دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

دستورکار آزمایشگاه سیستم‌عامل

دکتر علی بهلولی، دکتر مرجان کائدی

نسخه ۱.۲

پاییز 1401

تاریخچه بازبینی‌ها

شماره نسخه	تاریخ	شرح تغییرات
۱.۰	۱۳۹۶/۱۱/۰۷	نسخه اولیه
۱.۱	۱۴۰۰/۱۲/۰۵	اصلاحات کلی
۱.۲	۱۴۰۱/۰۹/۰۱	اصلاحات جزئی

پیشگفتار

هدف از این آزمایشگاه، آشنایی عملی دانشجویان با مفاهیم سیستم‌عامل است. دستورکار این آزمایشگاه در قالب ۱۰ آزمایش تدوین شده است که برمبنای سیستم عامل لینوکس است. به منظور جلوگیری از آسیب‌های احتمالی به سیستم عامل، از ماشین مجازی برای انجام آزمایش‌ها استفاده شده است. در این صورت دانشجو بدون نگرانی از آسیب دیدن سیستم‌عامل به انجام آزمایش‌ها خواهد پرداخت.

مقررات این آزمایشگاه نیز مطابق مقررات عمومی آزمایشگاه‌های دانشگاه است. جلسات آزمایشگاه بلافاصله بعد از حذف و اضافه شروع شده و به صورت هفتگی تا شروع امتحانات پایان ترم ادامه می‌یابد. دانشجویی که سه جلسه غیبت داشته باشد، مردود است.

روند کلی آزمایشگاه به این صورت است که هر یک از دانشجویان باید قبل از هر آزمایش، پیش‌آگاهی آن را از قبل مطالعه کرده، در ابتدای جلسه، در یک امتحان کوچک ۱۵-۱۰ دقیقه‌ای در مورد آن پیش‌آگاهی تسلط خود را نشان دهد (این کار به خاطر هدایت دانشجویان به مطالعهٔ قبلی مطالب و تقلیل احتمال تقلب در پیش گزارش‌ها در نظر گرفته شده است). پس از امتحان کوچک، دانشجویان هر آزمایش (که معمولا ۲ یا ۳ نفر هستند)، گام‌های مشخص شده در دستور کار را دنبال کرده، نتایج را در کاربرگ‌های آن آزمایش (که توسط مربی در محل آزمایشگاه در اختیار دانشجو گذاشته می‌شود) ثبت می‌کنند. به این ترتیب کاربرگ‌ها به صورت یک سری برای هر گروه کامل می‌گردد.

در ارزیابی نهایی هر دانشجو، علاوه بر نمرهٔ امتحانات کوچک هر جلسه و نمرهٔ گزارش جلسات (که برای تمام اعضای هر گروه یکسان است)، نظر مربی در مورد آن دانشجو و نتیجهٔ امتحان پایان ترم دانشجو نیز دخالت دارد.

در انتها از تمام همکاران و دانشجویان درخواست می‌کنم با طرح پیشنهادهای مشخص و مدون خود برای رفع معایب، ارتقای کیفی و پویایی به عنوان یک ویژگی ضروریِ آزمایشگاه سیستم‌عامل، ادامه دهندهٔ راهی باشند که اولین گام آن برداشته شده است. بر خود لازم می‌دانم از زحمات مهندس مریم طائبی، مهندس فاطمه شفیع زادگان و مهندس مهدی سمیع در بهبود کیفیت این دستورکار تشکر کنم.

دکتر علی بهلولی، دکتر مرجان کائدی

پاییز ۱۴۰۱

Emails: bohlooli@eng.ui. ac.ir, kaedi@eng.ui.ac.ir

فهرست مطالب

۲-۳ دستورکار

آزمایش چهارم: سیستم فایل (مجوزهای دسترسی)

۴-۱ پیش‌آگاهی

۴-۱-۱ کاربران در لینوکس

۴-۱-۲ نحوه تغییر اجازه‌های دسترسی

۴-۲ دستورکار

آزمایش پنجم: سیستم فایل (سلسله مراتب فایل‌ها)

۵-۱ پیش‌آگاهی

۵-۱-۱ دایرکتوری ریشه (/)

۵-۱-۲ دایرکتوری binary

۵-۱-۳ دایر کتوری configuration

۵-۱-۴ دایرکتوری data (بعدی ها زیر این هستند؟)

۵-۱-۵ دایرکتوری in memory

۵-۲ دستورکار

آزمایش ششم: مدیریت فرآیندها (مقدمه)

۶-۱ پیش‌آگاهی

۶-۱-۱ مشاهده مشخصات فرآیندها: فرمان‌های ps و top

۶-۱-۲ اجرای فرآیندها در پس‌زمینه (background)

۶-۱-۳ هستهٔ Linux

۶-۱-۴ پوستهٔ Linux

۶-۲ برنامه‌نویسی

۶-۲-۱ gcc (مترجم زبان C)

۶-۲-۲ اشکالزدایی با gdb

۶-۳ اتصال راه دور به سیستم‌عامل لینوکس

۶-۴ دستورکار

آزمایش هفتم: مدیریت فرآیندها (حالات فرآیند و ایجاد فرآیند)

۷-۱ پیش‌آگاهی

۷-۱-۱ حالات مختلف فرآیند (Process)

۷-۱-۲ ایجاد فرآیند جدید

۷-۱-۳ اجرای عملیاتی متفاوت با عملیات فرآیند اصلی در فرآیند ایجاد شده

۷-۱-۴ متوقف کردن یک فرآیند

۷-۱-۵ راهنمایی‌هایی برای انجام آزمایش

۷-۲ دستورکار

آزمایش هشتم: چندنخی

۸-۱ پیش‌آگاهی

۸-۱-۱ مفهوم چند وظیفه‌ای و چندنخی

۸-۱-۲ امکانات لینوکس برای کار با نخ‌ها

۸-۲ دستورکار

آزمایش نهم: مدیریت نخ‌ها (هماهنگی بین نخ‌ها)

۹-۱ پیش آگاهی

۹-۱-۱ توابع مربوط به هماهنگی نخ‌ها

۹-۲ دستورکار

آزمایش دهم: مدیریت فرآیندها (ارتباط و هماهنگی بین فرآیندها)

۱۰-۱ پیش‌آگاهی

آزمایش اول: آشنایی با ماشین مجازی و نصب لینوکس

۱-۱ پیش‌آگاهی

در این جلسه با ماشین مجازی و نحوه نصب سیستم‌عامل Ubuntu روی ماشین مجازی آشنا می‌شوید.

۱-۱-۱ ماشین مجازی

ماشین مجازی روشی برای ایجاد کامپیوترهای مجازی متعدد روی یک بستر سخت‌افزاری است. برای ماشین مجازی می‌توان دو دسته کاربرد در نظر گرفت:

الف) استفاده بهینه از سخت‌افزار

در کامپیوترهایی که حجم سخت‌افزار انبوهی دارند و معمولا در سرویس‌دهنده‌های وب استفاده می‌شوند (مثلا کامپیوترهای سرور که عموما دارای بیش از ۱۰۰ هسته پردازشی، حجم عظیم حافظه RAM و دیسک سخت هستند)، برای استفاده بهینه از این حجم سخت‌افزار، ماشین‌های مجازی متعدد ایجاد می‌شوند که هر کدام بخشی از سخت‌افزار واقعی را در اختیار دارند. این کار در واقع صرفه‌جویی در فضای فیزیکی است (زیرا مثلا به جای استفاده از ۱۰۰ کامپیوتر فیزیکی و مجزا، یک کامپیوتر قوی که تبدیل به ۱۰۰ کامپیوتر مجازی شده است استفاده می‌شود).

ب) شبیه‌سازی سیستم‌عامل‌های مختلف روی یک ماشین میزبان

در موارد متعدد نیاز است که چندین سیستم‌عامل روی یک کامپیوتر نصب شوند. نصب همزمان این سیستم‌ها به صورت واقعی روی دیسک ممکن است امکان‌پذیر نباشد و با ریسک‌هایی روبرو است؛ به خصوص وقتی که نصب سیستم‌عامل جدید به صورت موقت باشد و بعد از مدتی قصد حذف کردن آن را داشته باشید. به عنوان مثال می‌توان موارد زیر را ذکر کرد:

- قصد اجرا اجرای نرم‌افزاری را داشته باشید که در سیستم‌عامل کنونی اجرا نمی‌شود.
- قصد اجرای برنامه‌ای را داشته باشید که ممکن است عملکرد سیستم‌عامل را مختل کند.
- قصد یادگیری سیستم‌عامل جدیدی را داشته باشید (علت استفاده از ماشین مجازی در این آزمایشگاه نیز همین است).

ماشین مجازی به دو صورت پیاده‌سازی می‌شود:

- در روش اول، ماشین مجازی به صورت یک لایه روی سخت‌افزار پیاده‌سازی می‌شود (یعنی عملا به سیستم‌عامل میزبان نیاز ندارد). این روش عموما در سرویس‌دهنده‌های وب استفاده می‌شود.

– در روش دوم، از سیستم‌عامل میزبان استفاده می‌شود. در این روش، در واقع یک برنامه کاربردی روی سیستم‌عامل میزبان اجرا می‌شود که ماشین‌های مجازی را پیاده‌سازی می‌کند. برنامه vmware که در سیستم‌عامل ویندوز قابل استفاده است و همچنین برنامه VirtualBox که در سیستم‌عامل‌های ویندوز و لینوکس قابل استفاده است، از این نوع هستند.

۲-۱-۱ معرفی سیستم‌عامل Linux و نگاهی به ویژگی‌های آن

برای معرفی بهتر سیستم‌عامل Linux، ابتدا با تاریخچهٔ سیستم‌عامل Unix و انواع مختلف آن آشنا می‌شویم. سپس تاریخچهٔ سیستم‌عامل Linux را بررسی می کنیم.

تاریخچه سیستم‌عامل Unix

سیستم‌عامل Unix در سال ۱۹۶۹ میلادی توسط کن تامپسون و دنیس ریچی، بر روی کامپیوتر DEC PDP-7 در آزمایشگاه‌های بل با زبان اسمبلی طراحی شد. در سال‌های بعد تامپسون و ریچی، Unix را با زبان C بازنویسی کردند. زبان C قابل حمل بود وکمک کرد که Unix به سیستم‌عاملی تبدیل شود که می‌توانست بر روی انواع متفاوتی از کامپیوترها اجرا گردد. توسعهٔ سیستم‌عامل به آزمایشگاه‌های بل محدود نشد، بلکه در اواسط دههٔ ۱۹۷۰، Unix یک محصول تحقیقاتی بود که دانشگاه‌های بسیاری بر روی آن کار می‌کردند. از آن زمان تا کنون کارهای بسیاری بر روی Unix انجام گرفته است. با پیشرفت صنعت پردازنده‌ها و ظهور کامپیوترهای قوی‌تر، گونه‌های متعددی از آن بر روی کامپیوترهای گوناگون، توسط شرکت‌ها و مراکز تحقیقاتی مختلف دنیا ارائه شده است؛ مانند: Solaris برای پردازنده‌های Sparc، OSF/1 برای پردازنده‌های Alpha و Linux برای پردازنده‌های 80x86 (که در این آزمایشگاه مورد توجه ما است).

به‌طور کلی ویژگی‌های زیر را برای انواع مختلف سیستم‌عامل Unix می‌توان برشمرد:

طراحی مستقل از سخت‌افزار: از آنجا که قسمت اعظم کُد این سیستم‌عامل به جای اینکه با اسمبلی نوشته شود، به زبان C نوشته شده است، این سیستم‌عامل در مقایسه با سیستم‌عامل‌های دیگر دارای سرعت کم‌تر، ولی انعطاف و اطمینان بیشتری است. نگهداری چنین سیستم‌عاملی برای تولیدکنندگان آن آسان‌تر از سیستم‌عاملی است که با زبان اسمبلی تولید شده باشد. به همین دلیل روایت‌های زیادی از این سیستم‌عامل بر روی سخت‌افزارهای مختلف ایجاد شده است.

چندکاربره بودن: در یک زمان، یک یا چند کاربر می‌توانند از سیستم مبتنی بر Unix استفاده کنند. منابع سخت‌افزاری باارزش، مانند چاپگرها و سرویس‌دهنده‌های بزرگ (مانند سرویس‌دهنده‌های شبکه) توسط افراد بسیاری قابل استفاده است.

چندوظیفه‌ای بودن: هر کاربر می‌تواند همزمان چند وظیفهٔ مختلف انجام دهد (مثلا چند برنامه مختلف را به طور همزمان اجرا کند).

امکان شبکه شدن به صورت ذاتی: قابلیت اتصال کامپیوترهای کوچک و بزرگ و ایجاد شبکه‌های کامپیوتری، در نهاد این سیستم‌عامل تعبیه گشته است و برای ایجاد شبکه، احتیاجی به مدیر شبکهٔ دیگری نیست.

دارا بودن پایانه‌های متنی وگرافیکی: امروزه، اکثر سیستم‌های Unix دارای پایانه‌های متنی یا گرافیکی هستند. همان‌طورکه گفته شد، در آغاز Unix برای کامپیوترهای بزرگ طراحی شد. این کامپیوترها متشکل از یک یا چند پردازندهٔ قوی مرکزی و تعدادی پایانه (Terminal) بودند. دسترسی همزمان به کامپیوتر از طریق این پایانه‌ها انجام می‌گرفت. در واقع برای استفاده از قدرت چندکاربره بودن Unix، وجود این پایانه‌ها لازم بود. اولین گونه‌های این سیستم‌عامل دارای پایانه‌های متنی بودند. یک پایانهٔ متنی، دارای یک قالب متنی است که کوچک‌ترین جزء قابل تفکیک آن یک کاراکتر است (مانند محیط متنی DOS). با پیشرفت کامپیوترها و گرایش کاربران به واسط‌های کاربر گرافیکی، تحول بزرگی در زمینهٔ پایانه‌های Unix به وقوع پیوست: ظهور X Window System (یا به طور مخفف X). واسط گرافیکی X Window را می‌توان به محیط Microsoft Windows تشبیه کرد: X یک سیستم پنجره‌بندی گرافیکی است که یک واسط گرافیکی منویی و کاربرپسند (User Freindly) ارائه می‌کند. از ویژگی‌های بارز این واسط کاربر، سهولت و سادگی ارتباط کاربر با کامپیوتر است. در واسط‌های گرافیکی، کوچک‌ترین عنصر قابل تفکیک صفحهٔ نمایش، یک نقطه (Pixel) است. محیط X دارای قابلیت‌های فراوانی است که در اینجا از ذکر آنها صرف‌نظر می‌شود.

تاریخچهٔ سیستم‌عامل Linux

پیشرفت صنعت ریزپردازنده‌ها و ظهور ریزکامپیوترهای سریع و قوی، زمینه را برای به وجود آمدن گونه‌ای از Unix بر روی ریزکامپیوترها آماده ساخت. در مارس ۱۹۹۱ میلادی، Linus Torvalds یک سیستم Minix برای کامپیوتر ۳۸۶ خود خرید تا از آن برای طراحی سیستم‌عامل چند وظیفه‌ای خود استفاده کند. در ماه سپتامبر همان سال، او اولین نگارش آمادهٔ سیستم‌عامل خود را بر روی شبکهٔ اینترنت قرار داد و از تمام علاقه‌مندان، برای تکمیل آن دعوت به همکاری کرد. این سیستم‌عامل که ایدهٔ خود را از Unix گرفته بود، Linux نام گرفت. از این تاریخ، بسیاری از برنامه‌نویسان سراسر دنیا به کار بر روی Linux پرداختند. به این ترتیب پروژهٔ Linux آغاز شد و به تدریج قسمت‌های مختلف آن شکل گرفت و کامل گشت. اکنون این سیستم‌عامل بر روی ریزکامپیوترهای مبتنی بر پردازنده‌های 80x86 موجود است. در این بین، مؤسساتی مانند Free Software Foundation نیز مسؤولیت پشتیبانی و ارتقای Linux را بر عهده گرفته‌اند.

به‌طور کلی ویژگی‌هایی Linux را می‌توان به صورت زیر برشمرد:

- چند وظیفه‌ای و چند کاربره بودن، همچنین امکان شبکه شدن (که پیش‌تر برای انواع Unix بر شمردیم)، از جمله ویژگی‌های Linux است. با این سیستم‌عامل می‌توان شبکه‌های بسیار قوی طراحی کرد که امکان اتصال به شبکه‌های دیگر را نیز داشته باشند.

- Linux امکان استفاده از پایانه‌های متن و همچنین پایانه‌های گرافیکی مبتنی بر استاندارد X را دارد.

- این سیستم‌عامل را می‌توان بر روی یک کامپیوتر منفرد و جدا از شبکه نیز به‌کار گرفت و برای کار با آن وجود یک شبکهٔ متشکل از چندین ریز کامپیوتر الزامی نیست. البته در این حالت دیگر نمی‌توان از امکانات شبکه‌ای Linux استفاده کرد.

توزیع‌های مختلف لینوکس

امروزه صدها توزیع Linux در بازار موجود است. سه خانواده ی اصلی توزیع‌های سیستم‌عامل لینوکس عبارتند از:

- خانواده‌ی توزیع‌های Debian همچون توزیع Ubuntu

-خانواده‌ی توزیع‌های SUSE همچون openSUSE

- خانواده‌ی توزیع‌های Fedora همچون CentOS

راهنمای سیستم

به همراه گونه‌های جدید Unix و از جمله Linux، بستهٔ جامع راهنمای سیستم تحت عنوان صفحهٔ راهنما (Manual Page) ارائه شده است. هر جا که در کار کردن با امکانات سیستم‌عامل به اشکال برخوردید، سعی کنید از این صفحات راهنما با استفاده از فرمان man کمک بگیرید. این فرمان حداقل به یک پارامتر نیاز دارد و آن کلمهٔ کلیدی (keyword) درخواستی است. مثلاً برای گرفتن اطلاعات در مورد دستور ls باید فرمان زیر را صادر کنید:

```
$ man ls
```

این مجموعه راهنما در نه فصل سازماندهی شده است که سه فصل اول آن در این آزمایشگاه به کار می‌آید:

- فصل اول مربوط به فرمان‌هایی است که در اختیار کاربر قرار دارند.

- فصل دوم مربوط به کتابخانهٔ استاندارد فراخوانی‌های سیستم (system calls) در Linux است.

- فصل سوم توابع کتابخانهٔ زبان C را شرح می‌دهد.

می‌توانید با گزینهٔ S - به man بگویید که در کدام فصل، موضوع را جستجو کند. مثلا فرمان زیر در فصل سوم راهنماها به دنبال دستور exit (که در زبان C استفاده می‌شود) می‌گردد:

```
$ man -S 3 exit
```

ویرایشگرها

ویرایشگرهایی که در سیستم‌عامل Linux می‌توانید استفاده کنید عبارتند از vi، emacs و xemacs، vim و nano. ویرایشگرهای vi و vim برای کاربران حرفه‌ای سیستم‌عامل Linux است و کار کردن با آن‌ها مخصوصا برای کسانی که با سیستم‌عامل ویندوز کار کرده اند مشکل است. به همین خاطر در این آزمایشگاه تا حد ممکن از آن‌ها استفاده نخواهیم کرد. کار کردن با ویرایشگر nano در مقایسه با سایر ویرایشگرهای Linux آسان‌تر است (اگرچه ممکن است بازهم در کار با آن راحت نباشید). این ویرایشگر در محیط متن اجرا می‌شود ولی می‌تواند خود را با محیط گرافیکی هم تطبیق دهد. برای کار کردن با nano دانستن عملکرد چند کلید بسیار مفید است. برخی از این کلیدها در جدول ۱-۱ فهرست شده است.

جدول ۱-۲: چند کلید مفید در ویرایشگر nano

عملکرد	کلید
ذخیره فایل	Ctrl + O
خروج از ویرایشگر	Ctrl + X
رفتن به صفحه قبل	Ctrl + Y
رفتن به صفحه بعد	Ctrl + V
انتخاب متن	Alt + A
بریدن متن انتخاب شده	Ctrl + K
کپی کردن متن انتخاب شده	Alt + 6
چسباندن متن بریده شده یا کپی شده	Ctrl + U
نمایش مکان فعلی (شماره خط، سطر ..)	Ctrl + C

۲-۱ دستورکار

۱-فایل iso از سیستم‌عامل Ubuntu را از مربی آزمایشگاه دریافت کنید و روی ماشین مجازی نصب نمایید (مراحل نصب در بخش ۱-۴ آورده شده است).

