دستور کار آزمایشگاه سیستم عامل

دکتر لیلا شریفی

اکرم سلطانی

تابستان ۱۴۰۳

فهرست آزمایش‌ها

آزمایش 1

آشنایی و راه اندازی سیستم‌های لینوکسی

اهداف:

- آشنایی با مفاهیم اساسی لینوکس بعنوان یک سیستم عامل

ابزار و مفاهیم مورد نیاز:

- فایل نصبی یک توزیع از سیستم عامل لینوکس (ترجیحا آخرین ورژن اوبونتو)
- USB Flash/ DVD
- نرم افزارهای ایجاد فلش قابل بوت

محتوا:

- معرفی مفهوم نرم افزار آزاد
- اجزای تشکیل دهنده سیستم عامل گنو/لینوکس
- توزیع‌ها و کاربردهای لینوکس
- نصب لینوکس:
  - Live Distribution
  - نصب مجازی / محلی
  - مفهوم Dual Boot
  - ایجاد Bootable Flash
  - پارتیشن‌بندی در لینوکس و پارتیشن‌های سیستمی(Boot, Home, Swap)
  - Boot Loader

مفاهیم و شرح دستورکار

1-1- نرم افزار آزاد

شعاری در دنیای لینوکس هست که می گوید لینوکس یک سیستم عامل نیست، لینوکس یک فرهنگ است: فرهنگ آزادی.

این آزادی برمی گردد به ایده های آقای ریچارد استالمن که در ۱۹۸۳ با راه انداختن جنبشی به نام **جنبش نرم افزار آزاد** آن را شروع کرد و بعد در سال ۱۹۸۵ با تاسیس بنیادی به نام **بنیاد نرم افزار آزاد**، پایه های آن را مستحکم ساخت. نماد این بنیاد گوزن آمریکای شمالی است که گنو نام دارد.

از نظر بنیاد نرم افزار آزاد، هر برنامه برای اینکه آزاد شناخته شود باید چهار حق را برای دیگران قائل باشد. این چهار آزادی، آزادی های صفر تا سه نامیده می شوند:

- آزادی اجرای برنامه برای هر کاری (آزادی صفرم)
- آزادی مطالعه چگونگی کار برنامه و تغییر آن (پیش نیاز: متن برنامه) (آزادی یکم)
- آزادی تکثیر و کپی برنامه (آزادی دوم)
- آزادی تقویت و بهتر کردن برنامه و توزیع آن برای همگان (پیش نیاز: متن برنامه) (آزادی سوم)

دادن این آزادی‌ها با دادن سورس برنامه به دیگران قابل تامین است اما یک شرط نهایی وجود دارد که باعث می شود جنبش نرم افزار آزاد رشد کند: «اگر شما این آزادی ها را داشتید پس دیگران هم باید داشته باشند.»

به عبارت دیگر اگر شما یک برنامه آزاد را دریافت کنید، آن را تغییر دهید و بخواهید تغییرات خود را در اختیار دیگران بگذارید، باید آن را دوباره به شکل آزاد منتشر کنید.

اما این موضوع چطور ممکن است؟ در نرم افزارهای بسته (مثلا ویندوز یا فتوشاپ) قانون از تولید کننده حمایت می کند ولی اگر قرار باشد قانون همین حمایت را از آزادی ها هم انجام دهد، نیازمند متنی قانونی هستیم. این متن قانونی، GPL نام دارد.

جی پی ال یا **اجازه‌نامه عمومی همگانی (General Public Licence)**یک متن قانونی است که اگر نرم افزاری اشاره کند «این برنامه براساس مجوز جی پی ال منتشر شده»، قانون موظف است حقوق مولف را در مورد آن تضمین کند.

1-2- سیستم عاملی به نام گنو/لینوکس

دغدغه اصلی استالمن این بود که مردم باید کنترل کامل ابزاری که از آن استفاده می کنند را شخصا در دست داشته باشند و بتوانند با بررسی، بهبود و به اشتراک گذاشتن نرم افزارها، در خلق دنیایی بهتر و آزادتر سهیم باشند. مشخص است که این کار باید از پایین ترین لایه نرم افزاری شروع شود: سیستم عامل.

1-2-1-سیستم عامل آزاد گنو

استالمن و دیگران اسم پروژه خودشان را GNU گذاشتند (به فارسی گِنو). آنها بهترین معماری سیستم عامل موجود یعنی معماری UNIX که در آن زمان بر روی کامپیوترهای بزرگ به شکل تجاری و بسته موجود بود را انتخاب کردند و شروع کردند به نوشتن یک سیستم عامل کاملا آزاد براساس آن معماری پیشرفته.

براساس این معماری لازم بود تا این چهار جزء تکمیل شوند:

- کرنل (Kernel) که هسته اصلی سیستم عامل را تشکیل می داد و از آن انتظار می رفت با شناخت سخت افزارها بتواند با آنها ارتباط برقرار کند.
- محیط توسعه نرم افزار؛ مانند کمپایلرها و کتابخانه های مختلفی که بقیه باید از آنها برای ساخت برنامه در این سیستم عامل جدید استفاده کنند.

- دستورات عمومی زندگی روزمره مانند کپی فایل، برنامه هایی برای کارهای گرافیکی، پوسته ای متنی برای صادر کردن این دستورات، ابزارهایی برای فرمت کردن دیسک و. ...
- مستندات؛ راهنماهای سیستم عامل و توضیحات متنی پیرامون آن

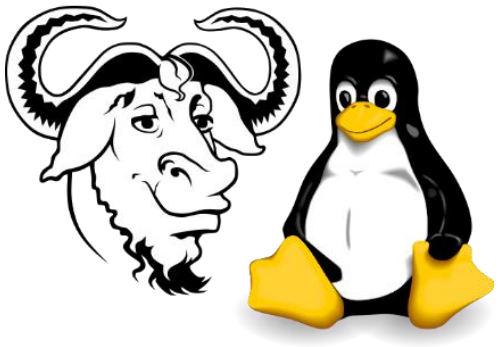
پروژه گنو به سرعت پیش رفت و سه جزء آخر را با کیفیتی باور نکردنی تکمیل کرد. کیفیت نرم افزارهای آزاد تولید شده برای سه بخش آخر در حدی بودند که بسیاری از سیستم عامل های دیگر (بخصوص یونیکسی ها) شروع به استفاده از آنها کردند.

1-2-2- تولید گنو/لینوکس

آخرین قطعه پازل برای ساخت یک سیستم عامل آزاد را فردی فنلاندی به نام **لینوس توروالدز** تکمیل کرد. لینوس که دانشجوی دانشگاه و یک برنامه نویس فوق العاده بود به عنوان تفریح شروع به نوشتن یک کرنل کرد که بعدها به نام **لینوکس** شناخته شد .

پس از اینکه لینوس توروالدز این پروژه را در سال ۱۹۹۱ با مجوز GPL منتشر کرد، تقاضا برای ادامه راه آنقدر زیاد شد که نه فقط سال های بعدی زندگی لینوس و هزاران نفر دیگر صرف تکمیل این هسته شده، که تا امروز هم شغل اصلی او کنترل این پروژه و هماهنگ کردن هزاران نفری است که این هسته را توسعه می دهند.

همراه شدن این هسته قوی و سه جزء دیگر یک سیستم عامل یونیکس که در پروژه گنو در حال پیگیری بودند، تشکیل چیزی را دادند که به شکل صحیح باید به آن «سیستم گنو/لینوکس» بنامیم .



تصویر 1-1- نماد لینوکس (Tux) و پروژه گنو (GNU)

1-3-توزیع ها

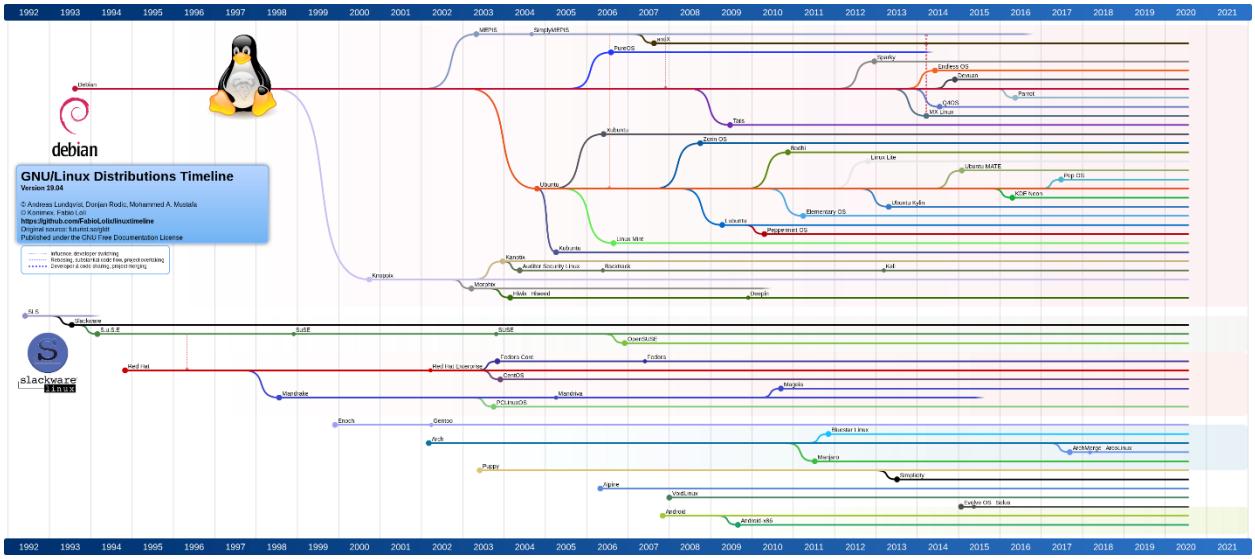
برای بوت کردن یک کامپیوتر با گنو/لینوکس سال ۱۹۹۲ باید کرنل را با یکی از ابزارهای گنو به فایل اجرایی تبدیل می کردید. بعد آن را از یک جای بخصوص هارد به بعد روی آن می‌نوشتید. فلگ‌هایی را تنظیم می کردید تا این برنامه بتواند کامپیوتر را بوت کرده و خط فرمان را اجرا کند و بعد تک تک برنامه های کاربردی و کتابخانه های نرم افزاری مورد نیاز آنها را جدا جدا باید کامپایل می کردید و براساس نظم و استانداردی که یونیکس استفاده می کند روی دیسک می‌نوشتید و اگر همه چیز درست و دقیق پیش رفته بود، سیستم با گنو/لینوکس بوت می شد. اینکار حتی حالا هم برای یک حرفه‌ای، یک پروژه جذاب پر از درگیری به حساب می آید!

برای حل این مشکل، افرادی شروع کردند به انجام تمام این مراحل پیچیده و وقت گیر و عرضه محصول نهایی به شکل یک مجموعه دیسک یا سی دی قابل نصب. این مجموعه ها، توزیع های لینوکس یا همانLinux Distro ها نامیده شدند. قدیمی ترین دیسترو (یا همان توزیع لینوکس) که هنوز هم زنده است، اسلکور (Slackware)نام دارد.

نه فقط هسته لینوکس و ابزارهای گنو که بسیاری برنامه های آزاد دیگر را هم روی توزیع‌ها عرضه می‌شدند. مثلا ممکن بود یک نفر تصمیم بگیرد که یک محیط کار گرافیکی، برنامه های لیبره آفیس، برنامه چت و فایرفاکس را در توزیع خود بگنجاند. در صورتی که شما این سی دی را در کامپیوتر می گذاشتید و آن را بوت می کردید با لینوکسی روبرو می شدید که این برنامه ها در آن نصب بود.

علاوه بر انتخاب بسته ها و تنظیمات عمومی، توزیع ها گاهی برنامه‌های مخصوص به خودشان را هم به کل سیستم عامل گنو/لینوکس اضافه می کنند. این برنامه ها معمولا برای تنظیمات راحت تر سیستم یا حذف و اضافه برنامه ها یا آپدیت برنامه های نصب شده بکار می روند.

از مشهورترین توزیع ها می شود به ردهت (RedHat) و دبیان (Debian) اشاره کرد که بسیاری از توزیع های مشهور دیگر از آنها منشعب شده اند. برای مثال اوبونتو از دبیان انشعاب یافته و فدورا (Fedora) هم به نوبه خود شاخه ای از ردهت به شمار می رود. در تصویر زیر توزیع‌های معروف و انشعاب‌هایشان را می‌بینید:



تصویر 1-2- تایم‌لاین پیدایش و تکامل توزیع‌های لینوکسی

1-4-کاربردهای لینوکس

سایت دیستروواچ مرجعی برای بررسی همه توزیع های لینوکس است و توزیع‌های فعال لینوکس را زیر نظر دارد. هر کدام از این توزیع ها با هدف متفاوتی ساخته شده‌اند:

- لینوکس سرور
- وب سرور
- سیستم های مخابراتی
- سازمان امنیت ملی آمریکا، پایگاه های پرتاب سفینه های فضایی و سیستم های کنترل ترافیک و کنترل پرواز
- سرور شرکت‌های انیمیشن سازی و جلوه های ویژه
- کامپیوترهای مین فریم(Main Frame)

- **سوپرکامپیوترها**

سریعترین سوپرکامپیوتر این لحظه (2024) جهان با نام Hewlett Packard Enterprise Frontier (OLCF-5، از سیستم عامل Cray که بر اساس توزیع SUSE Enterprise Linux توسعه داده شده است، استفاده می‌کند.

- لینوکس دسکتاپ

لینوکس روی دسکتاپ هم حضور دارد اما به گزارش TechSpot.com صرفا سهمی حدود 4٪ را به خود اختصاص داده است.

- لینوکس توکار(Embeded)

سیستم های توکار سیستم هایی هستند که در ظاهر یک کامپیوتر نیستند اما در باطن یک سیستم عامل مشغول اداره آنها است. مانند گوشی های موبایل اندرویدی که از کرنل لینوکس استفاده می کنند، خودروها و تلویزیون‌های هوشمند،

و حتی شتاب دهنده های ذرات و مریخ نوردها. این سیستم ها با لینوکس راه اندازی می شوند چرا که به خاطر آزاد و بازمتن بودن، این امکان هست که با حذف قسمت های غیر مورد نیاز، آن را تا حد ممکن کوچک کرد.

5-1- نصب لینوکس

در این قسمت با مراحل نصب لینوکس (هم بر روی ماشین مجازی و هم به صورت محلی به عنوان سیستم عامل دوم)، آشنا خواهیم شد.

Live Distribution -1-5-1

لینوکس یک سیستم زنده است. یعنی یک سی دی لینوکس، کامپیوتر را به شکل کامل بوت می کند و شما می توانید از لینوکس به شکل زنده و بدون نصب استفاده کنید. اینجا فرصت خوبی است برای محک زدن اینکه آیا از این توزیع خوشتان می آید یا نه. کافیست پس از بوت کردن سیستم با سی‌دی یا فلش نصاب، گزینه Live یا Try را انتخاب کنید و یا از ابتدا نسخه لایو را دانلود و سیستم را با آن بوت کنید.

سیستم عامل‌هایی که قابلیت اجرا به صورت "زنده" را دارند، بدون اینکه فرآیند نصب را انجام دهند، امکان اجرا شدن از روی سی دی یا فلش درایو را را دارند. همه اطلاعات برای بارگذاری سیستم عامل زنده، بر روی حافظه رم دستگاه قرار می‌گیرد و هیچ چیزی روی هارد درایو سیستم نوشته نمی‌شود. بنابراین به فایل‌ها یا سیستم عاملی که از قبل بر روی هارد نصب شده است، آسیبی نخواهد رسید. وقتی که کارتان با لینوکس زنده تمام شد می توانید سیستم را ریستارت کرده و فلش درایو یا سی دی را خارج کنید. کامپیوتر دوباره به همان حالت قبلی باز می گردد و با همان سیستم عامل پیشفرضش بارگذاری خواهد شد.

ویندوز نیز در نسخه enterprise 8 قابلیت اجرای زنده را اضافه کرده بود اما از ویندوز 10 به بعد آن را حذف کرده است.

1-5-2- نصب اوبونتو بر روی هارد دیسک به صورت Dual Boot

نکات مهم قبل از شروع نصب:

بسیاری از کاربران روش نصب native لینوکس بر روی دستگاه را انتخاب می کنند که دارای ویندوز نیز هست و از روش"Dual Boot" استفاده می کنند. که در آن، برای سوئیچ بین سیستم‌عامل‌ها، باید سیستم را ریبوت کنید و درمنوی بوت سیستم دلخواهتان را انتخاب کنید. برای نصب همزمان لینوکس و ویندوز، باید هارد دیسک خود را پارتیشن بندی کنید تا فضاهای جداگانه‌ای برای هر دو سیستم عامل ایجاد شود. اینجا فرض می‌کنیم ویندوز از قبل روی سیستم شما نصب شده و اکنون می‌خواهید سیستم عامل دوم را نصب کنید.

در ویندوز بر روی آیکون MyComputer راست کلیک کرده و گزینه Manage را انتخاب نمایید .در پنجره باز شده، وارد بخش disk management شده و یکی از درایوهای خود را به دو قسمت تقسیم کنید.(Shrink) یکی از قسمت ها باید حداقل 25 گیگ فضا داشته باشد. محتوای این درایو جدید را خالی کرده و درایو را پاک کنید. (راست کلیک روی درایو و انتخاب گزینه delete) به این ترتیب هنگام نصب تنها از این درایو استفاده خواهیم کرد و محتوای بقیه درایوها به خطر نمی‌افتند.

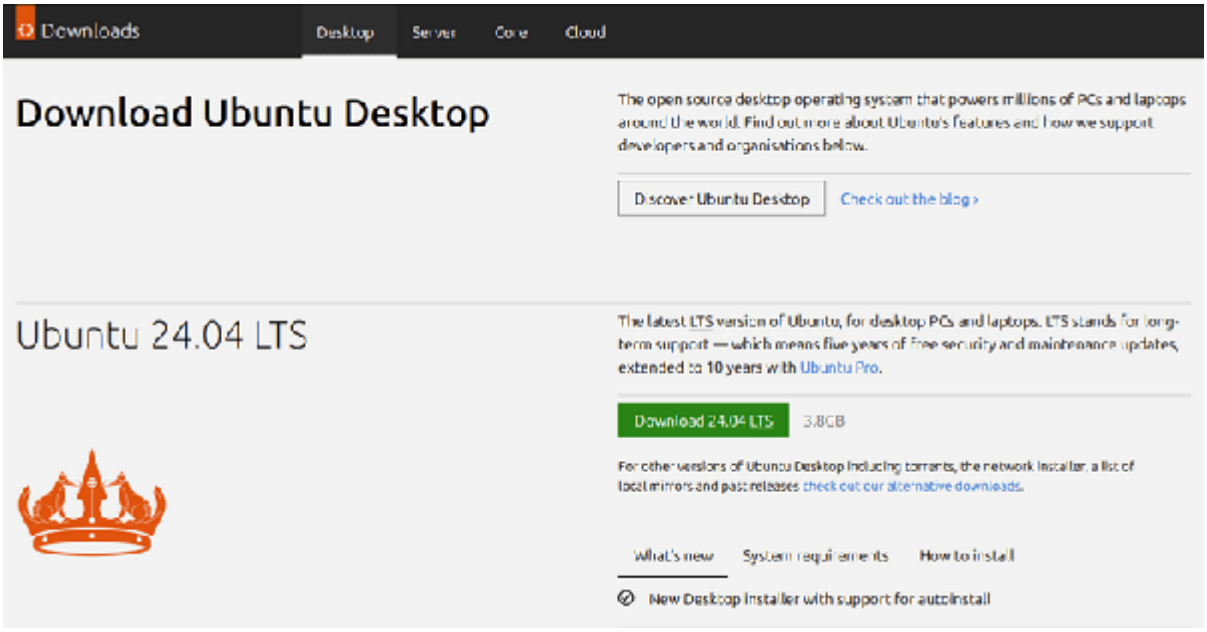
همچنین اگر قصد راه‌اندازی دوگانه (Dual Boot) یا راه‌اندازی Live یک توزیع لینوکس را دارید و ویندوز شما از Bitlockerبرای رمزگذاری داده‌ها استفاده می‌کند، حتماً کلیدهای رمزگذاری BitLocker خود را قبل از نصب لینوکس بازیابی و ذخیره کنید.

مراحل نصب:

1. فایل ISO لینوکس اوبونتو را دانلود کنید

از وب سایت رسمی اوبونتو آخرین نسخه Desktop Ubuntu را دانلود کنید. معماری مناسب (32 بیتی یا 64 بیتی) را بر اساس نوع سیستم خود انتخاب کنید. (دانلود سایر توزیع‌های محبوب)

در این راهنما از آخرین نسخه اوبونتو 24.04 LTS (منتشر شده در 25 آوریل 2024) استفاده شده است.



تصویر 1-3- دانلود اوبونتو

2. ایجاد USB قابل بوت (Bootable Flash)

برای این کار، نرم افزارهای مختلفی وجود دارد. از جمله Rufus, Balena Etcher, UNetBootin, UltraIso, Nero و غیره. اینجا از نرم‌افزار **balenaEtcher** استفاده خواهیم کرد، زیرا روی هر سه سیستم‌عامل های لینوکس، ویندوز و مک اجرا می‌شود. برای استفاده از آن، ابتدا نسخه ای را انتخاب کنید که با سیستم عامل فعلی شما مطابقت دارد، و آن را دانلود و نصب کنید.

فایل ISO دانلود شده خود را انتخاب کنید، سپس فلش USB خود را انتخاب کنید و روی گزینه Flash کلیک کنید.

3. بوت کردن از درایو فلشUSB

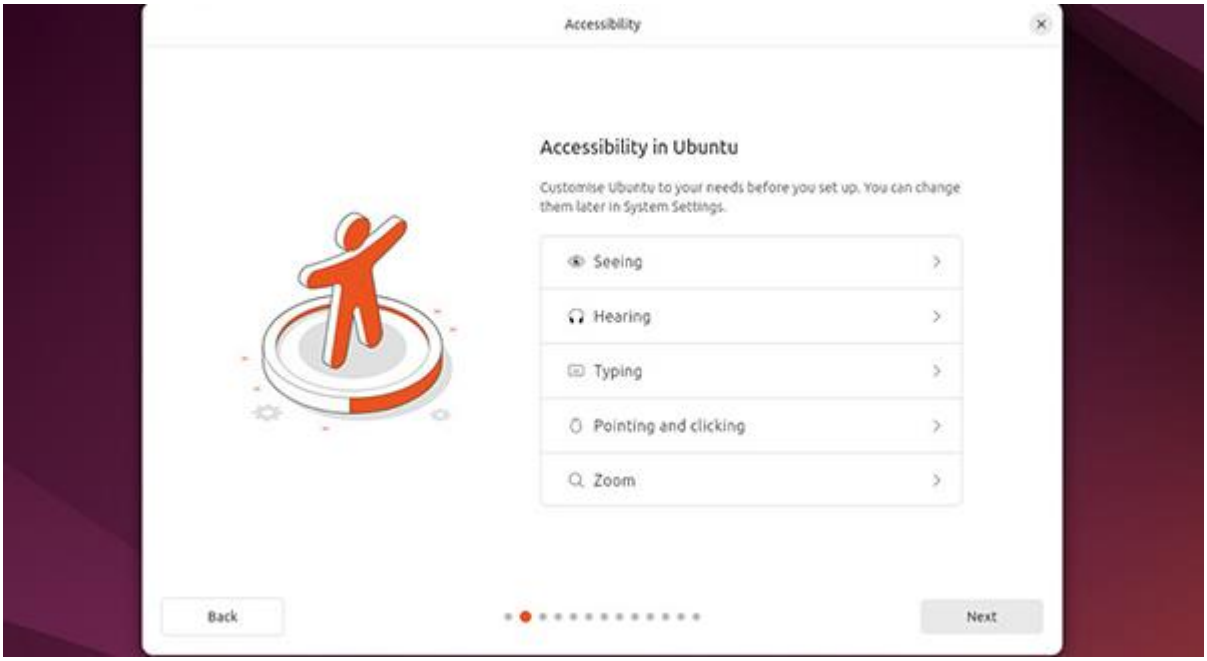
درایو فلش USB را به لپ تاپ یا سیستمی که می خواهید بر روی آن لینوکس اوبونتو را نصب کنید، متصل کنید و سیستم را راه اندازی مجدد کنید. سیستم باید رسانه نصب را به طور خودکار تشخیص دهد. اگر مشکلی پیش آمد و آن را تشخیص نداد، F12 را در هنگام راه‌اندازی نگه دارید و دستگاه USB را از منوی بوت مخصوص سیستم انتخاب کنید.

نکته : F12 رایج ترین کلید برای نمایش منوی بوت است، اما Escape ، Del, F2 و F10 نیز ممکن است در لپ‌تاپ ها یا سیستم‌های مختلف برای این کار استفاده شود

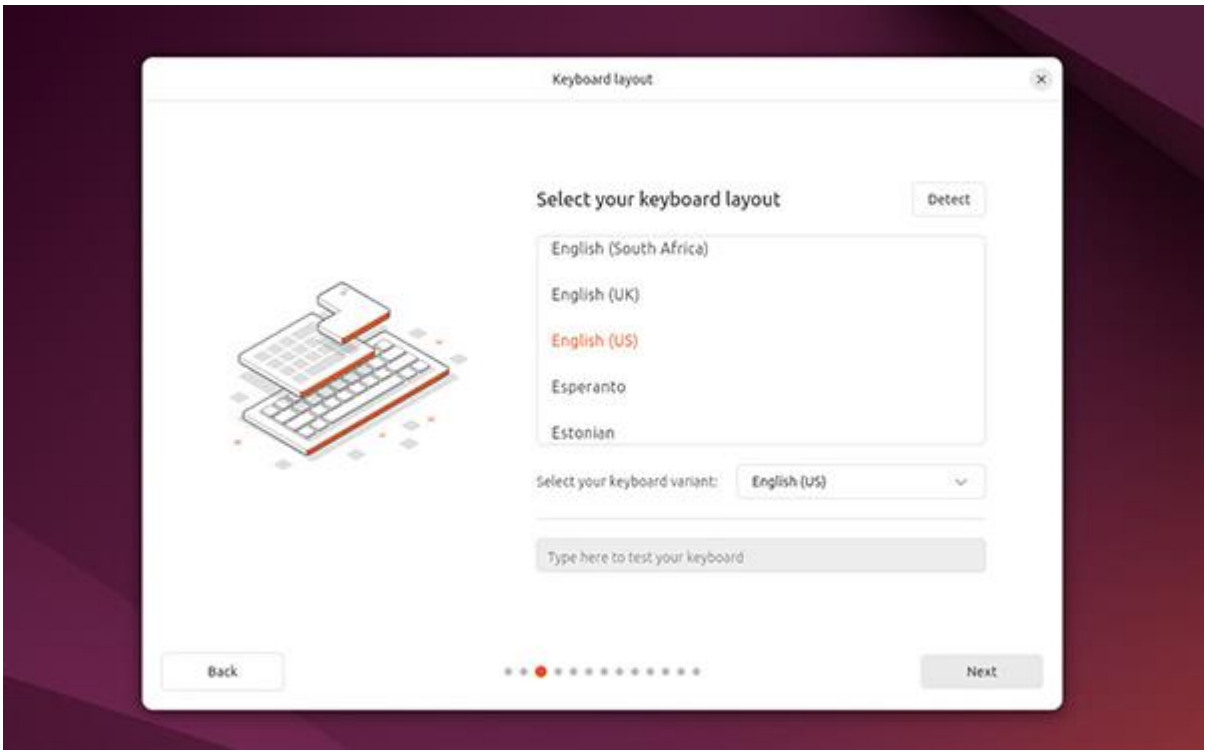
4. شروع تنظیمات نصب

هنگامی که نصب کننده راه اندازی شد، باید زبان خود را انتخاب کنید.

پس از انتخاب گزینه مورد نظر، بر روی next کلیک کنید تا تنظیمات دسترس پذیری را به شما نمایش دهد.

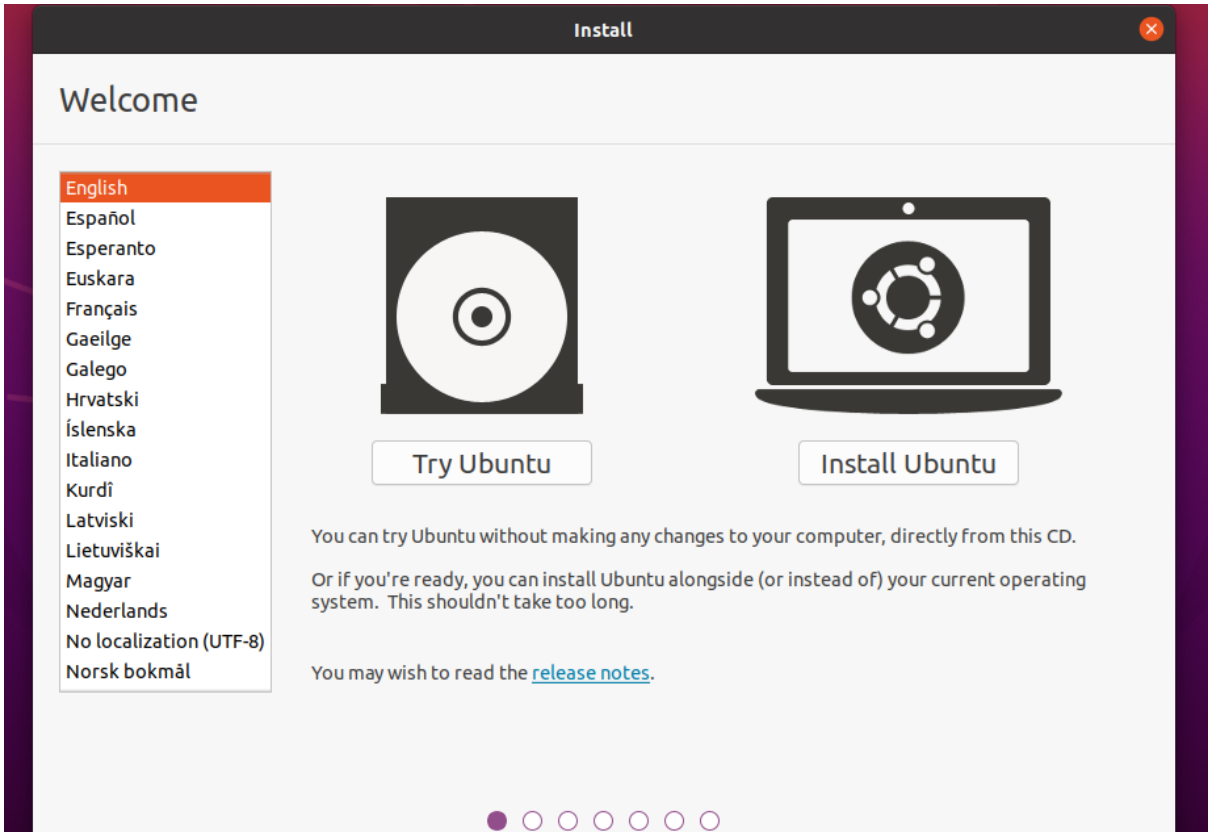


چیدمان صفحه کلید را انتخاب کنید.



