org یعنی 152.19.254.81 را در خود دارند. این نشانی‌ها همانند نشانی پستی خانه شما هستند. و روترها هم کاری شبیه به اداره پست انجام می‌دهند. همانطور که وقتی کسی به شما نامه می‌نویسد اداره پست نشانی روی پاکت را می‌خواند، محل شما را تشخیص می‌دهد و بهترین راه رساندن نامه را مشخص می‌کند.

اما لایه‌ی بالایی، یعنی پروتکل کنترل انتقال یا TCP (مخفف Transmission Control Protocol)، به شما اطمینان می‌بخشد. وقتی دو کامپیوتر یک تماس TCP برقرار می‌کنند (که این کار را با استفاده از IP انجام می‌دهند)، کامپیوتر دریافت کننده می‌داند که باید وصول بسته‌هایی را که دریافت می‌کند به فرستنده اعلام کند. اگر فرستنده در یک محدوده زمانی مشخص، اعلام وصول یک بسته را دریافت نکند، بسته را مجدداً می‌فرستد. به علاوه، فرستنده به هر بسته‌ی TCP یک شماره توالی یا sequence number نسبت می‌دهد، که دریافت‌کننده می‌تواند با استفاده از آن، بسته‌ها را سرهم کند زیرا ممکن است بسته‌ها بدون ترتیبِ درست به گیرنده برسند. (این حالت می‌تواند به آسانی در اثر نوسانات خطوط ارتباطی شبکه در هنگام تماس رخ دهد.)

بسته‌های TCP/IP همچنین حاوی عدد وارسی یا checksum هستند، به کمک این عدد می‌توان از تخریب احتمالی اطلاعات به خاطر اشکالات ارتباطی با خبر شد. (عدد وارسی با استفاده از بقیه‌ی بسته محاسبه می‌شود، به گونه‌ای که اگر بقیه‌ی بسته یا عدد وارسی تخریب شود، انجام دوباره محاسبه و مقایسه با عدد اولیه به احتمال بسیار قوی خطا را نشان دهد.)

(توضیح مترجم: checksum روشی برای کشف تغییر ناخواسته اطلاعات در حین انتقال است. فرستنده تمام اطلاعات را به اعداد دوکاراکتری تبدیل می‌کند و سپس همه اعداد را با هم جمع می‌کند. گیرنده عین همان محاسبه را انجام می‌دهد و checksum محاسبه شده را با checksum دریافتی مقایسه می‌کند. اگر این دو عدد مساوی نباشند گیرنده می‌فهمد که اطلاعات در جریان، اطلاعات تخریب شده است.)

بنابراین، برای هرکس که از TCP/IP و nameserverها استفاده می‌کند این روش راهی مطمئن برای تبادل بایت‌ها بین جفت‌های نام میزبان/شماره سرویس (hostname/service number pairs) است. کسانی که پروتکل‌های شبکه را می‌نویسند تقریباً هیچ وقت نیاز نیست بابت اندازه‌ی بسته‌ها، سرهم کردن دوباره‌ی بسته‌ها، خطایابی، وارسی عددی (checksumming) و انتقال مجدد که در سطح زیرین انجام می‌شود نگران باشند.

۱۲-۵. HTTP، یک پروتکل کاربردی

حالا بیایید به مثالی که در بالا زدیم برگردیم. مرورگرها و سرورهای وب از طریق یک پروتکل کاربردی یا application protocol با هم ارتباط برقرار می‌کنند. این پروتکل بر روی TCP/IP اجرا می‌شود و از آن، صرفاً برای جابجا کردن رشته‌های بایت استفاده می‌کند. نام این پروتکل HTTP (مخفف Hyper-Text Transfer Protocol به

معنی پروتکل انتقال فرامتنی) است، که در بالا یکی از فرمان‌های آن، یعنی فرمان GET را دیدیم.

وقتی فرمان GET با شماره سرویس 80 به وب‌سرور www.tldp.org می‌رود، به یک server daemon ،که در درگاه یا پورت 80 منتظر است، فرستاده می‌شود. (server daemon= برنامه سروری که در پس‌زمینه و به طور دائمی اجرا می‌شود - م) بیشتر سرویس‌های اینترنتی با استفاده از server daemonها عملیاتی می‌شوند.کار این daemonها چیزی جز آن نیست که در درگاه‌ها منتظر بمانند، و فرمان‌های ورودی به سیستم را دریافت و اجرا کنند.

اگر طراحی اینترنت یک قانون کلی و فراگیر داشته باشد، آن قانون این است که همه‌ی قسمت‌ها باید تا حد ممکن ساده و برای افراد، قابل دسترسی باشند. از این رو HTTP و هم خانواده‌هایش (از جمله SMTP ، پروتکل انتقال ساده‌ی ایمیل، که برای جابجا کردن نامه‌های الکترونیکی بین میزبان‌ها استفاده می‌شود) از فرمان‌های ساده‌ی متنی قابل چاپ، که در آخرِ آن‌ها علامت پایان خط یا رفتن به خط بعد(carriage-return) قرار می‌گیرد، استفاده می‌کنند.

این روش اندکی از کارایی می‌کاهد، در بعضی شرایط، در صورت استفاده از پروتکل‌های دودویی (binary) با کدنویسی فشرده و بسته، سرعت بیشتری به دست می‌آید. اما تجربه نشان داده است که داشتن فرمان‌هایی که برای انسان‌ها آسان و قابل فهم و توصیف باشند با ارزشتر از افزایش کارایی اندکی است که به قیمت پیچیده و مبهم کردن کارها به دست می‌آید.

بنابراین، آنچه که server daemon از طریق TCP/IP به شما برمی‌گرداند نیز، متن است. شروع پاسخ چیزی شبیه این است (چند سرصفحه (header) در اینجا مخفی شده‌اند):

```
HTTP/1.1 200 OK
Date: Sat, 10 Oct 1998 18:43:35 GMT
Server: Apache/1.2.6 Red Hat
Last-Modified: Thu, 27 Aug 1998 17:55:15 GMT
Content-Length: 2982
Content-Type: text/html
```

بعد از این سرصفحه‌ها (headers) یک خط خالی و بعد از آن متن صفحه‌ی وب می‌آید (که بعد از آن تماس قطع می‌شود). مرورگر شما فقط صفحه‌ی وب (webpage) را نمایش می‌دهد. سرصفحه‌ها به مرورگر می‌گویند که چگونه این کار را انجام دهد (به طور خاص، سرصفحه‌ی Content-Type به مرورگر می‌گوید که اطلاعات دریافتی واقعاً HTML است).