۲- سیستم‌عامل نصب شده را اجرا کنید و موارد زیر را تمرین کنید:

۲-۱ دسترسی به فایل‌ها و پوشه‌ها (با استفاده از آیکون کابینت )

۲-۲ باز کردن فایل‌های Microsoft Office

۲-۳ جستجوی فایل‌ها و برنامه‌ها

۲-۴ نمایش تاریخ و تنظیم صدا و تغییر زبان

۲-۵ خاموش و restart و قفل نمودن

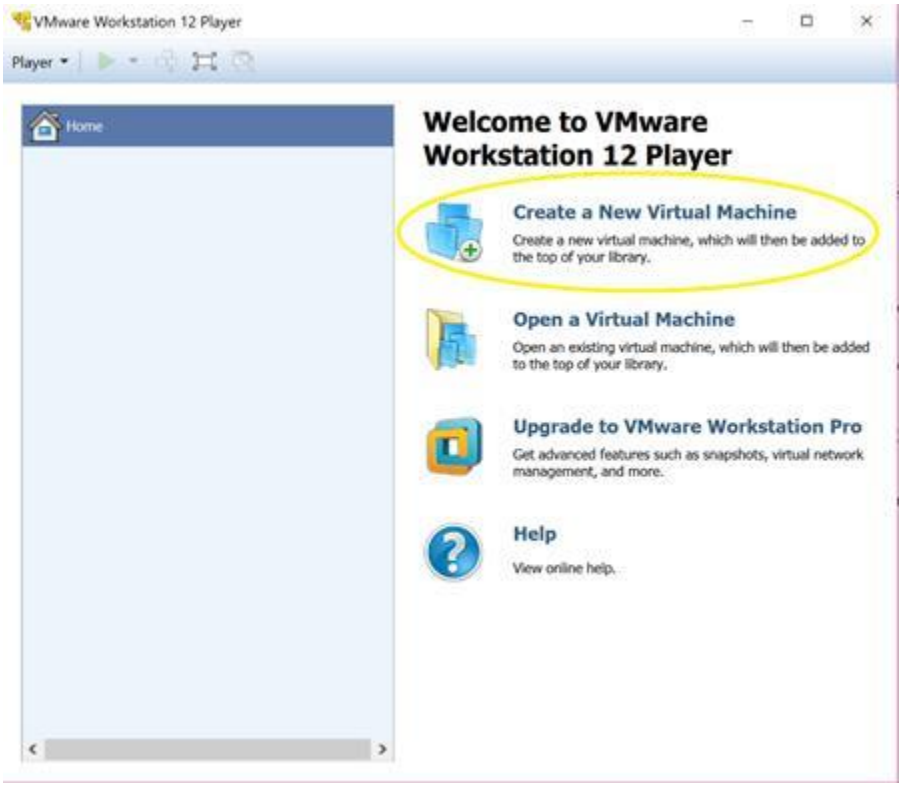
۲-۶ ساخت و مدیریت Userها

۲-۷ تغییر رمز عبور

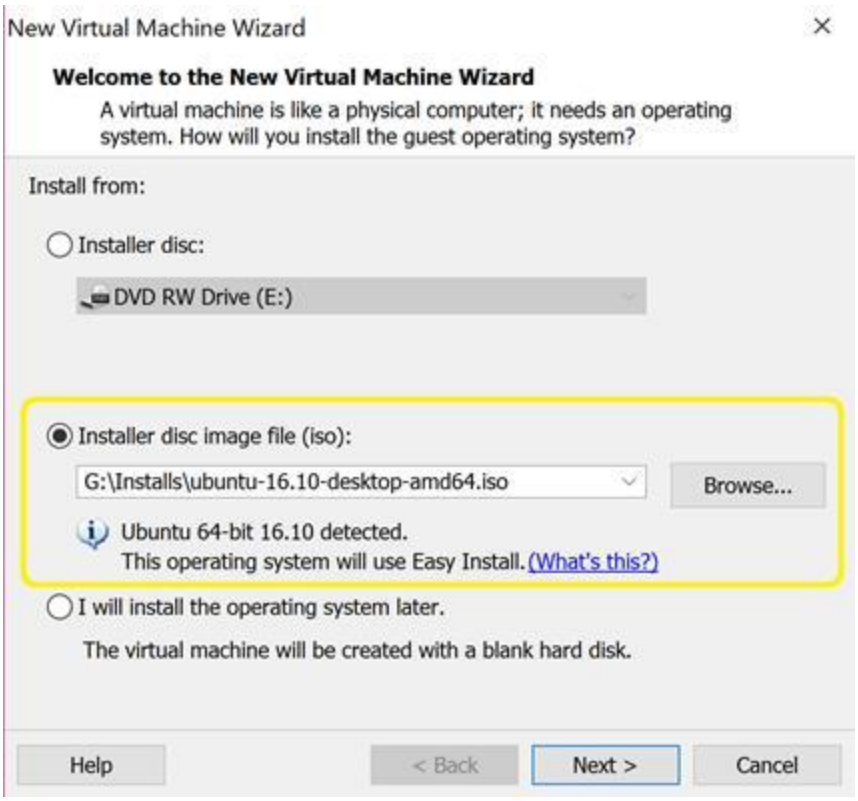
۳- با فشردن کلیدها Ctrl + Alt + t یک ترمینال باز کنید. سپس با استفاده از دستور nano، ویرایشگر را اجرا کنید و دستورات موجود در جدول ۱-۲ را تمرین کنید.

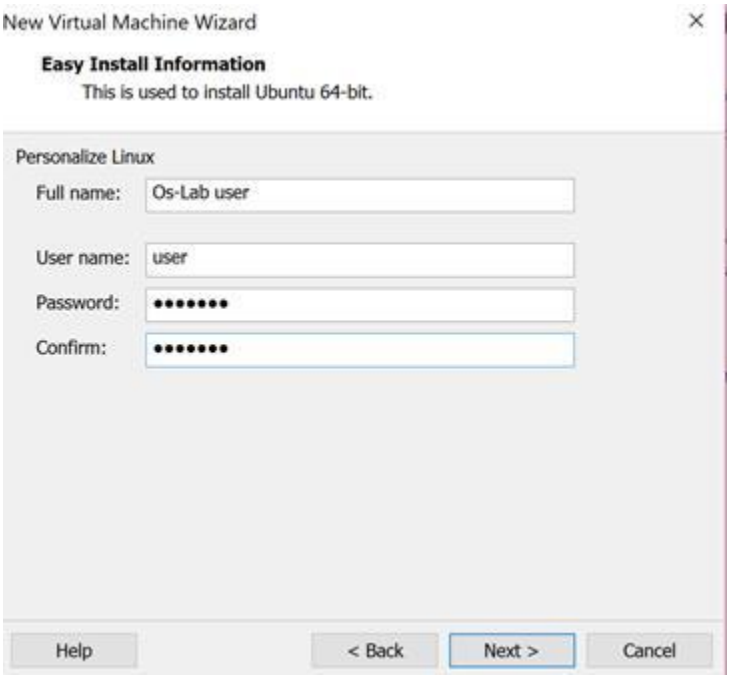
۳-۱ مراحل نصب سیستم‌عامل Ubuntu در VMware

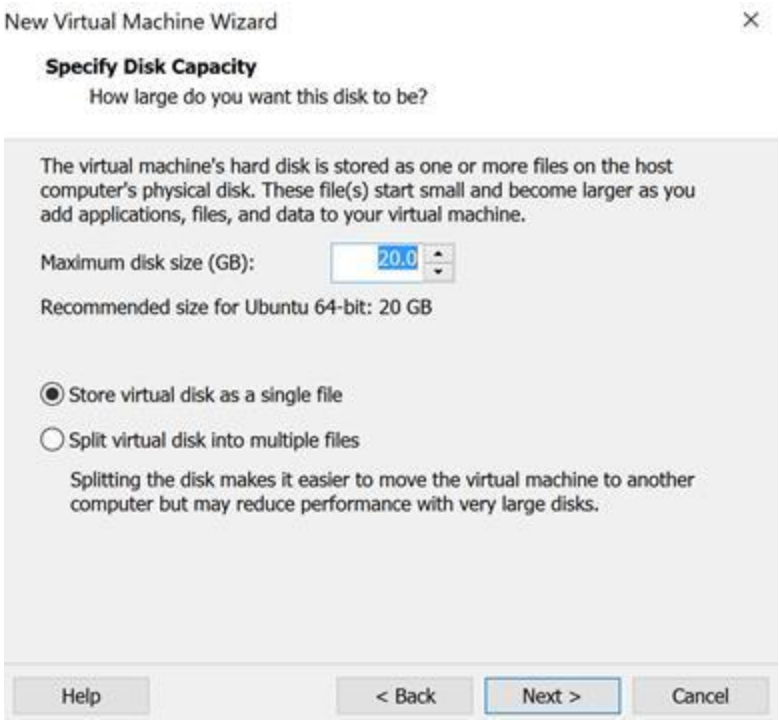
نرم‌افزار VMware را اجرا و سپس گزینه مشخص شده را انتخاب کنید:

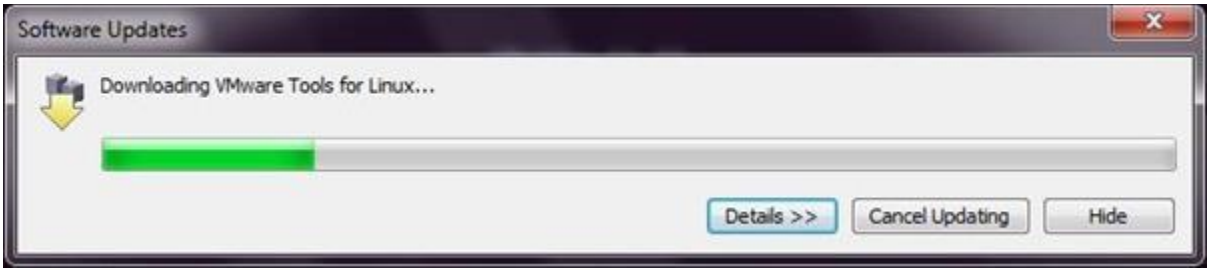


اکنون مراحلی که در ادامه نشان داده شده است را طی کنید.

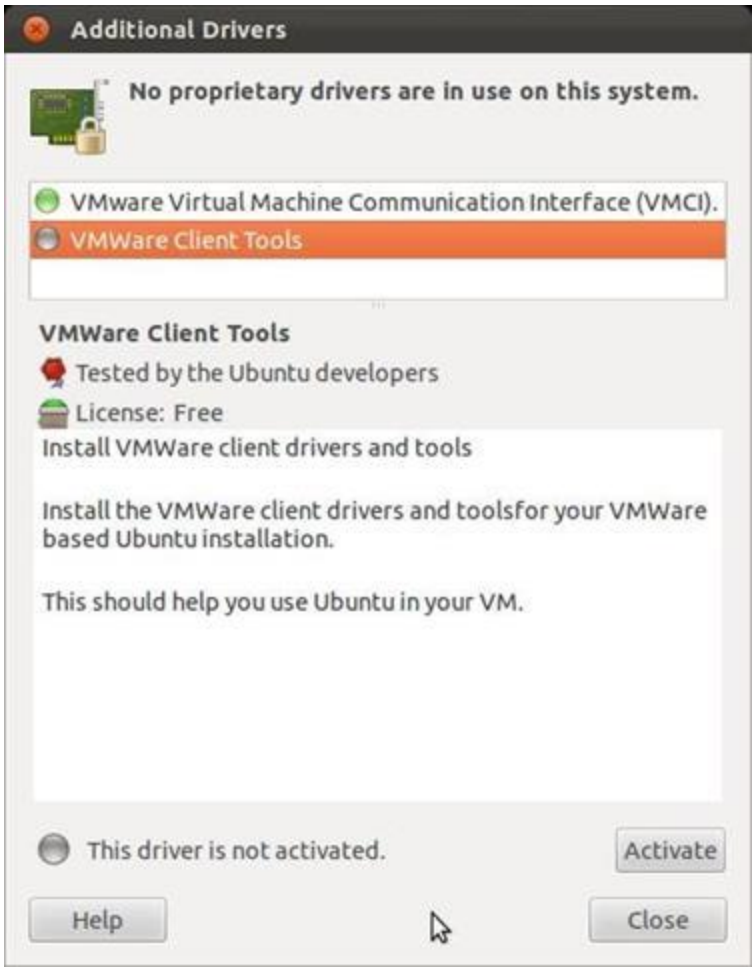

New Virtual Machine Wizard
Welcome to the New Virtual Machine Wizard
A virtual machine is like a physical computer; it needs an operating system. How will you install the guest operating system?
Install from:
Installer disc:
DVD RW Drive (E:)
Installer disc image file (iso):
G:\Installs\ubuntu-16.10-desktop-amd64.iso
Browse...
Ubuntu 64-bit 16.10 detected.
This operating system will use Easy Install. (What's this?)
I will install the operating system later.
The virtual machine will be created with a blank hard disk.
Help
< Back
Next >
Cancel


New Virtual Machine Wizard
Easy Install Information
This is used to install Ubuntu 64-bit.
Personalize Linux
Full name:
Os-Lab user
User name:
user
Password:
Confirm:
Help
< Back
Next >
Cancel

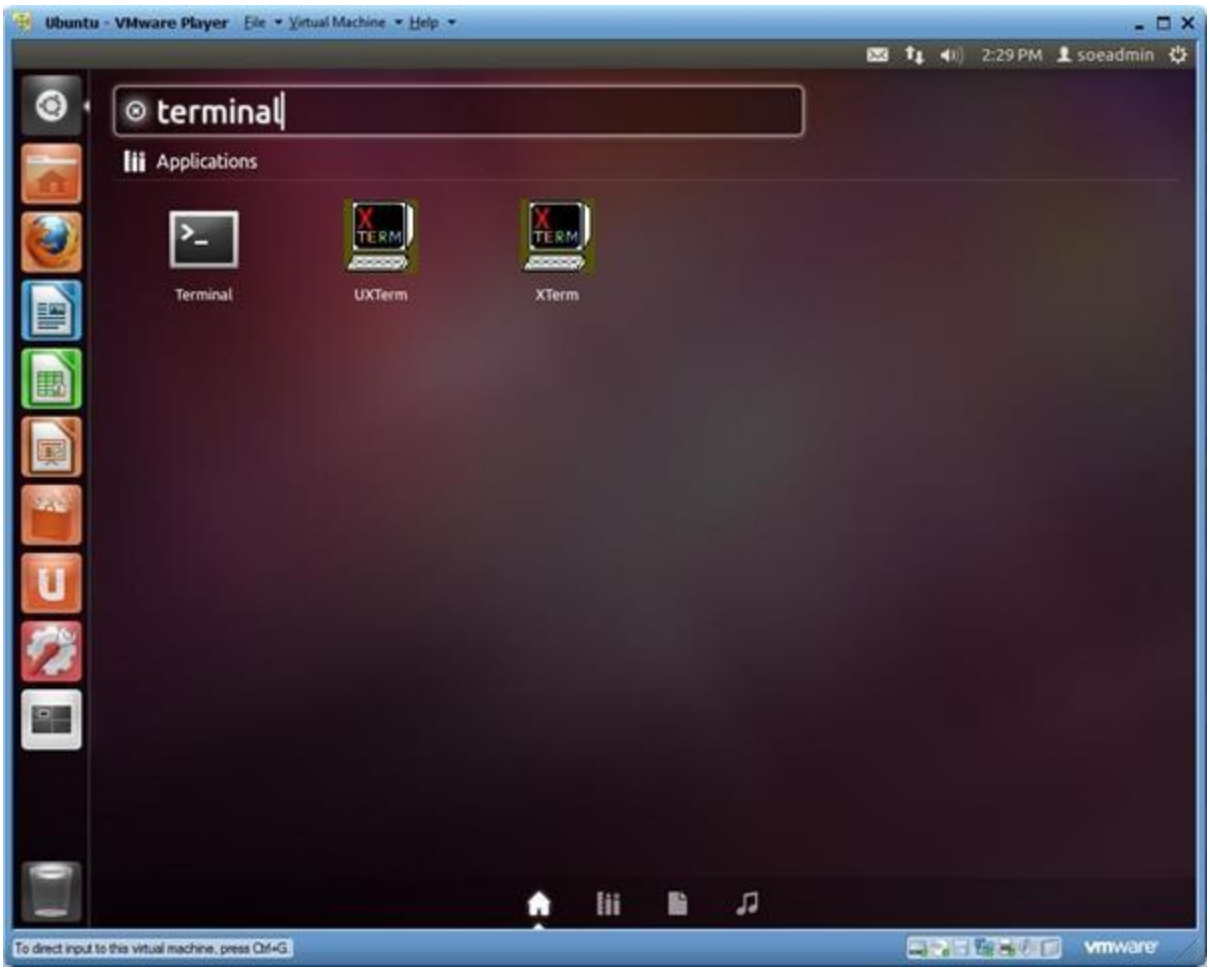

New Virtual Machine Wizard
Specify Disk Capacity
How large do you want this disk to be?
The virtual machine's hard disk is stored as one or more files on the host computer's physical disk. These file(s) start small and become larger as you add applications, files, and data to your virtual machine.
Maximum disk size (GB):
20.0
Recommended size for Ubuntu 64-bit: 20 GB
Store virtual disk as a single file
Split virtual disk into multiple files
Splitting the disk makes it easier to move the virtual machine to another computer but may reduce performance with very large disks.
Help
< Back
Next >
Cancel

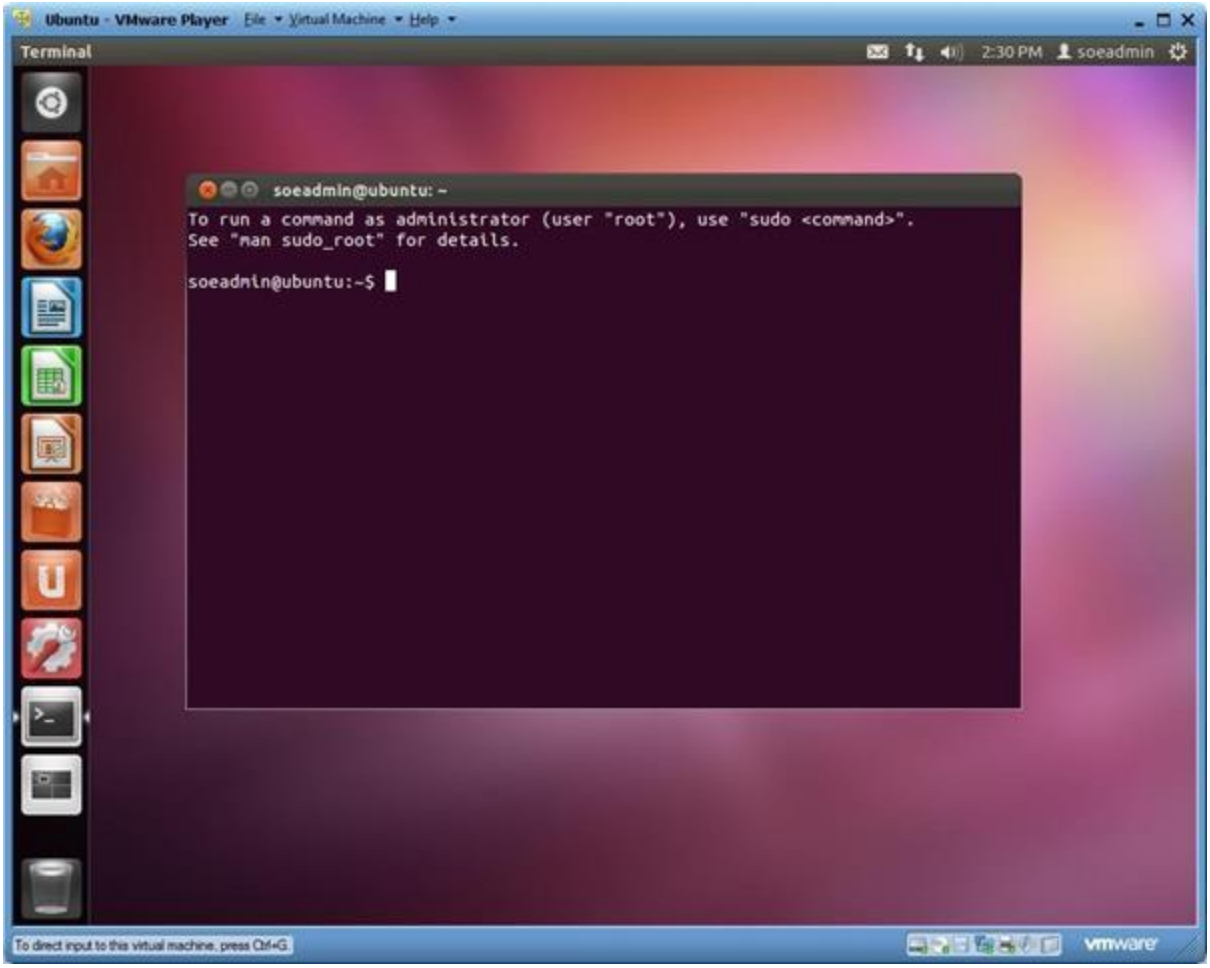

Software Updates
Downloading VMware Tools for Linux...
Details >>
Cancel Updating
Hide


Ubuntu - VMware Player
File
Virtual Machine
Help
Install
Welcome to Ubuntu
Ubuntu is packed with features to make your computer delightful to use. This version consolidates the Unity desktop interface; a brand new way to find and manage your applications. Now with more features and faster. We'd like to show some cool things to look for in your new operating system.
Copying files...
Click in the virtual screen to send keystrokes
Easy Install is installing Ubuntu.
Help
To direct input to this virtual machine, press Ctrl+G.
vmware


Additional Drivers
No proprietary drivers are in use on this system.
VMware Virtual Machine Communication Interface (VMCI).
VMWare Client Tools
VMWare Client Tools
Tested by the Ubuntu developers
License: Free
Install VMWare client drivers and tools
Install the VMWare client drivers and toolsfor your VMWare based Ubuntu installation.
This should help you use Ubuntu in your VM.
This driver is not activated.
Activate
Help
Close


Additional Drivers
Downloading and installing driver...
Cancel


Ubuntu - VMware Player
File
Virtual Machine
Help
2:29 PM
soeadmin
terminal
Applications
Terminal
UXTerm
XTerm
To direct input to this virtual machine, press Ctrl+G.
vmware



حالا سیستم‌عامل Ubuntu نصب شده است.

۱-٤ سوال

- هشت توزیع مختلف لینوکس را توضیح دهید و کاربردهای برجسته هر یک را عنوان نمایید.
- مزایا و معایب استفاده از ماشین‌های مجازی چیست؟
- چرا پیشنهاد می‌شود که کاربرهای آماتور از کاربر root استفاده نکنند؟
- لینوکس را با ویندوز مقایسه کنید و مزایا و معایب هر یک را بیان کنید.

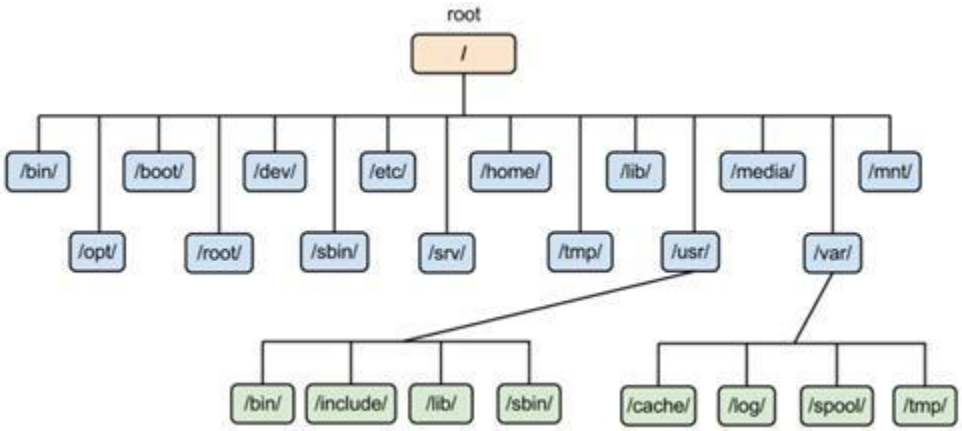
آزمایش دوم: سیستم فایل: مدیریت دایرکتوری

۲-۱ پیش‌آگاهی

سیستم فایل، یکی از بخش‌های مهم هر سیستم‌عاملی است. در این آزمایش و آزمایش بعدی با سیستم فایل سیستم‌عامل لینوکس آشنا می‌شوید.

۲-۱-۱ سیستم فایل (File System)

سیستم‌عامل Linuxنیز همانند همهٔ سیستم‌های عامل‌های دیگر به طرز چشمگیری بر اطلاعات ذخیره شده در پرونده‌ها تکیه می‌کند: اطلاعات کاربران مختلف، پرونده‌های اجرایی مورد نیاز کاربران، پرونده‌های داده‌ای مربوط به آن‌ها، کتابخانه‌های مورد نیاز برای برنامه‌نویسی، اطلاعات مربوط به تنظیم‌های سخت‌افزاری و امکانات موجود در سیستم، کُد اجرایی خود سیستم‌عامل و بسیاری اطلاعات دیگر همگی به صورت پرونده ذخیره می‌شوند. بنابراین با توجه به اهمیت و حساسیت اطلاعات فوق الذکر لازم است که این پرونده‌ها تحت یک نظام قوی و قابل اطمینان مدیریت و نگهداری شوند. در سیستم‌های عامل، انجام این وظایف بر عهدهٔ سیستم پرونده است. مثلاً در DOS سیستم پروندهٔ FAT و در سیستم‌عامل Windows، سیستم پروندهٔ NTFS برای این کار طراحی شده‌اند. گونه‌هایی از Unix که قبل از BSD نگارش ۴/۲ ایجاد شده‌اند، هر یک سیستم پروندهٔ مربوط به خود را داشتند. یکی از ویژگی‌های جالب توجه سیستم‌عامل Linux در نگارش‌های System V Release 4 به بعد این است که سیستم پرونده آن انواع سیستم‌های پروندهٔ موجود را می‌شناسد و قادر است اطلاعات موجود در پرونده‌هایشان را بخواند. سیستم‌های پروندهٔ پراستفاده در Unix عبارتند از: Extended File System 2 و System V File System. صرف نظر از نوع، سیستم پرونده باید اطلاعاتی را که سیستم‌عامل برای شناسایی کامل یک پرونده نیاز دارد مهیا کند.