در این بخش از تنظیمات می‌توانید با اتصال به شبکه، به اوبونتو این اجازه را بدهید که به‌روزرسانی‌ها و درایورهایی مانند درایورهای گرافیک NVIDIA را در حین نصب دانلود کند.

در این بخش به شما پیشنهاد می شود که لینوکس اوبونتو را امتحان یا نصب کنید.



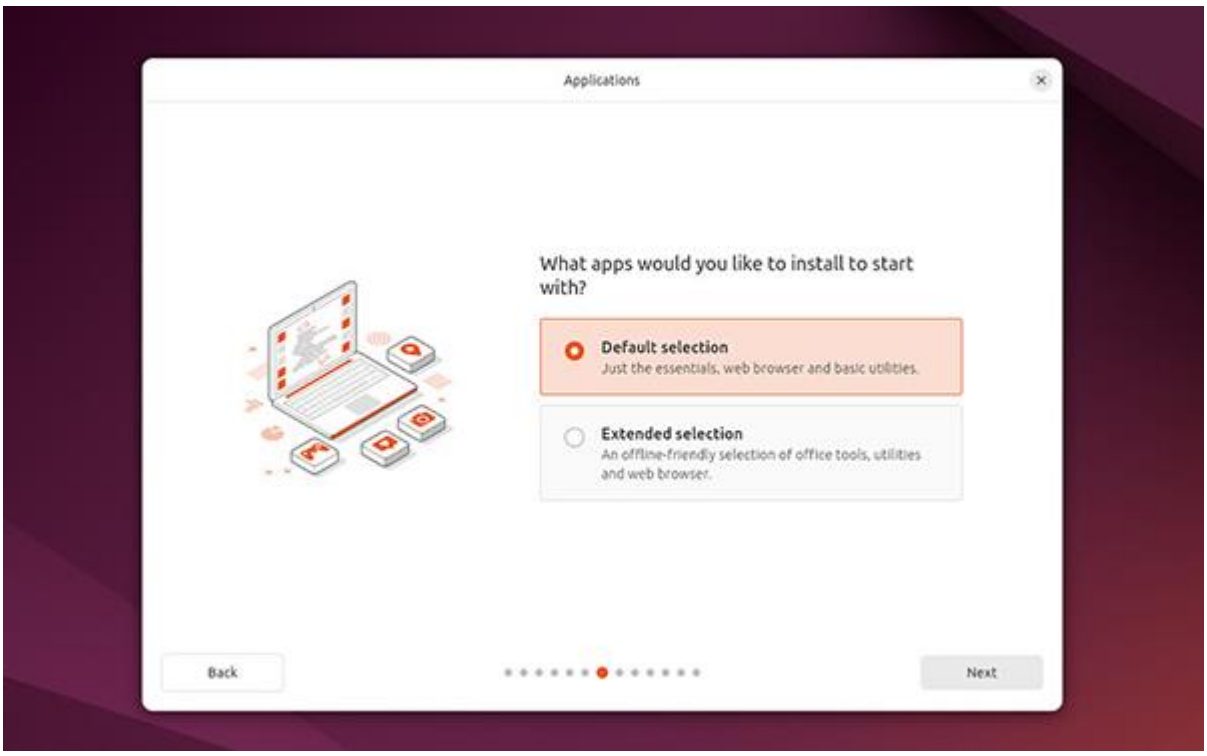
برای ادامه، روی Install Ubuntu کلیک کنید. اگر روی Try Ubuntu کلیک کنید، می‌توانید پیش‌نمایش (Live) اوبونتو را بدون ایجاد هیچ تغییری در سیستم خود مشاهده کنید.

توجه داشته باشید که برخی از کامپیوترها از Intel RST(فناوری ذخیره سازی سریع) استفاده می کنند که توسط اوبونتو پشتیبانی نمی شود. اگر سیستم شما نیز از این فناوری استفاده می‌کند، نمی‌توانید بدون غیر فعال کردن RST در منوی بایوس سیستم خود، از این مرحله جلوتر بروید.

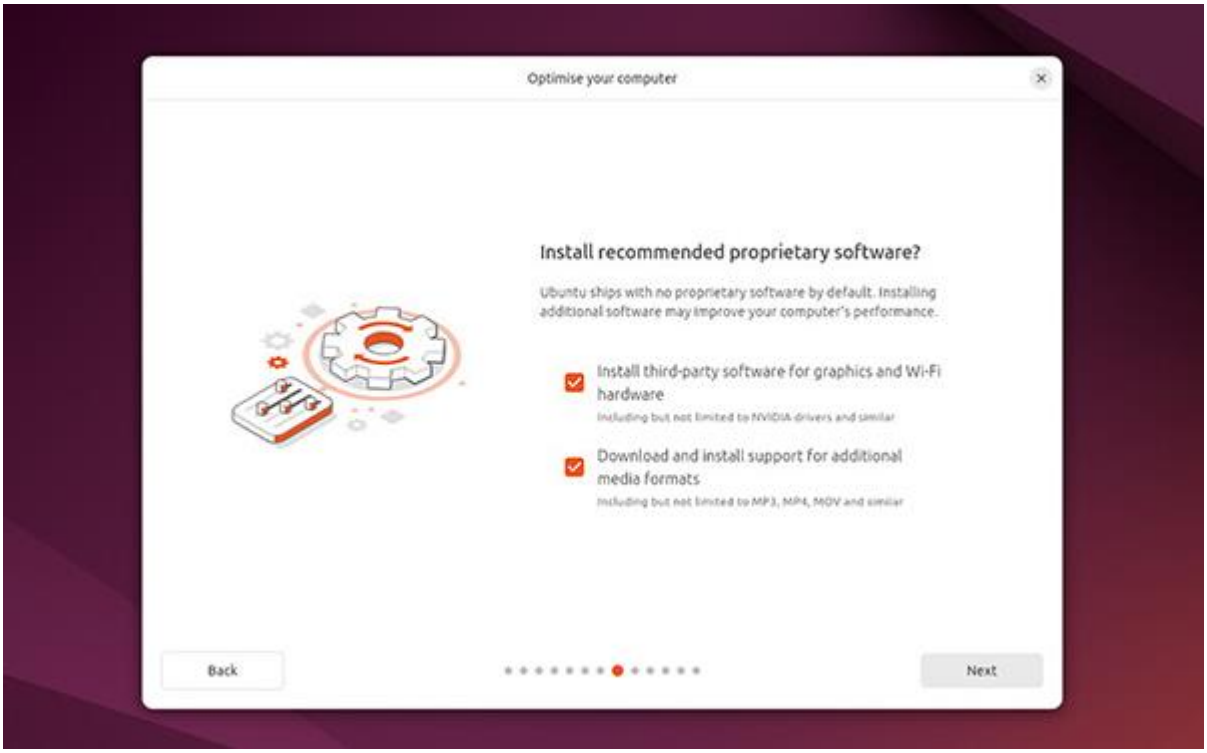
5. شروع نصب لینوکس اوبونتو

از شما خواسته می شود بین نصب تعاملی (Interactive installation) و نصب خودکار (Automated Installation) یکی را انتخاب کنید. گزینه تعاملی مسیر استاندارد تری است، اما گزینه نصب خودکار برای وارد کردن یک فایل پیکربندی از یک وب سرور برای استانداردسازی نصب‌های متعدد و سفارشی سازی های بیشتر استفاده می‌شود.

با مسیر نصب تعاملی ادامه می‌دهیم. در مرحله بعد از شما خواسته می شود که برای نصب نرم افزارها بین گزینه های مجموعه پیش فرض (Default selection) و مجموعه گسترده (Extended selection) یکی را انتخاب کنید. بهتر است گزینه پیش‌فرض را برای شروع انتخاب کنید. می‌توانید پس از نصب با استفاده از App Center موارد بیشتری را نیز بر روی سیستم خود نصب کنید.



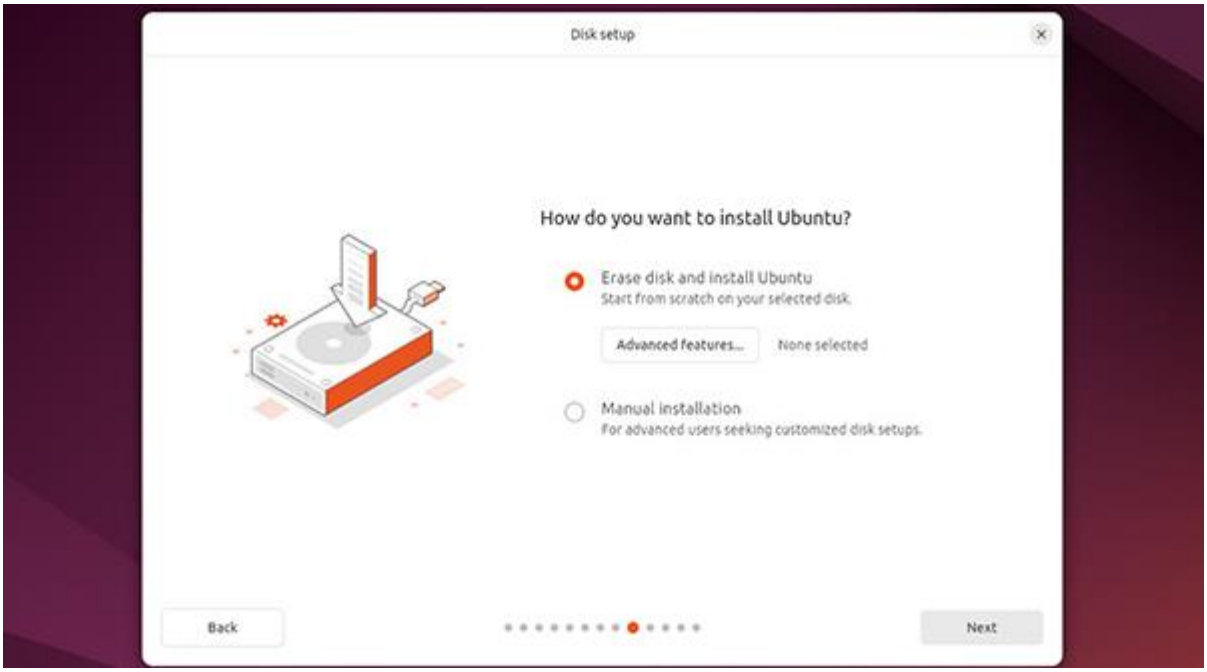
در مرحله بعد از شما خواسته می شود که نرم افزارهای third party را نصب کنید که می‌توانند پشتیبانی و عملکرد دستگاه را بهبود بخشند (به عنوان مثال، درایورهای گرافیکNvidia) و نرم‌افزارهای پشتیبانی از فرمت های رسانه. توصیه می شود هر دوی این کادرها را علامت بزنید.



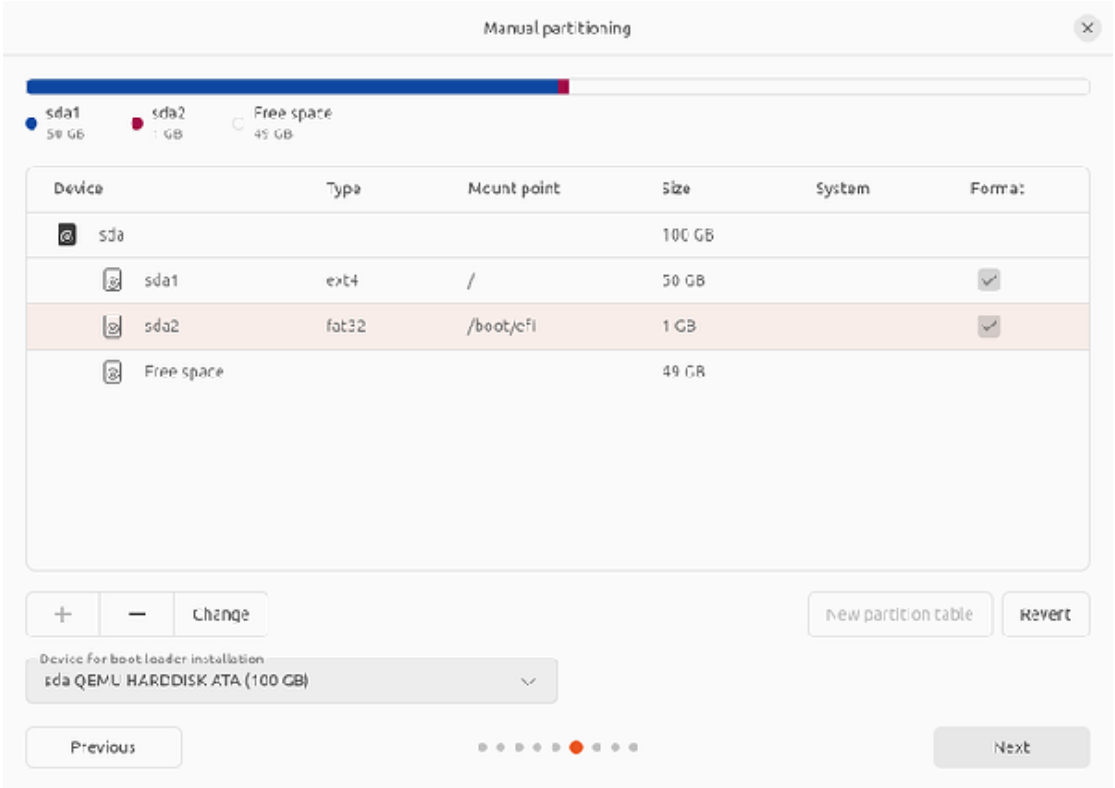
6. انتخاب محل نصب

در این بخش به دقت بیشتری نیاز دارید تا به اشتباه فایل‌ها یا سیستم عامل فعلی خود را پاک نکنید. در صورتیکه محل نصب ابونتو را مطابق روشی که در ابتدای دستورکار آمده، آماده کرده‌اید، گزینه Manual Installation را انتخاب کرده و ادامه بدهید تا اوبونتو در کنار ویندوز، روی سیستم شما نصب شود.

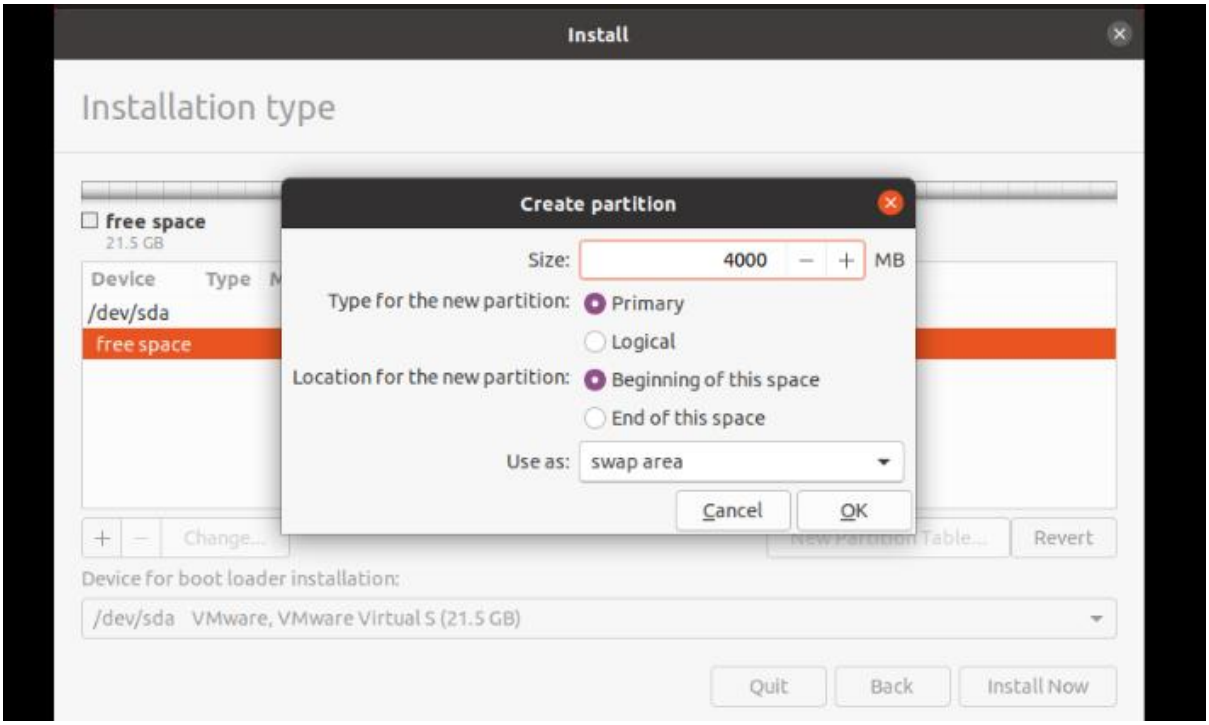
هشدار: در صورتیکه گزینه Erase Disk را بزنید کل هارد پاک خواهد شد. مگر اینکه سیستم دارای چند هارد درایو مجزا باشد که در این صورت نیز باید توجه کنید که درایو صحیح و موردنظر را انتخاب کرده باشید. همچنین در صورتیکه در حال نصب بر روی ماشین مجازی هستید این گزینه قابل انتخاب است.

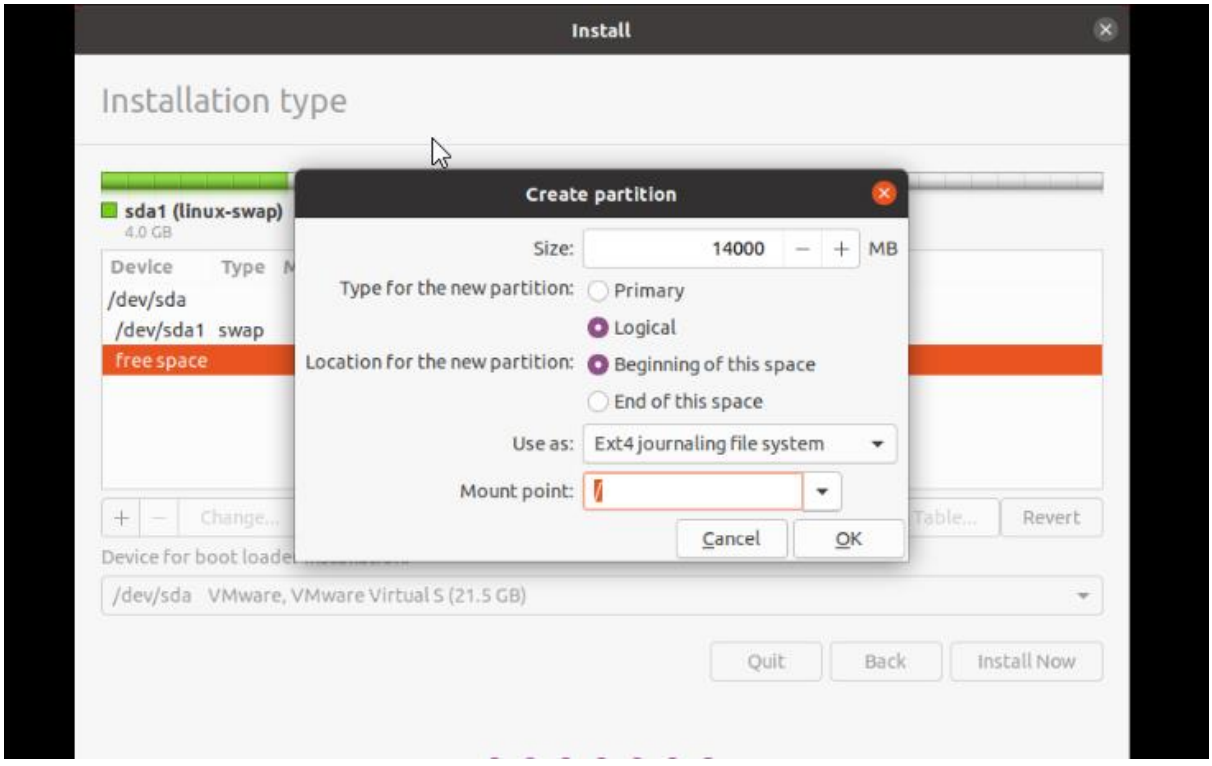


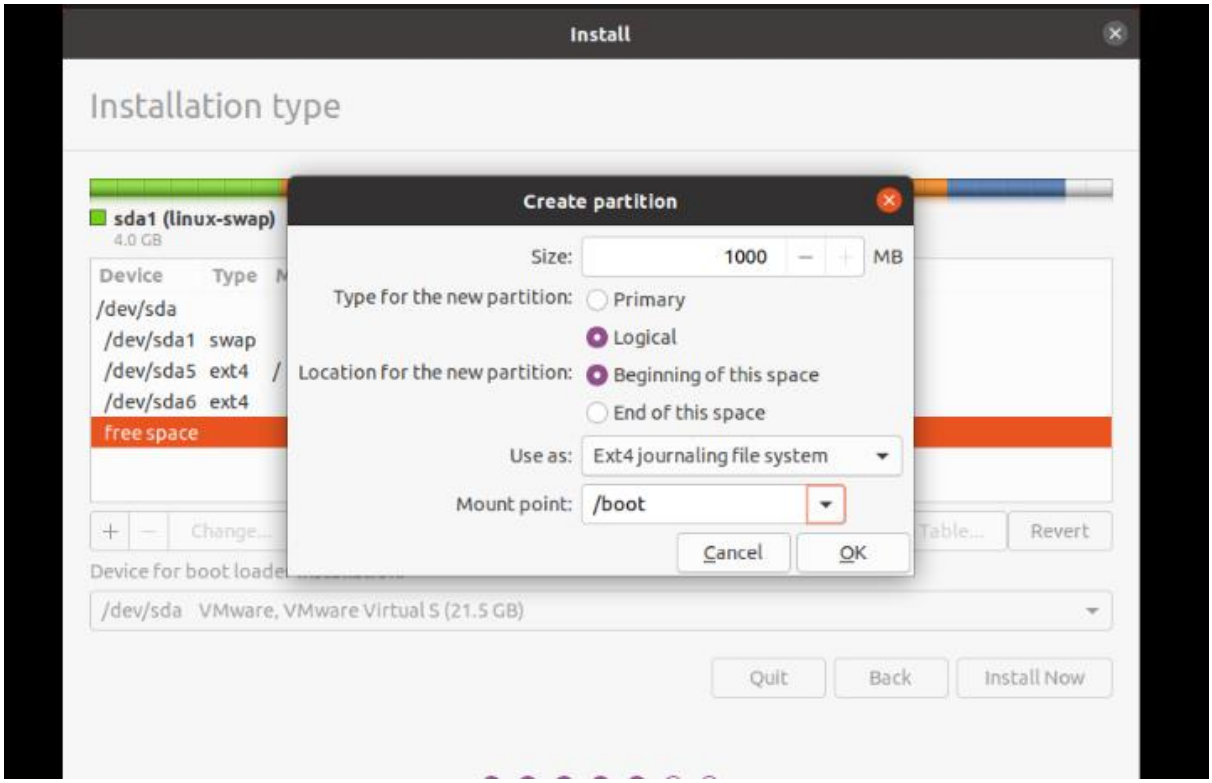
حالا به قسمت **Free space** رفته و بر روی بخش + کلیک می کنیم تا حافظه مورد نیاز به بخش های انتخاب شده اختصاص یابد؛ لازم است پارتیشن های **root**، **home**، **swap** و **boot** را ایجاد کنیم:

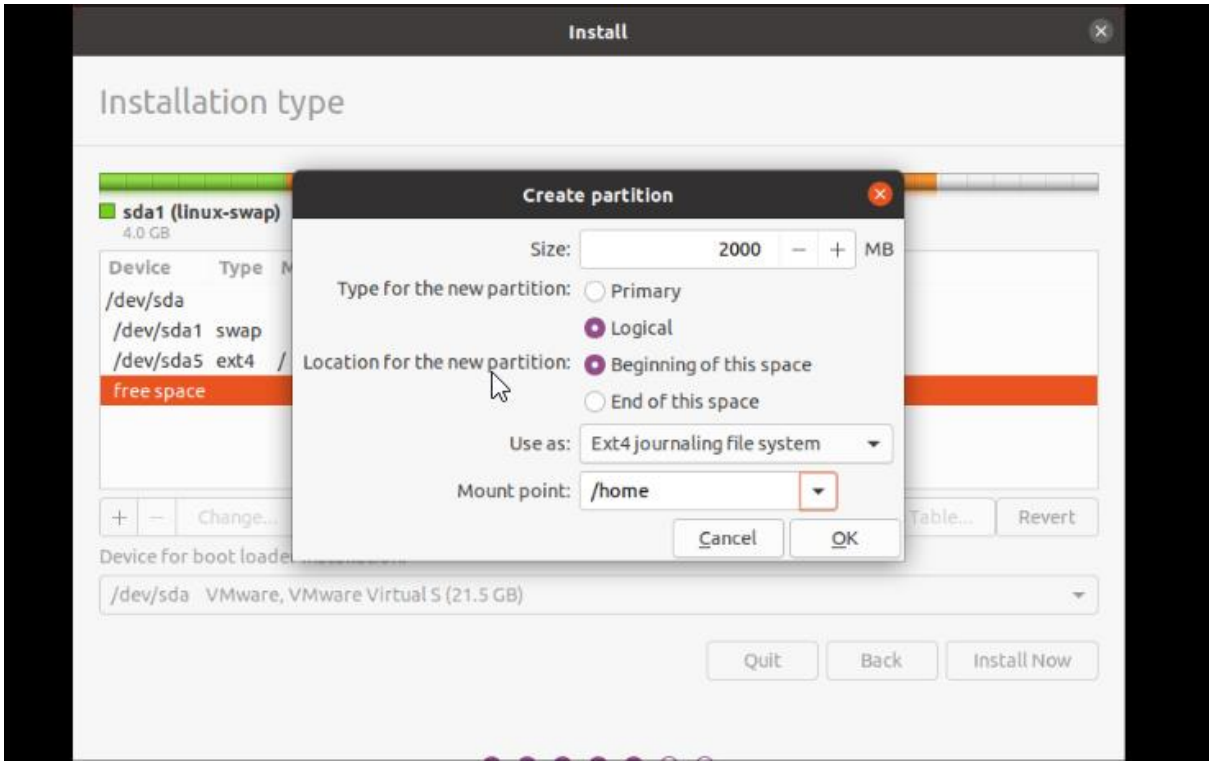


در این مرحله هر جا که یکی از مراحل را اشتباه انجام دادید، روی گزینه Revert کلیک کنید تا تنظیمات به حالت اولیه بازگردند.هر بار کلید + را زده و مطابق تصاویر زیر ادامه دهید:





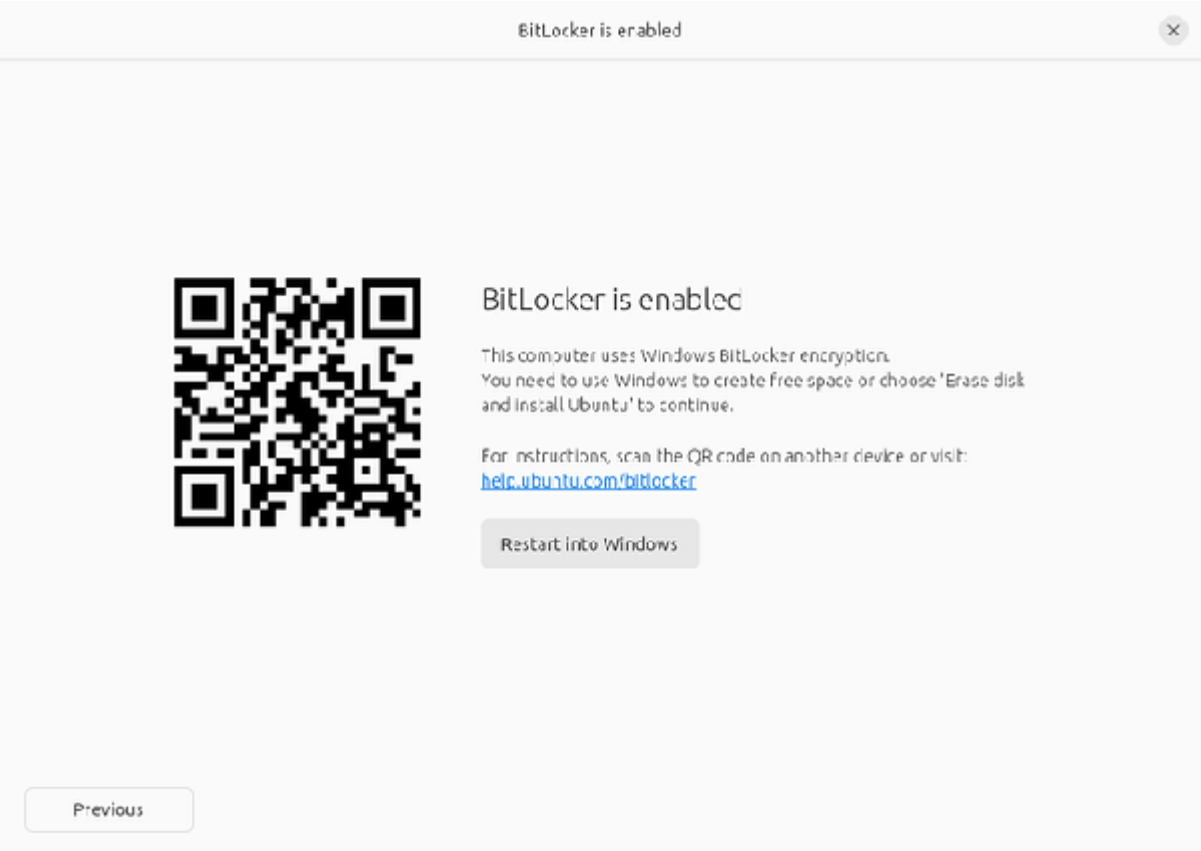

Install
Installation type
sda1 (linux-swap)
4.0 GB
Device Type
/dev/sda
/dev/sda1 swap
/dev/sda5 ext4 /
/dev/sda6 ext4
free space
Create partition
Size: 1000 MB
Type for the new partition: Primary
Logical
Location for the new partition: Beginning of this space
End of this space
Use as: Ext4 journaling file system
Mount point: /boot
Cancel
OK
Revert
/dev/sda VMware, VMware Virtual S (21.5 GB)
Quit
Back
Install Now


Install
Installation type
sda1 (linux-swap)
4.0 GB
Device Type
/dev/sda
/dev/sda1 swap
/dev/sda5 ext4 /
free space
Create partition
Size: 2000 MB
Type for the new partition: Primary
Logical
Location for the new partition: Beginning of this space
End of this space
Use as: Ext4 journaling file system
Mount point: /home
Cancel
OK
Revert
/dev/sda VMware, VMware Virtual S (21.5 GB)
Quit
Back
Install Now

پس از ساخت پارتیشن ها، بر روی گزینه Install now کلیک کنید و در پنجره باز شده گزینه Continue را انتخاب کنید. بعد از انتخاب گزینه continue امکان بازگشت و تغییر وجود ندارد.