سیستم فایل در linux برخلاف سیستم فایل در ویندوز که می‌تواند شامل چند درایو منطقی باشد (نظیر:a:،b:،c: و...)، تنها شامل یک بخش است که آن هم چون فقط یکی است، نام خاصی ندارد. شاخه ریشه این درایو با علامت / شناخته می‌شود و تمام فایل‌ها و

دایرکتوری‌های linuxدرشاخه/ قرار می‌گیرند. شاخه‌های اصلی سیستم که مستقیما درشاخه / قرار دارند، عبارتنداز: ,etc, lib, home bin ,root , mnt وچند شاخه دیگر. در اینجا ما تنها اشاره‌ای به شاخه‌های root و home می‌کنیم. ساختار درختی دایرکتوری‌های لینوکس در شکل ۲-۱ نمایش داده شده است. در آزمایش پنجم، جزئیات بیش‌تر این ساختار بررسی خواهد شد.

شکل ۲-۱: ساختار درختی دایرکتوری‌ها در لینوکس

برخی از دستورات لینوکس در مدیریت دایرکتوری‌ها در جدول ۲-۱ نمایش داده شده‌اند.

جدول ۲-۱: دستورات لینوکس برای مدیریت دایرکتوری‌ها

توضیحات	قالب فرمان
تغییر شاخهٔ جاری	path **cd**
گرفتن لیست پرونده‌ها و شاخه‌ها	path **ls**
انتقال پرونده‌ها وشاخه‌ها	newpath path/filename **mv**
کپی کردن پرونده‌ها وشاخه‌ها	target-file source-file **cp**
حذف شاخه‌ها	directory-name **rmdir**
ایجاد شاخه‌ها	directory-name **mkdir**
یافتن پرونده‌ها و نمایش آن‌ها	**print** -filename **name**- path **find**
دیدن محتویات پرونده‌ها	path/filename **cat**
حذف پرونده‌ها	path/filename **rm**
دیدن صفحه به صفحهٔ پرونده‌ها	path/filename **more**
نمایش مسیرکامل شاخه فعلی	**pwd**

Manual Pages

- man $command
- man $configfile
- man $daemon
- man -k (apropos)

- whatis
- whereis
- man man

Working with directories

- pwd
- cd
- cd $directory
- cd or cd ~
- cd ..
- cd .
- cd -


دایرکتوری جاری مکان شروع حساب می‌شود و نیازی به نوشتن / در مسیر نیست.

- ls
- -a --all all files even hidden ones
- -d --directory folder details
- -h --human readble
- -l long format
- -r --reverse decreasing alphabet order
- -s size
- -t last time of modifying
- ls
- ls -a
- ls -l
- ls -lh

- mkdir mydir

- mkdir -p
- rmdir emptydirectory
- rmdir -p

۲-۲ دستورکار

۱) اطلاعات داخل (لیست) دایرکتوری جاری خود را نمایش دهید. به زیر شاخه etc/ بروید و سپس دستور cd را بدون پارامتر اجرا کنید. چه اتفاقی می‌افتد؟

۲) به دایرکتوری etc/ تغییر مکان دهید و دستور ls را با پارامترهای زیر اجرا کنید (سعی کنید با استفاده از مشاهدات خود و دستور man، تاثیر هر پارامتر را بدست آورید).

ls –a

ls –l

ls –lh

۳) دوباره به دایرکتوری etc/ تغییر مکان دهید و با استفاده از فشردن تنها سه کلید از صفحه کلید، به دایرکتوری home بروید.

۴) با استفاده از فشردن تنها ۱۱ بار کلیدهای صفحه کلید، به دایرکتوری boot/grub/ بروید.

۵) به دایرکتوری والد دایرکتوری جاری بروید.

۶) به دایرکتوری ریشه بروید.

۷) محتویات داخل دایرکتوری root را لیست کنید.

۸) در دایرکتوری جاری باقی بمانید و لیست دایرکتوری etc/ را نمایش دهید.

۹) در دایرکتوری جاری باقی بمانید و لیست دایرکتوری‌های bin/ و sbin/ را نمایش دهید.

۱۰) کلیه فایل‌های موجود در دایرکتوری home (شامل فایل‌های مخفی) را نمایش دهید.

۱۱) فایل‌های موجود در دایرکتوری boot/ را به گونه‌ای نمایش دهید که برای انسان خواناتر باشد.

۱۲) یک دایرکتوری با نام testdir در دایرکتوری home خود ایجاد کنید. یک فایل متنی خالی در این مسیر قرار دهید.

۱۳) به دایرکتوری etc/ منتقل شوید، در آنجا بمانید و در home، دایرکتوری جدیدی به نام newdir ایجاد کنید.

۱۴) سه دایرکتوری ~/dir1/dir2/dir3 را به کمک یک دستور ایجاد کنید.

۱۵) دایرکتوری testdir را حذف کنید.

۱۶) تفاوت help،man و info چیست؟

آزمایش سوم: سیستم فایل: مدیریت فایل‌ها

۱-۳ پیش‌آگاهی

در آزمایش قبل با نحوه مدیریت دایرکتوری‌ها آشنا شدید. در این آزمایش با نحوه مدیریت فایل‌ها در لینوکس آشنا می‌شوید. مدیریت فایل‌ها شامل ایجاد، مشاهده، ویرایش، حذف، کپی و انتقال فایل‌ها است. در جدول ۳-۱ لیستی از دستورات لینوکس در مدیریت فایل‌ها نمایش داده شده است.

جدول ۳-۱: دستورات لینوکس برای مدیریت فایل‌ها

قالب فرمان	توضیحات
file filename	
touch filename	
rm filename	
rm -rf	
cp source targetfile/targetdirectory	
cp -r	
cp –i	
mv trgtfile/trgtdir newname	
head filename	
head -n filename	
head -cn filename	
tail filename	
tail -n filename	
cat filename	
cat filename1 filename2 …	

۱-۱-۳ انواع فایل‌ها در لینوکس

در Linux انواع مختلفی از پرونده‌ها وجود دارد. به هر نوع یک حرف کوچک انگلیسی متناظر اختصاص داده شده است. کاربرد این حروف را بعدا در

دستور ls خواهید دید. انواع مختلف پرونده به همراه حروف متناظرشان در جدول ۳-۱ فهرست شده‌اند.

جدول ۳-۱: انواع پرونده‌های موجود در Linux

نوع پرونده	حرف متناظر
ordinary file	-
directory	d
fifo	p
character device	c
block device	b
link to another file	l

نوع پرونده ordinary file، اغلب شامل پرونده‌های متن و پرونده‌های داده‌ای می‌شود. مثلاً پرونده‌های برنامه‌های زبان C از این نوع هستند.

نوع پرونده directory نیز که برای کاربران DOS و Windows، نوعِ پرونده شناخته شده‌ای است، برای دسته‌بندی مجموعه‌های پرونده استفاده می‌شود.

نوع پرونده fifo نوعی پرونده است که برای برقراری ارتباط بین فرآیندها استفاده می‌شود. یکی از راه‌های تبادل اطلاعات بین فرآیندها در Linux، استفاده از fifo است. ویژگی این کانال ارتباطی این است که پس از قطع ارتباط بین فرآیندها از بین نمی‌رود و پایدار باقی می‌ماند.

قبل از توضیح دادن نوع‌های character device و block device لازم است مطالبی راجع به نحوهٔ استفادهٔ Linux از سخت‌افزار بدانید. سیستم‌عامل Linux با هر سخت‌افزاری به صورت یک پرونده برخورد می‌کند. مثلاً ارسال یک بلوک داده به چاپگر معادل نوشتن آن بلوک داده در پروندهٔ متناظر با چاپگر است. به این پرونده، پروندهٔ دستگاه (device file) گفته می‌شود. عملیات ورودی و خروجی در سخت‌افزارهای مختلف به دو صورت انجام می‌شود:

- ارسال و دریافت بایت به بایت داده‌ها.

- ارسال و دریافت بلوکی داده‌ها (در یک انتقال بیش از یک بایت منتقل شود).

متناظر با این تقسیم بندی، پرونده‌های دستگاه نیز به دو دسته تقسیم می‌شوند:

- character device: مانند درگاه سری (Serial Port). پرونده‌های این توع را با حرف c نشان می دهند.

- block device: مانند دیسک سخت. پرونده‌های این توع را با حرف b نشان می دهند.

بعدا دربارهٔ این پرونده‌ها مطالب بیشتری خواهیم دانست.

مالک پرونده: مالک پرونده کسی است که پرونده را ایجاد می‌کند. مالک پرونده می‌تواند آن را به مالکیت کاربران دیگر درآورد. مدیر سیستم و کاربران ایجاد کنندهٔ پرونده، نمونه‌هایی از مالکان پرونده هستند.

گروه پرونده: در سیستم‌عامل Linux کاربران به گروه‌هایی تقسیم می‌شوند. برای یک پرونده علاوه بر مالک آن شمارهٔ گروهی از کاربران که می‌توانند به آن پرونده دسترسی داشته باشند نیز نگهداری می‌شود.

اجازه‌های دسترسی (Access Permissions): اجازه دسترسی نحوهٔ دسترسی افراد به پرونده‌ها را مشخص می‌کند. سه دسته از افرادی که می‌توانند به یک پرونده دسترسی داشته باشند، عبارتند از:

- مالک پرونده

- افراد درون گروه مربوط به آن پرونده

- بقیهٔ کاربران

برای هر کدام از افراد فوق سه نوع اجازهٔ دسترسی مطرح می‌شود:

- خواندن (Read)

- نوشتن (Write)

- اجرا کردن (eXecute)

بنابراین همراه اطلاعات هر پرونده، نُه اجازهٔ دسترسی نگهداری می‌شود. صاحب پرونده و همچنین، مدیر سیستم می‌توانند این اجازه‌ها را تغییر دهند. اگر کاربری اجازهٔ نوشتن در یک شاخه را نداشته باشد، آن کاربر نمی‌تواند به پرونده‌های درون آن شاخه دسترسی داشته باشد. برای دیدن اجازه‌های منتسب به یک پرونده دستور ls -l به کار می‌رود. در آزمایش چهارم به صورت مفصل در زمینه اجازه‌های دسترسی صحبت خواهد شد.

تاریخ: تاریخ اعمالِ آخرینِ تغییر و آخرین دسترسی به پرونده نیز در مشخصات آن نگهداری می‌شود.

اندازه: تمام انواع پرونده‌ها به جز پرونده‌های دستگاهی دارای اندازه برحسب بایت هستند. پرونده‌های دستگاهی نیز چون واقعا اطلاعاتی درون خود ندارند، اندازه‌شان صِفر است. برای این نوع پرونده‌ها به جای اندازه، دو عدد صحیح بزرگ‌تر از صفر ذخیره می‌شود که در قسمت هسته (Kernel) به آن‌ها خواهیم پرداخت.

نام پرونده: در نگارش استاندارد Unix، حداکثر طول نام پرونده ۱۴ کاراکتر است، اما بسته به نوع سیستم پرونده، این قابلیت وجود دارد که تا ۲۵۶ کاراکتر برای نام پرونده ذخیره شود. درLinux نام پرونده می‌تواند ۲۵۶ کاراکتر داشته باشد و تنها نویسهٔ غیرمجاز در نام پرونده '/' است. این کاراکتر جداکنندهٔ نام شاخه‌ها در مسیردهی به نام یک پرونده است؛ مثلاً: /usr/X11/bin/xinit.

۲-۱-۳ ویرایشگر vi

در آزمایش اول با ویرایشگر nano، برای ایجاد یک فایل متنی آشنا شدید. این ویرایشگر از لحاظ ورود متن خیلی مشابه ویرایشگرهایی است که در سیستم‌عامل‌های دیگر تجربه کرده‌اید. با این وجود این ویرایشگر در تمام توزیع‌های لینوکس وجود ندارد. در این بخش قصد داریم شما را با ویرایشگر vi آشنا کنیم. این ویرایشگر جزء قوی‌ترین ویرایشگرهایی است که در تمام توزیع‌های لینوکس و تمام نسخه‌های آن‌ها وجود دارد و برخی مواقع تنها ویرایشگری است که می‌توانید استفاده کنید.

با اجرای دستور vi filename وارد ویرایشگر خواهید شد. این ویرایشگر دارای دو حالت کاری، به نام‌های فرمان و درج است. در حالت فرمان، شما می‌توانید فرامینی نظیر حذف یک خط، حذف یک کلمه، رفتن سر کلمه بعدی، جستجوی متن، رفتن به صفحه بعد یا صفحه قبل را به ویرایشگر بدهید. در حالت درج، می‌توان کاراکترها را به متن اضافه کرد. برای خروج از حالت درج، کافی است کلید ESC را فشار دهید. برای رفتن به حالت درج باید اول کلید : و سپس کلید i و سپس کلید Enter را بزنید.

برای ذخیره کردن باید به حالت فرمان بروید و سپس w: را تایپ کنید و سپس کلید Enter بزنید. برای خروج از ویرایشگر باید به حالت فرمان بروید (فشردن کلید ESC) و سپس q: را تایپ کرده و Enter بزنید.

برخی از فرامین ادیتور vi در جدول ۳-۲ لیست شده‌اند.

جدول ۳-۲: برخی از فرامین ویرایشگر vi

توضیحات	فرمان
خروج از ویرایشگر	:q
ذخیره کردن	:w
رفتن به حالت درج	:i

حذف کاراکتر (البته باید در حالت فرمان باشید)	x
حذف خط جاری	dd
لود کردن فایل	:r filename<Enter>
ذخیره کردن در یک فایل تحت یک نام جدید	:w newfile<Return>
ذخیره خط ۱۲ تا ۳۵ در یک فایل جدید	:12,35w smallfile<Return>
جستجوی یک رشته	/string
کپی کردن خط جاری	yy
درج خط کپی شده در محل جاری	p

۲-۳ دستور کار

۱) فایل‌های موجود در دایرکتوری bin/ را نمایش دهید.

۲) نوع فایل‌های /bin/cat ، /etc/passwd و /usr/bin/passwd را نشان دهید.

۳) یک دایرکتوری با نام touched/ در دسکتاپ خود ایجاد کنید و وارد آن دایرکتوری شوید.

۴) فایل‌های today.txt و yesterday.txt را در دایرکتوری ایجاد شده در گام قبل، ایجاد نمایید.

۵) فایل yesterday.txt را در فایلی به نام copy.yesterday.txt کپی کنید.

۶) نام فایل copy.yesterday.txt را به new تغییر دهید.

۷) دایرکتوری دیگری تحت عنوان testbackup/ در دسکتاپ ایجاد کنید و تمام فایل‌های درون دایرکتوری touched را در آن کپی کنید.

۸) از یک دستور استفاده کنید و دایرکتوری testbackup/ را به همراه تمام فایل‌های درون آن حذف کنید.

۹) یک دایرکتوری etcbackup/ روی دسکتاپ ایجاد کنید و تمام فایل‌هایی با پسوند conf. از پوشه etc/ را در آن کپی کنید.

۱۰) دوازده خط اول /etc/services را نمایش دهید.

۱۱) خط آخر /etc/passwd را نمایش دهید.

۱۲) به کمک دستور cat فایلی با نام count.txt ایجاد کنید که محتویات آن به صورت زیر باشد:

one

Two

Three

۱۳) به کمک چه دستورهایی می‌توان محتویات داخل فایل count.txt را در فایل newcount.txt کپی کرد؟

۱۴) با راهنمایی مربی آزمایشگاه، با استفاده از ویرایشگر vi، یک فایل ایجاد کنید و متن زیر را داخل آن تایپ کنید و به نام viexample.txt ذخیره نمایید.

This is a sample Text

What is your opinion about vi text editor?!!!

آزمایش چهارم: سیستم فایل (مجوزهای دسترسی)

۴-۱ پیش‌آگاهی

در این آزمایش با مفهوم مجوزهای دسترسی به فایل‌ها و دایرکتوری‌ها و نحوه تغییر آن‌ها آشنا خواهید شد.

۴-۱-۱ کاربران در لینوکس

همانند سیستم‌عامل ویندوز که دو دسته کاربر دارد (کاربران عادی و مدیرسیستم با نام خاص Administrator)، Linux نیز همین دو دسته کاربر را دارد. در linux نام خاص کاربری برای مدیر سیستم به جای administrator، root است. بنابراین شاخه root/ شاخه مخصوص مدیر سیستم است وlinux اجازه ورود و تغییر محتویات این شاخه را به هیچ کاربری به غیر از مدیر سیستم نمی‌دهد و مدیر سیستم می‌تواند فایل‌ها و دایرکتوری‌های غیرمجاز برای مشاهده دیگران را در این شاخه قرار دهد.

همچنین برای هر کاربر جدیدی (نام کاربری جدید مثل user1) که توسط مدیرسیستم در linux ثبت می‌شود، یک دایرکتوری جدید با نام کاربری او در شاخه home/ ایجاد می‌شود (مثلاً home/user1/) که وقتی کاربر به سیستم login می‌کند، این شاخه به عنوان شاخه جاری او می‌باشد (که البته قابل تغییراست). قابل توجه است که اگر مثلاً کاربرuser1 فایل‌ها و دایرکتوری‌هایی رادرشاخه home/user1/ ذخیره کند، هیچ کاربر دیگری اجازه ورود به این شاخه و دیدن محتویات آن را ندارد مگرroot که اجازه ورود و تغییرهمه دایرکتوری‌های سیستم را دارد. در سیستم‌عامل لینوکس، برای هر پرونده، دو مفهوم مالک و گروه تعریف خواهد شد که توضیحات آن به صورت زیر است:

مالک پرونده: مالک پرونده کسی است که پرونده را ایجاد می‌کند. مالک پرونده می‌تواند آن را به مالکیت کاربران دیگر درآورد. مدیر سیستم و کاربران ایجاد کنندهٔ پرونده، نمونه‌هایی از مالکان پرونده هستند.

گروه پرونده: در سیستم‌عامل Linux کاربران به گروه‌هایی تقسیم می‌شوند. برای یک پرونده، علاوه بر مالک آن شمارهٔ گروهی از کاربران که می‌توانند به آن پرونده دسترسی داشته باشند نیز نگهداری می‌شود.

اجازه‌های دسترسی (Access Permissions): نحوهٔ دسترسی افراد به پرونده‌ها را مشخص می‌کند. سه دسته از افرادی که می‌توانند به یک پرونده دسترسی داشته باشند، عبارتند از:

مالک پرونده

افراد درون گروه مربوط به آن پرونده

بقیهٔ کاربران

برای هر کدام از افراد فوق سه نوع اجازهٔ دسترسی مطرح می‌شود:

خواندن (Read)

نوشتن (Write)

اجرا کردن (eXecute)

بنابراین همراه اطلاعات هر پرونده ۹ اجازهٔ دسترسی نگهداری می‌شود. صاحب پرونده و همچنین، مدیر سیستم می‌توانند این اجازه‌ها را تغییر دهند. اگر کاربری اجازهٔ نوشتن در یک شاخه را نداشته باشد، آن کاربر نمی‌تواند به پرونده‌های درون آن دسترسی داشته باشد. برای دیدن اجازه‌های منتسب به یک پرونده دستور ls -l را بکار ببرید.

۴-۱-۲ نحوه تغییر اجازه‌های دسترسی

با استفاده از دستور chmod (Change-mode) می‌توان اجازه‌های دسترسی مربوط به یک فایل یا دایرکتوری را تغییر داد. ساختار دستوری آن به صورت زیر است:

chmod [*references*][*operator*][*modes*] *file1* ...

در ساختار این دستور، چهار نوع پارامتر مختلف وجود دارد. پارامتر اول references است که نشان‌دهنده کلاس کاربری است که قرار است سطح دسترسی آن تغییر کند. همان‌طورکه گفته شد، سه کلاس کاربری موجود است: user (که به اختصار با حرف u مشخص می‌شود)، group (که به اختصار با حرف g مشخص می‌شود) و others (که به اختصار با حرف o مشخص می‌شود). اگر کلاس کاربر در دستور مشخص نشد، به صورت پیش‌فرض کلاس all یا همه در نظر گرفته می‌شود (که به اختصار با حرف a مشخص می‌شود). یعنی تغییرات برای همه سطوح کاربری اعمال می‌شود. در جدول ۴-۱ مشخصات کلاس کاربری و نحوه نمایش مختصر آن آورده شده است.

جدول ۴-۱: مشخصات کلاس کاربری و نحوه نمایش مختصر آن

Reference	Class	Description
u	user	the owner of the file
g	group	users who are members of the file's group
o	others	users who are not the owner of the file or members of the group
a	all	all three of the above, is the same as *ugo*

پارامتر دوم، عملگر یا operator است که دستور chmod از آن برای تنظیم مد فایل استفاده می‌کند. از علامت + برای اضافه کردن یک مد خاص به مدهای موجود فایل، از علامت – برای حذف یک مد خاص از مجموعه مدهای موجود فایل و از علامت = برای انتساب مد به فایل استفاده می‌شود. در جدول زیر علامت‌های مربوط به عملگر و کاربردشان آورده شده است.

جدول ۴-۲: علامت‌های مربوط به عملگر و کاربرد آن‌ها

Operator	Description
+	adds the specified modes to the specified classes
-	removes the specified modes from the specified classes
=	the modes specified are to be made the exact modes for the specified classes

پارامتر سوم، مد است که مشخص‌کننده نوع دسترسی به فایل است و به نوع مشخصی از کاربران داده می‌شود یا از آن‌ها گرفته می‌شود. سه نوع مد تعریف شده است که عبارتند از: خواندن (که به اختصار با حرف r نشان داده می‌شود)، نوشتن (که به اختصار با حرف w نشان داده می‌شود) و اجرا (که با اختصار با حرف e نشان داده می‌شود). در جدول زیر مدها و مشخصات آن‌ها آورده شده است.

جدول ۴-۳: معرفی مدها و مشخصات آن‌ها

Mode	Name	Description
r	read	read a file or list a directory's contents
w	write	write to a file or directory
x	execute	execute a file or recurse a directory tree

پارامتر سوم، نام فایلی (یا فایل‌هایی) است که باید مد آن تغییر کند. در ادامه نمونه‌هایی از نحوه به کارگیری دستور chmode آورده شده است:

Ø chmod g+r myfile

Ø chmod u-w myfile

Ø chmod g+r-w myfile

Ø chmod a=rw myfile

Ø chmod ug=rw myfile

Ø chmod =rw myfile

برای حالتی که از عملگر = استفاده می‌شود (یعنی می‌خواهیم ویژگی یا ویژگی‌هایی به فایل منتسب کنیم)، می‌توانیم از معادل باینری دستورات نیز استفاده کنیم. به این صورت که از چپ به راست کاربر user، group و other قرار می‌گیرد. برای هر کدام از کلاس کاربری سه نوع مد خواندن و نوشتن و اجرا از چپ به راست در نظر گرفته می‌شود. ۱ معادل با اجرا، ۲ معادل با نوشتن و ۴ معادل با خواندن است. برای هر کلاس کاربری، مقدار باینری محاسبه می‌شود و مقدار باینری محاسبه شده، جلوی دستور chmod قرار می‌گیرد. در جدول زیر مقادیر باینری و مدهای منتسب شده به کلاس‌های کاربری آورده شده است.

جدول ۴-۴: مقادیر باینری و مدهای منتسب شده به کلاس‌های کاربری

Symbolic Notation	Numeric Notation	English
`----------`	0000	no permissions
`-rwx------`	0700	read, write, & execute only for owner
`-rwxrwx---`	0770	read, write, & execute for owner and group
`-rwxrwxrwx`	0777	read, write, & execute for owner, group and others SECURITY RISK
`---x--x--x`	0111	execute
`--w--w--w-`	0222	write
`--wx-wx-wx`	0333	write & execute
`-r--r--r--`	0444	read
`-r-xr-xr-x`	0555	read & execute
`-rw-rw-rw-`	0666	read & write
`-rwxr-----`	0740	owner can read, write, & execute; group can only read; others have no permissions

۲-۴ دستور کار

۱) فولدری با نام test ایجاد کنید و به user و group آن حق نوشتن و اجرا را اضافه نمایید.

۲) برای تمامی کاربرانِ فولدر ایجاد شده، تنها حق خواندن را ایجاد کنید.

۳) برای فولدر test به user اجازه خواندن، نوشتن و اجرا و به group اجازه خواندن و اجرا و به other تنها اجازه خواندن را بدهید.

۴) دستورات زیر به user، group و other چه حق دسترسی‌هایی می‌دهد؟

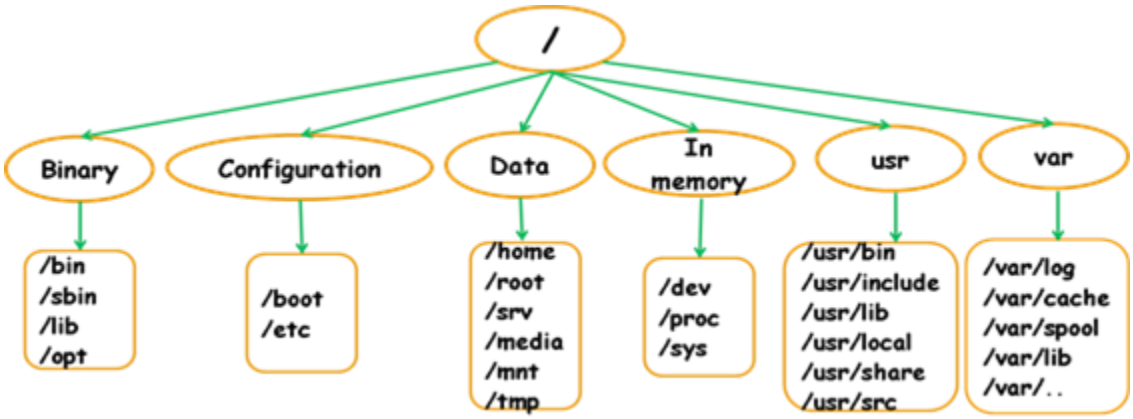
- Chmod 775
- Chmod 750
- Chmod 644
- Chmod 640
- Chmod 600
- Chmod 700

آزمایش پنجم: سیستم فایل (سلسله مراتب فایل‌ها)

۵-۱ پیش‌آگاهی

بیش‌تر توزیع‌های لینوکس از استاندارد سلسله مراتبی فایل سیستمی (File system Hierarchy Standard-FHS) پیروی می‌کنند. این استاندارد ساختار فایل، در توزیع‌های مختلف یونیکس و لینوکس را مشابه می‌کند که برای توسعه‌های آتی مفید است. استاندارد FHS به صورت آنلاین در در سایت http://www.pathname.com/fhs/ موجود است.

در توزیع‌های مختلف لینوکس، میان فایل سیستم تفاوت‌هایی وجود دارد. برای اطلاع یافتن از سلسله مراتب فایل سیستم در توزیع، از دستور man hier استفاده می‌شود. این راهنما درباره ساختار دایرکتوری روی سیستم موجود توضیح می‌دهد. در شکل ۵-۱، ساختار فایل رایج سیستم‌های لینوکسی آورده شده است. همه فایل‌ها، زیرمجموعه ریشه (/) هستند. سایر فایل‌ها را می‌توان به یکی از شش دسته binary، configuration، data، in memory، usr و var طبقه‌بندی کرد. در ادامه هر یک از این دسته‌ها به طور کامل بررسی می‌شوند.



شکل ۵-۱: ساختار فایل رایج سیستم‌های لینوکسی

۵-۱-۱ دایرکتوری ریشه (/)

ساختار فایلِ تمامی توزیع‌های لینوکس، با دایرکتوری ریشه شروع می‌شود. دایرکتوری ریشه با نماد / نشان داده می‌شود. تمامی فایل‌ها در لینوکس را می‌توان در ذیل این دایرکتوری پیدا کرد. در شکل ۵-۲، به محتویات موجود در دایرکتوری ریشه توجه کنید.

شکل ۵-۲: محتویات موجود در دایرکتوری ریشه

سوالی که ممکن است به ذهن برسد این است که رنگ‌های مختلف فایل‌ها به چه معنا است. همان‌طور که در سیستم‌عامل ویندوز، پسوند فایل نشان‌دهنده نوع فایل است، در لینوکس از روی رنگ فایل‌ها نیز می‌توان نوع فایل را تشخیص داد. رنگ آبی نشان‌دهنده دایرکتوری، رنگ سبز نشان‌دهنده فایل اجرایی، رنگ آبی آسمانی نشان‌دهنده فایل‌های ارجاعی، رنگ صورتی نشان‌دهنده فایل تصویری، رنگ قرمز نشان‌دهنده فایل زیپ شده، رنگ زرد با پس زمینه سیاه نشان‌دهنده درایور و رنگ سفید چشمک‌زن با پس زمینه قرمز نشان‌دهنده فایل ارجاعی نامعتبر است.

۵-۱-۲ دایرکتوری binary

فایل‌های باینری، فایل‌هایی هستند که شامل سورس کدهای کامپایل شده (کدهای ماشین) هستند. فایل‌های باینری را می‌توان روی کامپیوتر اجرا کرد. چهار نوع فایل نیز در این شاخه قرار می‌گیرند که در ادامه توضیح داده می‌شوند.

۵-۱-۲-۱ /bin

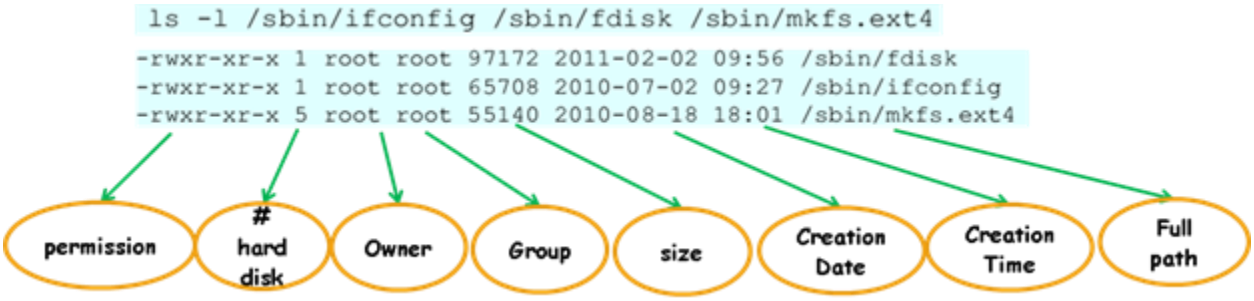
دایرکتوری /bin شامل دستورات باینری برای استفاده تمامی کاربران است. با توجه به استاندارد FHS، دایرکتوری /bin باید دارای /bin/cat و /bin/date باشد. در شکل ۵-۳، دستورات رایج در لینوکس مانند cat و sleep و ... در دایرکتوری /bin دیده می‌شود.

فولدر /bin را می‌توان در دایرکتوری‌های دیگری نیز دید. به عنوان مثال کاربری با نام shfz را در نظر بگیرید. این کاربر می‌تواند برنامه‌های خود را در مسیر /home/shfz/bin قرار دهد.

شکل ۵-۳: دستورات رایج در لینوکس مانند cat و sleep و ... در دایرکتوری /bin

۵-۱-۲-۲ /sbin

شامل دستورات باینری برای تنظیمات سیستم‌عامل است. بسیاری از دستورات باینری سیستمی نیاز به سطح دسترسی root دارند تا بتوانند وظیفه خاصی را انجام دهند. در شکل ۵-۴، دستورات باینری سیستمی برای تغییر آدرس IP، پارتیشن‌بندی دیسک و ایجاد یک فایل سیستم از نوع ext4 نشان داده شده است. اطلاعات کامل این دستورات که در شکل ۵-۴ نشان داده شده است، با استفاده از دستور ls –l اطلاعات قابل دسترسی است.



شکل ۵-۴: دستورات باینری سیستمی برای تغییر آدرس IP، پارتیشن‌بندی دیسک و ایجاد یک فایل سیستم از نوع ext4

۵-۱-۲-۳ /lib

دستورات باینری موجود در /bin و /sbin از کتابخانه‌هایی استفاده می‌کنند که در مسیر /lib قرار گرفته است. در زیر محتویات فولدر /lib نشان داده شده است.

```
paul@laika:~$ ls /lib/libc*
/lib/libc-2.5.so     /lib/libcfont.so.0.0.0  /lib/libcom_err.so.2.1
/lib/libcap.so.1     /lib/libcidn-2.5.so     /lib/libconsole.so.0
/lib/libcap.so.1.10  /lib/libcidn.so.1       /lib/libconsole.so.0.0.0
/lib/libcfont.so.0   /lib/libcom_err.so.2    /lib/libcrypt-2.5.so
```

۵-۱-۲-٤ /opt

از /opt برای ذخیره نرم‌افزارهای نصب شده (optional software) استفاده می‌شود. بیش‌تر اوقات این نرم‌افزارها در منبع نرم‌افزارهای نصبی وجود ندارند. در بسیاری از سیستم‌ها، دایرکتوری /opt خالی است. پکیج‌های بزرگ می‌توانند تمامی فایل‌هایشان را در زیر فولدرهای /bin، /lib و /etc فولدر /opt/$packagename ذخیره کنند. به عنوان مثال اگر پکیجی shfz نام داشته باشد در دایرکتوری /opt/shfz فایل‌های نصبی خود را قرار می‌دهد و مثلا دستورات باینری خود را در /opt/shfz/bin قرار می‌دهد.

۵-۱-۳ دایر کتوری configuration

۵-۱-۳-۱ /boot

دایرکتوری /boot شامل تمامی فایل‌هایی است که برای بوت شدن سیستم لازم است. این فایل‌ها غالبا چندان تغییر نمی‌کنند. در سیستم‌های لینوکسی معمولا دایرکتوری /boot/grub وجود دارد. این دایرکتوری شامل /boot/grub/grub.cfg است (در سیستم‌های قدیمی‌تر، این دایرکتوری شامل /boot/grub/grub.conf است) که منوی بوتی را تعریف می‌کند که قبل از شروع هسته نمایش داده می‌شود.

/etc ۲-۳-۱-۵

تمامی فایل‌های مربوط به تنظیمات خاص سیستم باید در دایرکتوری /etc قرار بگیرند. دایرکتوری /etc مخفف etcetera است. نام فایل تنظیمات غالبا مشابه نام برنامه، daemon یا پروتکل است که پسوند conf. به آن اضافه می‌شود.

```
paul@laika:~$ ls /etc/*.conf
/etc/adduser.conf        /etc/ld.so.conf       /etc/scrollkeeper.conf
/etc/brltty.conf         /etc/lftp.conf        /etc/sysctl.conf
/etc/ccertificates.conf  /etc/libao.conf       /etc/syslog.conf
/etc/cvs-cron.conf       /etc/logrotate.conf   /etc/ucf.conf
/etc/ddclient.conf       /etc/ltrace.conf      /etc/uniconf.conf
/etc/debconf.conf        /etc/mke2fs.conf      /etc/updatedb.conf
/etc/deluser.conf        /etc/netscsid.conf    /etc/usplash.conf
/etc/fdmount.conf        /etc/nsswitch.conf    /etc/uswsusp.conf
/etc/hdparm.conf         /etc/pam.conf         /etc/vnc.conf
/etc/host.conf           /etc/pnm2ppa.conf     /etc/wodim.conf
/etc/inetd.conf          /etc/povray.conf      /etc/wvdial.conf
/etc/kernel-img.conf     /etc/resolv.conf
```

۴-۱-۵ دایرکتوری data (بعدی ها زیر این هستند؟)

/home ۱-٤-۱-۵

کاربران می‌توانند فایل‌ها و داده‌های خود را در دایرکتوری /home ذخیره کنند. معمول است که نام کاربران به صورت یک دایرکتوری در /home قرار گیرد و هر کاربر بتواند فایل‌های خود را در دایرکتوری مربوط به خود ذخیره کند. به عنوان مثال:

```
paul@ubu606:~$ ls /home
geert  annik  sandra  paul  tom
```

علاوه بر این‌که /home یک دایرکتوری برای ذخیره اطلاعات شخصی هر کاربر روی سیستم است، در آن پروفایل کاربر نیز ذخیره می‌گردد. یک پروفایل یونیکسی رایج شامل تعدادی فایل مخفی است (فایل‌هایی که اسمشان با نقطه شروع می‌شود). فایل‌های مخفی پروفایل کاربر یونیکس شامل تنظیمات خاص برای آن کاربر است.

```
paul@ubu606:~$ ls -d /home/paul/.*
/home/paul/.              /home/paul/.bash_profile  /home/paul/.ssh
/home/paul/..             /home/paul/.bashrc        /home/paul/.viminfo
/home/paul/.bash_history  /home/paul/.lesshst
```

۵-۱-٤-۲ /root

در بسیاری از سیستم‌ها root/ مکان پیش‌فرض برای داده‌های شخصی و پروفایل کاربر root است. اگر چنین دایرکتوری وجود ندارد administratorها باید آن را ایجاد کنند.

۵-۱-٤-۳ /srv

از دایرکتوری srv/ برای داده‌هایی استفاده می‌شود که توسط سیستم شما مورد استفاده قرار می‌گیرند. طبق استاندارد FHS، داده‌های از نوع cv، rsync، ftp و www در این‌جا ذخیره می‌شوند. هم‌چنین از نام‌گذاری دایرکتوری به نام کاربر administrator نیز پشتیبانی می‌شود.

۵-۱-٤-٤ /media

دایرکتوری media/ به عنوان نقطه اتصال (mount point) برای اتصال دستگاه‌های قابل حمل مانند CDROM، دوربین و سایر دستگاه‌هایی که با usb به سیستم وصل می‌شوند، استفاده می‌شود. دایرکتوری media/ تقریبا تازه به دنیای یونیکس وارد شده و بدون استفاده از این دایرکتوری نیز می‌توان از سیستم استفاده کرد (مثلا در Solaris9 و Solaris10). اما امروزه بیش‌تر سیستم‌ها از این دایرکتوری برای نقطه اتصال دستگاه‌های قابل حمل استفاده می‌کنند.

```
paul@debian5:~$ ls /media/
cdrom  cdrom0  usbdisk
```

۵-۱-٤-۵ /mnt

دایرکتوری mnt/ باید خالی باشد و با توجه به استاندارد FHS از آن تنها برای نقاط اتصال موقتی استفاده می‌شود. کاربران admin دایرکتوری‌های زیادی را در این سمت ایجاد می‌کنند تا بتوان از آن‌ها برای سیستم فایل‌های محلی و remote مختلفی استفاده کرد.

۵-۱-٤-٦ /tmp

کاربران و برنامه‌های مختلف از tmp/ برای ذخیره داده‌های موقتی به هنگام لزوم استفاده می‌کنند. داده‌ای که در tmp/ قرار می‌گیرد، ممکن است روی دیسک سخت یا RAM ذخیره شود. داده ذخیره شده توسط سیستم‌عامل مدیریت می‌شود. از دایرکتوری tmp/ برای ذخیره داده‌های مهم به هیچ وجه استفاده نشود.

۵-۱-۵ دایرکتوری in memory

۵-۱-۵-۱ /dev

فایل‌های دستگاه (device files) در دایرکتوری /dev قرار می‌گیرند، اما واقعا روی دیسک سخت قرار نگرفته‌اند. فایل‌های هسته شناسایی سخت‌افزار در این دایرکتوری قرار می‌گیرند. تعدادی از دستگاه‌های فیزیکی معمول در ادامه معرفی می‌شوند.

سخت‌افزارهای رایجی همچون دیسک سخت، توسط فایل‌های دستگاه در /dev نشان داده می‌شوند. در زیر، نمونه‌ای از فایل‌های دستگاه SATA روی لپ تاپ و پارتیشن‌های IDE متصل به لپ تاپ آورده شده است.

```
#
# SATA or SCSI or USB
#
paul@laika:~$ ls /dev/sd*
/dev/sda  /dev/sda1  /dev/sda2  /dev/sda3  /dev/sdb  /dev/sdb1  /dev/sdb2

#
# IDE or ATAPI
#
paul@barry:~$ ls /dev/hd*
/dev/hda  /dev/hda1  /dev/hda2  /dev/hdb  /dev/hdb1  /dev/hdb2  /dev/hdc
```

علاوه بر نمایش سخت‌افزارهای فیزیکی، برخی از فایل‌های دستگاه، خاص و بسیار کاربردی هستند. برای مثال /dev/tty1 یک ترمینال یا کنسول متصل به سیستم را نشان می‌دهد (منظور یک واسط خط فرمان است). هنگامی‌که دستورات را در خط فرمان (در واقع بخشی از واسط گرافیکی مانند Gnome یا KDE است) می‌نویسید، ترمینال شما به صورت /dev/pts/1 (عدد ۱ می‌تواند تغییر کند) نمایش داده می‌شود.

از دیگر فایل‌های خاص، /dev/null است. می‌توان آن را مانند یک حفره سیاه در نظر گرفت که فضای ذخیره‌سازی نامحدودی دارد اما هیچ چیز از آن قابل بازیابی نیست. به صورت تخصصی‌تر هر چیزی که در این بخش نوشته می‌شود، نادیده (discarded) گرفته می‌شود. از این دایرکتوری زمانی استفاده شود که بخواهیم خروجی ناخواسته از دستورات، نادیده گرفته شود. تاکید می‌شود /dev/null مکان مناسبی برای ذخیره اطلاعات نیست.

۵-۱-۵-۲ /proc ارتباط با هسته

دایرکتوری /proc از جمله دایرکتوری‌های خاص است که فضایی از دیسک سخت نمی‌گیرد. در واقع دایرکتوری /proc یک چشم‌انداز از هسته است. این دایرکتوری آن‌چه که هسته مدیریت می‌کند را نشان می‌دهد و از آن برای تعامل مستقیم با هسته می‌توان استفاده کرد.

```
paul@RHELv4u4:~$ mount -t proc
```

```
paul@RHELv4u4:~$ mount -t proc
```

با لیست کردن محتویات دایرکتوری proc/ اعداد زیاد و فایل‌های جالبی در این دایرکتوری مشاهده می‌شود.

```
mul@laika:~$ ls /proc
1      2339   4724  5418  6587  7201       cmdline      mounts
10175  2523   4729  5421  6596  7204       cpuinfo      mtrr
10211  2783   4741  5658  6599  7206       crypto       net
10239  2975   4873  5661  6638  7214       devices      pagetypeinfo
141    29775  4874  5665  6652  7216       diskstats    partitions
15045  29792  4878  5927  6719  7218       dma          sched_debug
1519   2997   4879  6     6736  7223       driver       scsi
1548   3      4881  6032  6737  7224       execdomains  self
1551   30228  4882  6033  6755  7227       fb           slabinfo
1554   3069   5     6145  6762  7260       filesystems  stat
1557   31422  5073  6298  6774  7267       fs           swaps
1606   3149   5147  6414  6816  7275       ide          sys
180    31507  5203  6418  6991  7282       interrupts   sysrq-trigger
181    3189   5206  6419  6993  7298       iomem        sysvipc
182    3193   5228  6420  6996  7319       ioports      timer_list
18898  3246   5272  6421  7157  7330       irq          timer_stats
19799  3248   5291  6422  7163  7345       kallsyms     tty
19803  3253   5294  6423  7164  7513       kcore        uptime
19804  3372   5356  6424  7171  7525       key-users    version
1987   4      5370  6425  7175  7529       kmsg         version_signature
1989   42     5379  6426  7188  9964       loadavg      vmcore
2      45     5380  6430  7189  acpi       locks        vmnet
20845  4542   5412  6450  7191  asound     meminfo      vmstat
221    46     5414  6551  7192  buddyinfo  misc         zoneinfo
2338   4704   5416  6568  7199  bus        modules
```

حال بیایید در مورد ویژگی فایل‌ها در proc/ جستجو کنیم. زمان و تاریخ کنونی مطابق با زمان و تاریخ هسته است که به روز بودن فایل‌ها را نشان می‌دهد.

```
paul@RHELv4u4:~$ date
Mon Jan 29 18:06:32 EST 2007
paul@RHELv4u4:~$ ls -al /proc/cpuinfo
-r--r--r--  1 root root 0 Jan 29 18:06 /proc/cpuinfo
paul@RHELv4u4:~$
paul@RHELv4u4:~$  ...time passes...
paul@RHELv4u4:~$
paul@RHELv4u4:~$ date
Mon Jan 29 18:10:00 EST 2007
paul@RHELv4u4:~$ ls -al /proc/cpuinfo
-r--r--r--  1 root root 0 Jan 29 18:10 /proc/cpuinfo
```

بیش‌تر فایل‌ها در /proc حجمی ندارند اما داده دارند. برخی اوقات داده‌های زیادی هم دارند. می‌توانید این موضوع را با استفاده از دستور cat بر روی دایرکتوری /proc/cpuinfo بررسی کنید. این فایل شامل اطلاعاتی درباره CPU است. بیش‌تر فایل‌ها در /proc فقط خواندنی هستند و یا نیاز به سطح دسترسی root دارند. تعدادی از فایل‌ها هم قابل نوشتن هستند. تعداد زیادی فایل در /proc/sys قابل نوشتن هستند. در ادامه تعدادی از فایل‌ها در /proc مورد بررسی قرار می‌گیرند.

```
model name      : AMD Athlon(tm) 64 X2 Dual Core Processor 4600+
stepping        : 1
cpu MHz         : 2398.628
cache size      : 512 KB
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 1
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge...
bogomips        : 4803.54
```

```
paul@RHELv4u4:~$ file /proc/cpuinfo
/proc/cpuinfo: empty
paul@RHELv4u4:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : AuthenticAMD
cpu family      : 15
model           : 43
```

در معماری x86 در فایل /proc/interrupts وقفه‌های سیستم وجود دارند.

```
paul@RHELv4u4:~$ cat /proc/interrupts
           CPU0
  0:   13876877    IO-APIC-edge  timer
  1:         15    IO-APIC-edge  i8042
  8:          1    IO-APIC-edge  rtc
  9:          0   IO-APIC-level  acpi
 12:         67    IO-APIC-edge  i8042
 14:        128    IO-APIC-edge  ide0
 15:     124320    IO-APIC-edge  ide1
169:     111993   IO-APIC-level  ioc0
177:       2428   IO-APIC-level  eth0
NMI:          0
LOC:   13878037
ERR:          0
MIS:          0
```

روی سیستمی با دو CPU فایل بالا به صورت زیر خواهد بود:

```
paul@laika:~$ cat /proc/interrupts
           CPU0       CPU1
  0:     860013          0  IO-APIC-edge      timer
  1:       4533          0  IO-APIC-edge      i8042
  7:          0          0  IO-APIC-edge      parport0
  8:    6588227          0  IO-APIC-edge      rtc
 10:       2314          0  IO-APIC-fasteoi   acpi
 12:        133          0  IO-APIC-edge      i8042
 14:          0          0  IO-APIC-edge      libata
 15:      72269          0  IO-APIC-edge      libata
 18:          1          0  IO-APIC-fasteoi   yenta
 19:     115036          0  IO-APIC-fasteoi   eth0
 20:     126871          0  IO-APIC-fasteoi   libata, ohci1394
 21:      30204          0  IO-APIC-fasteoi   ehci_hcd:usb1, uhci_hcd:usb2
 22:       1334          0  IO-APIC-fasteoi   saa7133[0], saa7133[0]
 24:     234739          0  IO-APIC-fasteoi   nvidia
NMI:         72         42
LOC:     860000     859994
ERR:          0
```

حافظه فیزیکی با /proc/kcore نمایش داده می‌شود. از دستور cat برای این فایل استفاده نکنید. در عوض از یک اشکال‌یاب (debugger) استفاده کنید. اندازه /proc/kcore برابر با اندازه حافظه به اضافه ۴ بایت است.

```
paul@laika:~$ ls -lh /proc/kcore
-r-------- 1 root root 2.0G 2007-01-30 08:57 /proc/kcore
paul@laika:~$
```

۵-۱-۵-۳ Unix System Resource :/usr

اغلب، /usr مانند user تلفظ می‌شود اما توجه داشته باشید که usr مخفف Unix System Resources است. ساختار /usr داده‌هایی از نوع اشتراکی و فقط خواندنی است. برخی آن را به صورت فقط خواندنی در می‌آورند. این کار را می‌توان از روی پارتیشن خودش یا از روی یک NFS اشتراکی فقط خواندنی انجام داد. این دایرکتوری شامل دایرکتوری‌های مختلفی از جمله bin، include، lib و ... است.