دریافت هشدار فعال بودن Windows BitLocker

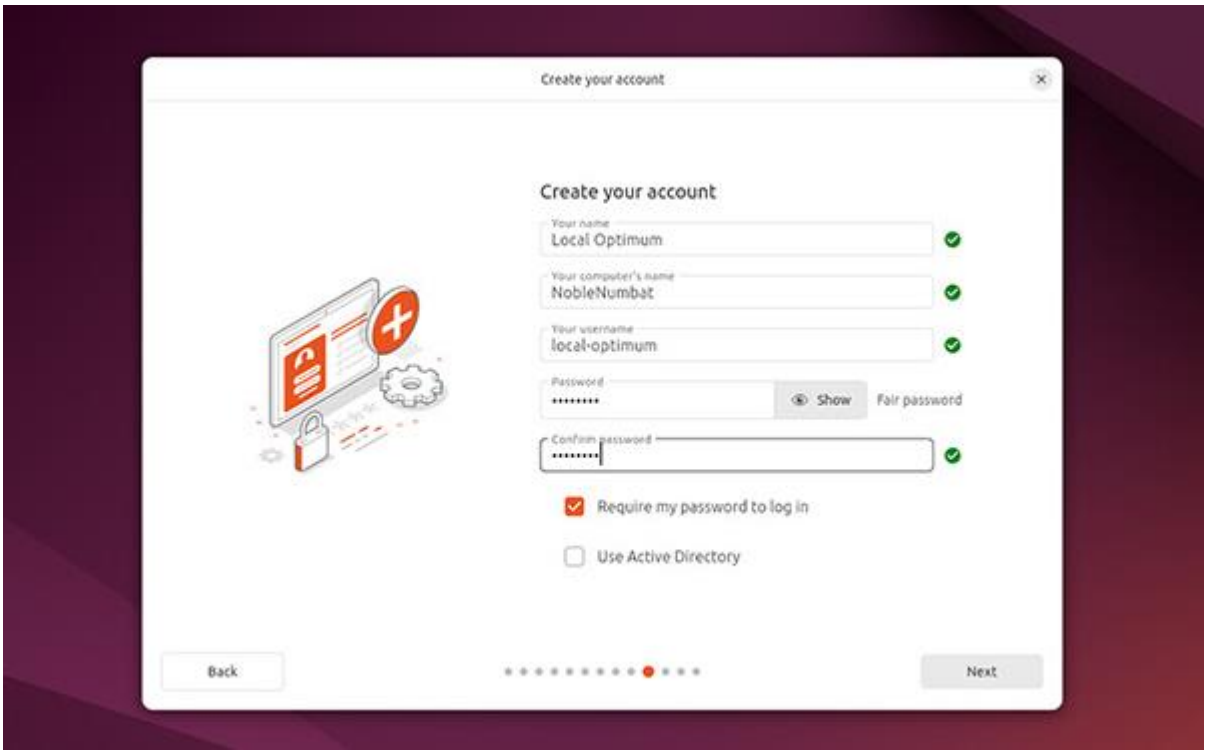
اگر سیستم عامل فعلی دستگاه شما ویندوز است، سیستم شما به احتمال زیاد Windows BitLocker Drive Encryption را فعال کرده است. در این صورت اوبونتو نمی‌تواند اطلاعات درایو مورد نیاز برای نصب امن لینوکس اوبونتو را در کنار ویندوز جمع‌آوری کند. بنابراین قبل از راه اندازی مجدد نصب کننده اوبونتو، یک درخواست برای غیرفعال کردن BitLocker در ویندوز دریافت خواهید کرد.



نکته: در صورت پاک کردن کامل ویندوز یا استفاده از یک درایو جداگانه و رمزگذاری نشده برای نصب اوبونتو، غیرفعال کردن BitLocker ویندوز لازم نیست.

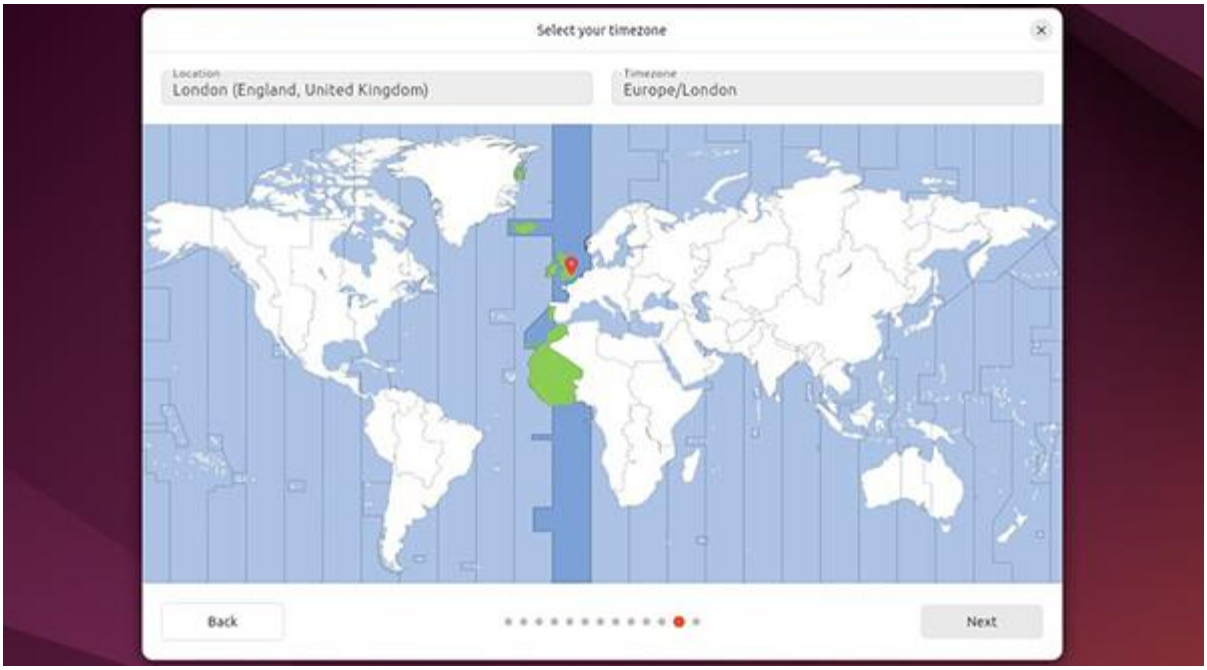
7. جزئیات Login خود را ایجاد کنید

در این بخش از شما خواسته می شود نام خود و نام رایانه خود را وارد کنید. در نهایت یک نام کاربری و یک رمز عبور قدرتمند ایجاد کنید. همچنین می توانید انتخاب کنید که از یک رمز عبور استفاده کنید. اگر در حین سفر از دستگاه خود استفاده می کنید، توصیه می شود گزینه "Require my password to log in" را تیک بزنید.



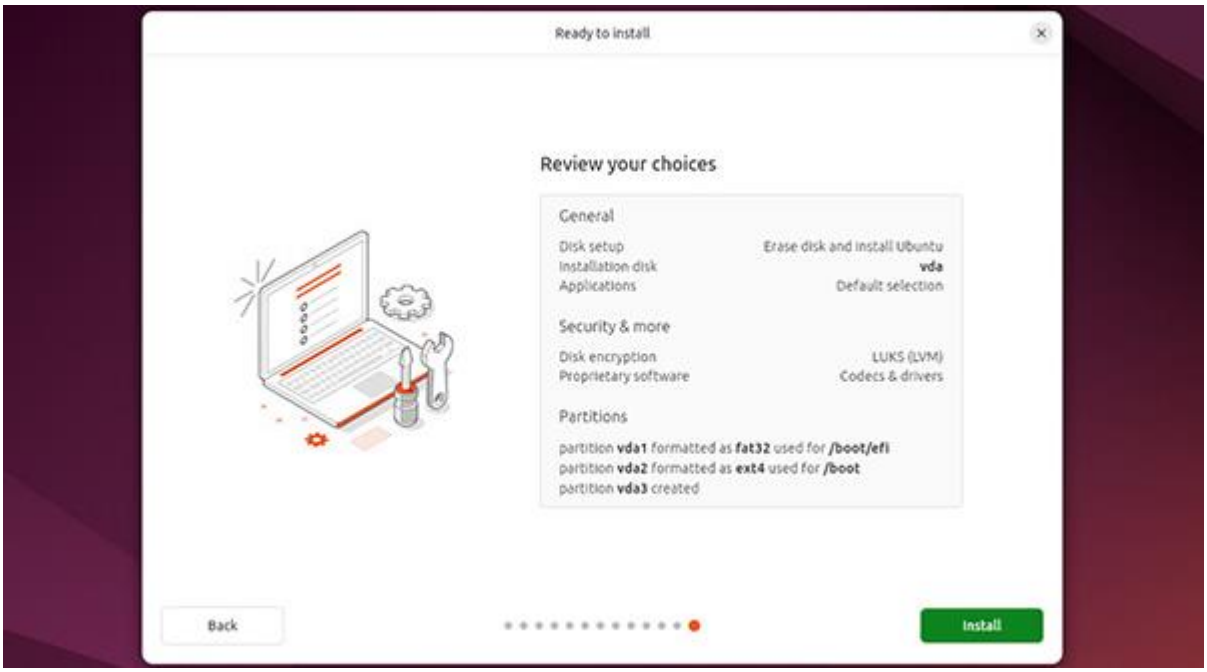
8. تعیین لوکیشن

لوکیشن و منطقه زمانی خود را بر روی نقشه مشخص کنید و روی Continue کلیک کنید. اگر به اینترنت متصل باشید، این اطلاعات به طور خودکار شناسایی می شوند.



9. نصب نهایی

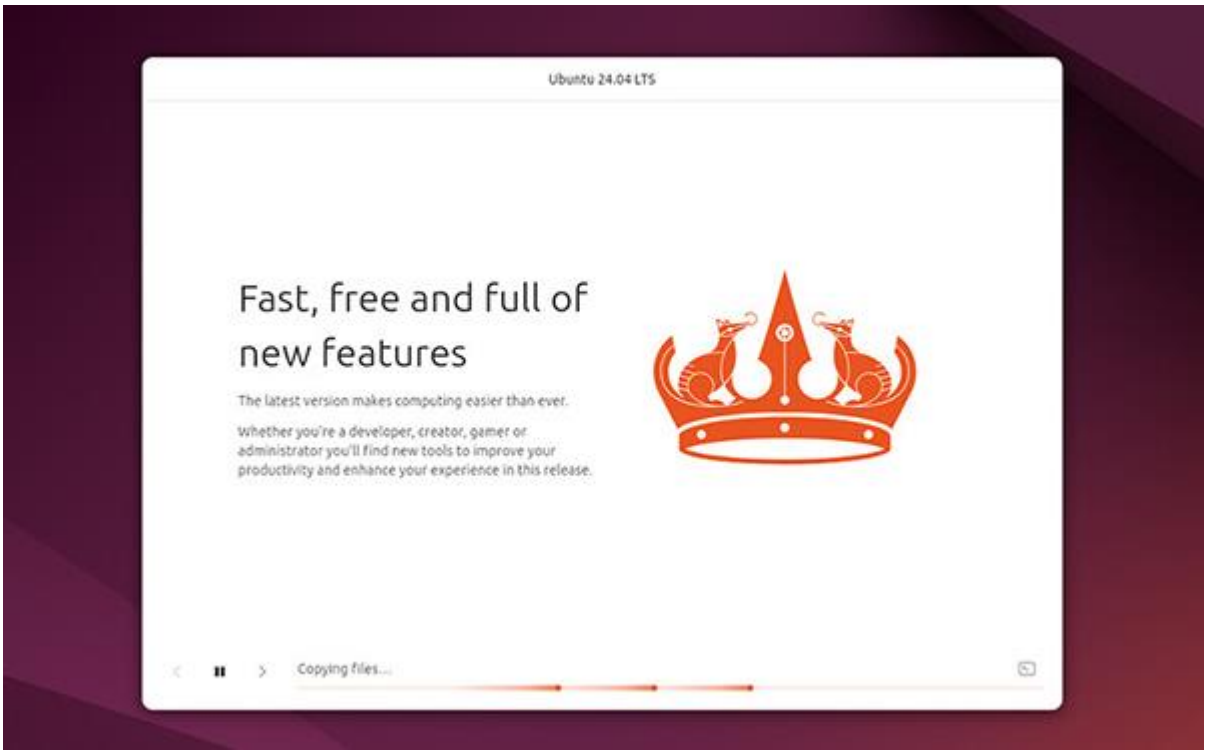
با کلیک بر رویNext ، خلاصه‌ای از پیکربندی نصب را به شما نمایش می‌دهد تا قبل از کلیک کردن بر رویInstall ، تنظیمات خود را بررسی و تأیید کنید.



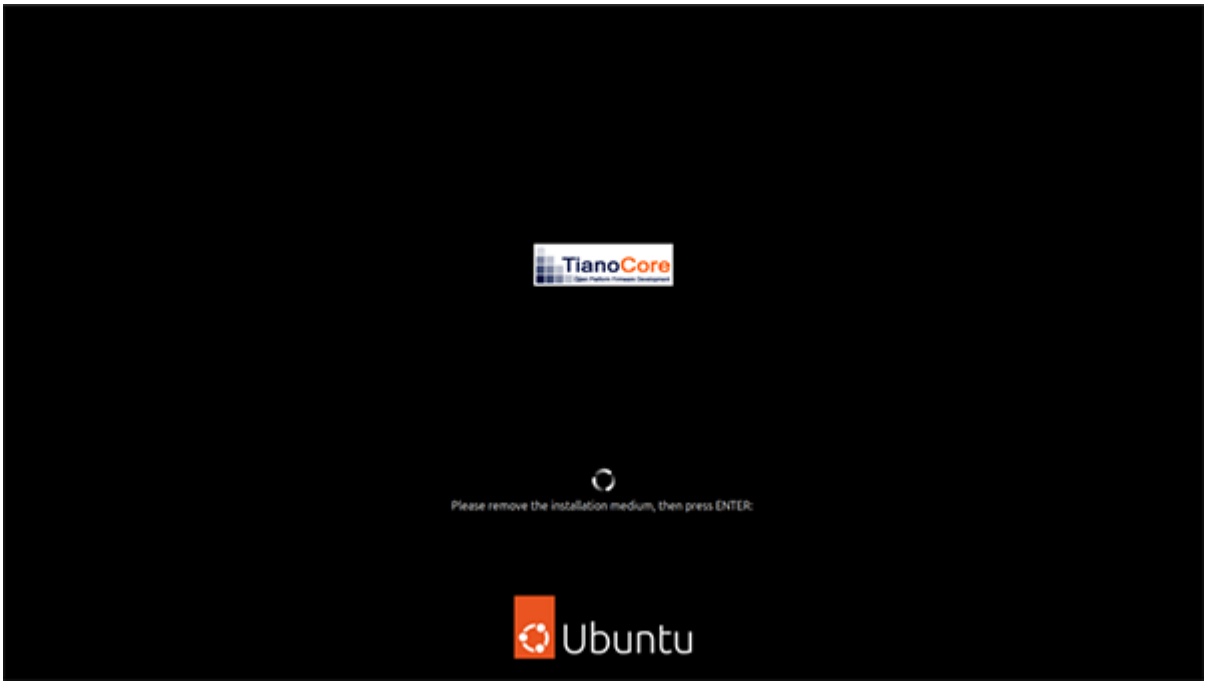
اگر در ابتدای فرآیند نصب، از راه پیکربندی نصب خودکار مراحل را انجام داده‌اید، بلافاصله به این صفحه هدایت می‌شوید تا تأیید کنید که پیکربندی شما به درستی انجام شده است. پس از کلیک بر رویinstall ، لینوکس اوبونتو فرآیند نصب را آغاز خواهد کرد.

10. تکمیل فرآیند نصب

فرآیند نهایی نصب ممکن است مقداری زمان‌بر باشد.

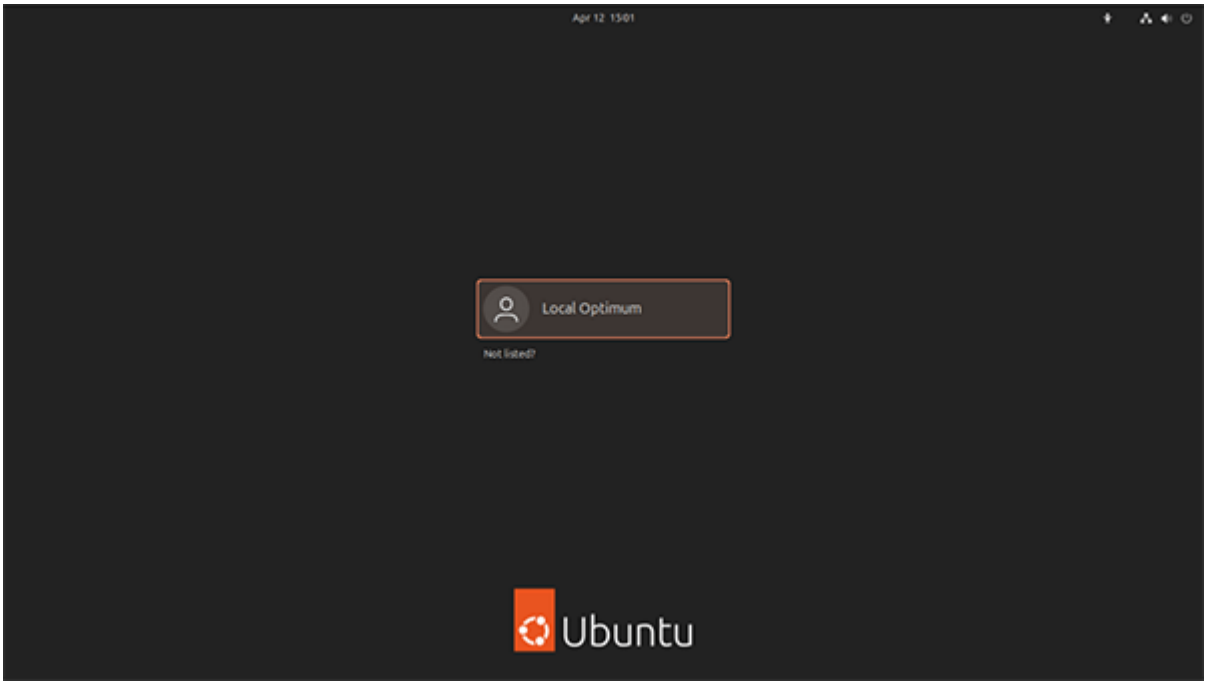


پس از راه اندازی مجدد، از شما خواسته می شود که فلش USB خود را از دستگاه جدا کنید. پس از انجام این کار، ENTERرا فشار دهید.



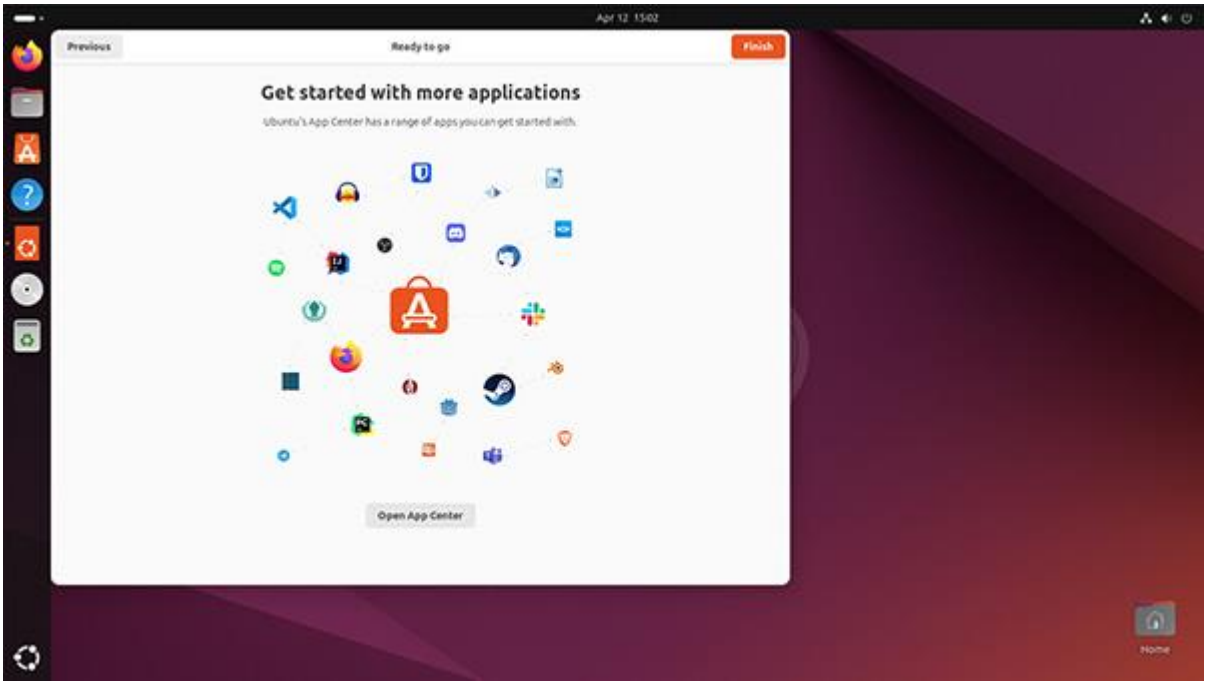
سیستم عامل های نصب شده روی سیستم توسط بوت لودر GRUB برایتان نمایش داده می شوند:

سپس صفحه ورود به سیستم به شما نمایش داده می شود که در آن می توانید نام کاربری و رمز عبور خود را وارد کنید.



و سپس وارد دسکتاپ اوبونتو می‌شوید:

ویجت خوش آمد گویی به شما در برخی از گزینه‌های راه‌اندازی کمک می کند و شما را با محیط آشنا می‌کند. به عنوان مثال، می‌توانید تعیین کنید که تصمیم به ارسال اطلاعات دستگاه به Canonical برای کمک به بهبود اوبونتو را دارید یا خیر. و همچنین دانلود نرم‌افزار های کاربردی بیشتر از Ubuntu Software.



1-5-3-مراحل نصب اوبونتو در ماشین مجازی (VMware)

1. VMware Workstation را نصب کنید

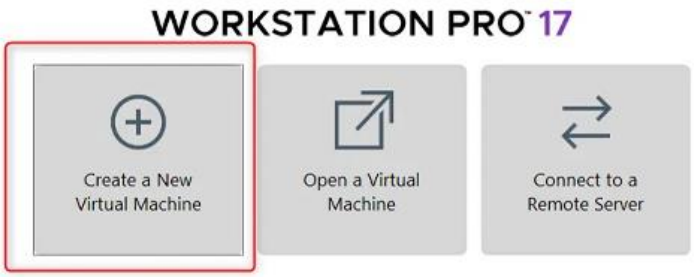
در صورتیکه از قبل این نرم افزار را بر روی سیستم خود ندارید آن را نصب نمایید. ممکن است لازم باشد پس از اتمام نصب، سیستم خود را مجددا راه اندازی کنید.

2. فایل ISO لینوکس اوبونتو را دانلود کنید.

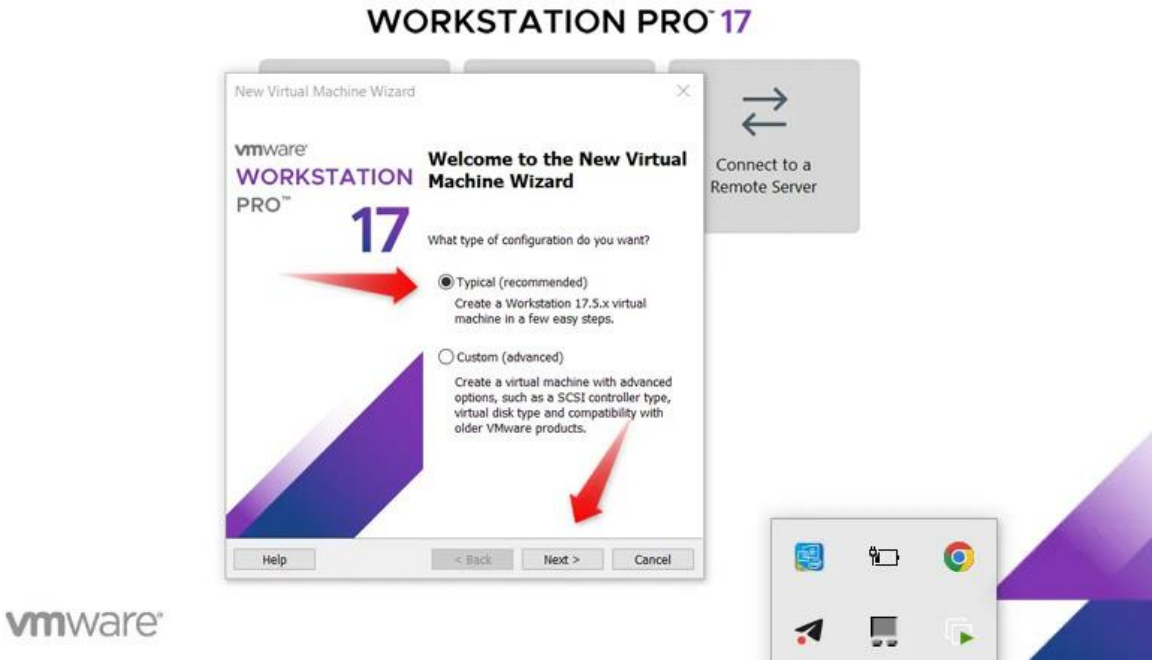
از وب سایت رسمی اوبونتو آخرین نسخه Desktop Ubuntu را دانلود کنید. معماری مناسب (32 بیتی یا 64 بیتی) را بر اساس نوع سیستم خود انتخاب کنید.

3. یک ماشین مجازی جدید در VMware Workstation ایجاد کنید:

برای اینکار ابتدا VMware Workstationرا اجرا کنید. سپس روی «File» در منو کلیک کنید و «New Virtual Machine»را انتخاب کنید.



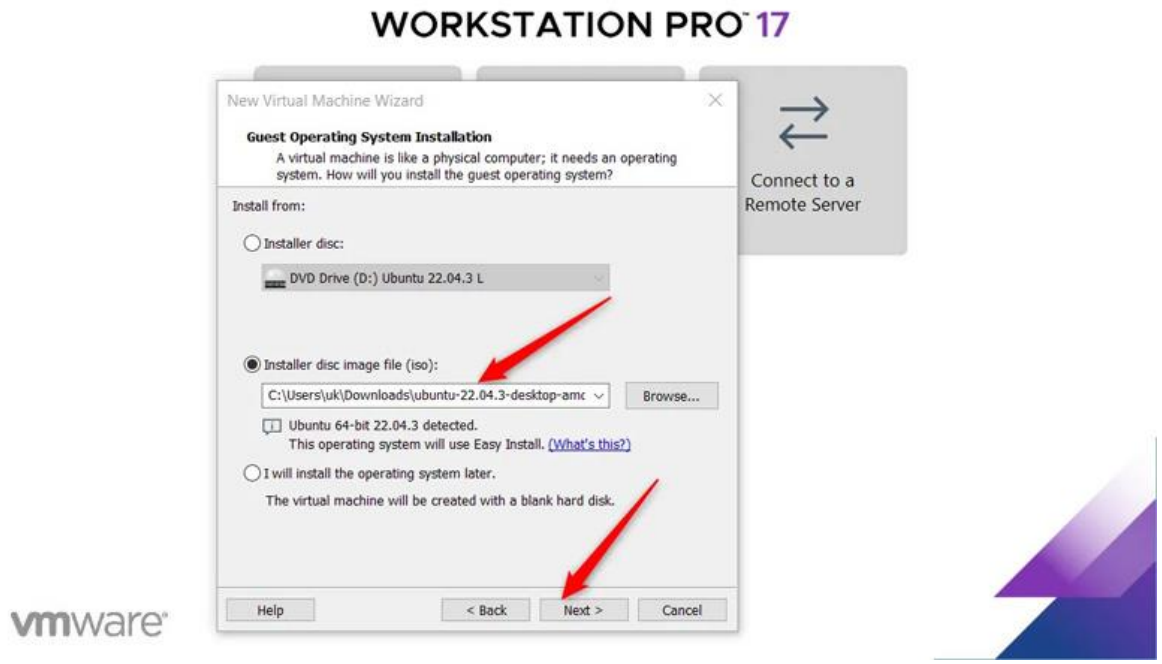
Wizard ایجاد ماشین مجازی جدید باز خواهد شد. گزینه **معمولی** را انتخاب کنید و روی «next» کلیک کنید.



4. فایل ISO اوبونتو را که دانلود کرده‌اید، برای نصب بر روی ماشین مجازی جدید، انتخاب کنید:

برای این کار ابتدا "Installer disc image file (iso)"را انتخاب کنید و روی "Browse" کلیک کنید. سپس به محلی که فایل ISO اوبونتو را در آن ذخیره کرده اید بروید و آن را انتخاب کنید.

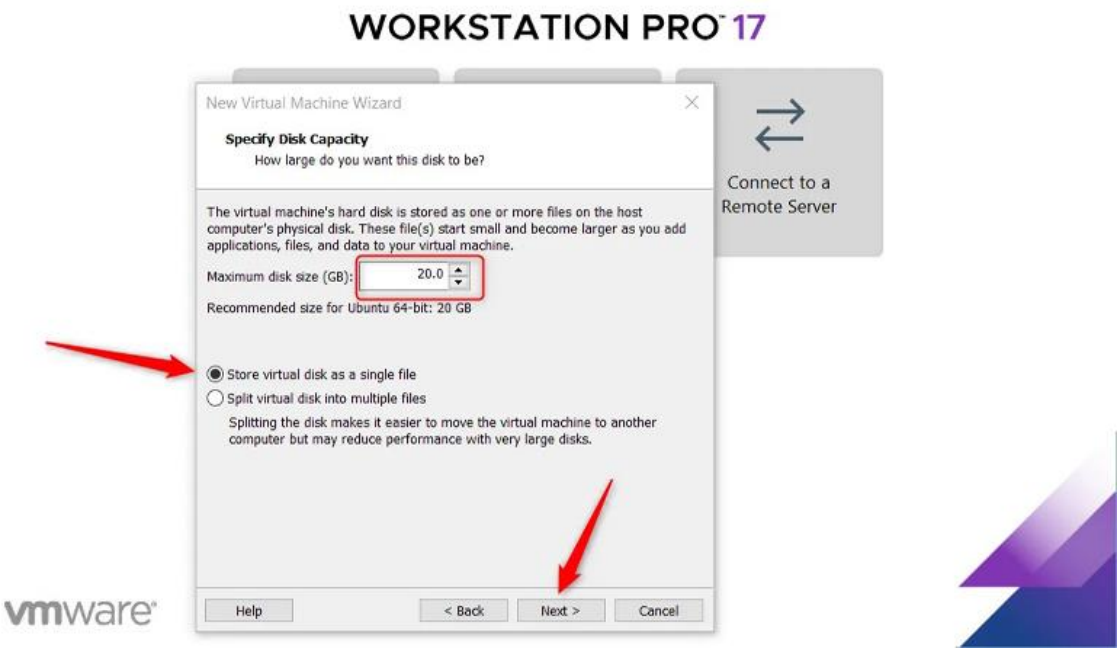
برای ادامه روی «Next» کلیک کنید.