۵-۱-۵-٤ variable data :/var

فایل‌هایی با اندازه غیرقابل پیش‌بینی مانند log، cache و spool در این دایرکتوری قرار می‌گیرند.

/var/log

این دایرکتوری شامل تمامی فایل‌های log است.

```
[paul@RHEL4b ~]$ ls /var/log
acpid           cron.2    maillog.2   quagga           secure.4
amanda          cron.3    maillog.3   radius           spooler
anaconda.log    cron.4    maillog.4   rpmpkgs          spooler.1
anaconda.syslog cups      mailman     rpmpkgs.1        spooler.2
anaconda.xlog   dmesg     messages    rpmpkgs.2        spooler.3
audit           exim      messages.1  rpmpkgs.3        spooler.4
boot.log        gdm       messages.2  rpmpkgs.4        squid
boot.log.1      httpd     messages.3  sa               uucp
boot.log.2      iiim      messages.4  samba            vbox
boot.log.3      iptraf    mysqld.log  scrollkeeper.log vmware-tools-guestd
boot.log.4      lastlog   news        secure           wtmp
canna           mail      pgsql       secure.1         wtmp.1
cron            maillog   ppp         secure.2         Xorg.0.log
cron.1          maillog.1 prelink.log secure.3         Xorg.0.log.old
```

/var/log/message

به طور پیش فرض این فایل حاوی اطلاعات رخدادها در سیستم است و در اوبونتو با /var/log/syslog شناخته می‌شود.

```
[root@RHEL4b ~]# tail /var/log/messages
Jul 30 05:13:56 anacron: anacron startup succeeded
Jul 30 05:13:56 atd: atd startup succeeded
Jul 30 05:13:57 messagebus: messagebus startup succeeded
Jul 30 05:13:57 cups-config-daemon: cups-config-daemon startup succeeded
Jul 30 05:13:58 haldaemon: haldaemon startup succeeded
Jul 30 05:14:00 fstab-sync[3560]: removed all generated mount points
Jul 30 05:14:01 fstab-sync[3628]: added mount point /media/cdrom for...
Jul 30 05:14:01 fstab-sync[3646]: added mount point /media/floppy for...
Jul 30 05:16:46 sshd(pam_unix)[3662]: session opened for user paul by...
Jul 30 06:06:37 su(pam_unix)[3904]: session opened for user root by paul
```

/var/cache

این دایرکتوری شامل داده‌های cache مربوط به برنامه‌های متعددی است.

```
paul@ubu1010:~$ ls /var/cache/
apt       dictionaries-common    gdm       man         software-center
binfmts   flashplugin-installer  hald      pm-utils
cups      fontconfig             jockey    pppconfig
debconf   fonts                  ldconfig  samba
```

/var/spool

چی؟ شامل دایرکتوری‌های spool برای ایمیل و کرون (cron) است. هم‌چنین به عنوان دایرکتوری والد برای سایر فایل‌های spool نیز استفاده می‌شود (مانند فایل‌های spool چاپ).

/var/lib

این دایرکتوری شامل اطلاعات حالت و وضعیت برنامه‌ها است.

/var/...

فایل‌های ID پروسه‌ها در var/run/، فایل‌های موقت برای reboot در var/tmp/ و اطلاعاتی درباره قفل فایل در var/lock/ قرار دارد. نمونه‌های زیادی برای استفاده var/ وجود دارد که مطالعه آن به خواننده واگذار می‌شود.

۲-۵ دستورکار

۱) آیا فایل‌های /bin/cat و /bin/echo وجود دارند؟ نوعشان چیست؟

۲) اندازه فایل هسته لینوکس (اول این فایل‌ها با vmشروع می‌شود) در /boot چقدر است؟

۳) دایرکتوری ~/test را ایجاد نمایید و دستورات زیر را اجرا کنید. حاصل این دستور چیست؟

۴) تابع random چه کاری انجام می‌دهد؟

۵) آیا می‌توانید وارد دایرکتوری ریشه شوید؟ چه فایل‌های مخفی در آنجا وجود دارد؟

۶) چرا فایل‌هایifconfig، fdisk، parted، shutdownو grub-installتنها در /sbin قرار دارند و در /bin قرار ندارند؟

۷) آیا /var/log دایرکتوری است یا فایل است؟

آزمایش ششم: مدیریت فرآیندها (مقدمه)

۶-۱ پیش‌آگاهی

در این آزمایش، مقدمه‌ای در مورد دستورات لینوکس برای دیدن اطلاعات مربوط به فرآیندها و نحوه نوشتن برنامه در این سیستم‌عامل ذکر خواهد شد و در آزمایش‌های بعدی به نوشتن برنامه‌های سیستمی پرداخته خواهد شد.

لینوکس یک سیستم‌عامل چندوظیفه‌ای است یعنی می‌تواند همزمان چند برنامه را روی یک پردازنده اجرا کند. متناظر با هر برنامه‌ای که در سیستم اجرا می‌شود، سیستم‌عامل یک فرآیند ایجاد می‌کند. در سیستم‌عامل Linux برای هر فرآیند اطلاعات زیادی نگهداری می‌شود که در ادامه، برخی از آن‌ها شرح داده می‌شود:

PID: هر فرآیند در سیستم، یک شماره (Process ID) دارد که از این پس با نام pid به آن اشاره خواهیم کرد. سه فرآیند ویژه در سیستم‌عامل Linux تعریف شده‌اند و هنگام شروع به کارِ سیستم ایجاد می‌شوند و هنگام پایان کار سیستم از بین می‌روند. این فرآیندها شماره‌های ثابت دارند و عبارتند از coordinator با شمارهٔ صفر، init با شمارهٔ یک و swapper با شمارهٔ دو.

PPID: هر فرآیند می‌تواند فرآیندهای دیگری را ایجاد کند. این فرآیند، پدر فرآیندهای جدید خواهد بود. هر یک از فرآیندهای فرزند، شمارهٔ فرآیند پدر خود (Parent Process ID) را به همراه دارند. تمام فرآیندها در سیستم، یک پدر دارند و پدر نهایی فرآیندها، init است. پس تنها فرآیندی که پدر ندارد، init است.

۶-۱-۱ مشاهده مشخصات فرآیندها: فرمان‌های ps و top

به منظور دیدن فرآیندهای موجود بر روی سیستم، از فرمان ps استفاده کنید. قالب کلی فرمان و نتیجهٔ اجرای آن به صورت زیر خواهد بود:

```
ps $

COMMAND STAT TIME TT PID
22 1 T 0:12 emacs
40 1 T 0:06 find
55 2 R 0:01 ps
```

قسمت‌های مختلف خروجی به ترتیب عبارتند از: شمارهٔ فرآیند، نام پایانه‌ای که فرآیند از آن اجرا شده، وضعیت فعلی فرآیند، زمانی که فرآیند اجرا شده و نام فرآیند. اگر بعد از فرمان ps گزینهٔ -l بکار برید، اطلاعات بیش‌تری راجع به فرآیندها خواهید دید.

فرمان ps فرآیندهای موجود را فقط یک بار گزارش می‌کند و سپس پایان می‌پذیرد. Linux، دارای فرمانی مشابه ps به نام top است. فرمان top پس از اجرا، ضمن نشان دادن فرآیندهای موجود، اطلاعات آن‌ها را نیز لحظه به لحظه به‌هنگام می‌کند. برای خارج شدن از فرمان top باید کلید q را فشار دهید.

۶-۱-۲ اجرای فرآیندها در پس‌زمینه (background)

با افزودن علامت & به آخر هر فرمان یا برنامه، می‌توان آن را در پس‌زمینه اجرا کرد. در این حالت، Linux علاوه بر اجرای برنامهٔ ذکر شده، با دادن اعلان خط فرمان آمادگی خود را جهت گرفتن فرمان یا برنامه‌ای دیگر اعلام می‌کند (بدون اینکه اجرای برنامهٔ قبلی پایان یافته باشد). بدین ترتیب دو یا چند فرمان را می‌توان با هم اجرا کرد.

۶-۱-۳ هستهٔ Linux

هستهٔ Linux وظایف مدیریت فرآیندها، مدیریت حافظه، مدیریت پرونده و ارتباط با سخت‌افزار را بر عهده دارد. هستهٔ سیستم‌عامل کاملاً وابسته به سخت‌افزار است و باعث استقلال برنامه‌های کاربردی از سخت‌افزار می‌شود. بنابراین برنامه‌نویسی برای این قسمت با برنامه‌نویسی عادی Linux کاملاً تفاوت دارد و نیاز به داشتن آگاهی از سخت‌افزار و نیز نحوهٔ پیاده‌سازی هسته دارد.

در Linux یک فرآیند می‌تواند با استفاده از فراخوان‌های سیستم (system call)، از امکانات هسته استفاده کند. مثلاً یک فرآیند برای باز کردن یک پرونده از فراخوان سیستم open استفاده می‌کند.

۶-۱-۴ پوستهٔ Linux

در Linux معمولاً برنامه‌ای از نوع پوسته، واسط ارتباطی کاربر با سیستم‌عامل می‌شود. این برنامه، فرمان کاربر را دریافت می‌کند و با استفاده از سرویس‌های قسمت‌های دیگر، آن را اجرا می‌کند. مثلاً وقتی کاربر دستور ls را برای دیدن لیست پرونده‌ها صادر می‌کند، پوسته ابتدا با استفاده از سرویس‌های سیستم پرونده، پرونده اجرایی ls را در شاخهٔ /usr/bin می‌یابد. سپس با استفاده از یک فراخوان سیستم (یعنی فراخوان fork)، یک فرآیند جدید ایجاد کرده، در آن /usr/bin/ls را اجرا می‌کند. تفسیر و اجرای فرمان‌هایی که در پای اعلان سیستم وارد می‌گردند به عهدهٔ پوسته است. با تعویض پوسته، قالب فرمان‌هایتان نیز ممکن است تغییر کند که برای فهمیدن این مطلب باید به راهنمای پوستهٔ آن سیستم مراجعه کنید. پوسته‌ای که در آزمایشگاه استفاده می‌شود، bash نام دارد.

همچنین پوسته، یک زبان برنامه‌نویسی تفسیری (مشابه پرونده‌های دسته‌ای (Batch Files) در DOS) در اختیار کاربر قرار می‌دهد.

۶-۲ برنامه‌نویسی

در سیستم‌عامل Linux، مفاهیم متداول در مبحث سیستم‌عامل به سادگی و زیبایی پیاده‌سازی شده است. اما در ابتدا ممکن است استفاده از آن‌ها برای برنامه‌نویسان عادی اندکی نامأنوس باشد. برنامه‌نویسی در Linux را می‌توان به دو دستهٔ کلی زیر تقسیم کرد:

- برنامه‌نویسی کاربردی (Application programming)

- برنامه‌نویسی هسته (Kernel programming)

نوع اول برنامه‌نویسی، برای ایجاد برنامه‌های عادی به کار می‌رود. مشخصهٔ اصلی چنین برنامه‌هایی این است که مطابق معماری Linux، کاملاً مستقل از سخت‌افزار هستند. هسته به عنوان یک پل ارتباطی بین سخت‌افزار و برنامه‌ها، این استقلال را تامین می‌کند. نوع دوم برنامه‌نویسی، برای افزایش قابلیت‌های سیستم‌عامل استفاده می‌شود. مثلاً برای اینکه سیستم‌عامل بتواند از سخت‌افزار جدیدی حمایت کند، باید برنامه‌هایی از این دست را به آن افزود.

در برنامه‌نویسی برای Linux، زبان C بیش‌ترین کاربرد را دارد و تقریباً تمام توابعی که در برنامه‌نویسی استفاده می‌شود به زبان C نوشته شده‌اند. برای نمونه، تمام فراخوان‌های سیستم به صورت توابع زبان C ایجاد شده‌اند.

به همراه Linux، کتابخانه‌های جانبی زبان C نیز وجود دارد که در مقایسه با کتابخانه‌های معمولی امکانات بیش‌تری در اختیار برنامه‌نویس قرار می‌دهند و برنامه‌نویس می‌تواند در مواقعی که استفاده از فراخوان‌های سیستم مشکل می‌شود، از توابع این کتابخانه‌ها استفاده کند.

۶-۲-۱- gcc (مترجم زبان C)

پس از آنکه برنامهٔ خود را توسط یک ویرایشگر ایجاد کردید، باید آن را توسط برنامهٔ gcc ترجمه کنید. قالب کلی این فرمان به صورت زیر است:

```
executable_filename o- $ gcc source_filename
```

به عنوان مثال اگر بخواهیم برنامهٔ myfork.c را ترجمه کنیم و نام پروندهٔ قابل اجرای آن myfok1 باشد، سطر زیر را وارد خواهیم کرد:

```
myfork1 myfork.c -o $ gcc
```

۶-۲-۲- اشکالزدایی با gdb

برای اشکال‌زدایی برنامه‌هایتان در Linux، می‌توانید از اشکال‌زدای gdb استفاده کنید. برای این کار ابتدا برنامهٔ خود را با گزینهٔ g- (مربوط به gcc) ترجمه کنید. سپس در پای اعلان Linux، فرمان gdb را به همراه نام پروندهٔ قابل اجرای برنامه وارد کنید. با این کار وارد محیط gdb شده، به کمک فرمان‌های این محیط می‌توانید به اشکال‌زدایی برنامهٔ خود بپردازید.

خلاصه‌ای از این فرمان‌ها با توضیحات مربوط در جدول ۶-۱ آمده است.

جدول ۶-۱: خلاصه‌ای از فرمان‌های gdb

توضیحات	شکل کلی فرمان
قرار دادن نقطهٔ شکست (شرطی یا غیرشرطی) در خط مورد نظر	**break** <line#> [if condition]

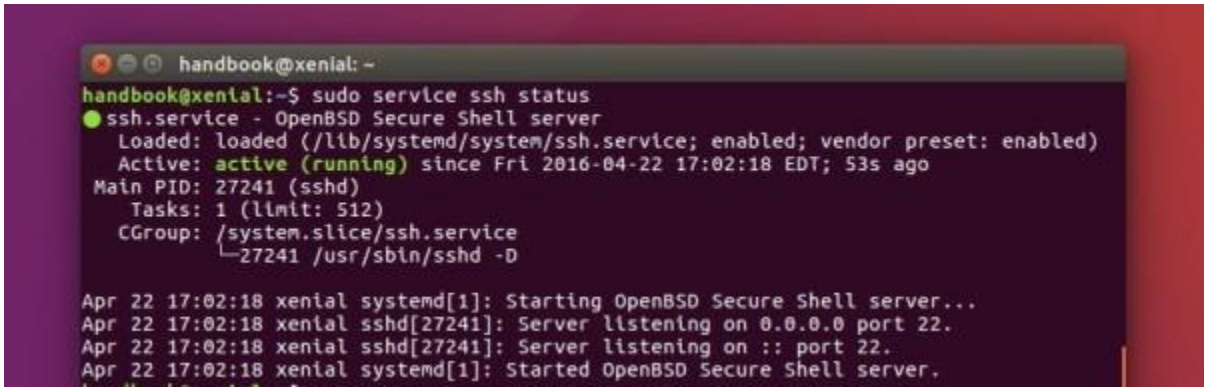
اجرای برنامه	run
اجرای یک سطر از برنامه	step
اجرای یک تابع بدون اجرای سطر به سطر دستورات آن	next
نشان دادن قطعاتی از برنامه	list
اجرای برنامه تا رسیدن به خط مورد نظر	**until** <#line>
چاپ مقدار یک متغیر	**print** <var-name>
فراخوانی تابع موردنظر و نشان دادن خروجی آن	**print** <function>
اجرای برنامه تا رسیدن به یک دستور بازگشت به سیستم‌عامل	finish
اجرای برنامه تا رسیدن به نقطهٔ شکست بعدی	continue
حذف نقطهٔ شکست از خط مورد نظر	clear <line#>
حذف تمام نقاط شکست	delete

۶-۳ اتصال راه دور به سیستم‌عامل لینوکس

یکی از قابلیت‌های مهم سیستم‌عامل لینوکس، موضوع اتصال از راه دور است. برای اینکه چنین ارتباطی برقرار شود، نیاز است روی سیستم‌عامل لینوکس برنامه ssh server نصب و فعال (active) شده باشد. برای اینکه متوجه شوید این برنامه نصب است یا خیر، کافی است در خط فرمان سیستم‌عامل دستور زیر را وارد نمایید:

```
sudo service ssh status
```

در صورت فعال بودن، پیغامی مشابه شکل ۶-۱ نمایش داده خواهد شد.



شکل ۶-۱: نحوه آزمایش فعال بودن ssh server

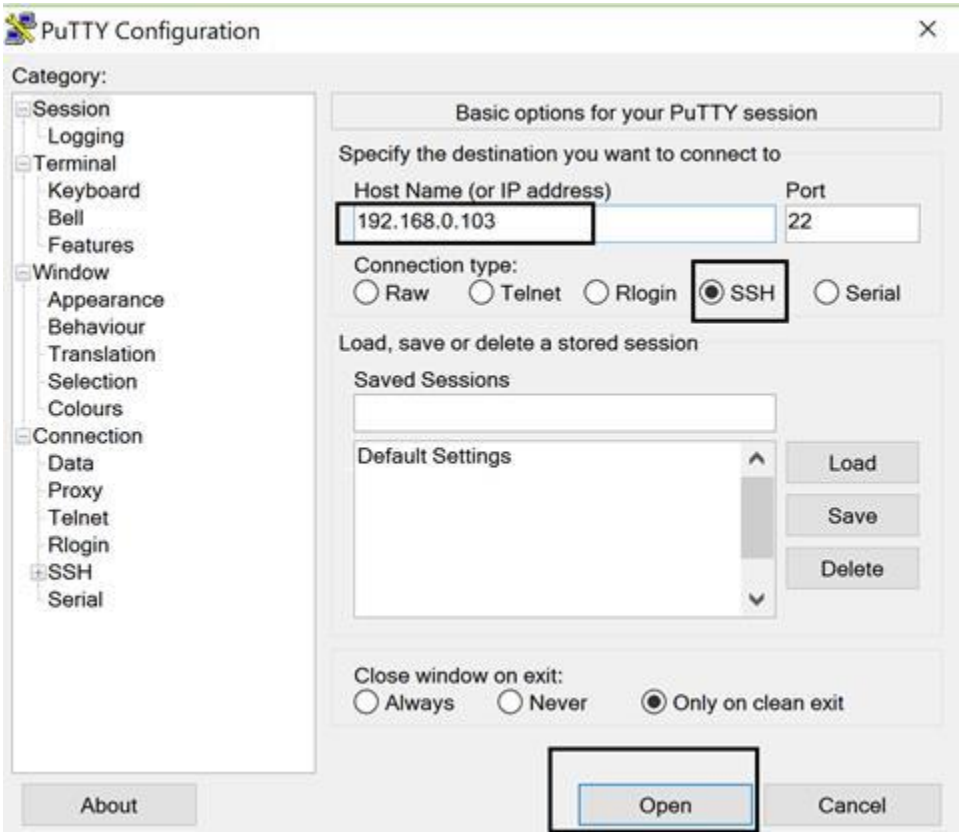
در صورت فعال نبودن ssh server باید ابتدا برنامه مربوطه را روی سیستم‌عامل نصب و فعال کنید:

```
sudo apt-get install openssh-server
sudo nano /etc/ssh/sshd_config
```

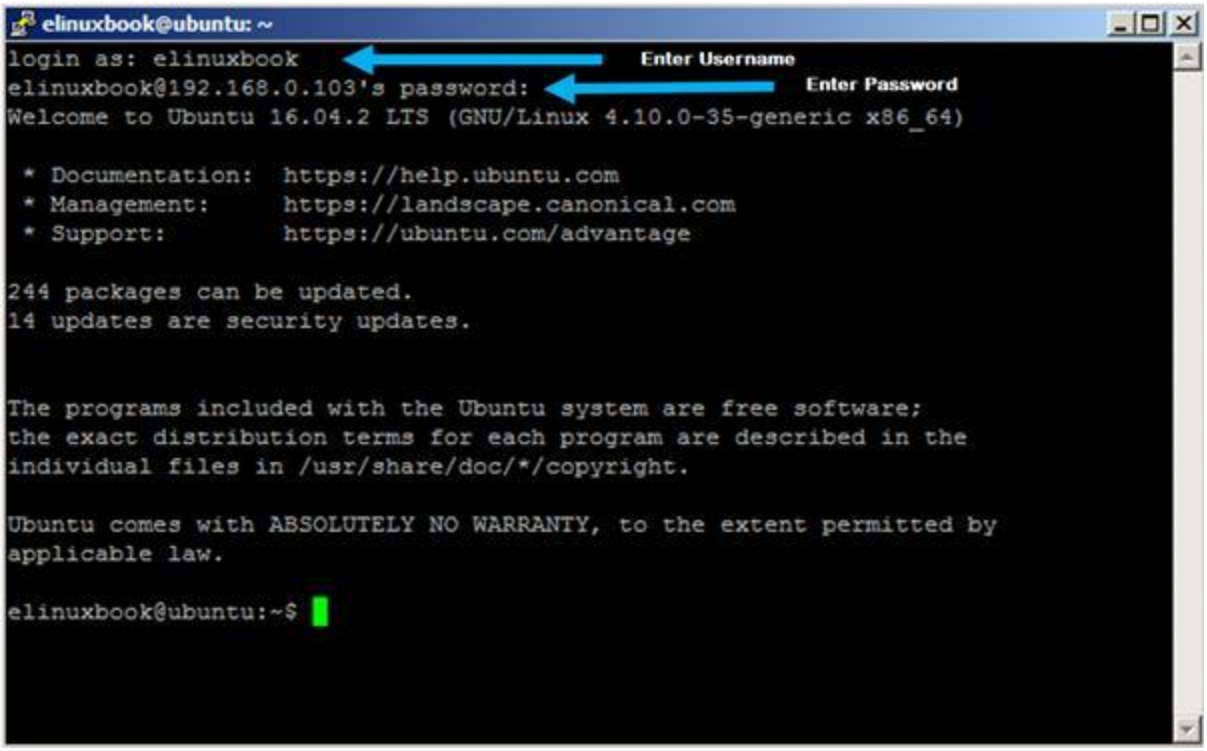
در فایل باز شده، دو خط مربوط به مشخص کردن شماره پورت و شماره پروتکل را از حالت کامنت خارج کنید (حذف کردن کاراکتر # در ابتدای این دو خط). سپس با فشردن کلید Ctrl+o آن را ذخیره کنید و با فشردن کلید ctrl+x خارج شوید.