5. انتخاب نام و تعیین فضای ذخیره‌سازی:

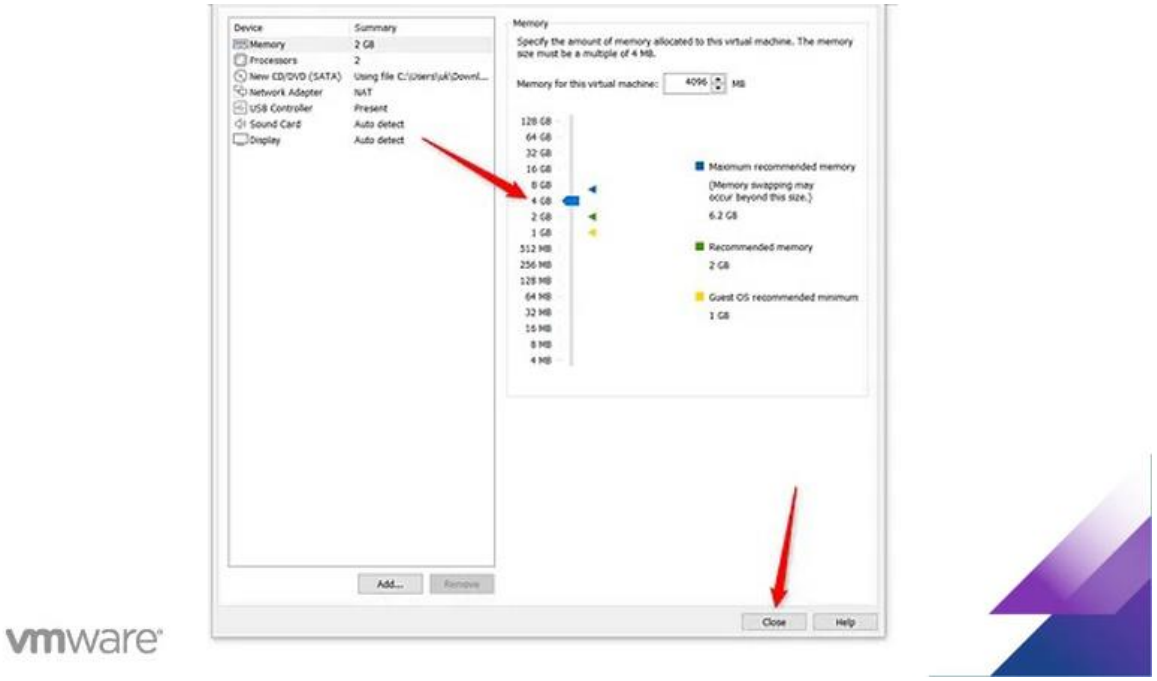
ابتدا یک نام برای ماشین مجازی خود وارد کنید. سپس مکانی را برای ذخیره فایل های ماشین مجازی خود انتخاب کنید. حال ظرفیت دیسک را مشخص کنید (حداقل 25 گیگابایت توصیه می شود). گزینه "Store virtual disk as a single file"را انتخاب کنید.

برای ادامه روی "Next" کلیک کنید.



6. سفارشی کردن سخت افزار (اختیاری)

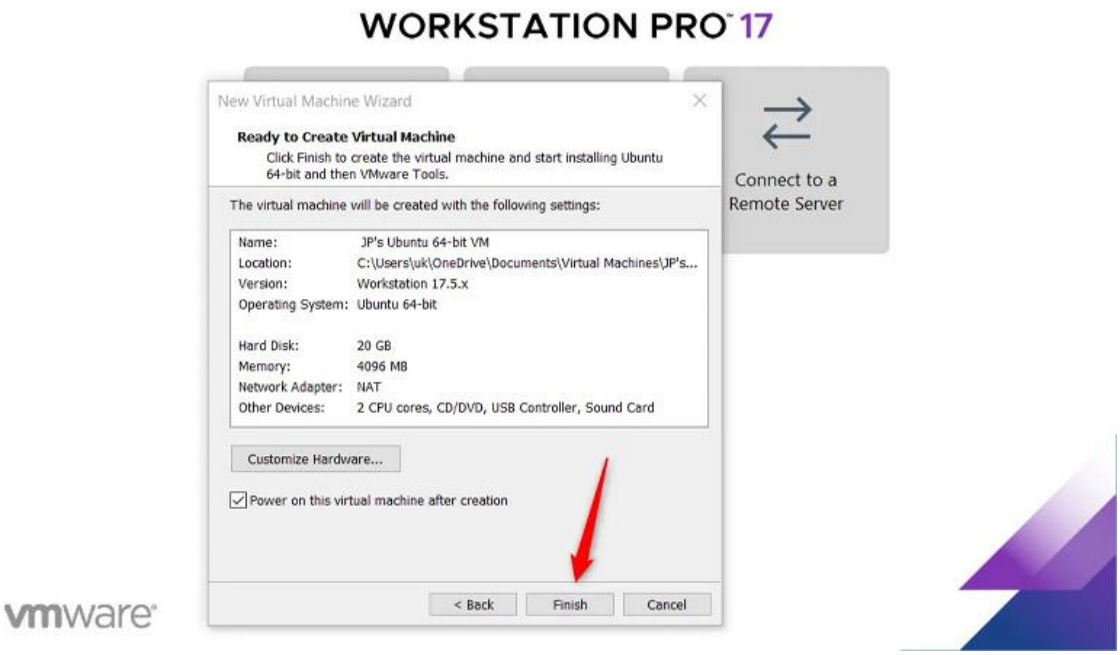
در صورت نیاز می‌توانید تنظیمات سخت افزاری ماشین مجازی را سفارشی کنید. تعداد پردازنده ها، حافظه و سایر تنظیمات را بر اساس قابلیت‌های سیستم خود تنظیم کنید.



7. پایان تنظیمات و شروع نصب اوبونتو

قبل از راه‌اندازی ماشین مجازی، ممکن است بخواهید به برگه "Options" بروید و در صورت لزوم تنظیمات اضافی را نیز پیکربندی کنید.

برای ایجاد ماشین مجازی روی "finish" کلیک کنید.

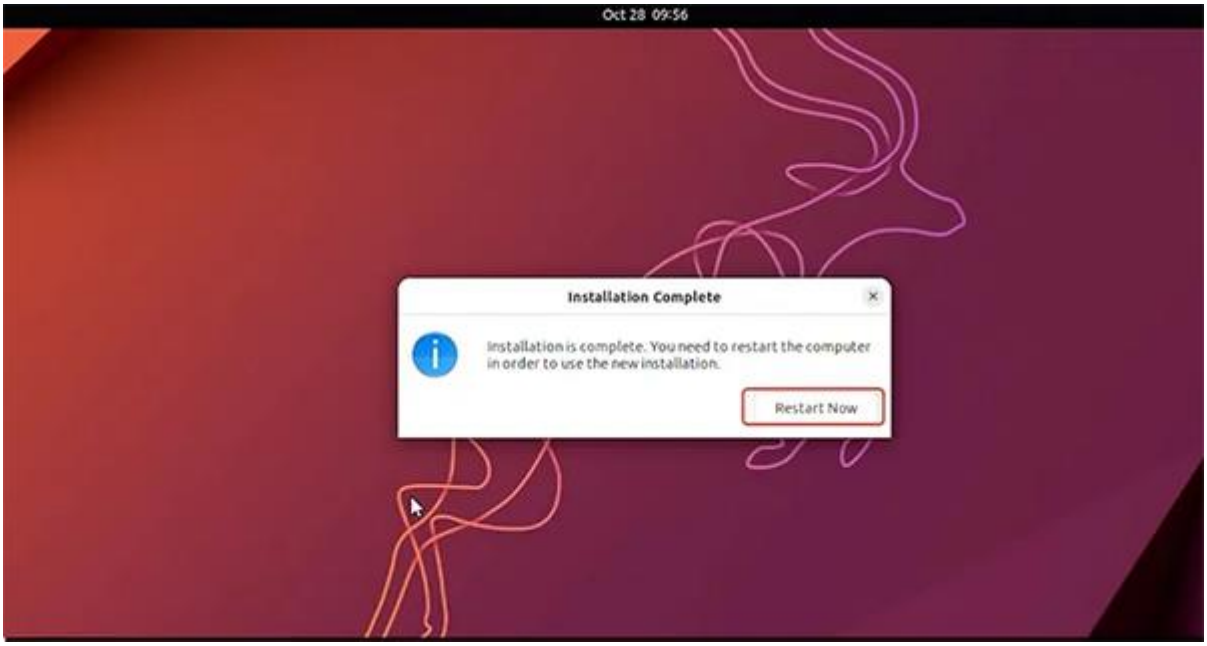


8. ماشین مجازی را استارت کنید

ماشین مجازی از ISO اوبونتو بوت می‌شود. دستورالعمل های روی صفحه را برای نصب اوبونتو دنبال کنید.(در صورتیکه در این بخش نیاز به راهنمایی دارید، به بخش 1-5-2 مراجعه و راهنمای نصب را از مرحله 4 به بعد دنبال کنید.

9. نصب اوبونتو را کامل کنید

دستورات نصب اوبونتو، از جمله زبان، نوع صفحه کلید و پارتیشن بندی دیسک را دنبال کنید. پس از اتمام نصب، ماشین مجازی را مجددا راه اندازی کنید.



10.ابزار VMware را نصب کنید (اختیاری اما توصیه می‌شود)

پس از نصب اوبونتو، برای ادغام و عملکرد بهتر، توصیه می‌شود VMware Tools را بر روی اوبونتو نصب کنید. در منویVMware ، به "VM" >> "Install VMware Tools" بروید. دستورالعمل های روی صفحه را در ماشین مجازی اوبونتو برای تکمیل نصب دنبال کنید.

گزارش کار:

- درباره نسخه WSL اوبونتو تحقیق کرده و آن را بر روی سیستم خود نصب نمایید.

منابع و مراجع:

- Fox R. Linux with operating system concepts. Chapman and Hall/CRC; 2021 Dec 28.
- Dalheimer MK, Welsh M. Running Linux. O'Reilly; 2002 Dec.
- darsnameh.com/linux1

آزمایش 2

تنظیمات و راهبری در سیستم‌های لینوکسی

اهداف:

- آشنایی با محیط، نحوه کار و تنظیمات در سیستم‌های لینوکسی

ابزار و مفاهیم مورد نیاز:

- سیستم عامل لینوکس
- مفاهیم و آموزش‌های دستورکار اول

محتوا:

- App Centers
- انواع کاربران :
- root, super, sudo, regular, service
- فایل سیستم
- پارتیشن‌ها
- run levels

مفاهیم و شرح دستورکار

2-1- راهبری عمومی در لینوکس

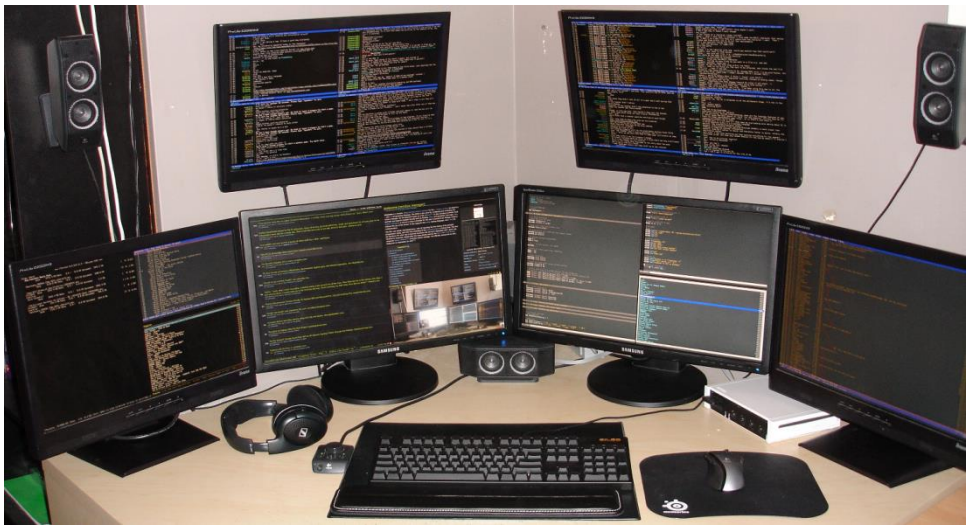
از بسیاری پیش، توزیع‌های لینوکسی مفهوم «منابع نرم افزاری» را به سیستم های خود اضافه کرده اند این منابع (که ریپوزیتوری یا منابع نرم افزاری نامیده می شوند)، فهرست های بزرگی هستند از حجم زیادی برنامه قابل نصب بر روی سیستم شما. مثلا اگر بخواهیم در همین لحظه برنامه چت اسکایپ را روی سیستم نصب کنیم کافی است وارد برنامه مدیریت نرم افزارها شویم، Skypeرا جستجو کنیم و با زدن تیک کنار آن و Apply کردن، به سیستم بگوییم که آن را از روی منابع نرم افزاری گنو/لینوکس مینت نصب کند. چند دقیقه بعد این برنامه دانلود شده و به شکل خودکار نصب خواهد شد.

برای انجام تنظیمات در سیستم به دنبال گزینه System Setting بگردید. این گزینه حاوی بخش بزرگی از تنظیمات عمومی سیستم است. مثلا برای اضافه کردن زبان فارسی به صفحه کلید روی Keyboard Layout کلیک کرده و برگهLayout را انتخاب کنید. علامت + را زده و Persian را انتخاب و Add را فشار بدهید

2-2- قابلیت تغییر و تنظیم

حق انتخاب در دنیای گنو/لینوکس، چیزی بسیار عمیق تر از شخصی سازی رنگ‌ها و اندازه آیکون‌ها است. در دنیای لینوکس می‌توانید تا عمیق ترین سطح سیستم (مثلا تا خود کرنل) را دستکاری کنید. می توانید براساس سخت افزار مورد استفاده سراغ تنظیمات سبک تری بروید و مثلا از یک میز کار خیلی کوچکتر برای یک کامپیوتر بسیار قدیمی که می خواهید در خانه همیشه به اینترنت وصل بماند استفاده کنید یا اگر کامپیوتری به روز دارید، سراغ یک میز کار سه بعدی با همه جذابیت های بصری ممکن بروید. اینجا حق کنار گذاشتن یک چیز و انتخاب چیز دیگر به جای آن را دارید. برای مثال اگر شما بعد از نصب لینوکس اوبونتو با میزکار یونیتی مواجه شدید، به سادگی می‌توانید آن را با میز کارKDE جایگزین کنید که کلا یک برنامه جدید برای مدیریت محیط گرافیکی است.

همچنین میزکارهایی مانند Awesome علاوه بر مدیریت خودکار پنجره ها در مانیتورهای متفاوت اجازه نمی دهد هیچ فضایی روی هیچ نمایشگری حرام شود یا پنجره ای با رفتن زیر پنجره دیگر، غیرقابل دید باشد:



تصویر 2-1- مدیریت میزکار در چندین مانیتور

اما این تنوع از کجا می آید؟

در یک نرم افزار تجاری، قبل از ارائه محصول چند نفر تصمیم می گیرند که برنامه دقیقا قرار است چه کاری انجام دهد و بعد تیمی از برنامه نویس ها تمام تلاش خود را می کنند تا دقیقا مشابه آن را پیاده کنند. اگر بعدها ایده های جدیدی مطرح شوند، شرکت تلاش خواهد کرد تا با اعمال آن‌ها در یک نسخه جدید و فروختن آن به مشتریان پول بیشتری در بیاورد و آنان را نیز راضی نگه دارد. این شرکت تجاری بنا به دیدگاه خود ممکن است سعی کند امکان تنظیم بعضی چیزها را به کاربر بدهد (مثلا ویندوز ممکن است تلاش کند تا با استفاده از تم ها، ظاهر سیستم را قابل تغییر کند) یا ممکن است با این ایده مخالف باشد و بگوید که کاربر نهایی نباید اجازه داشته باشد چیزی را بنا به سلیقه خودش تغییر بدهد چون چیزی که ما در اختیارش گذاشته ایم، بهترین چیز ممکن است (برخورد شرکت اپل که در سیستم عامل حداقل امکان تنظیمات غیر ضروری را به کاربر می دهد.)

در این دیدگاه تجاری، ما با یک طراح طرف هستیم که پیش از ساخت هر چیزی تلاش می کند کل جنبه های آن را پیش بینی کند و محصولی بسازد که با حداقل تلاش به بیشترین درآمد برسد. برای چنین تولید کننده ای منطقی نیست که سعی کند با دو محصول متفاوت به یک نیاز پاسخ بدهد یا سراغ محصولی برای یک بازار مثلا پنج هزار نفری

برود؛ اگر هم اینکار را بکند مجبور است قیمت را آنقدر بالا ببرد که تلاشش از نظر تجاری برای سرمایه گذاران قابل دفاع شود.

در دنیای آزاد، فلسفه از پایه متفاوت است. این دنیا با این ایده شروع شد که «تولید کننده، مصرف کننده است». کسی برنامه را می‌نویسد که خودش می خواهد از آن استفاده کند و در نتیجه سعی می کند بهترین حالت ممکن را ایجاد کند و دائما آن را براساس نیازهای روزمره اش بهینه کند و تغییر دهد. مشخص است که این استراتژی یک نقطه منفی هم دارد: طراحی اولیه. گاهی در بعضی برنامه ها برنامه نویس از روز اول شروع به طراحی کل جزییات برنامه نکرده و در نتیجه ممکن است تمام نیازهای کاربران آینده از اول پیش بینی نشده باشد. اما شیوه توسعه نرم افزار آزاد یک قدرت عظیم هم دارد: پیروی از طبیعت. در طبیعت گونه های زیستی دائما در حال بهبود هستند. هر گونه جهش هایی می کند و تغییراتی می یابد و در نسل بعد آنهایی که سازگاری بیشتری با محیط داشته اند باقی می مانند تا تبدیل به والدین نسل بعدی شوند. این دقیقا شیوه ای است که نرم افزار آزاد رشد می کند.

در طول میلیون ها سال هر جزء از هر موجود زنده در طی پروسه ای که داروین آن را «بقای اصلح» خوانده بود، در بهترین سطح خود تنظیم می شود. در دنیای آزاد هم یک برنامه نویس با نوشتن یک کد- مثلا برای نمایش یک صفحه وب به شکل گرافیکی- شروع می کند. اگر دیگران آن را مفید تشخیص دهند بنا به نیاز خودشان آن را تغییر می دهند و طبق مجوز GPL ، آن را با دیگران به اشتراک می گذارند. ممکن است همزمان ده نفر روی کد اولیه کار کرده باشند اما وقتی نسل بعدی می خواهد راه آنها را ادامه بدهد از شاخه ای ادامه می دهد که بهترین سازگاری و سریعترین اجرا را داشته و در نتیجه این نسخه بهتر، دوباره توسط برنامه نویس های جدید تکمیل می شود. این دقیقا همان شیوه ای است که در طبیعت دنبال می‌شود: تکامل.

حتی توزیع های گنو/لینوکس هم همینطور کار می کنند: یک گروه یک توزیع می سازند (مثلا دبیان) و گروه های بسیار متنوع از این توزیع، فرزندان جدید تولید می کنند (مثلا اوبونتو و پارسیکس) و آنی که محبوبیت بیشتری پیدا می کند (معمولا به خاطر سازگاری بیشتر و مفید بودن برای گروه های متنوع)، پایه لینوکس های جدید می شود که دوباره من و شما در نقش جامعه با استفاده کردن از یکی و کنار گذاشتن دیگری، در طی پروسه انتخاب اصلح، می گوییم نسل‌های بعدی باید از کدام انشعابات استفاده کنند و کدام توزیع‌ها محکوم به فنا هستند.

دقت کنید که این امر فقط از طریق آزاد بودن کد پیش می آید چرا که افراد، شرکت ها و گروه ها باید بتوانند در یک برنامه تغییرات جدید بدهند و برنامه هایی با قابلیت یا شکل جدید را در جامعه پخش کنند تا ببینند کدام یک توسط کاربران ترجیح داده خواهد شد. شرکت های تجاری هنوز باید به سیستم قدیمی «طراحی از قبل» خود وابسته باشند. مثلا اگر به مرکز نرم افزار اوبونتو نگاه کنید می بینید که در حوالی سال ۲۰۰۹ چند توزیع تغییراتی در سیستم قدیمی نصب بسته ها دادند و مفهوم software center را ساختند و بعد از دیدن علاقه کاربران به آن، توزیع های بزرگ دیگر هم به سراغ آن رفتند در حالی که اپل باید صبر می‌کرد تا در نسخه بعدی این ایده موفق را تکرار کند و مایکروسافت هم در ویندوز ۸، اینکار را به انجام رساند. در صورت شکست هم، در مورد سیستم های آزاد مبتنی بر تکامل، تنها یک شاخه مرده به وجود می آید اما در مورد سیستم های تجاری، طراحی های ضعیف منجر به ضررهای هنگفتی مانند ویندوز ویستا می شوند.

در نهایت همیشه هم لازم نیست که فقط یک سیستم بهتر از همه تشخیص داده شود و بقیه را کنار بزند. در دنیا افراد مختلف با نیازهای مختلفی وجود دارند و وقتی ما یک شرکت تجاری نباشیم که بخواهیم با حداقل محصولات حداکثر سود را کسب کنیم، لازم نیست برای همه سوال ها یک جواب ثابت داشته باشیم. ممکن است یک نفر بخواهد سیستمش حداکثر تنظیم پذیری را داشته باشد و حتی بتواند ایموجی‌های مورد استفاده در همه برنامه‌ها را از یک سیستم مرکزی کنترل کند. چنین فردی باید به سراغ محیط کار KDE برود. در مقابل شاید یکی طرفدار ایده «کمتر، بیشتر است» و «ساده زیباست» باشد و بخواهد در حین کار با کامپیوتر حداقل دکمه‌ها و کلیدهای ممکن را ببیند. برای چنین فردی محیط کار Gnome مناسب تر است. این بحث حتی فراتر از سلیقه است. شاید شما کامپیوتر ضعیفی داشته باشد که نیازمند یک محیط کار قابل کنترل با کیبورد و مصرف حداقل حافظه باشد و به همین دلیل به سراغ نصب محیط های بسیار سبکی مانند LXDE بروید که ضمن داشتن تمام ایده های مدرن و سازگاری با آخرین نرم افزارها، انرژی کمی را برای جلوه های بصری هدر می‌دهد یا حتی تصمیم بگیرید که محیط گرافیکی را کاملا کنار بگذارید و فقط با خط فرمان کار کنید!

2-3- خط فرمان

خط فرمان[1] یا Command Line روشی است برای ارتباط با سیستم عامل. این روش به ساده ترین حالت ممکن کار می کند: متن. در خط فرمان شما می توانید مستقیما دستوراتی را برای کامپیوتر تایپ کنید و بعد از زدن کلیدEnter ، کامپیوتر دستورات شما را اجرا خواهد کرد و در صورت نیاز به شکل متنی، به شما پاسخ خواهد داد. این موضوع در تقابل با رابط گرافیکی قرار دارد که در آن دستورات از طریق ماوس، منوها، کلیک کردن و موارد مشابه داده می شود و جواب ها از طریق تصاویر و نمودارها و غیره به نمایش درمی‌آید.

پس دسترسی به خط فرمان به شکل متنی و با نوشتن و دیدن دستورات و نتایج آنها انجام می شود. برای انجام این کار نیاز به یک برنامه است و این برنامه «شبیه ساز ترمینال» یا همان «Terminal Emulator» نام دارد که به سادگی به آن ترمینال می گوییم.

شاید یک مثال خوب برای درک اهمیت ترمینال، مقایسه آن با بالا زدن کاپوت خودرو و نگاه کردن به موتور باشد. سیستم های مشابه یونیکس (از جمله گنو/لینوکس ها) براساس یک فلسفه بنا شده اند. یک بخش از این فلسفه می گوید «همه چیز یا باید فایل باشد یا پروسس» و بخش دیگری تاکید دارد که «همه تنظیمات باید در فایل های متنی ذخیره شوند». ترکیب این دو یعنی قدرت فوق العاده ترمینال.

فرض کنید شما سعی داشته اید با یک رابط گرافیکی تغییری در شیوه نمایش پنجره هایتان بدهید و مثلا به جای دکمه دایره ای برای بستن پنجره، تمی نصب کنید که این دکمه را مثلثی نمایش می دهد و اینکار- به دلیل نصب برنامه ای خارج از منابع اصلی توزیع که می دانیم دلیلی دارد که وارد منبع اصلی نشده است- باعث به هم ریختن محیط گرافیکی تان شده است. برای این مشکل کسی که ترمینال را نشناسد باید تلاش کند تا با همان رابط گرافیکی مشکل را حل کند اما اگر کسی به ترمینال وارد باشد می‌تواند با زدن یک دستور، فایل مربوط به تنظیمات ظاهر پنجره ها را پاک کند (یا اگر نمی داند اول با یک دستور دیگر آن را پیدا کند) تا ظاهر پنجره ها به حالت پیش فرض برگردند.

[1] دستورات ونحوه کار با خط فرمان در بخش بعدی بطور کامل معرفی می‌شوند

درست مثل کسی که وقتی خودرو به درستی استارت نمی زند، بلد است کاپوت ماشین را بالا بزند و مشکل را در موتور رفع کند.

از طرف دیگر کار کردن در ترمینال کاراتر و سریع تر است. در عمل استفاده از ترمینال نه فقط قدرت و انعطاف پذیری که سرعت کار را هم بالا می‌برد.

فرض کنید بخواهیم یک فایل JPEG را بر روی دسکتاپ به PNG تغییر فرمت دهیم. در حالت گرافیکی نیاز به برنامه‌ای برای ادیت تصاویر داریم. در حالت خط فرمانی کافیست دستور زیر را اجرا کنیم:

```
convert     Desktop/sample.jpg   Desktop/sample.png
```

ترمینال معمولا با کلیدهای میانبر (اغلب F12) قابل دسترسی است. بعلاوه، خاصیت کامل کردن خودکار دستورات (Auto fill) که اغلب با کلید tab انجام می‌شود، سرعت تایپ را بالا می‌برد.

با اینکه برنامه های کنترل گرافیکی کامپیوترها از راه دور پیشرفت زیادی کرده اند اما هنوز اگر لازم باشد یک مدیر سیستم یا برنامه نویس، سروری را از راه دور تعمیر کند یا تغییری در آن بدهد یا آن را آپدیت کند، بهترین گزینه اجرای یک شبیه ساز ترمینال روی کامپیوتر خودش و وصل شدن از طریق محیط متنی به سرور است. در ضمن کامپیوتری که در ترمینال بوت می شود نیازی ندارد بخشی از منابع ارزشمند خودش را برای اجرای محیط گرافیکی هزینه کند.

از طرفی، هر برنامه کامپیوتری ممکن است باگ داشته باشد. بنابراین منطقی به نظر می رسد که سرورها اغلب از محیطهای گرافیکی، که نه تنها حجم زیادی از کدهایشان هیچ ربطی به کار واقعی سرور ندارد، بلکه لانه ای برای باگ های نرم افزاری می‌باشد، دوری می‌کنند.

تا اینجا دیدیم که خط فرمان سریع تر است، در سرورها منابع کمتری را به هدر می دهد، دسترسی ما به هر آنچه در یک سیستم هست و نیست را فراهم می کند و قابلیت کنترل بیشتری به کسی که پشت سیستم گنو/لینوکسی نشسته است می دهد.

2-4-رفع مشکل در لینوکس

تقریبا تمام جنبه‌های یک سیستم گنو/لینوکس خودشان دارای راهنماهای بسیار کاملی هستند که Man Pageیا صفحه راهنما نامیده می شوند. مثلا برای دیدن راهنمای کامل شیوه اجرای Firefox کافی است در یک ترمینال تایپ کنیدman firefox و تمام سوییچ ها و روش های ممکن برای اجرای فایرفاکس برای شما به نمایش در خواهد آمد (اگر اینکار را کردید با کلید q از راهنما خارج می شوید). خواندن این راهنماها در ابتدای کار راحت نیست، ولی فرق یک حرفه ی و یک آماتور دقیقا همین توان خواندن راهنما است.

تمرین کلاسی:

- بر روی سیستم خود از طریق system settings، تنظیمات مرتبط با دیسک را یافته و پارتیشن‌های موجود را بررسی نمایید.
- در منوی گراب، تایمر نمایش لیست را یافته و غیرفعال نمایید.

منابع و مراجع:

- Fox R. Linux with operating system concepts. Chapman and Hall/CRC; 2021 Dec 28.
- Dalheimer MK, Welsh M. Running Linux. O'Reilly; 2002 Dec.
- darsnameh.com/linux1

آزمایش 3

دستورات خط فرمان

اهداف:

- آشنایی با دستورات اساسی لینوکس

ابزار و مفاهیم مورد نیاز:

- Linux Terminal Emulator

محتوا:

- CLI
- شکل کلی دستورات خط فرمان:
- SYNOPSIS
- Switches / Options
- Navigation
- مجوزها
- صفحات Manual
- Sudo
- دستورات مدیریت فایل و دایرکتوری
- grep
- دستورات مدیریت و نظارت بر فرآیندها
- Repositories

مفاهیم و شرح دستورکار

3-1-دستورات خط فرمان

CLI (Command Line Interface) مکانیزمی برای برقراری ارتباط با سیستم عامل یا برنامه‌ها و اجرای دستورها بوسیله تایپ کردن است.

- هر دستور یک خلاصه (SYNOPSIS) دارد که نحوه کاربرد کلی دستور را مشخص می‌کند.
- دستورات لینوکس به بزرگی و کوچکی حروف حساسند.(case sensitive)
- حضور سوئیچ(switch/option)های مختلف، به هر دستور قابلیت‌های بیشتری می‌دهد و دستور را تعاملی‌تر و انعطاف پذیرتر می‌سازد .
- سوئیچ ها به کمک یک علامت - مشخص می‌شوند. اغلب می‌توان چند سوئیچ را بطور همزمان بکار برد که در آن صورت همه سوئیچ‌ها بعد از یک علامت - مشترک قرا می‌گیرند.
- برخی سوئیچ ها بعد از علامت -- قرار می‌گیرند .در اینحالت هر سوئیچ -- مجزا و مخصوص به خود را خواهد داشت.

دستورات پرکاربرد لینوکسی:

دستور pwd :

سرحرف print working directory است و مسیر جاری را نمایش می‌دهد.

```
pwd    [options]
```

دستور cd :

سرحرف کلمات change directory است و برای ورود به دایرکتوری و یا تغییر آن بکار می‌رود.

```
cd    [options]      نام و مسیر دایرکتوری
```

مثال : ورود به دایرکتوری دسکتاپ:

```
cd     Desktop
```

بازگشت به دایرکتوری ریشه(root) :

```
cd\
```

بازگشت به دایرکتوری بالاتر (معادل دکمه up در windows explorer):

```
cd ..
```

بازگشت سریع به آخرین دایرکتوری(معادل دکمهback در windows explorer):

```
cd -
```

دستور ls :

سرحرف list segments است و برای نمایش لیست فایل ها و دایرکتوری ها در مسیر جاریبا مسیر مشخص شده بکار می‌رود.

```
ls     [options]
```

برخی از آپشن‌ها:

-S

مرتب کردن خروجی بر اساس اندازه (در حالت پیش فرض مرتب سازی بر اساس حروف الفبا است)

-t

مرتب کردن خروجی بر اساس زمان تغییر

-l

نمایش لیست با جزئیات بیشتر (long list format)

-h

فایل های مخفی را نیز در لیست نمایش می‌دهد.

مثال زیر از سه سوئیچ بطور همزمان استفاده می‌کند:

```
$   ls      -lth

total 62K
drwxr-xr-x.       2     root  root  4.0K  Dec 3 22:25       Documents
drwxr-xr-x.       2     root  root  4.0K  Dec 2 20:47       Downloads
drwxr-xr-x.       5     root  root  4.0K  Nov 29 04:38      Progs
drwxr-xr-x.       2     root  root  4.0K  Nov 29 04:34      Desktop
-rw-r--r--.       1     root  root  46K   Nov 28 15:22      install.log
```

اطلاعاتی که در نتیجه‌ی اجرای دستور در ترمینال نمایش داده شده‌اند، به ترتیب عبارتند از:

permissions : مجوزها

number of links or directories inside this director : تعداد لینک ها یا دایرکتوری های موجود داخل دایرکتوری فعلی

The user that owns the file or directory : مالک

The group that file belongs to : نام گروه

The size in bytes : سایز برحسب بایت

The date of last modification : تاریخ آخرین تغییر

The name of the file or directory : نام

مجوزها:

هر مجوز از 4 فیلد مجزا تشکیل شده است، بعنوان مثال مجوزی که در سطر اول مثال قبل نمایش داده شده است را در نظر بگیرید.

d	rwx	r-x	r-x
فیلد ۱	فیلد۲	فیلد۳	فیلد۴

فیلد۱: اگر d بود یعنی آیتم مورد نظر دایرکتوری است. اگر علامت - بود یعنی آیتم مورد نظر فایل است.

فیلد 2 : مجوزهای مربوط به کاربر مالک فایل یا دایرکتوری را نمایش می‌دهد.

فیلد ۳ : مجوزهای مربوط به گروهی را نشان می‌دهد که فایل یا دایرکتوری مورد نظر به آن تعلق دارد.

فیلد ۴ : مجوزهای مربوط به سایر کاربران را نمایش می‌دهد.

هر کدام ازا این فیلدها از سه کاراکتر تشکیل شده اند .معنای هر کاراکتر در زیر آمده است:

r : اجازه خواندن(**r**ead) برای کاربر مورد نظر وجود دارد.

w : اجازه نوشتن (**w**rite) برای کاربر مورد نظر وجود دارد.

x : اجازه اجرا کردن (e**x**ecute) برای کاربر مورد نظر وجود دارد.

- : اجازه مورد نظر وجود ندارد.

دستور chmod :

سرحرف change mode است و برای تغییرمجوزهای فایل و دایرکتوری به کار می‌ود . مجوز مشخص می‌کند که چه افرادی چه مواردی را می‌توانند بخوانند، بنویسند و یا تغییر دهند.

```
chmod          [options]          value        file-path&name
```

همانطور که دیدیم، مجوز هر فایل یا دایرکتوری از سه فیلد (برای کاربر مالک، گروه‌ها و دیگران) تشکیل شده است . هر فیلد را با یک مقدار عددی (numeric Value) مشخص می‌کنیم:

Read : 4

Write : 2

E**x**ecute : 1

برای تخصیص چند مجوز همزمان به یک کاربر مقدار عددی آن مجوزها با هم جمع می‌شوند.

مثال :یک دایرکتوری با مجوز rwxr-x--x را درنظر بگیرید. می‌خواهیم مجوز این دایرکتوری را طوری تغییر بدهیم که کاربر همان مجوزهای قبلی را داشته باشد، اما گروه فقط مجوز خواندن و سایر افراد هیچ مجوزی نداشته باشند

۷	۵	۱	
owner user	group	others	
r+w+x	r+x	x	
۴+۲+۱	۴+۰+۱	۰+۰+۱	=۷۵۱

```
chmod   740      Desktop/mydirectory
```

آپشن‌ها:

-v

اطلاعات تغییر مجوز را نمایش می‌دهد.

-R

مجوز تعیین شده را به زیردایرکتوری‌ها و فایل‌ها نیز اعمال می‌کند.

برای مشاهده مجوزها در حالت گرافیکی روی فایل یا دایرکتوری راست کلیک کرده و از طریق گزینه properties زبانه permissions را ببینید.

دستور زیر تغییر مجوز را با حروف معادل، انجام می دهد. و مجوز فایلی به نام myfile را برابر با 754 تنظیم می نماید

```
chmod          u=rwx,g=rx,o=r          myfile
```

کاراکتر u نماینده کاربرمالک، g نماینده هم گروهی ها و o نماینده سایرین است. مجوزها نیز با همان کاراکتر های r,w,x مشخص شده اند. با اجرای دستور بالا، مجوزهای قبلی فایل از بین رفته و مجوز جدید (754) به فایل داده می‌شود.

اگر فقط قصد تغییر مجوزهای یکی از دسته‌های کاربران را داشته باشید، بصورت زیر عمل کنید. مثال زیر مجوز سایر کاربران را به r,w تغییر می‌دهد، اما مجوز مالک و هم گروهی‌ها به همان صورتی که بود باقی خواهد ماند. پس مجوز کل برابر 756 می شود.

```
chmod      o=rw          myfile
```

همچنین ممکن است بخواهید بدون تغییر باقی تنظیمات، برای یکی از دسته های کاربری مجوزی را اضافه یا کم نمایید

```
chmod      o+x          myfile
```

مثال بالا، مجوز اجرا را به سایرکاربران اضافه میکند. پس مجوز فایل برابر 757 خواهد شد

مثال بعدی مجوز خواندن را از هم گروهی ها می گیرد. پس مجوز فایل برابر 717 خواهد شد

```
Chmod     g-r       myfile
```

برای تغییر مجوز همه ی دسته های کاربران بصورت یکسان میتوان از کاراکتر a (all) استفاده کرد. مثال زیر مجوز نوشتن را به همه کاربران اعطا می‌کند. مجوز خواندن و اجرا برای هر دسته به همان صورتی که بود باقی خواهد ماند.پس مجوز فایل برابر با 737 خواهد شد.

```
chmod     a+w       myfile
```

دستور clear :

محتویات صفحه نمایش ترمینال را پاک می‌کند. برای مشاهده دستورات و محتوای قبلی، پنجره ترمینال را به سمت بالا جابجا (Scroll) نمایید.

دستور man :

صفحات manual (کتابچه راهنما)مربوط به دستور یا برنامه موردنظر را در صورت وجود نمایش می‌دهد.

```
man          [options]              command-or-application-name
```

راهنما در همان صفحه ترمینال نمایش داده می‌شود. برای جابجا شدن در صفحه از کلیدهای u و d و برای خروج از کلید q استفاده می‌شود.

مثال: اجرای دستور زیر راهنمای مربوط به دستور ls را نمایش می‌دهد:

```
man     ls
```

سوئیچ h– یا help--

تقریبا برای تمام دستورات لینوکس این سوئیچ تعریف شده است . استفاده از این سوئیچ ، راهنمای مربوط به یک دستور یا برنامه را نمایش می‌دهد.

```
application or command-name      -h    or    --help
```

مثال:

```
ls     --help
```

دستور su :

سرحرف switch user است. و برای تغییر و log in شدن با یک کاربر دیگر در ترمینال بکار می‌رود

```
su      user-name
```

با اجرای دستور su میتوان بین کاربران در در ترمینال سوئیچ کرد .پس از اجرای دستور پسورد کاربر موردنظر از شما درخواست میشود، مگر اینکه بخواهید از کاربر root به یک کاربر دیگر سوئیچ کنید .برای بازگشت به کاربر قبلی، کافیست دستور exit را اجرا نمایید.

دستور sudo :

برای اجرای برخی دستورات یا وظایف مدیریتی (administrative tasks) لازم است که مجوزهای کاربر root را داشته باشیم .علاوه بر su میتوان از دستور sudo نیز استفاده کرد .دستور sudo به کاربر معمولی اجازه میدهد دستورات را بعنوان root اجرا نماید، بدون اینکه رمز root را در اختیار داشته باشد و یا به کاربر روت سوئیچ کند.

در توزیع‌های مبتنی بر ردهت معمولاsudo بصورت پیشفرض برای کاربران غیر superuser فعال نیست. از طرفی، در اکثر توزیع های مبتنی بر دبیان مانند اوبونتو، کاربر root بصورت پیشفرض فعال نیست. و هنگام نصب، صرفا یک کاربر غیر root ایجاد می شود . این کاربر هنگام اجرای دستورات مدیریتی در ترمینال باید دستور sudo را در ابتدای خط ذکر نماید. پس از زدن کلید enter، از کاربر کلمه عبور وی درخواست خواهد شد. مشابه همین حالت، در انجام گرافیکیِ کارهای مدیریتی، وجود دارد. در این توزیع‌ها، دستور sudo بشکل پیشفرض برای کاربران فعال است و نیازی به انجام تنظیمات خاصی ندارد. برای فعال کردن کاربر root در اوبونتو، کافیست برای این کاربر پسورد تعریف کنیم:

```
[oslab@localhost ~ ]$ sudo passwd          root
[oslab@localhost ~ ]$ [sudo] password for oslab :
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

در این مثال، کاربر جاری oslab نام دارد. پس از اجرای دستور ابتدا باید پسورد oslab را وارد کنید و سپس پسورد انتخابی برای کاربر ریشه را دو بار تایپ نمایید. سپس می توانید با دستور زیر به کاربر ریشه سوئیچ نمایید:

```
su  -
```

دستور find :

برای یافتن فایل ها در سیستم سلسله مراتبی لینوکس به کار می رود. می توان جستجو را را بر اساس نام، مالک، گروه، نوع، مجوزها، تاریخ و دیگر مؤلفه های یک فایل انجام داد. جستجو در تمامی زیر شاخه های مسیر تعیین شده انجام خواهد گرفت.

```
find   [options]         [path]          file-expression
```

مثال اول : فایل های با نام test را در مسیر جاری جستجو میکند.

```
[root@localhost Desktop]# find        -name test
./test
```

مثال دوم: تمام فایل هایی که بخش ابتدایی نامشان test است را در مسیر جاری می یابد.

```
 [root@localhost Desktop]# find  -name           "test*"
./test
./test1
./test.txt
```

مثال سوم: فایل هایی با نام text.txt را در مسیر /home/oslab/ جستجو میکند.

```
[root@localhost Desktop]# find         /home/oslab/         -name
     text.txt
/home/oslab/text.txt
```

مثال چهارم: فایل linux.pdf را بدون توجه به بزرگ و کوچک بودن حروف نام فایل، جستجو می کند.

```
[root@localhost Desktop]# find -iname linux.pdf
./linux.pdf
./LiNuX.pdf
```

مثال پنجم: جستجوی فایل های با حجم 10 مگابایت یا بیشتر در مسیر جاری

```
 [root@localhost Desktop]# find  -size  +10M
./linux.pdf
```

دستور mkdir

یک دایرکتوری جدید ایجاد میکند.

```
mkdir         path-and-name-of-new-directory
```

دستور rmdir

یک دایرکتوری موجود را پاک میکند.

```
rmdir               path-and-name-of-new-directory
```

توجه :این دستور فقط دایرکتوری های خالی را پاک میکند. بنابراین اگر دایرکتوری مورد نظر حاوی زیردایرکتوری بود، باید از پاک کردن آنها شروع کنیم. همچنین این دستور قادر به پاک کردن فایل‌ها نمی‌باشد.

دستور rm

برای پاک کردن فایل ها استفاده می‌شود.

```
rm     path-and-name-of-file[s]
```

-r

پاک کردن یک دایرکتوری با تمام محتویاتش

دستور cp

کپی کردن فایلها و دایرکتوریها

```
cp   [options]    source-file-name&path    destination-file-name&path
```

-r

کپی کردن دایرکتوری

دستور mv

تغییر نام و مکان یک فایل بطور همزمان

```
mv        old-path-and-name-of-file          new-path-and-name-of-file
```

چند نکته کاربردی :

خاصیت auto fill معمولا در ترمینال فعال است. هنگام تایپیک دستور یا نام یک دایرکتوری، اگر بخشی از آن را تایپ کرده و کلید tab را بزنید بقیه دستور یا نام بطور اتوماتیک پر می‌شود.

اگر چند دایرکتوری با حروف ابتدایی یکسان وجود داشته باشد، فقط تا پایان قسمت مشترک پر می‌شود یا تمام گزینه-های موجود را نمایش می‌دهد. همچنین اگر دستورات دیگری با همان حروف اولیه وجود داشته باشند، تمام دستوراتی که با آن حروف شروع می‌شوند را نمایش می‌دهد .

اگر نام فایل یا دایرکتوری شامل کاراکتر فاصله باشد، هنگام تایپ در ترمینال باید کاراکتر فاصله بعد از کاراکتر \ قرار بگیرد در غیر اینصورت نام نوشته شده به منزله دو نام مجزا در نظر گرفته میشود.

دستور grep

برای جستجوی یک کلمه یا رشته درون فایلها استفاده میشود و تمام سطرهای داخل فایل را که حاوی کلمه مورد جستجو هستند، نمایش میدهد.

```
grep  [options]  pattern-word          path&file-name(s)-to-be-
searched
```

--color

کلمه مورد جستجو را در خروجی، بشکل رنگی نمایش میدهد.

مثال زیر کلمه configurationرا در فایلی به نام linux-commands که در مسیرDesktop قرار دارد، جستجو می‌کند و تمام سطرهای درون فایل که حاوی این کلمه هستند را چاپ می‌کند.

```
# grep     --color    configuration    Desktop/linux-commands
Yum uses a configuration file at /etc/yum.conf.
Additional configuration files are also read from the directories
Also this configuration is set to it,
The inclusion of external configuration files is supported.
```

-n

شماره سطر را نیز نمایش میدهد.

-c

فقط تعداد کل سطرهایی که حاوی کلمه موردنظر هستند را نمایش میدهد

-i

در هنگام جستجو، تفاوتی بین حروف بزرگ و کوچک قائل نمیشود(case-insensitive)

^

تمام سطرهایی را نمایش می‌دهد که با کلمه مورد نظر آغاز می‌شوند.

نتیجه اجرای مثال زیر، تمام سطرهایی خواهد بود که اولین کلمه آن‌ها there باشد. (توجه : اگر اولین کاراکتر یک سطر، کاراکتر فاصله باشد، آن سطر در نتایج نمایش داده نمیشود.)

```
grep              ^there          Desktop/linux-commands
```

برای جستجوی همزمان در چند فایل ، کافیست نام و مسیر هر فایل را با یک فاصله در ادامه دستور اضافه کنید و یا می‌توانید با کاراکتر * در بین تمام فایل‌ها با یک پسوند خاص جستجو کنید.

-l

فقط نام فایلهایی که حاوی کلمه مورد جستجو هستند را لیست میکند.

مثال :دستور زیر در بین تمام فایلهای موجود در دسکتاپ کلمه configuration را جستجو خواهد کرد. در نتایج جستجو، قبل از هر سطر، نام فایل مربوطه نیز ذکر میشود.

```
[root@localhost Desktop]# grep        configuration         *
```

برای جستجوی یک رشته، آن را در بین دو علامت '' قرار دهید.

```
[root@localhost Desktop]# grep        'configuration file'       *
```

دستور cat

محتوای فایل‌ها را به صورت پیوسته و پشت سر هم نمایش می‌دهد.

```
cat   options          path-and-name-of-files
```

-n

نمایش شماره سطر

دستور tar

ایجاد و مدیریت فایل‌های آرشیو

```
tar   [options]   new-file-name.tar        file-or-directory-name(s)
```

-c

ساختن فایل آرشیو(creat)

-f

نگاشتن آرشیو ساخته شده در یک فایل

-u

آپدیت کردن فایل آرشیو موجود بر اساس تغییرات ایجاد شده در فایل اصلی

-r

الحاق(اضافه کردن) فایل یا دایرکتوری جدید به آرشیو موجود .(این سوئیچ آیتم جدید را به داخل بالاترین دایرکتوری اضافه میکند)

مثال. دو فایل e1 و e2 را در یک فایل به نام myfiles1 آرشیو کرده و در مسیر root/Documents ذخیره نمایید.

```
# tar  -cf     Documents/myfiles1.tar     Desktop/e1     Desktop/e2
```

دستور who

کاربرانی که در حال حاضر به سیستم log in هستند را نمایش می‌دهد.

```
who
```

دستور whereis

مکان فایلهای باینری، سورس کدها و راهنماهای مربوط به یک برنامه یا دستور را مشخص میکند.

```
whereis          [options]          application/command-name
```

-m

فقط مکان فایلهای راهنما(manual) را نمایش میدهد.

-b

فقط مکان فایلهای باینری را نمایش میدهد.

-s

فقط مکان فایلهای سورس را نمایش میدهد.

دستور which

مشخص میکند با اجرای یک دستور یا برنامه کدام فایل از روی دیسک اجرا میشود. (فایل اجرایی دستور یا برنامه موردنظر در کجا قرار دارد)

```
which              application/command-name
```

دستور uname

اطلاعات سیستم را چاپ می‌کند.

```
uname            [options]
```

-i

نوع hardware platform را چاپ می‌کند .(به کمک این دستور میتوانید32 یا 64بیتی بودن سیستم را تشخیص بدهید)

-p

نوع پردازنده را مشخص میکند(.مشابه سوئیچ i عمل می‌کند)

-s

نام هسته (kernel) را چاپ میکند.

دستور arch

این دستور نیز معماری سیستم را نمایش می‌دهد.

```
arch
```

دستور service

برای آغاز کردن ، پایان دادن و یا راه‌اندازی مجدد یک سرویس موجود در سیستم استفاده می شود .

```
service          service-name       start/stop/restart
```

مثال زیر سرویس شبکه و اتصال به اینترنت را restart میکند.

```
# service   network   restart
```

دستور kill

با ارسال سیگنالهای مقتضی به یک فرایند (process)یا گروهی از فرایندها، به آن‌ها پایان می‌بخشد .

برای مشاهده process ID تمام فرایندهای فعال در سیستم، دستور top (که در ادامه به آن خواهیم پرداخت) را اجرا کنید.

```
kill        [options]              pid(process ID)
```

دستور killall

یک فرایند را با استفاده نامش پایان میبخشد.

```
killall          [options]          process-name
```

-9

سیگنال شماره ۹ برای از بین بردن اجباری فرایند استفاده می‌شودحتی اگر فرایند در حال استفاده باشد (force terminate)

دستور ps

وضعیت پروسس ها را گزارش میکند .این گزارش در لحظه اجرای دستور تولید میشود و تعاملی نیست .بعبارت دیگر، مانند یک snapshot تصویری از وضعیت پروسس ها در یک لحظه بدست می دهد.

```
ps      [options]
```

-e

نمایش تمام پروسه ها

-f

نمایش با جزئیات (full format)

-p process-ID

فقط پروسس هایی با شماره آی دی مشخص شده را لیست می‌کند.

--ppid Parent-Process-ID

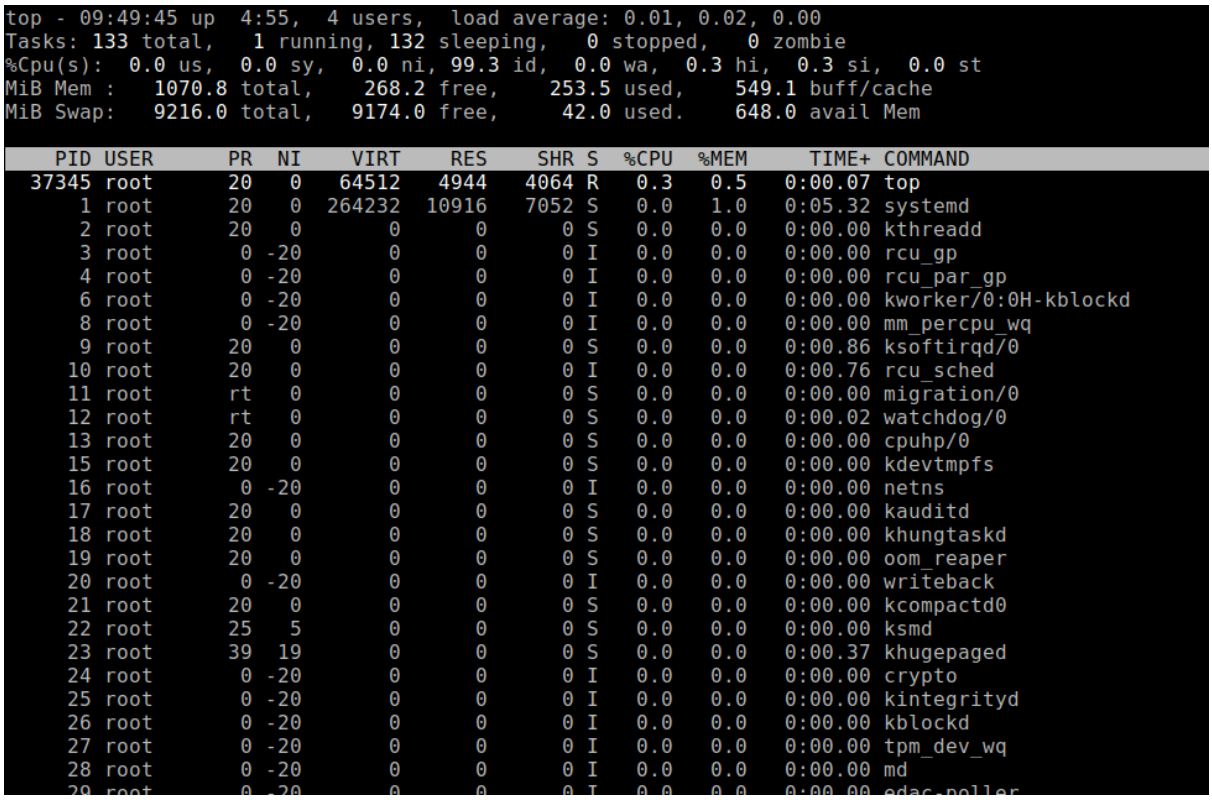
فقط پروسس‌هایی که شماره آی دی والدشان مشخص شده را لیست می‌کند.

مثال. میخواهیم فقط پروسس هایی که آی دی والدشان 2 است را مشاهده کنیم.

```
# ps  -f     --ppid     2
```

دستور top

یک برنامه نظارت بر عملکرد است. از دستور top برای نمایش تمام فرآیندهای در حال اجرا و فعال در زمان واقعی در یک لیست مرتب و با به روز رسانی منظم استفاده می‌شود. مقدار استفاده ازCPU ، استفاده از حافظه، Swap Memory، Cache Size، Buffer Size، Process PID، User، Commands و موارد دیگر را نمایش می‌دهد.



تصویر 3-1- خروجی دستور top

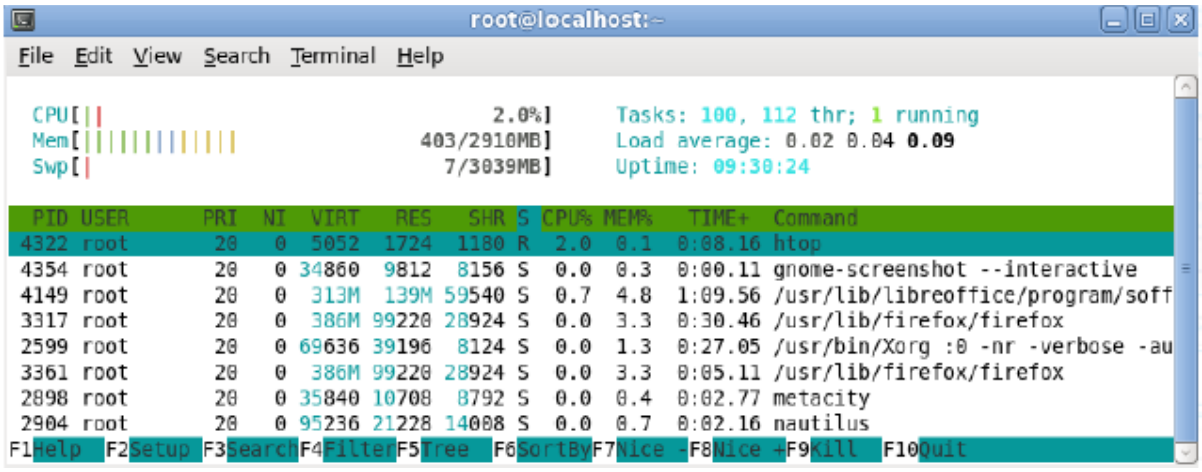
دستور htop

همان دستور top است که همراه با رابط کاربر بهتری ارائه شده است. ابزاری تعاملی است که بشکل بلادرنگ سیستم را مانیتور کرده و اطلاعاتی درمورد میزان مصرف cpu ، حافظه و ... ارائه می‌دهد.

درصورتیکه این ابزار در سیستم شما نصب نیست می‌توانید به شکل زیر آن را نصب نمایید.

```
sudo  apt-get          install          htop
```

پس از نصب، در ترمینال دستور htop را اجرا نمایید .این ابزار دارای واسط گرافیکی مستقل نیست و از ترمینال برای ارتباط با کاربر استفاده میکند . برای بالا و پایین رفتن در صفحه کلیدهای page up & down و برای خروج از q استفاده کنید.



تصویر 3-2- خروجی دستور htop

درقسمت بالای تصویر، اطلاعات مربوط به پردازنده، اطلاعات حافظه و swap نمایش داده می‌شوند. (swap فضایی از هارد است که بعنوان رم کمکی یا مجازی مورد استفاده قرار می‌گیرد .)

معنای رنگ خطوط برای حافظه و swap متفاوت است:

- سبز :میزان حافظه مصرفی توسط پروسس ها
- آبی :میزان بافر مصرفی
- زرد :میزانcache
- بالا سمت راست، تعدادtask های در حال اجرا/ فعال و میانگین بار و زمان uptime را میبینید .در قسمت میانگین بار سه عدد وجود دارد .اولی میانگین در بازه یک دقیقه، دومی در پنج و سومی در پانزده دقیقه است.
- در بخش وسط تصویر، موارد زیر قابل مشاهده اندکه باکلیک روی هر کدام، اطلاعات براساس آن ستون مرتب میشوند.
- PID: آی دی پروسس
- USER: مالک یا اجرا کننده پروسس
- PR: اولویت پروسس .هر چه عدد کوچکتر باشد، اولویت بالاتر است
- NI: معیار دیگری برای اولویت بندی پروسس هاست (nice)
- VIRT: میزان حافظه مجازی مصرفی توسط هر پروسس

- RES: میزان رم فیزیکی مصرفی توسط هر پروسس
- S: وضعیت هر پروسس که می تواند «در حال اجرا» یا در حالت«خواب» باشد
- %cpu : درصد زمانی استفاده از سی پی یو
- MEM : درصد رم مورد استفاده توسط هر پروسس
- TIME : مدت زمانی که پروسس وقت سی پی یو را گرفته است
- Command : دستوری که پروسس را آغاز کرده است
- SHR : میزان حافظه اشتراکی (shared)توسط هر پروسس

دستور netstat

یک ابزار خط فرمان برای نظارت بر آمار بسته‌ها و پورت‌های ورودی و خروجی شبکه است. یک ابزار بسیار مفید برای هر مدیر سیستم برای نظارت بر عملکرد شبکه و عیب‌یابی مشکلات مربوط به شبکه است.

```
netstat            – Network Statistics
```

مثال:

```
netstat        -a    |    more
```

```
[root@tecmint:~]# netstat -a | more
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:sunrpc          0.0.0.0:*               LISTEN
tcp        0      0 tecmint:domain          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:ssh             0.0.0.0:*               LISTEN
tcp        0      0 localhost:postgres      0.0.0.0:*               LISTEN
tcp        0      0 tecmint:ssh             192.168.0.124:45611     ESTABLISHED
tcp6       0      0 [::]:sunrpc             [::]:*                  LISTEN
tcp6       0      0 [::]:ssh                [::]:*                  LISTEN
tcp6       0      0 localhost:postgres      [::]:*                  LISTEN
udp        0      0 0.0.0.0:mdns            0.0.0.0:*
udp        0      0 localhost:323           0.0.0.0:*
udp        0      0 tecmint:domain          0.0.0.0:*
udp        0      0 0.0.0.0:bootps          0.0.0.0:*
udp        0      0 tecmint:bootpc          _gateway:bootps         ESTABLISHED
```

دستور Lsof

ابزاری قدرتمند برای مانیتورینگ عملکرد در سیستم‌عامل‌های لینوکس و یونیکس، فهرستی از تمام فایل‌ها و پردازش‌های باز را به شما نمایش می‌دهد. این ابزار اطلاعات کاملی از فایل‌های روی دیسک، سوکت‌های شبکه، پایپ‌ها، دستگاه‌های متصل و پردازش‌های در حال اجرا را ارائه می‌کند.

استفاده از این دستور در مقایسه با سایر ابزارهای مانیتورینگ عملکرد لینوکس، زمانی که جداسازی(unmount) دیسک از سیستم با خطاهای مربوط به عدم دسترسی یا نقص در سیستم فایل مواجه می‌شود، از مزایای قابل توجهی برخوردار است. این ابزار به شما امکان می‌دهد تا بدون نیاز به جداسازی دیسک به اطلاعات مربوط به عملکرد آن دسترسی پیدا کنید و مشکلات را به سرعت شناسایی و حل کنید. با این دستور به راحتی خواهید فهمید که چه فایل‌هایی در حال استفاده هستند و مانع جداسازی دیسک می‌شوند. فرمت استفاده از دستور به‌صورت زیر است:

```
# lsof
```

دستور df

برای بررسی فضای ذخیره‌سازی و دیسک سرور به کمک شما می‌آید.