برای اتصال به سیستم‌عامل از راه دور، به یک برنامه برای شبیه‌سازی ترمینال نیاز است. یکی از معروف‌ترین برنامه‌هایی که برای این کار استفاده می‌شود، putty است. مثلا اگر بخواهید از سیستم‌عامل ویندوز نصب شده روی کامپیوترتان به سیستم‌عامل Ubuntu نصب شده روی ماشین مجازی وصل شوید، کافی است برنامه putty را از سایت putty.org دانلود کنید. سپس همان‌طورکه در شکل ۶-۲ نمایش داده شده است، تنظیمات را انجام دهید و روی دکمه open کلید نمایید. ترمینالی باز می‌شود و رمز عبور سیستم‌عامل را از شما

سوال خواهد کرد. در صورت صحیح بودن رمز عبور، شکل ۳-۶ نمایش داده خواهد شد که همان ترمینال لینوکس خواهد بود و تمام فرامین را می‌توانید در این بخش وارد کنید.



شکل ۶-۲: اتصال به سیستم‌عامل لینوکس با استفاده از نرم‌افزار putty



شکل ۶-۳: اتصال موفقیت‌آمیز به سیستم‌عامل لینوکس با استفاده از نرم‌افزار putty

۶-۴ دستورکار

۱) فرمان ps را اجرا کنید. خروجی این فرمان را ببینید و دربارهٔ اطلاعاتی که توسط این فرمان در مورد فرآیندها نشان داده می‌شود را توضیح دهید.

۲) ویرایشگر vi را در پس زمینه اجرا کنید (vi &). فرمان ps را اجرا کنید. نتایج اجرای این فرمان با نتایج مشاهده شده در گام قبلی چه تفاوتی دارد؟ توضیح دهید.

۳) فرمان top را وارد کنید و در مورد آنچه که بر روی صفحه می‌بینید توضیح دهید.

۴) وارد ویرایشگر nano شوید. یک برنامهٔ کوچک C بنویسید تا پیغام‌هایی را بر روی صفحهٔ نمایش چاپ کند. آن را در پرونده‌ای با نام first.c ذخیره کنید.

۵) برنامهٔ first.c را ترجمه و اجرا کنید (در صورت اجرای برنامه gcc و صحیح بودن برنامه، فایلی اجرایی با نام first.o ایجاد خواهد شد که برای اجرای آن باید دستور ./first.o را در خط فرمان وارد نمایید).

۶) برنامه فوق را ویرایش کنید و یک حلقه بی‌پایان در انتهای برنامه قرار دهید. سپس برنامه را کامپایل کنید و سه بار برنامه را در پس زمینه اجرا کنید (با استفاده از &، برنامه را در پس زمینه اجرا کنید).

۷) با استفاده از دستور top، شماره PID فرآیندهای مربوط به سه بار اجرای برنامه فوق را بدست آورید. سپس با استفاده از دستور kill فرآیندهای فوق را از حافظه خارج کنید.

۸) با استفاده از دستور ifconfig در خط فرمان، آدرس IP سیستم‌عامل نصب شده روی virtual machine را به دست آورید. سپس طبق توضیحات انجام شده در بخش ۶-۳، از سیستم‌عامل ویندوز به سیستم‌عامل لینوکس نصب شده در virtual machine متصل شوید. اکنون برنامه first.o که قبلا ایجاد کرده بودید را اجرا کنید.

آزمایش هفتم: مدیریت فرآیندها (حالات فرآیند و ایجاد فرآیند)

۱-۷ پیش‌آگاهی

هدف از این آزمایش، آشنایی با فرآیند (Process) و حالات مختلف یک فرآیند در سیستم‌عامل لینوکس و سپس نحوه ایجاد فرآیند در این سیستم‌عامل است.

۱-۱-۷ حالات مختلف فرآیند (Process)

همان‌طورکه پیش‌تر دیدید، دستور ps می‌تواند وضعیت فرآیندها را نمایش دهد. یکی از مشخصه‌هایی که ps – al برای یک فرآیند نشان می‌دهد، STAT نام دارد که حالت فرآیند را به صورت یک کُدِ حداکثر سه حرفی نشان می‌دهد.

حرف اول این کُد می‌تواند یک از حروف زیر باشد:

R: برای فرآیندهای در حال اجرا

S: برای فرآیندهای منتظر (Waited)

Z: برای فرآیندهای جادویی (Zombie)

T: برای فرآیندهایی که متوقف (Stopped) یا ردگیری (Trace) شده‌اند (یا به اصطلاح آمادهٔ اجرا هستند).

D: برای فرآیندهایی که در حالت انتظار بدون انقطاع یا انتظار درحافظهٔ جانبی هستند.

اگر حرف دوم این کُد W باشد، نشان‌دهنده آن است که فرآیند مورد نظر هیچ صفحه‌ای در حافظهٔ اصلی ندارد و به‌طور کامل مبادله (Swap) شده است.

حرف سوم، مربوط به زمان‌بندی و اولویت فرآیندها است. اگر این حرف N باشد، نشان‌دهنده این است که ارزش مطلوب (Nice Value) فرآیند، بزرگ‌تر از صفر است (این نوع فرآیندها به وقت کم‌تری از پردازنده نیاز داشته و اولویت بیش‌تری دارند).

در سیستم‌عامل Linux یک فرآیند می‌تواند نُه حالت مختلف داشته باشد. حالات مختلف فرآیندها در جدول ۷–۱نشان داده شده است. وقتی فرآیند پدر با فراخوان سیستم fork، فرآیند جدیدی ایجاد می‌کند، فرآیند ایجاد شده بسته به مقدار حافظهٔ آزاد سیستم، وارد یکی از مراحل ۳ یا ۵ خواهد شد. برای سادگی، فرض کنید فرآیند وارد مرحلهٔ ۳ می‌گردد. وقتی زمان‌بند فرآیندها، به طور تصادفی آن را برای اجرا برگزید، فرآیند وارد مرحلهٔ اجرا در فضای هسته می‌شود و در این هنگام کار فراخوانِ سیستم fork به اتمام می‌رسد. پس از این مرحله ممکن است فرآیند به مرحلهٔ ۱ برود.

بعد از یک برش زمانی، به علت وقوع وقفهٔ ساعت سیستم، فرآیند به مرحلهٔ ۲ باز می‌گردد (چون تمام فرآیندها این وقفه را دریافت می‌کنند). هسته، از این وقفهٔ ساعت برای زمان‌بندی و اولویت‌بندی فرآیندها استفاده می‌کند. پس از زمان‌بندی، ممکن است فرآیند دیگری برای اجرا انتخاب گردد. در این هنگام، فرآیند قبلی وارد حالت ۷ می‌گردد و منتظر می‌ماند تا برای اجرا دوباره انتخاب گردد. اگر فرآیند در حال اجرا در فضای کاربر، بخواهد از سرویس‌های هسته استفاده کند، وارد مرحلهٔ ۲ می‌گردد. اگر این سرویس، کاری مانند ورودی/خروجی باشد، فرآیند وارد مرحلهٔ ۴ می‌شود و خود را متوقف می‌سازد. این توقف تا زمانی ادامه می‌یابد که به واسطهٔ اتمام کارِ ورودی/خروجی، وقفه‌ای به پردازنده برسد و سرویس روتین وقفه، فرآیند مربوطه را بیدار کند. این کار باعث می‌شود تا فرآیند به مرحلهٔ ۳ بازگردد و منتظر شود تا هسته آن را زمان‌بندی کند.

جدول ۷-۱: حالات مختلف فرآیندها

شماره	نام	شرح
۱	اجرا در فضای کاربر	فرآیند، موقع اجرای عملیات عادی خود در این حالت قرار دارد.
۲	اجرا در فضای هسته	فرآیند، هنگام استفاده از سرویس‌های هسته در این حالت قرار دارد. مانند اجرای فراخوان‌های سیستم و روال خدماتی وقفه‌ها.
۳	آماده به اجرا در حافظهٔ اصلی	فرآیند در حال اجرا نیست، ولی در حافظه اصلی قرار دارد و منتظر دریافت وقت پردازنده است تا اجرا شود.
۴	متوقف (Sleep) در حافظهٔ اصلی	فرآیند در حافظهٔ اصلی است و منتظر پایان یافتن عملی مانند ورودی/خروجی است تا دنبالهٔ اجرایش را از سر بگیرد.
۵	آماده به اجرا در حافظهٔ جانبی	فرآیند آمادهٔ اجرا است، ولی به علت کمبود حافظه، به حافظهٔ جنبی منتقل شده است و قبل از آن که وقت پردازنده به آن تخصیص داده شود، باید به حافظهٔ اصلی منتقل گردد.
۶	متوقف در حافظهٔ جانبی	مشابه حالت ۴ است، با این تفاوت که فرآیند به علت کمبود حافظه، توسط هسته به حافظهٔ جانبی منتقل شده است تا حافظهٔ کافی برای فرآیندهای در حال اجرا فراهم گردد.

۷	قبضه شده (Preempted)	فرآیند از حالت اجرا در فضای کاربر به حالت اجرا در فضای هسته منتقل می‌گردد. ولی در این هنگام زمان اختصاص یافته به آن پایان می‌یابد و زمان‌بند فرآیندها، فرآیند دیگری را برای اجرا انتخاب می‌کند.
۸	ایجاد	هر فرآیندی که ایجاد می‌شود، نخست در این حالت قرار می‌گیرد. در این حالت فرآیند نه در حال اجرا است و نه در حال توقف. این حالت نقطهٔ شروع تمامی فرآیندها (به جز فرآیند با pid صفر) است.
۹	جادویی (Zombie)	هر فرآیند توسط تابعexit، با به جا گذاشتن اطلاعاتی برای پدر خویش (فرآیندی که این فرآیند را اجرا کرده است)، به کار خویش خاتمه می‌دهد و در حالت جادویی قرار می‌گیرد. هسته، تمام منابع به جز کُد خروج فرزند را از این نوع فرآیندها می‌گیرد. این منابع شامل حافظه، پردازنده و . . . است. کُد خروج فرزند تا زمان فراخوانی wait یا waitpid توسط پدر معتبر است. این فراخوانی‌ها باعث می‌شوند پدر از وضعیت (کُد) خروج فرزندش مطلع شود. اگر پدر منتظر دریافت کُد خروج فرزندش نشود و کارش را به‌پایان برساند، درحالی‌که هنوز حیات فرزند ادامه دارد، فرزند بی‌سرپرست (Orphan) می‌شود. مدیریت این نوع فرآیندها به عهدهٔ هستهٔ سیستم‌عامل است.

اگر فضای کافی برای فرآیندهای در حال اجرا، در حافظه موجود نباشد، فرآیند مبادله‌کننده (swapper) (با شمارهٔ صفر) بعضی از فرآیندها را بر روی دیسک منتقل می‌کند و به این ترتیب فرآیند وارد مرحلهٔ ۵ می‌گردد. همین اتفاق ممکن است برای فرآیندهای متوقف در حافظه (حالت ۴) نیز اتفاق بیفتد. وقتی کار فرآیند به پایان رسید، وارد مرحلهٔ ۹ می‌شود.

۲-۱-۷ ایجاد فرآیند جدید

برای ایجاد یک فرآیند جدید، از یک فراخوان سیستم به نام fork استفاده می‌شود. خروجی fork از جنس pid_t (درحقیقت، یک عدد صحیح) است. این خروجی همان شناسهٔ فرآیند (شمارهٔ فرآیند PID) است.

با احضار فراخوان سیستم fork، فرآیند جدیدی ایجاد می‌شود که پدر آن، فرآیندی است که fork را فراخوانده است. فرآیند جدید در کنار بقیهٔ فرآیندهای موجود در سیستم قرار می‌گیرد و منتظر می‌شود تا وقت پردازنده به او اختصاص پیدا کند. کُد اجرایی این فرآیند دقیقاً همانند پدرش است و همان چیزی را اجرا می‌کند که پدرش اجرا می‌کرده است. فرآیند فرزند، محیط، متغیرها، پرونده‌های باز، شناسه‌های پروندهٔ پدر را به ارث می‌برد. البته تفاوت‌های اندکی نیز بین این دو فرآیند وجود دارد از جمله شمارهٔ فرآیند (PID) و شمارهٔ فرآیند پدر (PPID).

قطعه کُد زیر را در نظر بگیرید:

```
pid_t child_id;
child_id = fork();
printf("forked\n");
```

با اجرای این دستور، یک فرآیند جدید ایجاد خواهد شد که دارای کُدی به صورت فوق بوده، شمارندهٔ برنامهٔ (Program counter) آن، به دستورالعمل بعد از fork اشاره می‌کند. اکنون دو فرآیند وجود دارند که دستورالعمل بعدی را اجرا خواهند کرد و در نتیجه، کلمهٔ forked دو بار چاپ خواهد شد.

تذکّر: pid_t یکی از انواع تایپ‌های صحیح زبان C تحت Linux است که محتوای آن می‌تواند شمارهٔ یک فرآیند باشد. شمارهٔ فرآیند (PID)، مشخّصه‌ای است که سیستم‌عامل توسط آن یک فرآیند را می‌شناسد و به آن دسترسی پیدا می‌کند.

بنابر آنچه گفته شد، فراخوانی fork برای این است که یک فرآیند جدید ایجاد شود و در آن عمل دیگری به موازات عملیات فرآیند اصلی انجام گیرد.

۳-۱-۷ اجرای عملیاتی متفاوت با عملیات فرآیند اصلی در فرآیند ایجاد شده

توابع exec: فراخوان‌های exec مجموعه توابع مفیدی هستند که اجرای یک دستور سیستم‌عامل را (که قاعدتاً بایستی از خط فرمان اجرا شود) در داخل برنامه امکان‌پذیر می‌سازند. برای اطلاعات بیش‌تر در مورد این توابع به کتاب‌های آموزش زبان C مراجعه کنید. اجرای exec شبیه فراخوانی یک روال معمولی است و پس از اتمام کار، به بعد از محل فراخوانی exec برمی‌گردیم. اما در Linux، تمام شدن اجرای exec به اجرای برنامه (که شامل تابع main و فراخوانی exec است) خاتمه خواهد داد.

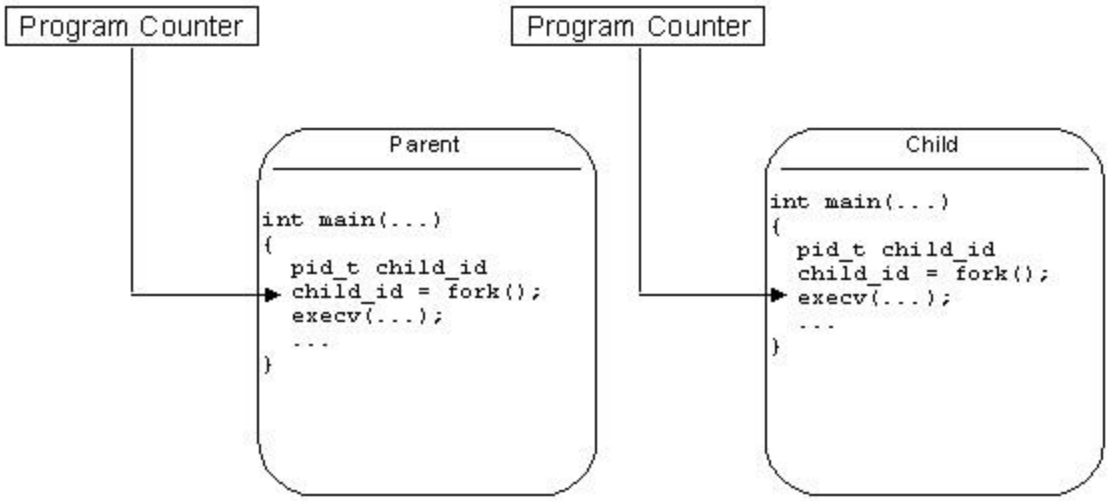
در اینجا به نظر می‌رسد که اگر بتوان تابع exec را در مسیری جدا از مسیر برنامهٔ اصلی اجرا کرد، مشکل قابل حل خواهد بود. اگر برای ایجاد این مسیر جدید، بعد از fork یک فراخوانی exec قرار دهیم (مطابق شکل ۷–۱)، این تابع در فرآیند پدر نیز اجرا خواهد شد که مورد نظر ما نیست (مانند آنچه در مورد printf در قطعه کُد بالا داشتیم).

برای رفع مشکل ذکر شده، راه‌حلی به شرح زیر ارائه می‌شود:

راه‌حل: عبارت()child_id = fork را در نظر بگیرید:

اگر عملیات ایجاد فرآیند با موفقیت انجام شود و فرآیند فرزند ایجاد شود، متغیر child_id که در دو فرآیند پدر و فرزند وجود دارد، در هر دوی آن‌ها مقدار خواهند گرفت. در فرآیند پدر مقداری بزرگ‌تر از صفر (به عنوان شمارهٔ فرآیند فرزندِ ایجاد شده) در child_id خواهدگرفت و در فرآیند فرزند، مقدار صفر در این متغیر ذخیره خواهد شد.

اگر اشکالی در ایجاد فرآیند فرزند به وجود آید، fork مقدار ۱- را برگشت خواهد داد که در متغیر child_id از فرآیند پدر ذخیره خواهد شد (فرآیند فرزند به‌وجود نیامده و بنابراین متغیری متناظری هم نخواهد داشت).



شکل ۷–۱: وضعیت فرآیند اصلی (پدر) و فرآیند ایجاد شده (فرزند) بلافاصله بعد از fork

با توجه به مطالب بالا می‌توان بعد از fork قسمت‌های اختصاصی پدر و فرزند را مشخص کرد.

قطعه کُد شکل ۷–۲ را در نظر بگیرید. قسمت‌هایی که مقدار برگشتی از fork، ۱– نبوده است، می‌توانند به عنوان قسمت اشتراکی بین هر دو فرآیند در نظر گرفته شوند.

شکل ۷–۲: مشخص کردن قسمت‌های اختصاصی فرآیندهای پدر و فرزند

```
child_id = fork();
if (child_id == -1)
{
*/معمولا در این حالت تابع با یک کُد خطا ترک می‌شود. خطا در ایجاد فرآیند فرزند: */
}
if (child_id == 0)
{
*/ قسمت اختصاصی فرزند:
```

```
کارهای مربوط به فرآیند فرزند را در این قسمت می‌توان انجام داد. این قسمت با اینکه
درفرآیند پدر نیز وجود دارد، هرگز در فرآیند پدر اجرا نمی‌شود. به این علت که درصورت
در فرآیند پدر مقدار بزرگ‌تر از صفر به خود می‌گیرد. child_id، forkموفقیت
*/}
else
{
*/ قسمت اختصاصی پدر:
کارهای اختصاصی فرآیند پدر نیز در این قسمت انجام می‌گیرد. زیرا در صورت
در فرآیند فرزند مخالف صفر نخواهد شد child_id، forkموفقیت
*/
}
```

البته مانند شکل ۷-۲ می‌توان قسمت فرآیندهای پدر و فرزند را از هم جدا کرد. کد شکل ۷-۲ را می‌توان به صورت زیر هم نوشت:

```
child_pid=fork();
switch(child_pid)
{
*/ :1- caseخطا در ایجاد فرآیند */
*/ :0 caseفرآیند فرزند */
/* execute one file */
*/ :defaultفرآیند پدر */
}
```

۷-۱-۴- متوقف کردن یک فرآیند

فرض کنید که یک فرآیند، با استفاده از fork، فرآیند جدیدی را به وجود آورده است. به محض تولید فرآیند فرزند، این دو فرآیند به موازات هم اجرا خواهند شد. ولی گاهی این عمل مورد نظر نیست بلکه فرآیند پدر باید منتظر فرزند بماند تا کار او تمام شود، آنگاه به کار خود ادامه دهد. برای این کار، فرآیند پدر باید از فراخوان سیستمِ waitاستفاده کند. این تابع به صورت زیر تعریف شده است:

int wait(int* status)

فرآیندی که این تابع را احضار کرده باشد، تا اتمام کار یکی از فرزندانش صبر خواهد کرد (مقدار خروجی این تابع همان PID فرآیند فرزندی است که کارش تمام شده است و کد خروج این فرزند در پارامتر status ظاهر خواهد شد). اگر فرآیند احضارکننده هیچ فرزندی نداشته باشد، این تابع بلافاصله بازگشت می‌کند.