```
df     -h
```

-h

برای نمایش خروجی به‌شکل ساده‌تر و با واحد اندازه‌گیری استفاده می‌شود.

در خروجی، جدولی با مقادیر زیر مشاهده خواهید کرد:

- Filesystem : نام هر درایو
- Size : اندازه‌ی هر درایو
- Used : میزان دیسک استفاده‌شده توسط آن درایو
- Avail : میزان دیسک آزاد در آن درایو
- Use٪ : درصد استفاده‌شده از دیسک در آن درایو
- Mounted on : دایرکتوری که درایو به آن مانت شده است

دستور chkconfig

برای مشاهده، تنظیم یا تغییر لیست سرویس هایی یکار می‌رود که در زمان sartup به شکل خودکار آغاز می‌شوند. اجرای دستور بدون هیچگونه سوئیچی یا همراه با list-- سرویس های مذکور را لیست می‌کند.

```
chkconfig  --list
cpuspeed   0:off      1:on    2:on     3:on    4:on    5:on   6:off
vsftpd     0:off      1:off   2:off    3:off   4:off   5:off  6:off
network    0:off      1:off   2:on     3:on    4:on    5:on   6:off
```

اعداد 0 تا 6، سطوح اجرایی برنامه ها را مشخص می‌کنند.

دستور reboot

سیستم را restart می‌کند.

```
reboot
```

دستور halt

سیستم را shut down می‌کند.

```
halt
```

3-2-مخازن نرم افزاری

repositoryها (مخزن‌ها)یا بشکل خلاصه repoها مخازن نرم افزاری هستند که می توان بسته‌های نرم افزاری را از آن‌ها دریافت و بر روی سیستم نصب و یا بسته‌های موجود را به روز رسانی نمود .

هر توزیع لینوکس مخازن مربوط به خود را دارد. همچنین برای یک توزیع معمولا مخازن مختلفی وجود دارد. آدرس مخزن‌های اصلی در فایل /etc/apt/sources.list قرار دارد.

دستور apt-get

برای نصب، حذف و مدیریت بسته های نرم افزاری در لینوکس های deb-base مورد استفاده قرار می‌گیرد. لینوکس-هایی از قبیل Debian ،Mint و Ubuntu از جمله توزیع‌های deb-based هستند. این دستور معادل دستور yum در توزیع‌های redhat-based است.

دستور apt-get به صورت چند فرمانی است یعنی ابتدا apt-get نوشته می‌شود و سپس دستور فرعی مورد نظر مثلاً نصب یا حذف و یا ... را ذکر می‌کنیم. مثلا برای نصب یک بسته نرم افزاری به صورت زیر عمل می‌کنیم.

```
apt-get    install          packageName
```

به عنوان مثال برای نصب برنامه unrar که وظیفه استخراج فایلهای rar را دارد به این صورت عمل میکنیم:

```
$ sudo            apt-get      install          unrar
```

همانطور که می بینیدچون کاربر root نیست(با توجه به علامت prompt) بنابراین دستور sudo پیش از این دستور آورده شده است.

دستور زیر یک بسته نرم افزاری نصب شده را پاک می‌کند، اما فایل های پیکره بندی (Config) آن را نگه می دارد.

```
apt-get           remove       packageName
```

برای مثال برنامه unrar را پاک می‌کنیم:

```
$ sudo            apt-get          remove            unrar
```

برای اینکه فایل‌های پیکربندی نیز حذف شوند:

```
apt-get     --purge      remove             packageName
```

تمرین کلاسی:

- دستورهای خط فرمانی را در گروه خود تمرین کنید.

گزارش کار:

- موارد زیر را به ترتیب و صرفا از طریق خط فرمان انجام داده، و گزارش کار تهیه نمایید.
- وارد دایرکتوری boot بشوید.
- در دایرکتوری جاری و زیرشاخه‌هایش به دنبال فایل‌هایی که حاوی عبارت grub در اسمشان هستند بگردید.
- درون این فایل‌ها به شکل همزمان و با استفاده از دستور grep وآپشن های مقتضی، سطرهایی را که با کاراکتر # شروع نمی‌شوند، لیست کنید.

منابع و مراجع:

- Purcell J, Hat R. Linux complete command reference. Sams; 1997 Dec 1.
- Dalheimer MK, Welsh M. Running Linux. O'Reilly; 2002 Dec.

آزمایش 4

آشنایی با برنامه‌نویسی پوسته

اهداف:

- آشنایی با مفاهیم پایه برنامه نویسی پوسته (Shell Scripting)

ابزار و مفاهیم مورد نیاز:

- CLI
- ابزارهای ویرایش متن
- انواع پوسته

محتوا:

- آشنایی با مفهوم و کاربردهای برنامه نویسی پوسته
- پیش‌نیازهای برنامه‌نویسی پوسته
  - ابزارهای ویرایش متن
  - عملگرهای هدایتگر
  - مدیریت مجوز فایل و دایرکتوری در لینوکس
- انواع پوسته
- ایجاد و اجرای یک اسکریپت ساده

مفاهیم و شرح دستورکار

4-1-برنامه نویسی پوسته

به زبان ساده، اسکریپت پوسته یک برنامه کامپیوتری است که توسط پوسته اجرا می‌شود. پوسته نیز یک مفسر[۲] است که دستورات خط فرمان را دریافت و اجرا می‌نماید. زبان‌های مختلفی که برای نوشتن این اسکریپت‌ها مورد استفاده قرار می‌گیرند، زبان‌های اسکریپت نویسی نام دارند. عملیاتی که عموما توسط اسکریپت‌ها قابل انجام هستند عبارتند از: ایجاد تغییر در فایل سیستم، اجرای برنامه‌ها و چاپ پیغام‌ها.

هر خط در یک برنامه‌ی پوسته شامل یک دستور استاندارد پوسته یا لینوکس است. دستورات لینوکسی همان دستوراتی هستند که در فصل قبل با آن‌ها آشنا شدید و هر یک از این دستورات به تنهایی و مستقیما از خط فرمان نیز قابل اجرا هستند. با دستورات پوسته نیز در ادامه آشنا خواهید شد. مزیت ایجاد اسکریپت در این است که فقط با اجرای یک فایل اسکریپت، تمام دستوراتی که در فایل هستند اجرا خواهند شد و دیگر نیازی به تایپ مجدد دستورات در دفعات بعدی نیست. برنامه‌های پوسته کامپایل نمی‌شوند بلکه توسط پوسته تفسیر و اجرا می‌گردند.

4-1-1-چرا باید برنامه نویسی پوسته را یاد بگیریم؟

یادگیری مفاهیم اسکریپت نویسی برای مدیر سیستم(آدمین)، موضوعی ضروری است. زیرا حتی اگر ادمین نخواهد به معنای واقعی اسکریپت نویسی کند، برای درک و تحلیل رفتار سیستم به آن نیاز دارد. مثلا هنگامی که یک سیستم لینوکسی بالا می‌آید، اسکریپت‌های سیستم را که در /etc/rc.d قرار دارند، اجرا می‌کند. این اسکریپت‌ها حاوی تنظیمات

2. interpretor

سیستم و سرویس‌های سیستم هستند. مدیر سیستم برای اینکه بتواند رفتار سیستم را تحلیل کند، باید درباره‌ی این اسکریپت‌های راه انداز[3] و طرز کارشان آشنایی کافی کسب کرده و در صورت لزوم بتواند در آن‌ها تغییراتی ایجاد نماید.

معمولا از برنامه نویسی پوسته برای خودکارسازی کارهای روتین، گزارش خطا، مانیتورینگ سیستم، سینک کردن و مواردی از این دست استفاده می‌شود.

4-1-2-چه زمانی نباید از برنامه نویسی پوسته استفاده کرد؟

بطور معمول برای برنامه‌های سنگین، یعنی برنامه‌هایی که نیازمند منابع محاسباتی یا انجام عملیات ریاضی زیاد هستند، اسکریپت نویسی پوسته گزینه‌ی مناسبی نمی‌باشد. همچنین برای ایجاد برنامه‌های پیچیده یا برنامه‌های گرافیکی، زبان‌های برنامه نویسی و اسکریپت نویسی مناسب‌تری وجود دارند.

4-2-پیش‌نیازهای برنامه نویسی پوسته

برای شروع اسکریپت نویسی در عمل، علاوه بر دانشی که از بخش قبل به آن دست پیدا کردیم، لازم است با ابزارهایی نیز آشنا شویم.

4-2-1- ابزارهای ویرایش متن

برای ایجاد و ویرایش فایل‌های متنی در لینوکس، دستورات خط فرمانی و برنامه‌های گرافیکی متعددی وجود دارند. در ادامه با یک نمونه از هر مورد آشنا می‌شویم.

الف) ویراشگرهایی که واسط گرافیکی مجزا دارند. مثل kwriter، emacs ،gedit و... معمولا ظاهری شبیه Notepad در ویندوز دارند ولی در باطن قابلیت‌های بسیار بیشتری را پشتیبانی می‌کنند. مثلا اگر از این ویرایشگرها برای کدنویسی

3. Startup scripts

استفاده نمایید، کدهای شما را تشخیص داده و مثل یک IDE بخش‌های مختلف کد را بصورت رنگی نمایش می‌دهند.

در تصویر 4-1 محیط ویرایشگر gedit را مشاهده می‌نمایید.

تصویر4-1- ویرایشگر gedit

این برنامه معمولا بطور پیش‌فرض در اکثر توزیع‌های لینوکسی نصب می‌شود و در منوی Applications → Accessories → Text Editor قرار دارد. اما اگر موفق به پیدا کردن برنامه نشدید، می‌توانید آن را از خط فرمان فراخوانی نمایید. دستور زیر یک فایل خالی و جدید (با نام myFile.txt) را در مسیر جاری[4] ایجاد می‌نماید.

```
gedit   myFile.txt
```

درمورد سایر ویرایشگرها نیز همین دستورالعمل صادق است.

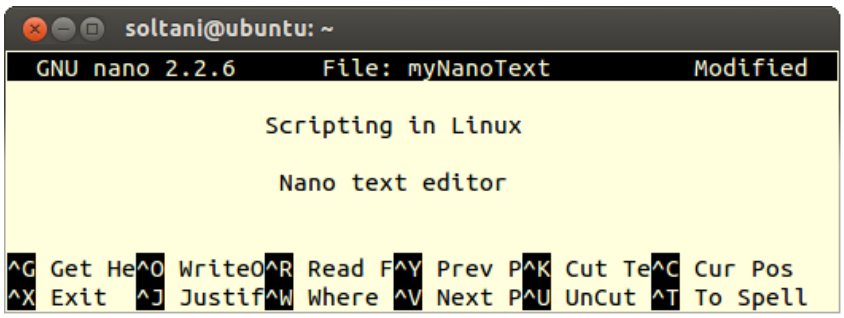
ب)ویرایشگرهای خط فرمانی که از پنجره ترمینال بعنوان واسط کاربری خود استفاده می‌نمایند. مثل vim و nano.

برای ورود به محیط ویرایشگر نانو دستور nano و پس از یک فاصله نام و مسیر فایل دلخواهتان را تایپ نمایید. مثال زیر فایل myNanoText را در مسیر جاری ایجاد می‌نماید.

```
nano   myNanoText
```

4 . برای دیدن مسیر جاری در سیستم خود دستور pwd را اجرا نمایید

نکته: اگر فایلی با همین نام و در همین مسیر از قبل موجود باشد، نانو فایل موجود را باز می‌کند. در غیر اینصورت، یک فایل جدید و خالی با این نام ایجاد خواهد کرد. در تصویر 4-2 محیط ویرایشگر نانو را مشاهده می‌کنید. قسمت بالای ویرایشگر برای تایپ در نظر گرفته شده و در قسمت پایین، راهنمای امکانات موجود در نانو، نمایش داده می‌شوند.



تصویر 4-2- ویرایشگر nano

بلافاصله پس از باز شدن فایل در محیط نانو، می‌توانید تایپ کردن را شروع کنید. برای خروج از ویرایشگر و بازگشت به ترمینال، باید از کلیدهای میانبری که در قسمت پایین ویرایشگر نمایش داده می‌شوند استفاده کنید. بدین منظور، ابتدا کلیدهای Ctrl و x را بطور همزمان از صفحه کلید بفشارید. یک پیغام در قسمت پایین نانو نمایش داده می‌شود و از شما می‌پرسد که آیا قصد ذخیره تغییرات را دارید؟ در صورتیکه پاسختان مثبت است، کلید y و در غیر اینصورت کلید n را بفشارید.

مجددا یک سوال دیگر از شما پرسیده می‌شود که در آن، مسیر و نام فایل نمایش داده شده است. اگر قصد Save As کردن فایل را ندارید، کافیست کلید Enter را بزنید. حالا از محیط نانو خارج شده و به ترمینال بازگشته‌اید! می‌توانید به محلی که فایل را در آن ایجاد کرده بودید رفته و وجود فایلتان را چک کنید!

انتخاب ویرایشگر برای اسکریپت نویسی کاملا سلیقه‌ای است و نوع ویرایشگر هیچ تاثیری در نتیجه و اجرای اسکریپت نخواهد داشت. پس با هر ویرایشگری که می‌توانید راحت‌تر کار کنید، همان را برای ادامه کار انتخاب نمایید.

ویرایشگر vim: برای کار با ویرایشگر vim شاید نیاز به کمی تمرین بیشتر داشته باشید. اما نکات کلیدی زیر برای شروع کار بسیار مفید خواهند بود. ابتدا دستور زیر را اجرا کنید تا یک فایل جدید در محیط vim باز شود.

```
vim   file1.txt
```

در vim دو حالت تایپ و دستور وجود دارد. همانطور که از اسمشان پیداست، از حالت تایپ برای ویرایش متن و از حالت دستور برای فرمان دادن به برنامه (مثلا ذخیره کردن یا خروج از برنامه) استفاده می‌شود. برای فعال شدن حالت تایپ، پس از ورود به محیط vim کافیست یکبار کلید i را بفشارید. حتما متوجه شدید که قبل از زدن این دکمه، هر چه تایپ می‌کردید اتفاقی در ویرایشگر نمی‌افتاد! پس از اتمام تایپ و برای ذخیره کردن تغییرات و خروج از برنامه ابتدا کلید Esc را بزنید تا حالت دستوری فعال شود و سپس همزمان با پایین نگه‌داشتن کلید shift، دو بار کلید z را بفشارید. در واقع باید دوبار Z بزرگ را بزنید تا دستور لازم برای ذخیره و خروج صادر گردد.

4-2-2- عملگرهای هدایتگر [5]

با استفاده از عملگرهای هدایتگر می‌توان خروجی یک دستور یا اسکریپت را به یک دستور دیگر یا به یک فایل هدایت کرد. به این ترتیب دیگر نتایج در ترمینال نمایش داده نخواهند شد. در ادامه با سه هدایتگر مهم آشنا می شویم.

الف) هدایتگرِ >

کاراکتر > در زمینه‌های دیگر مثل ریاضیات برای مقایسه و مشخص کردن عدد بزرگتر بکار می‌رود. اما در اینجا یک هدایتگر است که خروجی حاصل از یک دستور را به یک فایل هدایت می‌نماید. بنابراین خروجی دستور در ترمینال نمایش داده نخواهد شد، بلکه در فایل مشخص شده نوشته می‌شود. این عملگر به صورت زیر به کار می‌رود:

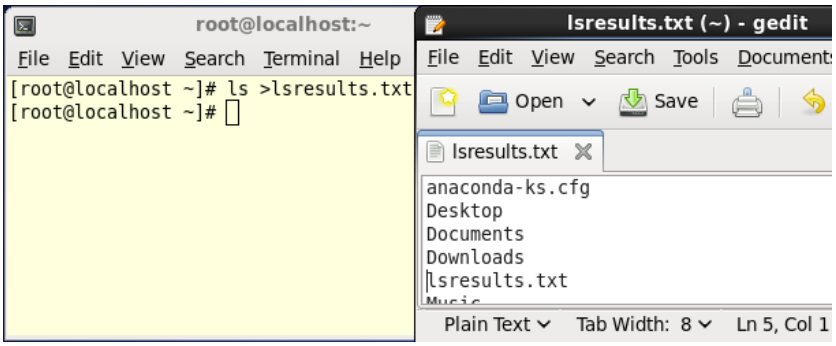
```
نام و مسیر فایل   >   دستور
```

مثال:

```
ls >lsresults.txt
```

5. redirection

دستور ls محتویات فولدر جاری(در اینجا پوشه روت) را لیست کرده اما همانطور که در تصویر زیر می‌بینید در ترمینال چیزی نمایش داده نمی‌شود، زیرا نتیجه در فایلی به نام lsresults.txt نوشته شده است.



تصویر 4-3- استفاده از هدایتگر > و فایل حاصل

نکته : دستور ls را یکبار بسادگی در ترمینال اجرا کرده و نتیجه را با آنچه که در فایل lsresults.txt ذخیره کرده‌اید مقایسه نمایید. ملاحظه خواهید کرد که در ترمینال، اسم آیتم‌ها پشت سر هم نمایش داده می‌شوند اما در فایل، هر آیتم در یک سطر مجزا چاپ شده است. تفاوت دوم این است که هنگام استفاده از هدایتگر، اسم فایل lsresults.txt نیز در لیست نتایج وجود دارد! علت این است که ابتدا این فایل ایجاد می‌شود و سپس دستور ls اجرا شده و نتیجه را در فایل می‌ریزد.

ب) هدایتگرِ <

این کاراکتر یک هدایتگر است که اطلاعات داخل فایل را خوانده و بعنوان ورودی به یک دستور تحویل می‌نماید. این عملگر به صورت زیر به کار می‌رود:

```
دستور  <  نام و مسیر فایل
```

مثال:

```
wc  -w  <lsresults.txt
```

فایل lsresults.txt که در بخش قبل ایجاد کردیم، بعنوان ورودی به دستور wc داده می‌شود. این دستور به همراه آپشن w– تعداد کلمات داخل فایل را شمرده و نمایش خواهد داد.

نکته: ورودی دستور از فایل گرفته می‌شود ولی نتیجه‌ی دستور در ترمینال نمایش داده می‌شود.

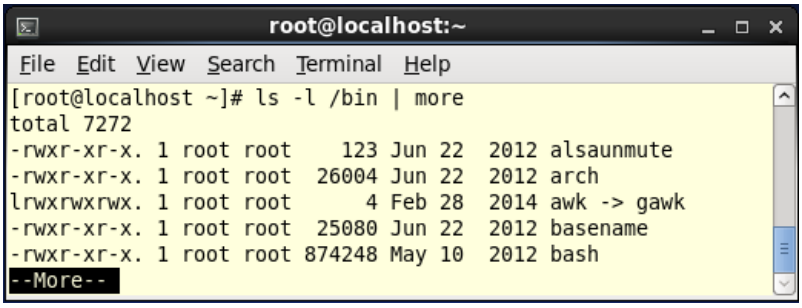
ج) هدایتگرِ |

این هدایتگر، پایپ نام دارد و خروجی یک دستور را بعنوان ورودی به یک دستور دیگر هدایت می‌کند.

```
دستور1  |  دستور2
```

مثال:

```
ls   -l   /bin  | more
```



تصویر 4-4-عملگر پایپ

نتیجه‌ی دستور ls در مثال بالا، یک لیست بلند بالا و چند صفحه ای است. یعنی اگر این دستور را به تنهایی در خط فرمان اجرا کنید، باید در پنجره ترمینال اسکرول کنید تا بتوانید تمام نتایج را مطالعه نمایید. دستور more، کار مطالعه نتایج را آسانتر می‌کند. به این ترتیب که ابتدا نتایج حاصل از دستور ls را توسط هدایتگر پایپ دریافت نموده و بخش اول نتایج را نمایش می‌دهد. سپس با هر بار فشرده شدن کلید Enter(یا Space) سطربعدی(یا صفحه بعدی) نمایش داده خواهد شد.

در مثال زیر نیز با کمک هدایتگر پایپ، نتایج حاصل از دستور سمت چپ(ls) بعنوان ورودی به دستور سمت راست(grep) داده می‌شود. درست مثل این است که نتیجه ls را در یک فایل متنی ذخیره نماییم و سپس دستور grep را بر روی آن فایل اجرا نماییم. تمام سطرهایی که حاوی کلمه server هستند، در ترمینال نمایش داده خواهد شد.

```
ls   -l   /bin  | grep server --color
```

نکته 1: فقط خروجی حاصل از دستور دوم در ترمینال نمایش داده می‌شود.

نکته 2: دستور زیر را اجرا کرده و نتیجه‌ی آن را با مثال بالا مقایسه کنید.

```
find   /bin   -name   "*server*"
```

بعنوان آخرین مثال از این بخش، استفاده همزمان از دو هدایتگر را می‌بینند. دستور sort، برای مرتب کردن نتایج بکار می‌رود.

```
ps | sort > pssort.txt
```

خروجی ps بعنوان ورودی به sort داده می‌شود و پس از مرتب سازی در فایلی به نام pssort.txt ذخیره می‌گردد.

4-2-3- پوسته[6] ها

همانطور که قبلا هم دیدید، در لینوکس دستورها به وسیله‌ی ترمینال به سیستم عامل ارسال می‌شوند. اما در پشت سر ترمینال، یک مفسر[7] وجود دارد. این مفسر است که دستورها را ترجمه و برای سیستم عامل قابل فهم می‌سازد. وقتی دستوری را تایپ کرده و کلید اینتر را می‌زنید، ترمینال دستور شما را دریافت، ترجمه و به سیستم عامل تحویل

6. shell
7. command-line interpreter

می‌دهد و سپس پاسخ سیستم عامل و نتیجه‌ی اجرای دستور را برای ترمینال تفسیر کرده و به فرمتِ قابل نمایش برای کاربر تبدیل می‌کند.

در مورد اسکریپت‌ها نیز همینطور است. دستورات اسکریپت، توسط مفسر ترجمه و تفسیر شده و سپس برای اجرا به سیستم عامل تحویل داده می‌شوند. در یک مقایسه ساده شاید بتوان مفسر خط فرمان را مشابه کامپایلرها در محیط‌های برنامه نویسی در نظر گرفت.(این دو به هیچ وجه یکسان نیستند)

مفسر خط فرمان را پوسته نیز می‌نامند. زیرا مانند یک پوسته بین سیستم عامل و ترمینال قرار گرفته است. برای سیستم‌های لینوکسی، پوسته‌های مختلفی توسعه داده شده‌اند. از آن جمله می‌توان به موارد زیر اشاره کرد:

پوسته‌ی بورن[8]: این پوسته با نام sh شناخته می‌شود. قدیمی ترین پوسته‌ی لینوکسی است و توسط آقای اِستفان بورن در آزمایشگاه‌های بِل نوشته شده است. این پوسته فاقد ویژگی‌های تعاملی و ساختارهای پیچیده‌ی برنامه نویسی است. اما در طی سالیان تبدیل به استانداردی در این زمینه شده است.

پوسته‌ی بَش[9]: این پوسته که نسخه ی توسعه یافته‌ی پوسته‌ی بورن است، بعنوان پوسته‌ی پیش‌فرض در اکثر توزیع‌های لینوکس عرضه می‌شود. از پروژه‌ی گنو با لینوکس همراه شده و بصورت متن باز ارائه می‌شود. شباهت‌های بسیاری نیز با پوسته‌ی کورن دارد.

پوسته‌ی C و هم خانواده‌هایش : توسط آقای بیل جوی در Berkeley UNIX ایجاد شده است و پس از بَش و کورن سومین پوسته‌ی پرکاربرد می‌باشد.

پوسته‌ی کورن : توسط آقای دیوید کورن نوشته شده و پوسته‌ی پیش‌فرض در سیستم‌های لینوکسی تجاری است.

4-3- اولین اسکریپت

8. Bourne Shell
9. Bourne Again Shell (BASH)

حال زمان آن فرا رسیده تا ابزارهایی را که در بخش‌های قبل با آن‌ها آشنا شدید، بکار ببندید. در ادامه با نحوه‌ی ایجاد و اجرای یک اسکریپت ساده آشنا خواهید شد. روند کلی کار به این صورت است که ابتدا در یک ویرایشگر، اسکریپت مورد نظر را نوشته و ذخیره می‌نماییم. سپس مراحل لازم برای تفسیر و اجرا را انجام خواهیم داد. جزئیات بیشتر را در ادامه ببینید:

ابتدا ویرایشگر متنی مورد علاقه خود را باز کرده و سطرهای زیر را در آن تایپ نمایید.

```
#!/bin/sh
echo "Hello World! "
exit 0
```

قبل از رفتن به سراغ مرحله بعد، ببینیم هر یک از این سطرها چه معنایی دارند. با سطر آخر شروع کنیم! در این سطر دستور exit بامقدار 0 بکار رفته است. دستور exit برای پایان دادن به اسکریپت است و مقدار 0 هم به سیستم اعلام می‌کند که اسکریپت با موفقیت به پایان رسیده است.

در سطر دوم، یکی از دستورات پوسته به نام echo استفاده شده است. این دستور برای چاپ نتایج در خروجی بکار می‌رود. همانطور که در مثال بالا می‌بینید، برای چاپ پیغام در خروجی، پس از دستور echo با یک فاصله پیغام مورد نظر را داخل علامت نقل قول می‌آوریم. توضیح کامل این دستور و بقیه دستورات را در فصل سوم خواهید دید.

```
echo    "پیغام مورد نظر"
```

تا اینجا متوجه شده‌اید که با اجرای این اسکریپت، فقط یک پیغام در خروجی چاپ خواهد شد. حال برگردیم سراغ سطر اول که با کاراکتر هَش(#) و علامت تعجب(!)[10] شروع شده است. در این سطر، مسیر پوسته‌ای را مشخص می‌کنیم که اسکریپت را تفسیر و اجرا خواهد کرد.

10 . علامت # به انگلیسی، شارپ و علامت !، بَنگ خوانده می‌شود. بنابراین ترکیب این دو کاراکتر را به اختصار شابنگ(shabang) می‌گویند.

علامت !# به مفسر اعلام می‌کند که این سطر چه وظیفه‌ای بر عهده دارد و مسیری که در ادامه نوشته می‌شود، محل قرارگیری پوسته‌ی مورد نظر را مشخص می‌سازد. اگر سطر !# را حذف کنید، از پوسته‌ی پیش‌فرض سیستم استفاده خواهد شد.

برویم سراغ مرحله دوم از ایجاد اسکریپت. فایل ایجاد شده را با نام دلخواه و با پسوند sh. ذخیره نمایید. برای تخصیص پسوند به فایل‌ها در لینوکس کافیست در انتهای نام فایل، یک نقطه و سپس پسوند مورد نظر خود را تایپ کنید. البته توجه داشته باشید که لینوکس برای تشخیص نوع فایل‌ها از پسوند استفاده نمی‌کند! بلکه بدین منظور از مکانیزم‌های متفاوتی بهره می‌برد. تخصیص پسوند در اینجا، بیشتر برای راحتی کاربر در تشخیص نوع فایل‌ها می‌باشد.

تصویر 4-5- ایجاد اولین فایل اسکریپت با نام hello.sh

4-3-1-اضافه کردن کامنت

برای اضافه کردن توضیحات به کدها، از یک علامت # در ابتدای سطر استفاده می‌شود. در مثال زیر، اضافه شدن کامنت به مثال قبل را می‌بینید:

```
# این سطر کامنت است و از دید مفسر پنهان خواهد ماند.
#!/bin/sh
echo "Hello World! "
exit 0
```

نکته: علامت # کل یک سطر را به حالت کامنت در می‌آورد. برای تبدیل سطر بعدی به کامنت، باید از یک علامت # دیگر استفاده کرد.

```
# این سطر کامنت است و از دید مفسر پنهان خواهد ماند.
```

```
# این سطر نیز یک کامنت دیگر است.
#!/bin/sh
echo "Hello World! "
exit 0
```

4-3-2-اجرای اسکریپت

پس از نوشتن یک برنامه مثلا به زبان c++، آن را کامپایل می‌کنید و سپس فایل اجرایی با پسوند exe. قابل اجرا بر روی سیستم خواهد بود. همانطور که می دانید در ویندوز، فایل‌های اجرایی عمدتا پسوند exe. دارند اما دیدید که در لینوکس پسوندها خیلی مورد توجه سیستم عامل نیستند. بنابراین برای اجرای اسکریپت، روش متفاوتی وجود دارد. برای اینکه یک اسکریپت تبدیل به یک فایل اجرایی شود، کافیست به آن مجوز اجرا بدهید. بعنوان مثال اسکریپتی که در ابتدای همین بخش با نام hello.sh نوشتیم را در نظر بگیرید. همانطور که در ادامه خواهید دید، ابتدا به این فایل مجوزهای لازم برای اجرا را تخصیص می دهیم و سپس اسکریپت را در خط فرمان اجرا و نتیجه آن را مشاهده خواهیم کرد.

تخصیص مجوز اجرا: (توجه: فایل در مسیر جاری قرار دارد، بنابراین تنها ذکر نام فایل کافی خواهد بود)

```
chmod   755   hello.sh
```

نکته 1: اینکه به کاربران مجوز خواندن و نوشتن هم تخصیص بدهید یا نه به شما بستگی دارد اما به این نکته توجه کنید که حتما مجوز اجرا (برابر با عدد 1) به کاربر مورد نظر شما تخصیص داده شود. در مثال بالا، برای تمام کاربران مجوز اجرا در نظر گرفته شده است. (مالک: تمام مجوزها = 7، هم‌گروهی‌ها: خواندن و اجرا = 5، سایرین: خواندن و اجرا = 5)

نکته 2: ممکن است بخواهید در مورد یک اسکریپت خاص، سایر کاربران(افرادی غیر از کاربر مالک و هم‌گروهی‌هایش) امکان اجرای اسکریپت را نداشته باشند. در آن صورت مقدار مجوز، بجای 755 بصورت 750 خواهد بود.

حال که مجوزهای لازم را به فایل اسکریپت تخصیص دادیم، می‌توانیم آن را اجرا نماییم. برای اجرای اسکریپت دو حالت وجود دارد :

الف) اگر فایل اسکریپت در دایرکتوری جاری باشد، آنگاه برای اجرای اسکریپت کافیست نام اسکریپت را بعد از علامت ./ بیاوریم:

```
./hello.sh
```

ب) فایل اسکریپت در مسیری متفاوت از دایرکتوری جاری باشد، باید آدرس کامل و نام فایل را ذکر نماییم. که در مورد مثال ما، اسکریپت در پوشه‌ی روت قرار دارد.