فراخوان سودمند دیگر برای متوقف کردن یک فرآیند، waitpidاست. در مورد آن می‌توان از صفحات راهنمای موجود در Linux کمک گرفت. این تابع، فرآیند جاری را قادر می‌سازد تا منتظر اتمام کار فرآیند فرزندی شود که شمارهٔ آن فرآیند فرزند را یه عنوان پارامتر به تابع waitpid ارائه کرده است.

۵-۱-۷ راهنمایی‌هایی برای انجام آزمایش

همان‌طورکه در پیش‌آگاهی ملاحظه کردید، توابع exec پس از انجام کار خود، در صورت موفقیت، خاتمه یافته و باعث برگشت به برنامه صدا کننده نمی‌شوند. در طی انجام این آزمایش به دنبال راه حلی برای استفاده مناسب از این توابع هستیم.

تابع fork_cmd مطابق تعریف به صورت زیر:

pid_t fork_cmd(char* cmd, char* argv[]);

در پرونده fork.c تعریف شده است. شما می‌توانید با تکمیل این تابع شرایطی را فراهم کنید که دستور ارسالی در پارامتر cmd با پارامترهای ارسالی احتمالی در argv اجرا شود. مثلاً با صدا کردن تابع زیر:

fork_cmd("/usr/bin/ls","-al");

می‌توان اجرای دستور ls -al را مشاهده کرد. برای امتحان درستی fork_cmd، پروندهٔ fcd.c را با توجه به توضیحات آن، تکمیل کنید.

در قسمت بعد و در ادامه fork_cmd در پرونده‌ای با نام fork.c، یک تابع fork_proc بنویسید که عین عملیات fork_cmd را برای انجام یک تابع proc انجام دهد. برای امتحان درستی fork_ proc، پروندهٔ fpd.c را با توجه به توضیحات آن تکمیل کنید. (دستور کار؟)

تذکر: مقادیر خروجی (مقادیر برگشتی به فرآیند پدر) برای fcd.cو fpd.c، در صورت موفقیت، شمارهٔ PID فرزند خواهد بود و در صورت عدم موفقیت به شرح زیر خواهد بود:

۱ : اشکال در اجرای fcd.c یا fpd.c

۰ : اجرای صحیح fcd.c یا fpd.c

۲-۷ دستور کار

۱) عملکرد تابع getpid را از صفحات راهنما به وسیلهٔ دستور man بیابید. با استفاده از این دستور در پرونده pid.c برنامه‌ای بنویسید که شمارهٔ PID خود را چاپ کند. این برنامه را با استفاده از gcc ترجمه و به دفعات اجرا کنید. چه نتیجه‌ای می‌گیرید؟

۲) پروندهٔ fork.c را باز کرده، تابع fork_cmd را تکمیل کنید (با توجه به راهنمایی موجود در پیش‌آگاهی) .

۳) با فراخوانی تابع fork_cmd در پرونده fcd.c، صحت قسمت قبل را بررسی کنید. چنانچه در نوشتن fork_cmdاشکالی نداشته باشید، پس از اجرای fcd بایستی خروجی فرمان ps –al را ببینید. خروجی برنامه را توجیه کنید.

۴) با استفاده از تابع wait، خروجی برنامه fcdرا اصلاح کنید (به مربی نشان دهید).

۵) در پرونده fork.c، تابع fork_proc را تکمیل کنید. با فراخوانی تابع fork_proc در پرونده fpd.c صحت قسمت قبل را بررسی کنید (به مربی نشان دهید).

۶) برنامه system_derive.cرا چنان تکمیل کنید که بتوان با استفاده از تابع system دستور ps –a را اجرا کرد. آیا تابع system از fork استفاده می‌کند؟ چرا؟

آزمایش هشتم: چندنخی

۸-۱ پیش‌آگاهی

هدف از این آزمایش آشنایی با مفهوم نخ و نحوه برنامه نویسی چند نخی در سیستم‌عامل لینوکس است.

۸-۱-۱ مفهوم چند وظیفه‌ای و چندنخی

توانایی یک سیستم‌عامل در اجرای چندبرنامه به طور موازی، چندوظیفه‌ای نام دارد. در واقع، سیستم‌عامل از ساعت سخت‌افزاری برای اختصاص برهه‌های زمانی (Time slice) به فرآیندهایی که به طور همزمان اجرا می‌شوند، استفاده می‌کند. اگر این برهه‌های زمانی به حد کافی کوچک باشند و تعداد کارهایی که می‌خواهیم همزمان انجام شوند، زیاد نباشد، به نظر می‌رسد که کارها به طور همزمان اجرا می‌شوند.

چندوظیفه‌ای چیز تازه‌ای نیست. چندوظیفه‌ای حتی در Mainframeها نیز وجود دارد. MainFrameها این توانایی را دارند که صدها ترمینال به آن‌ها متصل شوند و کاربر هر ترمینال احساس می‌کند که ماشین منحصراً در اختیار او است. به علاوه، سیستم‌عامل MainFrameها اغلب به کاربران این امکان را می‌دهد که کارهایی را به زمینه (Background) بفرستند. این کارها زمانی انجام می‌شوند که کاربر می‌تواند روی چیزهای دیگر کار کند.

چندنخی قابلیتی است که به یک برنامه اجازه می‌دهد درون خودش، چندوظیفه‌ای شود. برنامه می‌تواند به چند نخ اجرای جداگانه که به طور همزمان اجرا می‌شوند، شکسته شود. در ابتدا این امر چندان مهم جلوه نمی‌کند. ولی این برنامه‌ها می‌توانند از چندنخی برای اجرای کارهای طولانی در زمینه استفاده کنند، بدون آنکه کاربر مجبور باشد مدت زیادی کار با ماشین را رها کند. در محیط‌های چندنخی، برنامه‌ها می‌توانند به تکه‌هایی (که نخ اجرا نام دارند) شکسته شوند که همزمان اجرا شوند.

در پیاده‌سازی، یک نخ با یک تابع نشان داده می‌شود که ممکن است توابع دیگری را فراخوانی کند. اجرای یک برنامه با نخ اصلی (اولیه)اش شروع می‌شود که در برنامه‌های سنتی C، main نام دارد. در حین اجرا، برنامه می‌تواند با مشخص کردن نام یک تابع در یک فراخوانی سیستم، نخ جدیدی ایجاد کند. سیستم‌عامل به همان طریقی که بین فرآیندها رفت و آمد می‌کند، به طور نوبه‌ای بین نخ‌ها رفت و آمد می‌کند.

۸-۱-۲ امکانات لینوکس برای کار با نخ‌ها

در لینوکس توابع مربوط به نخ‌ها در کتابخانه pthread.h تعریف شده‌اند. این توابع جزء توابع کتابخانه‌ای استاندارد C نیستند. بنابراین هنگام استفاده و لینک کردن فایل‌های object باید از پارامتر lpthread- استفاده شود. به عنوان مثال اگر نام برنامه‌ای که در آن از کتابخانه نخ‌ها استفاده شده است prog1.c باشد، آنگاه برای کامپایل و اجرای آن باید دستورات زیر را استفاده کرد:

```
gcc –c prog1.c
```

```
gcc –o prog1 prog1.o –lpthread
./prog1
```

از تابع pthread_create برای ایجاد نخ استفاده می‌شود که پارامترهای آن به صورت زیر است:

```
pthread_create(& th_id, NULL, &function_name, NULL);
```

توضیحات مربوط به پارامترهای این تابع در ادامه آمده است.

پارامتر اول، th_id، شماره شناسایی thread ایجاد شده است. این شماره شناسایی، توسط تابع pthread_create ایجاد و به این متغیر انتساب داده می‌شود. این متغیر باید در برنامه به صورت زیر از نوع pthread_t تعریف شود:

```
pthread_t th_id;
```

پارامتر دوم که در این مثال با NULL مقداردهی شده است، یک استراکچر است که حاوی ویژگی‌هایی است که نخ طبق آن ایجاد می‌شود. برخی از این ویژگی‌ها عبارتند از: اندازه پشته، اولویت، سیاست زمان‌بندی. در صورتی‌که این پارامتر NULL گذاشته شود به این مفهوم است که باید از مقادیر پیش‌فرض استفاده شود.

پارامتر سوم: نام تابعی است که قرار است در این نخ اجرا شود. هر نخ وقتی ایجاد شد باید شروع به اجرای کد تخصیص داده شده به آن نماید. نحوه تعریف این تابع به صورت زیر است:

```
void * function_name(void *param)
```

همان‌طورکه در تعریف تابع دیده می‌شود، تعداد پارامترهای تابع فقط یکی است. سوالی که مطرح می‌شود این است که اگر تابع یک نخ نیاز به بیش از یک پارامتر داشته باشد آنگاه چطور می‌توان آن‌ها را برای تابع ارسال کرد؟ (مثلا فرض کنید قرار باشد تابع نخ، دو عدد را دریافت کند و محاسباتی را روی آن انجام دهد و نتیجه را چاپ کند). برای ارسال بیش از یک پارامتر به تابع، باید یک استراکچر تعریف کنیم و پارامترهایی را که قصد داریم به تابع ارسال کنیم، به عنوان فیلدهای آن استراکچر تعریف کنیم.

مثلا فرض کنید می‌خواهیم نخی ایجاد کنیم که رشته‌ای را دریافت کند و به تعداد دفعاتی که تعیین می‌شود آن را چاپ کند. بنابراین باید استراکچری به صورت زیر تعریف شود:

```
struct param {
char str[255];
int count;
}
```

پارامتر چهارم که در این مثال با NULL مقداردهی شده است، اشاره‌گر به پارامترهایی است که قرار است برای تابع نخ ارسال گردد. اگر این پارامتر NULL باشد به مفهوم این است که تابع نخ نیاز به پارامتری ندارد.

۲-۸ دستورکار

۱-برنامه زیر را با استفاده از ویرایشگر nano وارد و سپس کامپایل و اجرا کنید. خروجی برنامه را توجیه کنید.

```
#include<pthread.h>
#include<stdio.h>
void *printx(void * th_param)
{
while(1)
printf("x");
return NULL;
}
()main
{
;int i
;pthread_t th_id
;pthread_create(&th_id,NULL,&printx,NULL)
while(1)
;printf("0”)
;return 0
}
```

۲-تابعی به نام printy، بنویسید که کاراکتر y را در یک حلقه بینهایت چاپ کند. سپس برنامه سوال۱ را به نحوی تغییر دهید که دو نخ ایجاد شود، یکی تابع printx و دیگری تابع printy را اجرا کند. سپس برنامه را کامپایل و اجرا کنید. خروجی برنامه را توجیه نمایید.

۳-در برنامه قبلی، حلقه بینهایت که در تابع main وجود دارد را محدود کنید(مثلا حلقه ۱۰۰۰۰۰ بار اجرا شود) برنامه را کامپایل و اجرا کنید. با توجه به مشاهدات خود از اجرای برنامه چه نتیجه ای می گیرید؟

۴-در مثالهای قبل تابعی که به نخ انتساب داده می شد پارامتر نداشت. حال می خواهیم به جای توابع printx و printy که در سوال ۲ نوشته شد تابعی به نام print_char بنویسید که کاراکتری را به عنوان آرگومان دریافت و آن را چاپ کند. در برنامه اصلی با استفاده از دستورات زیر دو نخ ایجاد شوند که کاراکترهای a و b را چاپ نمایند.

```
char th_arg1='a';
char th_arg1='b';

;pthread_create(&th_id1,NULL,&print_char,&th_arg1)
;pthread_create(&th_id1,NULL,&print_char,&th_arg2)
```

راهنمایی: در تابع print_char با توجه به اینکه به صورت پیشفرض، پارامتر ورودی از نوع * void است باید با استفاده از دستور زیر، آن را به نوع * charتغییر دهید سپس با استفاده از دستور printf("%c",*ch) آنرا چاپ کنید.

```
char *ch=(char *)th_param;
```

۵-در برنامه قبل، پارامتر ارسال شده برای تابع نخ، فقط یک کاراکتر بود، حال می خواهیم برنامه سوال ۴ را به گونه ای تغییر دهیم که علاوه بر کاراکتر، تعداد دفعات چاپ آن نیز ارسال گردد(در واقع دو پارامتر برای تابع ارسال گردد) با توجه به توضیحات داده شده در بخش پیش گزارش، با تعریف استراکچری به صورت زیر، برنامه سوال ۴ را به گونه ای بازنویسی کنید که در تابع print_char کاراکتر دریافت شده را به تعداد دفعات ذکر شده در پارامتر چاپ کند.

```
struct th_param {
char ch;
int count;
}
```

نحوه فراخوانی در تابع اصلی:

```
struct th_param th_arg1, th_arg2;
th_arg1.ch='a';
th_arg1.count=100000;
th_arg2.ch='b';
th_arg2.count=300000;
;pthread_create(&th_id1,NULL,&print_char,&th_arg1)
;pthread_create(&th_id1,NULL,&print_char,&th_arg2)
```

آزمایش نهم: مدیریت نخ‌ها (هماهنگی بین نخ‌ها)

۹-۱ پیش آگاهی

در آزمایش ۸، با مفهوم نخ و نحوه ایجاد آن آشنا شدید. در این آزمایش نحوه هماهنگ سازی نخهای متعدد را خواهید دید و نحوه استفاده از سمافور برای هماهنگ کردن نخهای متعدد را آزمایش خواهید کرد.

۹-۱-۱ توابع مربوط به هماهنگی نخ‌ها

در این بخش، توابع مربوط به هماهنگی نخها، معرفی و نحوه استفاده از آنها آورده می شود.

تابع join

این تابع، شماره شناسایی یک نخ را دریافت می کند و برنامه فراخواننده خود را معلق می کند تا اینکه نخ مشخص شده به پایان برسد. تعریف این تابع به صورت زیر است:

```
#include <pthread.h>
int pthread_join(pthread_t thread, void **value_ptr);
```

در حالت ساده شده، مقدار پارامتر دوم را می توان NULL گذاشت.

سمافور mutex

Mutex یک سمافور باینری است که برای مدیریت همزمانی دسترسی به ناحیه بحرانی استفاده می شود. برای استفاده از این سمافور ، باید ابتدا یک متغیر از نوع mutex به صورت زیر تعریف شود(با توجه به اینکه این متغیر، یک متغیر سراسری باید باشد که توسط توابع نخ و برنامه اصلی قابل مشاهده باشد بنابراین تعریف آن باید در ابتدای برنامه باشد(بلافاصله بعد از #include های برنامه))

pthread_mutex_t *my_mutex_name* : **PTHREAD_MUTEX_INITIALIZER**;

در دستور فوق، نامی که برای mutex انتخاب شده است *my_mutex_name* است.

بعد از اینکه متغیر تعریف شد، با استفاده از دو دستور زیر میتوان متغیر mutex را قفل یا آزاد کرد

pthread_mutex_lock(&*my_mutex_name*)

pthread_mutex_unlock(&*my_mutex_name*)

۲-۹ دستور کار

۱-تابع main، سوال آخر از آزمایش ۸ را به صورت زیر بنویسید

```
()main
{
;int i
;pthread_t th_id1, th_id2
struct th_param th_arg1, th_arg2;
th_arg1.ch='a';
th_arg1.count=100000;
th_arg2.ch='b';
th_arg2.count=100000;
;pthread_create(&th_id1,NULL,&print_char,&th_arg1)
;pthread_create(&th_id1,NULL,&print_char,&th_arg2)
printf("\nTwo Threads are generated");
pthread_join((&th_id1,NULL);
pthread_join((&th_id2,NULL);
printf("\nThreads have been finished");
}
```

الف) برنامه را اجرا کنید و مشاهدات خود را بنویسید.

ب) اگر در برنامه فوق، دو دستور pthread_join را حذف کنید چه اتفاقی می افتد؟

ج) اگر در برنامه فوق، جای دو دستور pthread_join را جابجا کنید چه اتفاقی می افتد؟

د) اگر دو دستور join را حذف کنید و قطعه کد زیر را جایگزین کنید، نتیجه اجرا چه خواهد شد؟ توجیه کنید

```
While1(1)
{
```

```
Printf(“1”);
pthread_join((&th_id1,NULL);
Printf(“2”);
pthread_join((&th_id2,NULL);
Printf(“3”);
}
```

ه) تکه برنامه فوق را به گونه ای تغییر دهید که در صورت اتمام دو نخ، از حلقه خارج شود.

و) روشی ارائه دهید که در انتهای برنامه main، پیغامی چاپ کند که نشان دهد کدام یک از دو نخ زودتر به پایان رسیده اند.

۲-برنامه زیر را در نظر بگیرید. همانگونه که ملاحظه می کنید ترتیب چاپ شدن x و y ها تحت کنترل برنامه نویس نیست و کاملا وابسته به نحوه زمانبندی CPU توسط سیستم عامل است. در این بخش قصد داریم با استفاده از mutex ترتیب چاپ شدن را تحت کنترل برنامه نویس در بیاوریم.

```
()main
{
;int i
;pthread_t th_id1,th_id2
;pthread_create(&th_id1,NULL,&printx,NULL)
;pthread_create(&th_id2,NULL,&printy,NULL)
pthread_join((&th_id1,NULL);
pthread_join((&th_id2,NULL);
;return 0
}
```

```
#include<pthread.h>
#include<stdio.h>
void *printx(void * th_param)
{
for(int i=0; i<100; ++i)
printf("x");
return NULL;
}
void *printy(void * th_param)
{
for(int i=0; i<100; ++i)
printf("y");
} return NULL;
```

الف) چگونه با تعریف دو mutex میتوان برنامه را به نحوی تغییر داد که چاپ x و y ها به این صورت باشد: .....xyxyxyxyxy.

ب) برنامه را به گونه ای تغییر دهید که خروجی زیر را تولید کند.

۵۰تا ۵۰تا ۵۰تا ۵۰تا

xx...x yy...y xx...x yy...y

۳-برنامه سوال ۲ را با تغییرات زیر به صورت مساله کلاسیک تولید کننده- مصرف کننده تبدیل نمایید.

تابع printx را به produce_y تغییر نام دهید. سپس یک آرایه ۱۰۰ عنصری از نوع کاراکتر تعریف و آنرا با '0' مقدار دهی نمایید. دو متغیر به نامهای front و tail از نوع صحیح تعریف نمایید(برای پیاده سازی یک بافر حلقوی). تابع produce_y، در یک حلقه، با ایجاد یک تاخیر بین ۱ تا ۱۰ ثانیه، یک کاراکتر در بافر می نویسد و این کار را در یک حلقه ۱۰۰۰ تایی تکرار می کند و تابع printy، دائما در حال چک کردن بافر حلقوی است به محض دریافت یک کاراکتر آن را چاپ می کند.

برای ایجاد تاخیر از کتابخانه و تابع زیر استفاده کنید:

```
#include <unistd.h>
int usleep(useconds_t usec);
```

برای تولید عدد تصادفی از کتابخانه و تابع زیر استفاده کنید:

```
#include <stdlib.h>
int rand(void);//یک عدد صحیح بین صفر تا ۳۲۷۶۸ برمیگرداند
```

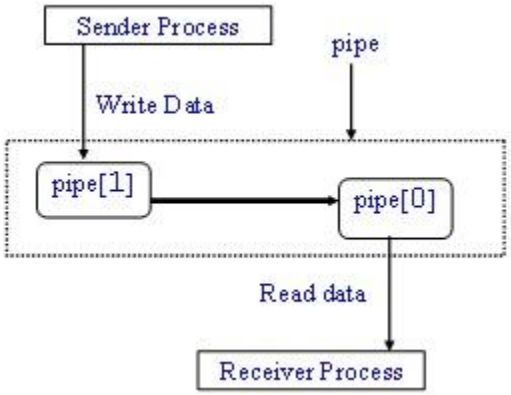
۴-یک نخ جدید به برنامه سوال ۳ اضافه کنید. کار این نخ به این صورت باشد که اگر صف خالی بود، کل بافر را با صفر پر کند سپس مادامی که صف خالی است، کاراکتر '0' را با فاصله زمانی یک ثانیه به صورت مرتب چاپ کند.

آزمایش دهم: مدیریت فرآیندها (ارتباط و هماهنگی بین فرآیندها)

۱-۱۰ پیش‌آگاهی

هدف از این آزمایش، آشنایی با ارتباط فرآیندها با لوله (Pipe) و دوطرفه‌کردن (Dup)، آشنایی با سیگنال‌ها و چگونگی فراخوانی آن‌ها، همزمان کردن فرآیندها با استفاده از سیگنال‌ها و پیاده‌سازی یک الگوریتم موازی با استفاده از فرآیندها و سیگنال‌ها است.

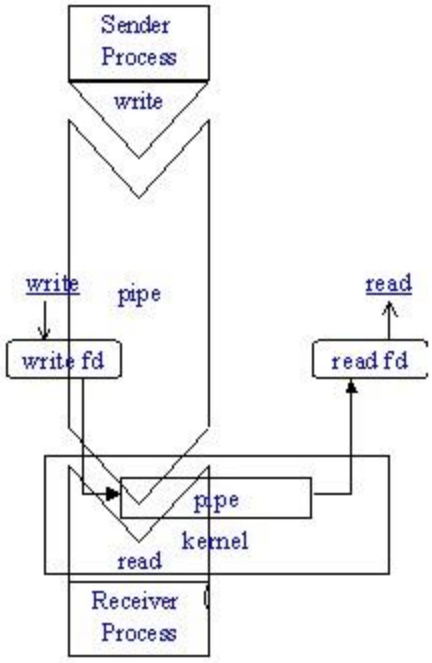
۱-۱-۱۰ لوله (Pipe)



شکل ۱۰-۱: نحوهٔ عملکرد لوله

ج)

الف)



یکی از روش‌های انتقال اطلاعات بین فرآیندها، لوله (Pipe) است. یک فرآیند، اطلاعات را در یک سوی لوله می‌نویسد و فرآیند دیگر آن‌ها را از سوی دیگر لوله می‌خواند. لوله از دو پرونده تشکیل شده است که یکی از این پرونده‌ها فقط خواندنی و دیگری فقط نوشتنی است. بنابراین فرآیندی که می‌خواهد اطلاعات را ارسال کند، باید در پروندهٔ فقط نوشتنی بنویسد تا فرآیند دریافت‌کنندهٔ اطلاعات، آن‌ها را از پروندهٔ فقط خواندنی بخواند. خط‌لوله معمولاً بین فرآیند پدر با فرزندش برقرار می‌شود. نحوهٔ عملکرد لوله را با استفاده از شکل ۱۰-۱ بهتر می‌توان درک کرد. همان‌طورکه در شکل ۱۰-۱-ب نشان داده شده است، اطلاعاتی که از یک لوله عبور می‌کند، از داخل هسته می‌گذرد.

برای ایجاد لوله از فراخوان سیستم با نام pipe استفاده می‌شود. این ب)
فراخوان سیستم آدرس یک آرایهٔ دوتایی از جنس int را دریافت
می‌کند و در صورت موفقیت، دو شناسهٔ پرونده در آن‌ها قرار می‌دهد. شناسهٔ اولین پرونده، فقط خواندنی و شناسهٔ دومی، فقط نوشتنی است.