```
/root/hello.sh
```

تمرین کلاسی:

- یک اسکریپت نوشته و اجرا کنید که نام و شماره دانشجویی خودتان را در خروجی چاپ کند.

منابع و مراجع:

- Cooper M. Advanced Bash Scripting Guide: An In-depth Exploration of the Art of Shell Scripting. editor no identificat; 2014.
- Matthew N, Stones R. Beginning linux programming. chapter 2. John Wiley & Sons; 2008.
- Soltani A. Scripting in Linux, simply write Linux Scripts. Naghoos; 2018
- Bash Shell Scripting. Wiki Books. reached 2024, Aug.

آزمایش 5

دستورات تکمیلی در برنامه‌نویسی پوسته

اهداف:

- آشنایی تکمیلی با برنامه نویسی پوسته

ابزار و مفاهیم مورد نیاز:

- CLI
- ابزارهای ویرایش متن
- انواع پوسته

محتوا:

- دستورات تکمیلی برنامه نویسی پوسته
- مرور مفاهیم جلسه گذشته
- متغیرها
- عملگرهای منطقی
- دستورات شرطی
- حلقه‌های تکرار
- متغیرهای خاص

مفاهیم و شرح دستورکار

5-1-مرور مفاهیم پایه

در جلسه قبل دیدیم که با استفاده از برنامه نویسی پوسته، می توان برنامه های نسبتا کوچک و سبکی نوشت که توسط خود ترمینال اجرا می‌شوند ونیاز به مفسر مجزا ندارند. معمولا ازبرنامه نویسی پوسته برای خودکارسازی کارهای روتین، گزارش خطاها و مانیتورینگ سیستم، همگام‌سازی و ... استفاده می شود.

چه زمانی نباید از برنامه نویسی پوسته استفاده کرد؟ برای برنامه های سنگین که نیازمند منابع یا عملیات ریاضی زیادی هستند، برنامه های پیچیده و برنامه های گرافیکی.

اسکریپت‌های پوسته مختص استارت آپ، در مسیر /etc/rc.d قرار دارند و حاوی تنظیمات سیستم و سرویس‌های تنظیمات می‌باشند.

5-2- متغیرها

در بَش نیازی به اعلان متغیرها نمی‌باشد. هر گاه در یک اسکریپت، برای اولین بار از یک متغیر استفاده کنید یا آن را مقداردهی نمایید، متغیر بطور خودکار اعلان می‌شود و در استفاده‌های بعدی بدون مشکل شناسایی خواهد شد.

5-2-1-مقداردهی به متغیر

```
نام متغیر  =  مقدار مورد نظر برای متغیر
```

مثال :

```
x=2
f=3.14
```

```
name='Ali'
lname="Irani"
```

نکته: مقادیر رشته ای و کاراکتری، باید داخل کوتیشین یکانی(' ') یا جفت کوتیشن (" ") قرار بگیرند.

چاپ در خروجی:

برای چاپ پیغام‌ها و متغیرها در خروجی، از دستور echo استفاده می‌شود. مثال زیر یک پیغام برای کاربر چاپ می‌کند:

```
echo    "پیغام مورد نظر"
```

توجه : هر آنچه که در بین دو علامت نقل قول بنویسید عینا در خروجی چاپ خواهد شد.

چاپ متغیرها

برای چاپ مقدار متغیرها در خروجی از اشاره گرها استفاده می‌کنیم تا بتوانیم به مقدار داخل متغیر دسترسی پیدا کنیم. برای اشاره به مقدار یک متغیر، در ابتدای نام آن متغیر یک علامت $ قرار بدهید.

مثال:

```
echo   $x
```

چاپ ترکیبی پیغام و متغیر در یک دستور:

مثال :

```
چاپ نام  و مقدار متغیر #
#!/bin/bash
x=2
echo    " x = $x "
exit 0
```

همانطور که در مثال بالا می‌بینید، عبارتی که در مقابل دستور echo نوشته شده، داخل کوتیشن قرار گرفته است. بنابراین انتظار داریم که عبارت عینا چاپ شود. در مورد x اول این اتفاق می افتد اما در مورد x دوم بعلت وجود علامت $، مقدار متغیر x جایگزین شده و چاپ می‌شود. خروجی این اسکریپت بصورت زیر خواهد بود:

```
x = 2
```

5-2-2- دریافت از ورودی

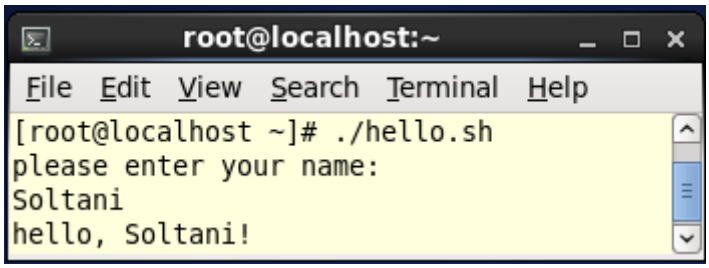
برای دریافت مقادیر متغیرها از کاربر، از دستور read استفاده می‌شود.

```
read    نام متغیر
```

مثال : اسکریپتی بنویسید که با دریافت نام کاربر، یک پیغام به نام وی چاپ نماید.

```
#!/bin/bash
یادگیری متغیرها #
echo "please enter your name: "
با چاپ یک پیام از کاربر میخواهیم که نام خود را وارد نماید #
read name
آنچه که کاربر تایپ کرده را دریافت و در متغیری به نام name قرار می دهیم #
echo "hello, $name!"
پیغام "سلام، نام کاربر" را چاپ می‌کنیم #
exit 0
```

در تصویر زیر نتیجه‌ی اجرای اسکریپت بالا را مشاهده می‌کنید.



تصویر 5-1- دریافت متغیر از ورودی

دریافت دو یا چند متغیر با یک دستور

```
read   x   y
```

نکته: کاربر پس از تایپ مقدار اول معمولا یک کاراکتر فاصله یا کلید اینتر را می‌زند و سپس مقدار دوم را وارد می‌نماید.

5-2-3- محاسبات ریاضی بر روی متغیرها

در بَش برای انجام محاسبات ریاضی از چند روش استفاده می‌شود. حالت اول، استفاده از let است.

```
let x=x+1
```

نکته: در این حالت قبل و بعد از علامت + نباید از کاراکتر فاصله استفاده کرد.

روش دوم استفاده از جفت پرانتز است. در این حالت وجود کاراکترهای فاصله‌ی اضافی، موجب بروز خطا نخواهد شد!

```
((x=5))
((x=x+1))
```

عملگرهای ریاضی تعریف شده برای بش، در جدول زیر آمده است:

علامت عملگر	نام عملگر	علامت عملگر	نام عملگر
+	جمع	*	ضرب
-	تفریق	/	تقسیم
++	افزاینده	%	باقیمانده تقسیم
--	کاهنده	**	توان

جدول 5-1- عملگرهای ریاضی

5-3- ساختار شرطی If

برای تست برقراری شرط می‌توان از ساختار If استفاده کرد. شکل ساده این دستور به صورت زیر می‌باشد.

```
if    [    عبارت شرطی    ]
then
      دستورات 1
else
      دستورات 2
fi
```

نکته ها:

1. در صورتیکه عبارت شرطی صحیح باشد، دستورات 1 و در غیر اینصورت، دستورات 2 اجرا خواهند شد.
2. وجود بخشelse الزامی نیست.
3. برای مشخص کردن بلوک دستورات از کلمه کلیدی then استفاده می‌شود.
4. عبارت fi پایان دهنده‌ی ساختار if است.

قبل از اینکه به سراغ یک مثال در این زمینه برویم، توجه داشته باشید که عملگرهای مقایسه‌ای درعبارات شرطی کمی متفاوت از عملگرهای متداول ریاضی است. مثلا در مورد مقایسه عبارات حسابی از عملگرهای زیر استفاده می‌شود.

عملگرهای مقایسه‌ای عبارات حسابی	
تساوی	e1 -eq e2
عدم تساوی	e1 -ne e2
بزرگتر	e1 -gt e2
بزرگتر مساوی	e1 -ge e2
کوچکتر	e1 -lt e2
کوچکتر مساوی	e1 -le e2

جدول 5-2- مقایسه دو عبارت حسابی e1 و e2

همانطور که در جدول بالا مشاهده می‌کنید، برای مقایسه تساوی دو عبارت حسابی با نام‌های e1 و e2 از عملگر eq– استفاده می‌شود. اگر e1 و e2 مساوی باشند نتیجه‌ی این مقایسه True و در غیر اینصورت False است.

مثال : اسکریپتی بنویسید که بر اساس معدل، کلمه pass یا fail را چاپ نماید.

```
#!/bin/sh
رد یا قبول؟ #
echo "please enter the average: "
read ave
if [ $ave -ge 10 ]
then
```

```
  echo "pass"
else
  echo "fail"
fi
exit 0
```

نکته: وجود یک کاراکتر فاصله بین هر برکت و عبارت شرطی الزامی است و در صورتیکه رعایت نشود، با خطای مفسر مواجه خواهید شد.

5-3-1-ساختار شرطی if ... elif

هرگاه بخواهیم چند شرط را تست نماییم، از این ساختار به شکل زیر استفاده می‌کنیم.

```
if    [    عبارت شرطی 1     ]
then
      دستورات 1
elif    [    عبارت شرطی 2    ]
then
      دستورات 2
else
      دستورات 3
fi
```

مثال : اسکریپتی بنویسید که یک عدد را از ورودی دریافت و مثبت یا منفی بودن آن را بررسی نماید.

```
#!/bin/sh
# مثبت یا منفی؟
echo "please enter a number: "
# با چاپ یک پیام از کاربر می خواهیم یک عدد وارد کند
read x
# عدد را دریافت و در متغیر x قرار می‌دهیم

if [ $x -gt 0 ]
# اگر x بزرگتر از 0 باشد، آنگاه
then
  echo "$x is positive!"
# چاپ پیام مبنی بر اینکه عدد مثبت است

elif [ $x -lt 0 ]
# اگر x کوچکتر از 0 باشد، آنگاه
then
  echo "$x is negative!"
# چاپ پیام مبنی بر اینکه عدد منفی است

else
# در غیر اینصورت عدد برابر 0 است
  echo "It's a zero!"

fi    # پایان ایف
exit 0
```

نکته: در صورتیکه بخواهیم چند حالت شرطی را بررسی نماییم، می‌توانیم به تعداد مورد نیاز از elifهای بیشتری استفاده کنیم.

5-4- عملگرهای منطقی :

ترکیب عبارت‌ها یا شرط‌ها با استفاده از عملگرهای منطقی قابل انجام است. از زوج کاراکتر || بعنوان عملگر OR و از && بعنوان عملگر AND استفاده می‌شود.

مثال : اسکریپتی بنویسید که با دریافت دو عدد تشخیص بدهد که حاصل ضربشان مثبت است یا منفی.

```
#!/bin/bash
# بررسی مثبت یا منفی بودن حاصل‌ضرب دو عدد
echo "insert two integers: "
read x y

if [ $x -lt 0 ] && [ $y -lt 0 ]
then
  echo "the product of $x and $y is  positive!"

elif [ $x -eq 0 ] || [ $y -eq 0 ]
then
  echo "the product of $x and $y is  zero!"

elif [ $x -gt 0 ] && [ $y -gt 0 ]
then
  echo "the product of $x and $y is  positive!"

else
  echo "the product of $x and $y is negative!"
fi
exit 0
```

مقایسه رشته ها

عملگرهای مقایسه‌ای رشته‌ها	
string1 = string2	تساوی
string1 != string2	عدم تساوی
-n string	Null نبودن رشته
-z string	Null بودن رشته(رشته‌ی خالی)

جدول 5-3- عملگرهای مقایسه‌ای رشته‌ها

مثال : اسکریپتی بنویسید که پاسخ کاربر به یک سوال را دریافت و سپس بررسی کند که پاسخ مثبت است یا منفی.

```
#!/bin/bash
echo "do u want to continue?(y/n)"
# پرسش از کاربر و درخواست از وی برای وارد کردن کاراکتر y یا n
read ans
# دریافت و قرار دادن پاسخ کاربر در متغیری به نام ans
if [ $ans = "y" ]
# در کنار برکت‌ها باید فاصله قرار بدهیم.
then
echo "let's go"
elif [ $ans = "n" ]
then
echo "see you later"
else
# اگر کاربر کاراکتری غیر از y یا n وارد کند آنگاه:
echo "please type y/n!"
fi
exit 0
```

نکته: یکبار دیگر اسکریپت را اجرا کنید. این بار در پاسخ به سوال، بدون اینکه کاراکتری را وارد کنید فقط کلید اینتر را بزنید(پاسخ پوچ). چه اتفاقی می افتد؟

علت بروز خطا این است که وقتی کاربر هیچ کاراکتری را وارد نمی‌کند، برنامه باید متغیری با مقدار پوچ را با یک کاراکتر (y/n) مقایسه کند و بنابراین دچار مشکل می‌شود. برای حل این مشکل، در عبارات شرطی متغییرans را داخل کوتیشین قرار بدهید.("$ans")

5-5-حلقه تکرار for

```
for     متغیر      in     مقادیر
do
      دستورات
done
```

ساختار حلقه‌ی for کمی متفاوت از ساختارهای مشابه در زبان‌هایی مانند c++ به نظر می‌رسد. در این ساختار، دستور for به ازای هر یک از مقدارهایی که در بخش مقادیر برایش ذکر شود حلقه را تکرار خواهد کرد.

مثال: با استفاده از حلقه for اعداد 1 تا 5 را در خروجی چاپ نمایید.

```
#!/bin/bash
#  for اسکریپت چاپ اعداد متوالی با استفاده از
for   x   in   1  2  3  4  5
do
#شروع دستورات حلقه
echo $x
#پایان دستورات حلقه
done    #پایان حلقه
exit 0
```

در هربار تکرارِ حلقه، به ترتیب یکی از مقادیر، درون متغیر قرار می‌گیرد و دستورات حلقه به ازای آن مقدار، اجرا می‌شوند. در تکرار بعدی، مقدار بعدی و همین طور تا آخرین مقدار، حلقه تکرار خواهد شد. در مثال بالا نیز، ابتدا مقدار 1 درون متغیر x قرار گرفته و چاپ شده، سپس مقدار 2 و الی آخر.

شاید این ساختار در نگاه اول خیلی سودمند به نظر نرسد، اما یک راهکار بسیار جالب در اسکریپت نویسی برای فایل‌ها و دایرکتوری‌هاست که در بخش‌های بعدی به آن خواهیم پرداخت.

6-5-حلقه تکرار while

```
while    [    شرط    ]
do
دستورات
done
```

مثال: اسکریپتی بنویسید که با دریافت پسورد از کاربر آن را با پسورد صحیح (در اینجا secret) مقایسه نماید و تا دریافت پسورد صحیح، تکرار حلقه ادامه یابد.

```
#!/bin/sh
echo "Enter password: "
read pass
                                    دریافت رشته از کاربر و قراردادن در متغیری به اسم ans #
while [ "$pass" != "secret" ]
تا زمانیکه ورودی کاربر با پسورد صحیح یکسان نباشد حلقه تکرار می‌شود #
do
شروع دستورات حلقه #
  echo "Sorry, try again"
  read pass
پایان دستورات حلقه #
```

```
done    #پایان حلقه
exit 0
```

حال که استفاده از حلقه‌ها و حالات شرطی را یاد گرفتید، در مثال بعدی این دو را با هم ترکیب خواهیم کرد.

مثال : مثال "دریافت پاسخ از کاربر" را طوری تغییر بدهید که تا زمانیکه کاربر یکی از کاراکترهای مجاز (y/n) را وارد نکرده، برنامه تکرار شده و از کاربر درخواست پاسخ بکند.

```
#!/bin/bash
#درخواست تایپ پاسخ صحیح از کاربر
x=1
while [ $x -eq 1 ]
do     #شروع دستورات حلقه‌ی وایل
  echo " do u want to continue?(y/n)"
  read ans
  if [ "$ans" = "y" ]    #شروع بخش شرطی
  then
    echo "let's go!"
    x=0
  elif [ "$ans" = "n" ]
  then
    echo "see you later!"
    x=0
  else
    echo "please type y/n!"
  fi     #پایان بخش شرطی
done   #پایان حلقه‌ی وایل
exit 0
```

تمرین کلاسی:

- اسکریپتی بنویسید که آیتم‌های داخل یک دایرکتوری دلخواه را چک کرده و مشخص کند که هر آیتم فایل است یا دایرکتوری.

گزارش کار:

- درباره Cron Jobs تحقیق کرده و سپس یک Cron Job تنظیم کنید تا اسکریپت دلخواهتان را در زمان‌بندی مورد نظر شما اجرا کند.

منابع و مراجع:

- Cooper M. Advanced Bash Scripting Guide: An In-depth Exploration of the Art of Shell Scripting. editor no identificat; 2014.
- Matthew N, Stones R. Beginning linux programming. chapter 2. John Wiley & Sons; 2008.
- Soltani A. Scripting in Linux, simply write Linux Scripts. Naghoos; 2018
- Bash Shell Scripting. Wiki Books. reached 2024, Aug.

آزمایش 6

فرآیندها و رشته‌ها

اهداف:

- درک مفاهیم چند رشته‌ای
- multithreading با استفاده از pthread

ابزار و مفاهیم مورد نیاز:

- کامپیوتر دارای IDE و یا ابزارهای خط فرمانی کامپایل

محتوا:

- تعریف و اهمیت چند رشته‌ای
- آشنایی با Pthreads
- کتابخانه رشته‌های POSIX (pthreads)
- توابع pthread رایج :
  - pthread_create
  - pthread_join
  - pthread_exit

6-1- ایجاد کردن و از بین بردن رشته‌ها

واضح است که اولین مرحله مورد نیاز برای درک چگونگی ساخت یک برنامه چند رشته‌ای، درک چگونگی ایجاد و از بین بردن رشته ها است. معمولاً علاوه بر ایجاد یک رشته، می‌خواهیم پارامترهایی نیز به آن رشته ارسال کنیم. بعلاوه ، هنگامی که رشته به پایان می رسد، معمولاً می‌خواهیم مقادیری که از رشته بازگردانده شده است را بازیابی کنیم. نحوه انجام این کارها را در این بخش خواهیم دید.

6-1-1- ایجاد رشته‌ها

برای ایجاد یک رشته جدید باید از تابع pthread_create() استفاده کنید. کد مثال 1، با نام listing1، فرم کلی یک برنامه برای ایجاد یک رشته را نشان می‌دهد که هیچ کاری انجام نمی دهد و سپس منتظر می ماند تا thread خاتمه یابد.

Listing 1:

```
#include <pthread . h>

/*
* The function to be executed by the thread should take a
* void* parameter and return a void* result.
*/

void_thread *function (void_*arg )
{
// Cast the parameter into whatever type is appropriate.
int *incoming = (int *) arg ;

// Do whatever is necessary using *incoming as the argument.
// The thread terminates when this function returns.
```

```
return NULL;
}

int main (void)
{
pthread_t thread_ID ;
void  *thread_result ;
int value ;

// Put something meaningful into value.
value = 42 ;

// Create the thread, passing &value for the argument.
pthread create (&thread_ID , NULL, thread_function , &value ) ;

// The main program continues while the thread executes.

// Wait for the thread to terminate.
pthread_join ( thread_ID , &thread_result ) ;

// Only the main thread is running now.
return 0 ;
}
```

تابع pthread_create() یک شناسه thread را برمی‌گرداند که در فراخوانی‌های دیگر قابل استفاده است. پارامتر دوم یک اشاره گر به یک thread attribute object است که می توانید از آن برای تنظیم ویژگی‌های رشته استفاده کنید. اشاره گر تهی به معنای استفاده از ویژگی های پیش فرض است که برای بسیاری از موارد مناسب است. سومین پارامتر یک اشاره گر به تابعی است که thread باید اجرا کند. پارامتر آخر، آرگومان ارسال شده به تابع thread است. با استفاده از نشانگرهای void، به طور بالقوه هر نوع داده‌ای را می‌توان اختصاص داد، مشروط بر اینکه castهای مناسب اعمال شوند.

در مثال بعد، نشان می‌دهیم که چگونه یک عدد صحیح می‌تواند به عنوان یک آرگومان رشته استفاده شود، اما در عمل ممکن است یک اشاره‌گر به ساختاری حاوی چندین آرگومان، به رشته ارسال شود.

در نقطه‌ای از برنامه‌تان باید منتظر بمانید تا هر رشته پایان یابد و با فراخوانی ()pthread_join نتیجه‌ای را که تولید می‌کند جمع‌آوری کنید. یا می توانید یک رشته جدا شده(detached) ایجاد کنید. نتایج بازگردانده شده توسط چنین موضوعاتی دور ریخته می شوند. مشکل رشته‌های جدا شده این است که هرگز از زمان تکمیل شدن آنها مطمئن نیستید، مگر اینکه ترتیبات خاصی انجام دهید. معمولاً می‌خواهید قبل از اینکه فرآیند با بازگشت نتیجه(return) از ()main پایان یابد، مطمئن شوید که تمام رشته‌های شما کاملاً خاتمه یافته‌اند. وقتی در تابع ()main خط return اجرا می‌شود، هر رشته در حال اجرا به طور ناگهانی قطع می‌گردد. گرچه ممکن است این کار در برخی موارد مناسب باشد، اما این خطر را دارد که کارهای مهمی که توسط یک رشته انجام در حال انجام بوده‌اند تکمیل نشده باشند.

اگر می خواهید یک رشته را قبل از اینکه تابع thread به طور معمول بازگردد بکشید، می توانید از ()pthread_cancel استفاده کنید. با این حال، مشکلاتی در انجام این کار وجود دارد.

باید مطمئن شوید که رشته قبل از اینکه واقعاً بمیرد، منابعی را که به دست آورده است، آزاد کرده است. به عنوان مثال اگر یک رشته به صورت پویا حافظه را تخصیص داده باشد و قبل از اینکه بتواند آن حافظه را آزاد کند آن را لغو کنید، برنامه شما دچار نشت حافظه خواهد شد. این متفاوت از زمانی است که یک فرآیند کامل را بکشید. سیستم عامل معمولاً منابع (خاص) معلقی را که توسط فرآیند معلق مانده اند پاک می کند. به ویژه، کل فضای آدرس یک فرآیند بازیابی می شود.

اما سیستم عامل این کار را برای یک رشته انجام نمی دهد، زیرا رشته‌های یک فرآیند، از منابع به صورت اشتراکی استفاده می‌کنند. سیستم عامل می‌داند که حافظه اختصاص داده شده توسط یک رشته توسط رشته دیگر نیز استفاده خواهد شد. به همین علت، لغو صحیح رشته‌ها دشوار می‌شود.

6-1-2- برگرداندن نتایج از رشته‌ها

در مثال بخش قبل نشان دادیم که چگونه می توانید در صورت لزوم یک آرگومان را به تابع رشته خود منتقل کنید. در این بخش نحوه برگرداندن نتایج از توابع رشته را خواهیم دید.

توجه داشته باشید که توابع thread طوری اعلان شده‌اند که یک اشاره گر به void برمی‌گردانند. با این حال، برخی از مشکلات در استفاده از این اشاره گر وجود دارد. کد زیر تلاش میکند یک کد وضعیت عدد صحیح از یک تابع رشته را بازگرداند.

```
void    *thread function (void *)
{
int code = DEFAULT_VALUE;
// Set the value of 'code ' as appropriate.
return (void *) code;
}
```

این روش فقط روی ماشین‌هایی کار می‌کند که درآن‌ها می‌توان اعداد صحیح را به یک اشاره‌گر تبدیل کرد و سپس بدون از دست دادن اطلاعات به یک عدد صحیح برگشت. در برخی از ماشین‌ها، چنین تبدیل‌هایی خطرناک هستند. در واقع این روش در تمام مواردی که فرد تلاش می کند یک شی، مانند ساختاری را که بزرگتر از یک اشاره گر است، برگرداند، شکست خواهد خورد.

در مقابل، کد زیر مشکلی با سیستم انواع داده ندارد. و یک اشاره گر به بافر داخلی، که مقدار بازگشتی در آن ذخیره شده است، برمی گرداند. در حالی که در مثال، آرایه‌ای از کاراکترها برای بافر نشان داده می‌شود، می‌توان به راحتی آن را آرایه‌ای از هر نوع ضروری یا یک شی منفرد مانند یک کد وضعیت عدد صحیح یا ساختاری با اعضای متعدد تصور کرد.

```
void *thread_function ( void *)
{
char  buffer [ 64 ] ;
```

```
// Fill up the buffer with something good.
return       buffer;
}
```

متاسفانه کد بالا شکست می‌خورد زیرا بافر داخلی خودکار است و به محض بازگشت مقدار از تابع رشته، ناپدید می‌شود. اشاره‌گر که به رشته فراخواننده بازپس داده می‌شود، به حافظه تعریف نشده اشاره خواهد کرد. که مثال دیگری از خطای نشانگر معلق کلاسیک است.

در مثال بعدی، بافر ایستا می شود به طوری که حتی پس از پایان تابع thread به وجود خود ادامه می دهد. این کار مشکل نشانگر معلق را حل می‌کند.

```
void  *thread_function ( void *)
{
static     char   buffer [ 64 ] ;
// Fill up the buffer with something good.
return     buffer ;
}
```

این روش ممکن است در برخی موارد رضایت بخش باشد، اما در حالت معمولی که چندین رشته یک تابع رشته را اجرا می‌کنند، کار نمی کند. در چنین شرایطی رشته دوم بافر استاتیک را با داده های خود بازنویسی می کند و داده های باقی مانده از رشته اول را از بین می برد. داده های عمومی از همین مشکل رنج می برند زیرا همیشه طول زمانی ثابتی دارند.

تمرین کلاسی:

- برنامه ای بنویسید که 10 رشته ایجاد کند. از هر رشته بخواهید یک تابع مشابه را اجرا کند و به هر رشته یک عدد منحصر به فرد ارسال کند. هر رشته باید سه بار "Hello, World (thread n)" را چاپ کند که در آن n با شماره رشته جایگزین شود.
- از آرایه ای از اشیاء pthread_t برای نگهداری شناسه‌های رشته‌های مختلف استفاده کنید. مطمئن شوید که برنامه تا زمانی که تمام رشته ها کامل نشده‌اند، خاتمه نمی‌یابد.

منابع و مراجع:

- Love R. Linux system programming: talking directly to the kernel and C library. " O'Reilly Media, Inc."; 2007 Sep 18.
- Wall K. Linux programming unleashed. Sams; 2000 Dec 1.

آزمایش 7

سمافور

اهداف:

- درک مفاهیم بن بست و سمافور

ابزار و مفاهیم مورد نیاز:

- کامپیوتر دارای IDE و یا ابزارهای خط فرمانی کامپایل

محتوا:

- سمافور
- mutexes
- condition variables
- مساله تولید کننده-مصرف کننده

مفاهیم و شرح دستورکار

7-1-سمافور

Semaphore اساساً شمارنده های عدد صحیح glorified هستند. آنها از دو عملیات اصلی پشتیبانی می کنند. یک عملیات، called down یا wait ، تلاش می کند شمارنده را کاهش دهد. عملیات دیگر، called up، signal ، یا post ، تلاش برای افزایش شمارنده است. چیزی که سمافورها را خاص می کند این است که اگر یک رشته سعی کند روی سمافور صفر منتظر بماند، در عوض مسدود می شود. بعداً وقتی رشته دیگری سمافور را پست می‌کند، رشته مسدود شده فعال می‌شود در حالی که سمافور روی صفر باقی می‌ماند. در واقع، ارسال باعث می شود که سمافور افزایش یابد اما سپس رشته مسدود شده‌ای که در تلاش برای انجام یک انتظار بود، اجازه می‌گیرد تا ادامه یابد و باعث می شود که سمافور بلافاصله دوباره کاهش یابد.

اگر چندین رشته در انتظار روی سمافور مسدود شده باشند، سیستم یکی را برای رفع انسداد انتخاب می کند. اینکه دقیقاً چگونه این انتخاب انجام می شود به طور کلی به سیستم بستگی دارد. نمی توانید فرض کنید که به ترتیب FIFO خواهد بود. با این حال، ترتیبی که تاپیک‌ها بر اساس آن مسدود می شوند، معمولاً نگران کننده نیست. اگر اینطور باشد، برنامه شما ممکن است خیلی خوب طراحی نشده باشد.

سمافور با مقدار اولیه یک، می‌تواند مانند یک mutex استفاده شود. هنگامی که یک رشته بخواهد وارد بخش حیاتی خود شود و به یک ساختار داده مشترک دسترسی پیدا کند، یک عملیات انتظار بر روی سمافور انجام می دهد. اگر رشته دیگری در بخش بحرانی خود نباشد، سمافور مقدار اولیه یک را خواهد داشت و انتظار بلافاصله برمی‌گردد. سپس سمافور صفر خواهد شد. اگر رشته دیگری سعی کند در این مدت روی سمافور منتظر بماند، مسدود خواهد شد. هنگامی که رشته اول اجرای بخش بحرانی خود را پایان می‌دهد، یک عملیات پست روی سمافور به انجام می‌رساند. با این کار یک رشته در انتظار از حالت انسداد خارج می‌شود، یا اگر رشته‌ی منتظری وجود نداشته باشد، سمافور را به مقدار اولیه خود یعنی یک افزایش می‌دهد. سمافور مورد استفاده در این روش سمافور باینری نامیده می شود زیرا دقیقاً دو حالت دارد.

با این حال، چون سمافورها اعداد صحیح هستند، می توانند مقادیر بزرگتر از یک را بگیرند. بنابراین آنها اغلب برای شمارش منابع کمیاب استفاده می شوند. به عنوان مثال، یک رشته ممکن است روی یک سمافور منتظر بماند تا یک آیتم از یک منبع را ذخیره کند. اگر هیچ آیتمی باقی نماند، سمافور صفر می شود و عملیات انتظار مسدود می شود.

هنگامی که یک رشته با استفاده از یک آیتم از یک منبع، پایان می‌یابد، سمافور را ارسال می‌کند تا تعداد آیتم‌های موجود را افزایش دهد یا به یک رشته مسدود شده اجازه دسترسی به آیتم موجود را بدهد. سمافور مورد استفاده در این روش را سمافور شمارشی می نامند.

API سمافور POSIX واقعاً بخشی از API pthread معمولی نیست. بلکه POSIX سمافورها را تحت یک API متفاوت استاندارد می کند. سیستم‌های سنتی یونیکس از حافظه مشترک، صف‌های پیام و سمافورها به عنوان بخشی از آنچه که ارتباطات بین فرآیندی system V نامیده می‌شود(System V IPC) پشتیبانی می‌کنند. POSIX همچنین حافظه مشترک، صف‌های پیام و سمافورها را به‌عنوان بسته‌ای که با استاندارد قدیمی‌تر رقابت می‌کند یا جایگزین آن می‌شود، ارائه می‌کند. عملکرد دو سیستم مشابه است اگرچه جزئیات دو API متفاوت است.

توجه داشته باشید که سمافورهای POSIX، مانند سمافورهای System V IPC، می توانند برای همگام سازی دو یا چند فرآیند مجزا استفاده شوند. این نسبت به mutexes pthread متفاوت است. یک mutex فقط می تواند توسط رشته‌ها در همان فرآیند استفاده شود. از آنجایی که سمافورهای POSIX را می توان برای ارتباطات بین فرآیندی استفاده کرد، آن‌ها می‌توانند نامگذاری شوند. یک فرآیند می تواند یک سمافور را تحت یک نام خاص ایجاد کند و سایر فرآیندها می توانند آن سمافور را با نام باز کنند.

فرم کلی یک برنامه در listing 2، نحوه مقداردهی اولیه، پاکسازی و استفاده از سمافور POSIX را نشان می دهد. این برنامه نه رشته‌های در حال ایجاد یا پیوستن را نشان می‌دهد و نه مدیریت خطا را نشان می‌دهد.

تفاوت دیگر بین pthread mutex و سمافور این است که برخلاف mutex، یک سمافور می‌تواند در یک رشته متفاوت از رشته‌ای که عملیات انتظار را انجام می‌دهد پست شود. این هنگام استفاده از سمافور برای شمارش نمونه های یک منبع کمیاب ضروری است. برنامه listing 2 از سمافور مانند mutex استفاده می کند. این کار برای ساده سازی انجام شده تا توابع مورد استفاده برای دستکاری سمافور واضح باشد.

برای مشاهده استفاده از سمافورها، مثال کلاسیک تولید کننده/مصرف کننده را در نظر بگیرید. در این مساله یک رشته در حال تولید آیتم ها است (مثلاً اشیایی از نوع * void که ممکن است به اشیاء دیگری با پیچیدگی دلخواه اشاره کنند) در حالی که رشته دیگر آن موارد را مصرف می کند. Lisitng 3 یک نوع داده انتزاعی را نشان می دهد که یک بافر را پیاده سازی می کند که می تواند برای نگهداری آیتم ها هنگام عبور از یک رشته به رشته دیگر استفاده شود.

راه حل در واقع به دو سمافور نیاز دارد. یکی برای شمارش تعداد اسلات های آزاد در بافر و دیگری برای شمارش تعداد اسلات‌های استفاده شده.

listing 2: Semaphore Example

```
#include <semaphore.h>
int   shared ;
sem_t  binary_sem ; // Used like a mutex.
void  *thread_function ( void  *arg )
{
sem_wait (&binary_sem ) ; // Decrements count.
// Used shared resource.
sem_post (&binary_sem ) ; // Increments count.
}
void main ( void)
{
sem_init (&binary_sem , 0 , 1 ) ; // Initial count of 1.