اگر پس ازpipe یک fork نیز انجام شود، فرآیند فرزند دو پروندهٔ متعلق به لوله را (که در فرآیند پدر وجود دارند) در اختیار خواهد داشت. اکنون اگر فرآیند پدر بخواهد برای فرزند اطلاعات بفرستد، باید یک لوله ایجاد کند و اطلاعات را در پرونده فقط نوشتنی بنویسد. فرآیند فرزند با خواندن از پروندهٔ فقط خواندنی، این اطلاعات را دریافت می‌کند. فرآیند پدر چون به پروندهٔ فقط خواندنی احتیاج ندارد، آن را می‌بندد. فرآیند فرزند نیز پروندهٔ فقط نوشتنی را می‌بندد.

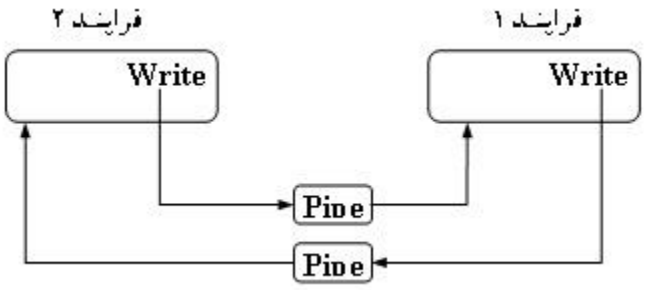
```
H> .#include <unistd

H> .#include <stdio

int main(void)

{

int pfd[2];

char *s = "OS-LAB";

char p[6];

if(pipe(pfd) == -1)

{

perror("Pipe()");

return 1;

}

if(write(pfd[1], s, strlen(s) + 1) == -1)

{

perror("write()");

return 2;

}

if(read(pfd[0], p, strlen(s)+1) == -1)

{

perror("read()");

return 3;

}

printf("Send string = %s\n", s);

printf("Received string = %s\n", p);

return 0;

}
```

شکل ۱۰-۲: ایجاد لوله

به برنامه شکل ۹-۲ توجه کنید. این برنامه یک لوله ایجاد کرده و ابتدا در پروندهٔ فقط نوشتنی آن، می‌نویسد و سپس سعی می‌کند از پروندهٔ فقط خواندنی، بخواند. شاید این مورد چندان اهمیت لوله را نشان ندهد، ولی اگر بعد از این مراحل یک فرزند ایجاد شود، آن نیز می‌تواند محتویات پروندهٔ فقط خواندنی را که توسط پدرش در پروندهٔ فقط نوشتنی قرار گرفته است، بخواند.

اگر پس از ایجاد لوله، یک فرزند ایجاد شود، فرآیندهای پدر و فرزند (هر دو)، یک پروندهٔ فقط نوشتنی و یک پروندهٔ فقط خواندنی خواهند داشت. پدر می‌تواند در پروندهٔ فقط نوشتنی بنویسد و خودش یا فرزندش از پروندهٔ فقط خواندنی بخوانند و یا بالعکس فرزند در پروندهٔ فقط نوشتنی بنویسد و خودش یا پدرش از پروندهٔ فقط خواندنی بخوانند. البته اینکه چه موقع یک فرآیند شروع به خواندن کند تا از وجود اطلاعات معتبر در پرونده مطمئن شود، نیازمند مدیریت کافی فرآیندها و پرونده‌ها است (مثلاً پدر خود را منتظر نگه دارد کند تا فرزند اطلاعات را بنویسد و...).

لازم به ذکر است که معمولا ایجاد لوله به خاطر برقرار ساختن ارتباطی یک طرفه از پدر به فرزند یا بالعکس است. ایجاد ارتباطات دیگر نیاز به مدیریت قوی فرآیندها و پرونده‌ها دارند (مثل ایجاد ارتباط دو طرفه بین دو فرآیند با یک لوله) و یا چندان لازم نیستند (مثل ارتباط یک فرآیند با خودش). برای برقراری ارتباط دوطرفه بین دو فرآیند، بهتر است دو لوله در جهات مخالف هم ایجاد شود (مطابق شکل ۳-۱۰).



شکل ۱۰-۳: ایجاد ارتباط دوطرفه بین فرآیندها

۱۰-۱-۲-۱ دوطرفه‌کردن (Dup)

معمولاً خروجی فرآیندهای پدر و فرزند همه در یک جا ارائه می‌گردد و باعث بروز مشکلاتی می‌شود. اکثر فرمان‌های Linux، خروجی خود را در پروندهٔ خروجی استاندارد (stdout) می‌نویسند. اما گاهی لازم می‌شود که خروجی در جای دیگری مثلاً یک پرونده نوشته شود: می‌خواهیم که یک فرآیند فرزند یکی از فرمان‌های پوسته، مثل ps یا ls را اجرا کند تا خروجی آن از طریق یک لوله به فرآیند

پدر منتقل شود. ولی خروجی این دستورات به‌طور طبیعی در خروجی استاندارد (صفحهٔ نمایش) نوشته می‌شود که مطلوب ما نیست. برای رفع این مشکلات، Linux، امکان دوطرفه کردن (Duplicat) شناسهٔ یک پرونده را در اختیار گذاشته است.

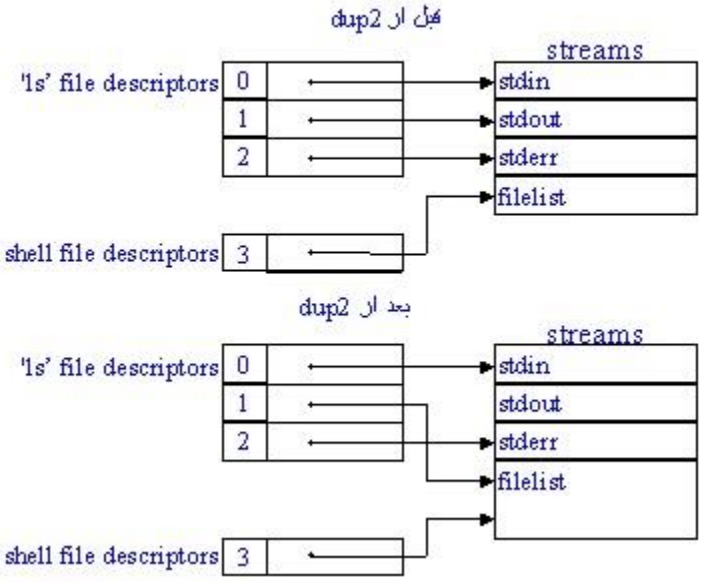
فراخوان سیستمی دوطرفه‌کردن (fd)، اولین پروندهٔ بسته (بلااستفاده) را متناظر با fd می‌کند (عملاً پروندهٔ fd از دو طریق قابل دسترسی است؛ اول از طریق خودش و دیگری از طریق پروندهٔ متناظر شده). بنابراین اگر ما پروندهٔ شمارهٔ ۱ را که مخصوص خروجی استاندارد (stdout) است، ببندیم و پروندهٔ شمارهٔ صفر هنوز باز باشد، عبارت دوطرفه‌کردن (fd) باعث خواهد شد که از این به بعد هر چیز که قرار است در پروندهٔ شمارهٔ ۱ (stdout) نوشته شود، به پروندهٔ fd منتقل شود و دیگر چیزی بر صفحهٔ نمایش نشان داده نشود. اگر این پرونده (fd)، پروندهٔ فقط نوشتنیِ یک لوله باشد، فرآیند آن سوی لوله نیز از چیزی که قرار بود به صفحه نمایش منتقل شود، آگاه خواهد شد.

گونهٔ بهتر شده‌ای از دوطرفه‌کردن با نام dup2 نیز وجود دارد که دو پارامتر می‌گیرد. به‌جای آنچه که در بخش قبلی با dup انجام گرفت، کافی است عبارت dup2(fd , 1) نوشته شود. در این حالت پروندهٔ مشخص شده با پارامتر دوم در صورت باز بودن بسته می‌شود و متناظر با پرونده پارامتر اول می‌شود.

برای مثال اگر بخواهید خروجی ls را در یک پرونده ذخیره کنید، فرمان زیر را وارد کنید:

```
$ls > test1
```

با این فرمان، خروجی ls به جای خروجی استاندارد، در پروندهٔ test1 نوشته می‌شود. اما در پشت پرده کاری مطابق شکل ۴-۱۰ انجام می‌گیرد:



شکل ۱۰-۴: نحوهٔ عملکرد dup2

فرآیند پوسته با استفاده از fork فرمان ls را اجرا می‌کند و فرآیند فرزند، قبل از احضار یک تابع exec، پروندهٔ test1 را ایجادکرده و dup2 را احضار می‌کند تا آنچه که در شکل ۱۰-۵ نشان داده شده است، انجام گیرد. ملاحظه می‌شود که نیازی به کپی‌کردن محتویات پروندهٔ مربوط به صفحه نمایش (stdout) در پروندهٔ test1 نیست و همهٔ کارها از طریق شناسهٔ پرونده‌ها انجام می‌گیرد. سهولت این کار به این دلیل است که همه چیز از دیدگاه Linux، پرونده است.

برای مثال مذکور، باید عملیاتی شبیه به قطعه کُد شکل ۱۰-۵ در shell اجرا شود.

```
pid = fork();
switch(pid)
{
case -1: /* error */
case 0: /* child */
{
int fd;
fd = creat("filelist",0666); /*creat write & read file*/
if(fd == -1)
exit(1);
if(dup2(fd, 1) == -1)
exit(2);
if(execl("/bin/ls", NULL) == -1)
exit(3);
}
default: /* parent */
```

```
}
```

شکل ۱۰-۵: نحوهٔ عملکرد shell در موقع تغییر مسیر دادن خروجی

به نحوهٔ احضار dup2 توجه کنید. این تابع نیز مثل بقیهٔ فراخوان‌های سیستم در صورت بروز خطا -۱ باز می‌گرداند. ملاحظه می‌کنید که قبل از انجام execl لازم است عملیاتی انجام شود. در این مثال لازم است که ابتدا یک پرونده ایجاد شده و dup2 احضار شود تا نوبت به execl برسد.

۱۰-۱-۳- سیگنال‌ها و فراخوانی آن‌ها در Linux

سیگنال پیامی است که یک فرآیند به دیگری می‌فرستد و فرآیند دیگر با دریافت آن، کار به خصوصی را انجام می‌دهد. معانی و کارکردهای از پیش تعریف شده‌ای برای اغلب سیگنال‌ها وجود دارند. برای مثال، سیگنال SIGFPE به یک فرآیند اطلاع می‌دهد که یک خطای floating point اتفاق افتاده است و سیگنال SIGKILL فرآیند را مجبور می‌کند تا به کار خود پایان دهد.

یک فرآیند را می‌توان طوری سازمان داد تا بعضی از سیگنال‌ها را نادیده بگیرد و یا در جواب آن‌ها کار دیگری انجام شود. این کار به وسیلهٔ فراخوان سیستم signal امکان‌پذیر است که دراینجا ما به آن نمی‌پردازیم.

یک سیگنال را می‌توان از داخل برنامه به وسیلهٔ فراخوان سیستم kill، به فرآیند یا فرآیندهای دیگر ارسال کرد. قالب کلی آن به شکل زیر است:

```
pid , int sig) int kill (pid_t
```

pid شماره فرآیندی است که می‌خواهیم سیگنال به آن فرستاده شود و sig شمارهٔ سیگنال مورد نظر است. می‌توان به جای sig، نام سیگنال مورد نظر را ذکر کرد. در صورت موفقیت، مقدار صفر و در غیر این صورت یک ۱- برگردانده خواهد شد.

در این آزمایش، ما از دو سیگنال با نام‌های SIGCONT و SIGSTOP استفاده خواهیم کرد. سیگنال SIGCONT، یک فرآیند متوقف شده را دوباره به کار می‌اندازد و سیگنال SIGSTOP، یک فرآیند در حال کار را متوقف می‌سازد.

۱۰-۱-۴- الگوریتم مرتب‌سازی موازی زوج‌فرد (OddEven)

مرتب‌سازی موازی زوج‌فرد که در شکل ۱۰-۶ نشان داده شده است، نمونهٔ ساده‌ای از الگوریتم‌های موازی است. الگوریتم‌های موازی، الگوریتم‌هایی هستند که قسمت‌های مختلف آن می‌توانند به موازات هم انجام شوند. این الگوریتم‌ها از سرعت بالایی برخوردار هستند و پیاده‌سازی واقعی آن‌ها در یک محیط چندپردازنده‌ای امکان‌پذیر است، ولی می‌توان در سیستم‌های چندوظیفه‌ای نیز آن‌ها را پیاده‌سازی کرد و بخش‌های موازی این الگوریتم‌ها را بر عهدهٔ فرآیندهای مختلف گذاشت. این الگوریتم‌ها در سیستم‌های چند وظیفه‌ای که تنها یک پردازنده دارند، سرعت واقعی خود را نخواهند داشت، زیرا در نهایت تمامی عملیات توسط یک پردازنده انجام می‌شود. اکنون به شرح این الگوریتم می‌پردازیم و سپس با تغییرات اندکی، آنرا بوسیلهٔ فرآیندها در Linux پیاده‌سازی خواهیم کرد:

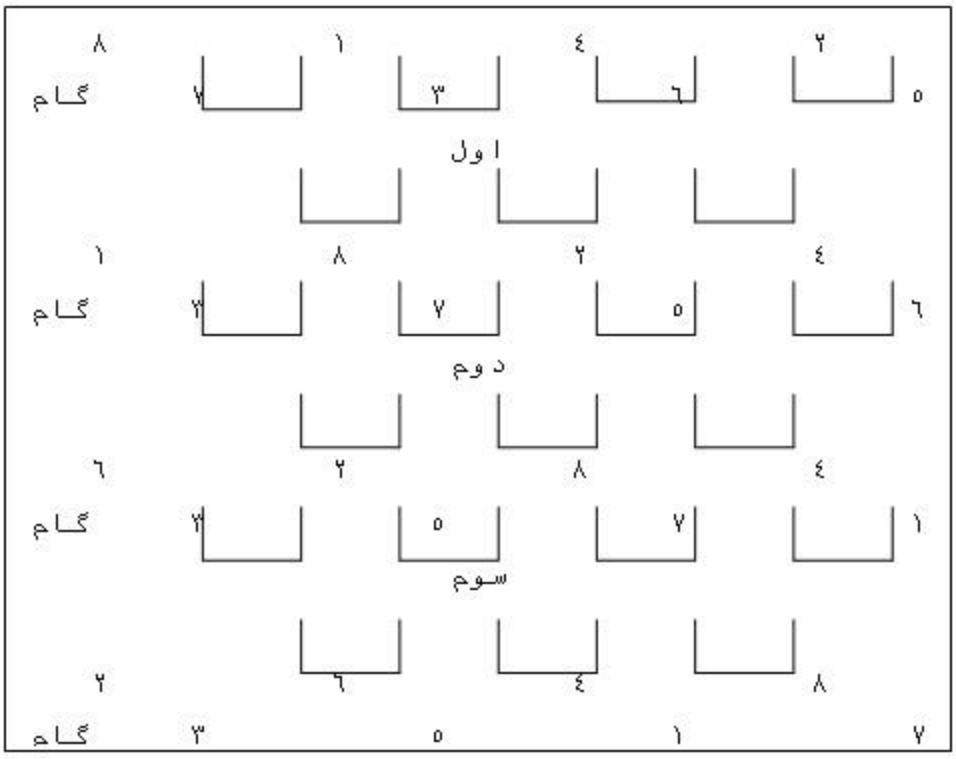
فرض می‌کنیم n پردازنده با نام‌های P1، P2، ... و Pn و n ورودی با نام‌های X1، X2، ... و Xn وجود دارند. هر پردازنده یک ورودی می‌گیرد و آن را در تنها ثبات خود قرار می‌دهد. هدف، مرتب‌سازی ورودی‌ها در میان پردازنده‌ها است، به‌طوری‌که کوچک‌ترین ورودی در ثبات P1 و دومین کوچک‌ترین ورودی در P2 و بالاخره بزرگ‌ترین ورودی در Pn قرار گیرد. در حالت تعمیم یافته که بیش‌تر مورد نظر ماست، هر پردازنده می‌تواند بیش از یک ورودی بگیرد و در ثبات‌های خود قرار دهد که در این مورد بعدا بحث خواهد شد.

پردازنده‌ها به شکل خطی به هم متصل هستند و هر پردازنده فقط می‌تواند با همسایهٔ سمت راست خود ارتباط برقرار کند. بنابراین عمل مقایسه و تعویض بر روی عناصری می‌توانند انجام شوند که در صف پردازنده‌ها پشت سر هم باشند.

همان‌طورکه گفتیم، هر پردازنده عدد موجود در تنها ثبات خود را با عدد موجود در ثبات همسایهٔ سمت راستش مقایسه می‌کند و بنابراین تعویض، زمانی اتفاق می‌افتد که ترتیب اعداد موجود در ثبات‌های دو پردازندهٔ به‌دنبال هم، درست نباشد. سپس این روند برای اعداد بعدی تکرار خواهد شد تا محتوای موجود در ثبات پردازنده‌ها ترتیب صحیحی داشته باشند. به نظر می‌رسد که الگوریتم در بدترین حالت باید n-1 بار انجام شود و آن وقتی است که یک ورودی از یک طرف صف پردازنده‌ها به سوی دیگر آن حرکت کند.

گام‌های موجود در الگوریتم به گام‌های زوج و فرد تقسیم می‌گردند. در گام‌های زوج، پردازنده‌های با شماره‌های زوج، اعداد خود را با همسایگان راستشان مقایسه می‌کنند و در گام‌های فرد، پردازنده‌های با شمارهٔ فرد همین کار را انجام می‌دهند. اگر پردازنده‌ای همسایهٔ متناظر نداشته باشد، در آن مرحله هیچ عمل مقایسه‌ای انجام نمی‌دهد.

یک مثال عددی از این الگوریتم در شکل ۱۰-۷ آمده است. در این مثال، مرتب سازی پس از شش مرحله کامل گشته است. در حالت کلی، پیش‌بینی تعداد مراحل اجرای الگوریتم کار سختی است و در این آزمایش بهتر است اجازه دهیم تا الگوریتم در بدترین حالتِ اجرای خود، کار کند.



شکل ۱۰-۶: الگوریتم مرتب‌سازی موازی زوج‌فرد (OddEven)

```
Algorithm Odd-Even sort (X, n)
Input: X (an array in the range 1 to n, such that Xi resides at Pi)
Output: X (the array in sorted order, such the i th smallest element is in Pi)
begin
do in parallel [n/2] times
P2i-1 and P2i compare their elements and exchange them if necessary;
for all i, such that 1 <= 2i <= n
P2i and P2i+1 compare their elements and exchange them if necessary;
for all i, such that 1<= 2i <= n
if n is odd, then this step is done only [n/2] time
end
```

شکل ۱۰-۷: مثال عددی از مرتب‌سازی زوج‌فرد

اکنون فرض کنید n فرآیند با نام‌های P_1، P_2، ... و P_n و آرایه‌ای شامل m داده وجود دارند (m>n). هدف، مرتب کردن عناصر این آرایه است به‌طوری‌که کوچک‌ترین عنصر در اولین خانهٔ آرایه و بزرگ‌ترین عنصر در آخرین خانهٔ آرایه قرار گیرند. مقایسهٔ داده‌ها و تعویض محل آن‌ها وقتی ممکن است که این داده‌ها در آرایه پشت سر هم باشند. در این الگوریتم، آرایه به n قسمت حتی‌الامکان مساوی تقسیم می‌شود. سپس هر قسمت در اختیار یکی از فرآیندها قرار می‌گیرد و فرآیند مربوطه، قسمت مورد نظر خود را مرتب می‌کند. برای مثال اگر سه فرآیند داشته باشیم، یک آرایهٔ ۱۶ عنصری را می‌توان در میان این فرآیندها به دو آرایهٔ پنج عنصری و یک آرایهٔ شش عنصری تقسیم کرد. هر کدام از این فرآیندها بر روی یکی از این بخش‌ها کار مرتب‌سازی را به شکل مقایسهٔ دو عدد دنبال هم در آرایه و تعویض آن‌ها در صورت نیاز، ادامه می‌دهد. پس از اینکه هر کدام از فرآیندها یک بار حلقهٔ خود را طی کرد، باید منتظر فرآیندهای دیگر بماند تا آن‌ها نیز کار خود را تمام کنند. سپس تمام فرآیندها با هم دور بعدی حلقه را شروع می‌کنند. در مرحلهٔ دوم، هر کدام از فرآیندها بخشی از آرایه را که در اختیار دارند یک خانه به جلو تغییر می‌دهند. برای مثال، فرآیند اول در گام اول روی عناصر اول تا ششم و در گام بعدی روی عناصر دوم تا هفتم و در گام سوم دوباره روی عناصر اول تا ششم کار خواهد کرد و این روند تا آخر تکرار خواهد شد.

مهم‌ترین مشکل در پیاده‌سازی این الگوریتم، هماهنگ‌سازی و کنترل همزمانی فرآیندها است که این کار می‌تواند به وسیلهٔ سیگنال‌ها انجام شود.

۱۰-۲-۲ دستورکار

۱۰-۲-۲-۱ آزمایش‌های مربوط به لوله و Dup

۱) قطعه برنامه‌ای را که در قسمت لوله آمده است، در پروندهٔ Pipe1.c قرار داده‌ایم. آن را ترجمه و اجرا کنید و مشاهدات اجرای برنامه را بنویسید.

۲) در پروندهٔ dup.c برنامه‌ای بنویسید که معادل دستور زیر باشد. قبل از نوشتن برنامه، طرح خود را آماده و در صورت لزوم با مسؤل آزمایشگاه در میان بگذارید:

```
ls -l > test1
```

چگونه می‌توان عملیات زیر را انجام داد:

```
$ ls >> test1
```

تذکر: علامت >> برای اضافه (Append) کردن به‌کار می‌رود.

۱۰-۲-۲-۲ آزمایش‌های مربوط به بخش signal

۱) برنامه‌ای بنویسید که پس از ایجاد فرآیند فرزند، آن را متوقف کرده و عملیات خود را انجام دهد (عملیات اختیاری است) و پس از انجام عملیات خود فرآیند فرزند را راه‌اندازی کند.

۳-۲-۱۰ بخش اختیاری آزمایش (الگورریتم Oddeven Sort)

۱) برنامه‌ای بنویسید که سه فرآیند فرزند ایجاد کند و سپس آن‌ها را با سیگنال‌ها هماهنگ کند. کار هر کدام از فرآیندهای فرزند این است که در داخل یک حلقه، ۱۰ بار پیغامی را چاپ کنند. می‌خواهیم طوری فرآیندها را هماهنگ کنیم که تا بارِ اول حلقهٔ همهٔ آن‌ها تمام نشده است، وارد بار دوم حلقه نشوند.

۲) پروندهٔ oddevensort.c را باز کرده و قسمت‌های مختلف آن را ببینید. تابع مرتب‌سازی بر اساس الگوریتم مرتب‌سازی موازی زوج‌فرد پیاده‌سازی شده است. با توضیحات مربی آزمایشگاه این برنامه را تکمیل کنید.