// Start threads here.
sem_wait (&binary_sem ) ;

// Use shared resource.
sem_post (&binary_sem ) ;

// Join with threads here.
```

```
sem_destroy(&binary_sem ) ;
return 0 ;
}
```

listing 3: Producer/Consumer Abstract Type

```
#ifndef PCBUFFER_H
#define PCBUFFER_H

#include <pthread. h>
#include <semaphore. h>

#define PCBUFFER_SIZE 8

typedef struct {
 void  *buffer [PCBUFFER_SIZE ] ;
 pthread_mutex_t  lock ;
 sem_t                used ;
 sem_t                free ;
 int             next_in ; // Next available slot.
 int             next_out ; // Oldest used slot.
}pcbuffer_t ;

void pcbuffer_init ( pcbuffer_t *) ;
void pcbuffer_destroy ( pcbuffer_t *) ;
void pcbuffer_push (pcbuffer_t *, void *) ;
void * pcbuffer_pop ( pcbuffer_t *) ;

#endif
```

این امر ضروری است زیرا زمانی که بافر پر است باید تولیدکننده را مسدود کنیم و زمانی که بافر خالی است باید مصرف کننده را مسدود کنیم. با این حال، سمافورها تنها زمانی تماس گیرنده خود را مسدود می کنند که فرد سعی کند آنها را به زیر صفر برساند، و وقتی افزایش می یابند هرگز مسدود نمی شوند.

توابع اولیه و پاکسازی ساده هستند. در مقابل، توابع push و pop بسیار حساس هستند. ابتدا لازم است در هر یک از آن‌ها، یک واحد از منابع محدود رزرو شود. در مورد pcbuffer_push() باید یک اسلات آزاد را رزرو کنیم. اگر هیچ اسلات آزادی در دسترس نباشد، فراخوانی sem_wait (&p->free) مسدود می‌شود تا زمانی که مصرف‌کننده پس از حذف یک مورد، آن سمافور را پست کند.

هنگامی که یک اسلات رزرو شد، بافر را قفل می کنیم تا مطمئن شویم که هیچ رشته دیگری نمی تواند با تغییر همزمان آن را خراب کند. در نهایت، پس از باز کردن قفل، سمافور دیگر را پست می‌کنیم تا در صورت لزوم، رشته دیگر را رفع انسداد کنیم.

به عنوان مثال، فراخوانی sem_signal (&p->used) یک مصرف کننده منتظر را برای رسیدگی به آیتم ذخیره شده در مخزن، مسدود می کند.

گزارش کار:

با استفاده از POSIX mutex و متغیرهای شرطی ، یک سمافور abstract type را پیاده سازی کنید. به عنوان مثال، یک فایل هدر حاوی موارد زیر را در نظر بگیرید.

```
typedef struct {
// Fill in members as appropriate.
} semaphore_t ;
void semaphore_init (
struct semaphore *s , int initial_count ) ;
void semaphore_destroy ( semaphore_t *s ) ;
void semaphore_up ( semaphore_t *s ) ;
void semaphore_down ( semaphore_t *s ) ;
```

توابع اعلان شده در بالا را پیاده‌سازی کنید. این نشان می دهد که خیلی ضروری نیست که سمافورها بخشی از یک API سطح پایین باشند.

منابع و مراجع:

- Love R. Linux system programming: talking directly to the kernel and C library. " O'Reilly Media, Inc."; 2007 Sep 18.
- Wall K. Linux programming unleashed. Sams; 2000 Dec 1.

آزمایش 8

معرفی و دستورات پایه Perf

اهداف:

- یادگیری مفهوم و کاربردهای ابزار Perf
- نصب و تنظیمات اولیه
- استفاده از دستورهای ساده Perf برای جمع آوری اطلاعات کارایی

ابزار و مفاهیم مورد نیاز:

- ابزار Perf نصب شده روی سیستم عامل لینوکس
- برنامه‌های کاربردی نصب شده برای تحلیل کارایی

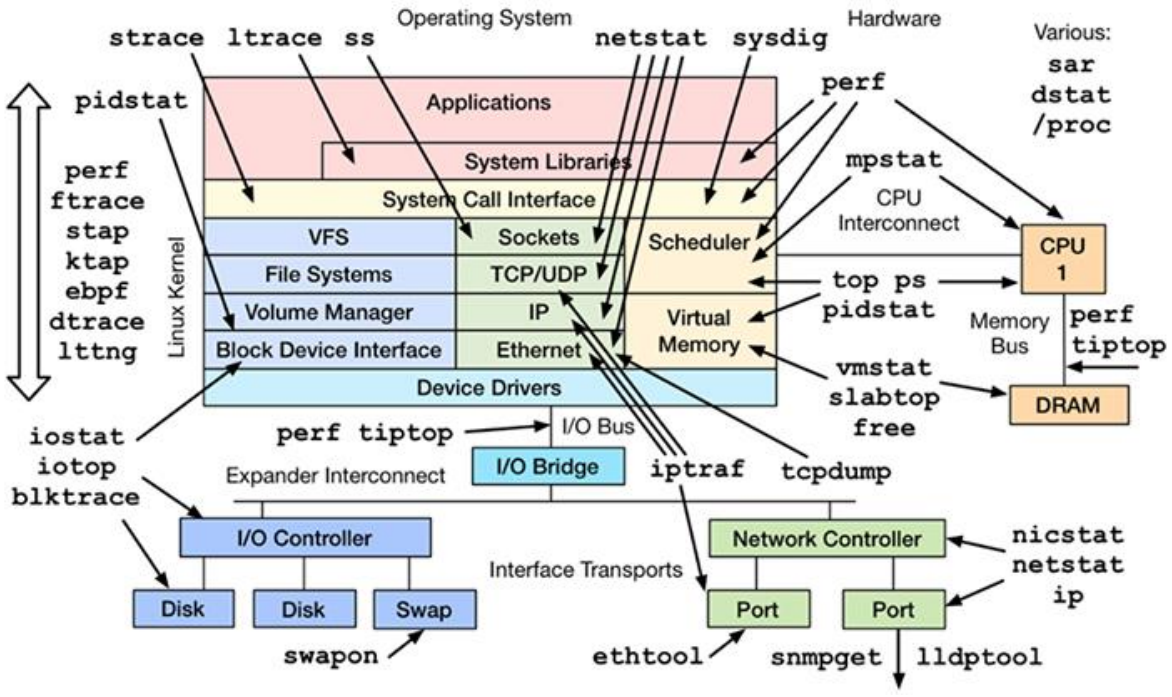
محتوا:

- مقدمه
  - تحلیل کارایی در لینوکس
  - Perf و کاربردها
- نصب :
  - تنظیم مجوزهای کاربران معمولی برای استفاده از ابزار Perf
- دستورات فرعی
- اجرای دستورات پایه:
  - perf stat, perf list, perf record, perf report

مفاهیم و شرح دستورکار

8-1-معرفی:

Perf یک شمارنده کارایی برای لینوکس ((Performance Counters for Linux(PCL) است. یک ابزار تجزیه و تحلیل کارایی که از هسته لینوکس نسخه 2.6.31 از سال 2009 در دسترس است. این ابزار نظارتی ، برای جمع آوری اطلاعات در مورد هسته لینوکس، CPU و سخت افزار مورد استفاده قرار می‌گیرد. بعلاوه از ردیابی پویا و شمارنده‌های کارایی سخت‌افزار، tracepoints، شمارنده‌های کارایی نرم‌افزار (مثلا hrtimer) و Probeهای پویا (مانند kprobes یا uprobes) پشتیبانی می‌کند و می تواند به عنوان واحد نظارت بر عملکرد Performance Monitoring Unit (PMU) در لینوکس استفاده شود. ابزار Perf با زبان C نوشته شده، تحت مجوز GNU GPL منتشر گردیده و با یک رابط خط فرمانی ساده ارائه می‌شود. Perf شامل تعدادی دستور فرعی(Subcommands) می‌باشد و قادر است کل سیستم (هم هسته و هم کدهای سمت کاربر) را به صورت آماری پروفایل کند.



شکل.7-1- ابزارهای مشاهده کارایی لینوکس

پیاده سازی

رابط بین ابزار perf و هسته فقط از یک syscall تشکیل شده است و از طریق یک توصیفگر فایل و یک منطقه حافظه نگاشت شده انجام می شود. به service daemons نیازی ندارد، زیرا بیشتر قابلیت‌هایش در هسته یکپارچه شده‌اند. هنگامی که بافر پر می شود، ابزار perf داده های خام را از بافر map شده، به دیسک تخلیه می کند. پروفایل کردن توسط perf هزینه سربار بسیار کمی دارد.

8-2-نصب:

Perf همراه با سایر بسته‌های متداول لینوکس در مخازن مربوطه عرضه می‌شود. برای نصب این ابزار باید به کاربر روت و یا یک کاربر sudoer دسترسی داشته باشید. مراحل نصب در اوبونتو/ دبیان را در ادامه آمده است.

ابتدا، مخزن سیستم را به روز رسانی کنید.

```
sudo apt update
```

سپس با دستور aptitude زیر ابزارهای رایج لینوکس را روی دستگاه خود نصب کنید. این دستور نیز به سطح دسترسی ریشه نیاز دارد. هنگامی که نصب به پایان رسید، می‌توانید بسته های Perf را در پوشه /usr/bin/perf پیدا کنید.

```
sudo apt install linux-tools-common
```

از آنجایی که Perf از جمله بسته‌های مشترک لینوکس است، لازم است اطمینان حاصل کنیم که با هسته لینوکس سیستم سازگار است. برای بررسی هسته سیستم دستور زیر را اجرا کنید.

```
uname -r
```

نسخه هسته را یادداشت کنید و با دستور زیر ابزارهای لینوکسی مناسب هسته سیستم خود را نصب کنید . بجای عبارت you_kernel_version عددی که از مرحله قبل یادداشت کردید را جایگزین کنید. مثلا (linux-tools-5.8.0-50)

```
sudo apt-get install linux-tools-you_kernel_version
```

می‌توانید برای اطمینان از نسخه Perf نصب شده، دستور زیر را اجرا کنید. خواهید دید که نسخه Perf وKernel یکسان است.

```
perf -v
```

مراحل نصب ذکر شده در بالا را می‌توان بصورت ترکیب شده در یک دستور واحد، به صورت زیر انجام داد!

```
sudo apt install linux-tools-$(uname -r) linux-tools-generic
```

8-3-کاربردها:

```
usage: perf [--version] [--help] [OPTIONS] SUBCOMMAND [ARGS]
```

7-3-1-تنظیم مجوز کاربران معمولی برای استفاده از ابزار Perf

استفاده از دستور perf به شکل پیش‌فرض نیاز به امتیاز sudo دارد. برای اینکه کاربران معمولی نیز بتوانند از perf استفاده کنند، مراحل زیر را طی کنید:

به کاربر روت سوئیچ کنید:

```
sudo su -
```

دستور زیر را اجرا کنید

```
echo 0 > /proc/sys/kernel/perf_event_paranoid
```

این دستور به کاربران معمولی اجازه می دهد در نشست حاضر از ابزار perf استفاده کنند.

با دستور زیر به کاربر معمولی برگردید.

```
exit
```

برای اینکه تغییراتی که ایجاد کردید، دائمی شود، به صورت زیر عمل کنید:

فایل تنظیمات sysctl را با یک ویرایشگر متنی باز کنید :

```
sudo nano /etc/sysctl.conf
```

خط زیر را به فایل اضافه کنید:

```
kernel.perf_event_paranoid = 0
```

تغییرات را ذخیره کرده و فایل را بندید.

8-3-2-ثبت پروفایل فرآیندها

از دستور perf record می‌توان برای ثبت پروفایل کارایی یک فرآیند موجود نیز استفاده کرد. به این طریق می‌توانیم رفتار و ویژگی های کارایی یک فرآیند در حال اجرا را بدون نیاز به راه اندازی مجدد آن تجزیه و تحلیل کنیم.

```
sudo perf record -p pid
```

ثبت پروفایل‌های فرآیند، هنگام تجزیه و تحلیل کارایی یک فرآیند طولانی یا اشکال‌زدایی بلادرنگ از یک موضوع خاص، مفید است. با این کار می‌توانیم مشخصات کارایی را بدون ایجاد وقفه در فرآیند، ثبت و بررسی کنیم.

خروجی دستور perf record معمولا کوتاه است و میزان پیشرفت دستور و محل قرارگیری فایل perf.data را نشان می‌دهد.

```
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.001 MB perf.data (3612 samples>
                                                                ]
```

تمرین کلاسی:

- در خط فرمان اپلیکیشن‌های نصب شده روی سیستم خود را لیست کنید.
- دستورات پایه perf را بر روی سه مورد از اپلیکشن‌های مذکور اجرا نمایید.

گزارش کار:

- ابزار Perf را روی توزیع لینوکسی خود نصب کرده و با استفاده از آن، کارایی یک اپلیکیشن ساده را بررسی و تحلیل نمایید.

منابع و مراجع:

1. https://perf.wiki.kernel.org/index.php/Tutorial, visited 2024, Aug.
2. https://phoenixnap.com/kb/linux-perf, visited 2024, Aug.
3. https://thelinuxcode.com/linux-perf-commands, visited 2024, Aug.

آزمایش 9

کاربردهای پیشرفته و تحلیل نتایج با ابزار Perf

اهداف:

- یادگیری دستورات و آپشن‌های پیشرفته Perf
- تفسیر نتایج و تصمیم‌گیری

ابزار و مفاهیم مورد نیاز:

- سیستم عامل لینوکس با ابزار Perf نصب شده
- برنامه‌های کاربردی نصب شده برای تحلیل کارایی

محتوا:

- یادآوری دستورهای پایه
- دستورهای بیشتر :
  - perf trace : رهگیری فراخوانی‌های سیستمی
  - perf top : نظارت بر کارایی به صورت زنده
  - perf annotate : حاشیه نویسی کد منبع با داده‌های کارایی
  - perf script : تولید گزارش‌های شخصی‌سازی شده
- آپشن‌های perf
- تحلیل کارایی با استفاده از داده‌های حاصل از perf annotate و perf report

مفاهیم و شرح دستورکار

9-1-دستورات بیشتر

دستور فرعی trace

از دستور فرعی اسکریپت برای فهرست کردن همه رویدادها از perf.data استفاده می‌شود. به عنوان مثال:

```
script      sudo        perf
```

خروجی جزئیات perf.data را به ترتیب زمانی چاپ می کند. از دستور فرعی script به عنوان داده‌های پس از پردازش استفاده کنید.

نمایش هدر trace

برای نمایش تمام رویدادها از perf.data همراه با اطلاعات اضافی trace header، دستور زیر را اجرا کنید:

```
sudo        perf        script      --header
```

خروجی اطلاعات هدر فایل را نشان می دهد، مانند زمان شروع ردیابی، مدت زمانی که طول کشیده است، اطلاعات CPU و فرمانی که داده‌ها را واکشی کرده است. لیست رویدادها بعد از اطلاعات هدر آمده است.

تخلیه(Dump) داده های خام

برای حذف داده های خام به صورت هگز از فایل perf.data، از آپشن D- استفاده کنید:

```
sudo        perf        script      -D
```

نتیجه، اطلاعات ردیابی رویداد خام در قالب ASCII است. این گزینه برای اشکال زدایی رویداد مفید است.

حاشیه نویسی داده‌ها

برای حاشیه نویسی داده ها و جداسازی بیشتر، از دستور فرعی annotate استفاده کنید:

```
sudo perf annotate --stdio -v
```

آپشن v- خروجی پر جزئیاتی را ارائه می دهد. نتیجه اجرا، کد منبع و disassembly رویدادها را نمایش می دهد.

9-2- جدول برخی از دستورات فرعی پرکاربرد Perf

annotate

perf.data را که توسط perf record تولید شده، می‌خواند و کد مشروح (annotated) رانمایش می‌دهد.

archive

با فایل objectهای حاوی شناسه‌های توکار که در فایل perf.data قرار دارند، آرشیو ایجاد می‌کند.

bench

چارچوب کلی برای مجموعه‌های benchmark

config

متغیرها را در یک فایل پیکربندی دریافت و تنظیم می‌کند.

daemon

record sessions را در پس زمینه اجرا می‌کند.

iostat

معیارهای کارایی I/O را نشان می‌دهد.

list

همه رویدادهای قابل اندازه گیری را فهرست می‌کند.

mem

دسترسی به حافظه را پروفایل می‌کند

record

یک دستور را اجرا کرده و پروفایل آن را در perf.data ضبط می‌کند.

report

perf.data را که توسط perf record ایجاد شده خوانده و پروفایل را نمایش می‌دهد.

script

perf.data را که توسط perf record ایجاد شده خوانده و trace را نمایش می‌دهد.

stat

یک دستور را اجرا کرده و آمار شمارنده کارایی را جمع آوری می‌کند.

top

ابزار پروفایل کردن سیستم

version

نسخه perf را نمایش می‌دهد.

برای دریافت اطلاعات بیشتر در مورد هر دستور فرعی، perf help COMMAND را اجرا کنید.

تمرین کلاسی:

- تمرین دستورهای perf

گزارش کار:

- با کمک perf، اپلیکیشنی که بیشترین میزان استفاده از CPU سیستمتان دارد را شناسایی کنید.

منابع و مراجع:

1. http://perf.wiki.kernel.org, visited 2024, Aug.
2. https://phoenixnap.com/kb/linux-perf, visited 2024, Aug.
3. https://thelinuxcode.com/linux-perf-commands, visited 2024, Aug.

آزمایش 10

ماژول‌های هسته لینوکس

اهداف:

- ایجاد، کامپایل و مدیریت ماژول‌های هسته لینوکس

ابزار و مفاهیم مورد نیاز:

- سیستم عامل لینوکس
- ابزارهای کامپایل

محتوا:

- معرفی ماژول‌های هسته
- مزایای ماژول‌های شخصی بر ماژول‌های build-in
- ایجاد یک ماژول ساده
- کدنویسی
- ایجاد makefile و کامپایل
- load و unload کردن ماژول در هسته و چک کردن وضعیت ماژول

مفاهیم و شرح دستورکار

10-1- ماژول‌های هسته

ماژول‌ها تکه کدهایی هستند که در حین اجرای هسته لینوکس می توانند وارد آن شده و یا از آن خارج شوند. این تکه کدها عملکرد هسته را بدون نیاز به راه اندازی دوباره کامپیوتر توسعه می‌دهند. به عنوان مثال یک نوع از ماژول‌ها device driver ها هستند که به هسته امکان استفاده از قابلیت سخت افزار ها را می‌دهند. اگر ماژول‌ها وجود نداشتند، برای هر قابلیتی که می‌خواستیم به هسته اضافه کنیم یا از آن کم کنیم، می‌بایستی یک بار هسته را کامپایل می‌کردیم و برای استفاده از ان قابلیت یا حذف آن یک بار سیستم را از نو راه اندازی می‌کردیم.

10-1-1-ماژول ها چگونه به هسته وارد می شوند؟

می‌توانید با اجرای دستور lsmod ماژول‌هایی که هم اکنون در هسته وارد شده‌اند و اطلاعات آن‌ها را ببینید این دستور اطلاعات خود را از فایل /proc/modules/دریافت می‌کند.

هنگامی که هسته، به امکان و عملکردی نیاز دارد که هم اکنون در آن نیست، یکی از deamon های آن به نام kmod دستور modprobe را اجرا می‌کند تا ماژول مربوطه که آن عملکرد را دارد، وارد هسته شود . هنگامی که modprobe اجرا می شود به آن یک رشته کاراکتر به دو صورت زیر داده می‌شود:

- نام ماژول مثلا softdog یا ppp
- یک مشخصه کلی مانند char-major-10-30

اگر حالت اول به modprobe داده شود، این دستور به دنبال فایلی به نام softdog.ko یا ppp.ko با روشی که در ادامه می‌آید می‌گردد. ولی اگر حالت دوم به modprobe داده شود، این دستور ابتدا به دنبال رشته کاراکتر در فایل /etc/modprobe.conf می گردد و اگر توانست alias یا مستعاری مانند:

alias char-major-10-30 softdog

پیدا کند، متوجه می‌شود که این نام کلی که در اینجا char-major-10-30 است به ماژول softdog اشاره می‌کند که فایل ماژول آن softdof.ko می‌باشد.

در مرحله بعد modprobe فایل /lib/modules/version/modules.dep را باز کرده و به دنبال ماژول‌هایی می‌گردد که باید قبل از ماژول مورد نظر به هسته وارد شوند. این فایل به وسیله دستور depmod -a ایجاد می‌شود و حاوی وابستگی بین ماژول هاست.

به عنوان مثال اگر به دنبال ماژول msdos.ko در این فایل بگردید خواهید دید که به ماژول دیگری به نام fat.ko وابسته است یعنی برای اینکه msdos.ko وارد هسته شود حتما باید قبل از آن fat.ko وارد شده باشد. این مساله برای fat.ko نیز تکرار شده تا به مرحله‌ای برسیم که دیگر وابستگی موجود نباشد. در نهایت modprobe دستور insmod را به کار می‌برد تا ابتدا وابستگی‌ها را به هسته وارد کند و در نهایت ماژول مورد نظر ما به هسته وارد می‌شود.

پس modprobe وظیفه پیدا کردن ماژول، تعیین وابستگیهای آن و وارد کردن آن به هسته به وسیله صدا کردن insmod را دارد در حالی که insmod فقط وظیفه وارد کردن آن ماژول به هسته را دارد.

به عنوان مثال اگر بخواهیم به صورت دستی msdos.ko را وارد هسته کنیم به صورت زیر عمل می کنیم :

```
# insmod   /lib/modules/2.6.11/kernel/fs/fat/fat.ko
# insmod   /lib/modules/2.6.11/kernel/fs/fat/msdos.ko
```

معادل دو دستور بالا با modprobe به صورت زیر است:

```
# modprobe msdos
```

توجه کنید که insmod مسیر کامل تا فایل ماژول را می‌خواهد در حالی که modprobe فقط نام ماژول را می‌گیرد.

نکات:

۱) modversioning : یک ماژول که برای یک هسته خاص کامپایل شده است بر روی هسته دیگر load نخواهد شد مگر اینکه شما CONFIG_MODVERSIONS را در هسته فعال کنید.

۲) ماژول‌ها نمی‌توانند چیزی به غیر از خطاها و هشدارها را بر روی صفحه نمایش نشان دهند. آن‌ها برای نشان دادن اطلاعات خود، آن‌ها را در log فایل‌ها می‌نویسند.

۳) پیشنهاد می‌شود که برای جلوگیری از برخی مشکلات مانند احتمال وجود فایل‌های ناقص Header برای هسته، خودتان هسته را کامپایل کنید.

10-2-ماژول Hello World

فایلی به نام hello-1.c باز کرده و کد C زیر را در آن بنویسید:

```
#include  <linux/module.h>  /*needed by all modules  */
#include  <linux/kernel.h>  /*needed for Macros like KERN_INFO */

int  init_module(void)  /* this
function is called as initialization for all modules */
{
     printk(KERN_INFO "Hello World1.\n");
     /* if this function returns non zero means init_module failed
and
this module can't be loaded.
*/
     return  0;
}
void  cleanup_module(void) /* it is
called when module is terminated and unloaded */
{
     printk( KERN_INFO "Goodbye World1.\n");
}
```

هر ماژول هسته، حداقل بایستی ۲ تابع داشته باشد. اول تابع شروع که ()init_module نامیده می‌شود و هنگام load شدن ماژول در هسته صدا زده می‌شود و دیگری تابع پایان که ()cleanup_module نامیده می‌شود و هنگام unload شدن ماژول از هسته صدا زده می‌شود. با اینکه بعد از هسته 2.3.13 شما می‌توانید هر نام دیگری برای این دو تابع قرار دهید، اما خیلی از افراد همچنان از این استاندارد قدیمی استفاده می‌کنند. دو فایل Header در این کد ضمیمه شده‌اند. یکی linux/module.h می‌باشد که برای هر ماژولی مورد نیاز است و تعریف خیلی از توابع را در خود دارد و دیگری linux/kernel.h می‌باشد که حاوی تعدادی ماکرو می‌باشد مانند .KERN_INFO

printk()

این تابع چیزی در صفحه نمایش چاپ نمی کند و برای کار با کاربر نیست. این تابع برای مکانیزم log هسته به کار می‌رود . هر ()printk با یک اولویت می‌آید که در این مثال ماکروی KERN_INFO برای این منظور به کار رفته است. تعداد ۸ اولویت وجود دارند که به صورت ماکرو در فایل linux/kernel.h تعریف شده‌اند. اگر شما این اولویت را تعیین نکنید به طور پیش فرض DEFAULT_MESSAGE_LOGLEVEL به آن تخصیص می‌یابد .

اگر این اولویت کمتر از اولویت int console_loglevel (که در linux/kernel.h تعریف شده است) باشد، message در دستور ()printk بر روی صفحه ظاهر می‌شود. اگر syslogd و یا klogd در سیستم در حال اجرا باشند این message در فایل /var/log/messages/ نوشته می‌شود.

3-10-کامپایل ماژول های هسته

برای اینکه یک ماژول هسته را به درستی کامپایل کنید، نیاز به تنظیمات بسیار زیادی دارید. با پیچیده‌تر شدن ماژول‌ها این تنظیمات پیچیده‌تر می‌شوند. مکانیزمی به نام kbuild وجود دارد که تمام این تنظیمات را انجام می‌دهد. برای اگاهی بیشتر از این مکانیزم می‌توانید به فایل‌های مستند هسته که در آدرس

linux/Documentation/kbuild/modules.txt کد منبع هسته موجود است مراجعه کنید. برای اینکه بتوانیم از مکانیزم kbuild استفاده کنیم، بایستی یک Makefile مطابق با استاندارد آن بنویسیم. برای این کار یک فایل به نام Makefile باز کرده و دستورات زیر را در آن بنویسید:

```
obj-m  +=  hello-1.o
all :
     make  -C  /lib/modules/$(shell uname -r)/build  M=$(PWD)
modules
clean:
     make  -C  /lib/modules/$(shell uname -r)/build  M=$(PWD)
clean
```

حال با اجرای دستور make ماژول خود را کامپایل کنید. در هسته 2.6 به بعد از پسوند ko برای نامیدن ماژول‌های هسته استفاده شده است که به راحتی قابل تمییز از o که پسوند فایل های object است می‌باشد.

برای بدست آوردن اطلاعاتی از ماژول خود دستور زیر را اجرا کنید:

```
# modinfo hello-1.ko
```

برای وارد کردن ماژول خود در هسته از دستور زیر استفاده کنید:

```
# insmod ./hello-1.ko
```

اگر بعد از اجرای این دستور فایل /var/log/messages را باز کرده و به انتهای آن بروید، خواهید دید که ماژول hello-1 در هسته load شده است. با دستور lsmod نیز ماژول load شده را خواهید دید. برای unload یا خارج کردن ماژول خود از هسته از دستور rmmod به صورت زیر استفاده کنید :

```
# rmmod hello-1
```

دوباره اگر فایل /var/log/messagesرا باز کنید و به انتهای آن بروید خواهید دید که ماژول hello-1 از هسته خارج شده است.

ایجاد یک ماژول ساده دیگر. ماژولی بنویسید که هنگام بارگیری و خروج از هسته، پیام‌هایی را ثبت کند.

یک فایل با نام simple_module.c ایجاد کنید و کد ماژول را در آن کپی کنید.

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>


MODULE_LICENSE("GPL");
MODULE_AUTHOR("Your Name"); MODULE_DESCRIPTION("A simple Linux
kernel module");
MODULE_VERSION("1.0");


static int __init
simple_module_init(void) {
printk(KERN_INFO "Simple Module: Initialization\n");
return 0;
}


static void __exit
simple_module_exit(void) {
printk(KERN_INFO "Simple Module: Exit\n");
}


module_init(simple_module_init); module_exit(simple_module_exit);


Makefile obj-m += simple_module.o
all:
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
```

```
clean:
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

در همان دایرکتوری یک فایل به نام Makefile ایجاد کنید و محتوای Makefile را در آن کپی کنید.

ترمینال را باز کنید و به دایرکتوری حاوی simple_module.c و Makefile بروید.

دستور make را اجرا کنید.

با دستور زیر ماژول را بارگیری کنید

```
sudo  insmod       simple_module.ko
```

گزارش هسته را بررسی کنید تا پیام چاپ شده را ببینید.

```
dmesg | tail
```

ماژول را unload کنید

```
sudo  rmmod      simple_module
```

گزارش هسته را بررسی کنید تا پیام خروج را ببینید.

```
dmesg | tail
```

تمرین کلاسی:

- ماژول helloworld را تغییر دهید بطوریکه هنگام بارگیری و خروج ماژول پیام‌هایی را در لاگ چاپ نماید.

گزارش کار:

- ماژول هسته ای را که در کلاس نوشتید طوری تغییر دهید که هنگامی که یک صفحه کلید USB (یا فلش) به سیستم وصل می‌شود، ماژول به طور خودکار توسط ابزارهای hotplug مناسب فضای کاربر (که بسته به توزیعی که استفاده می کنید، توسط depmod / kmod / udev / mdev / systemd پیاده سازی می شود) بارگذاری گردد.

منابع و مراجع:

- Love R. Linux kernel development. Pearson Education; 2010 Jun 22.
- Salzman PJ, Burian M, Pomerantz O. The Linux Kernel Module Programming Guide. CreateSpace; 2022 Dec 25.

آزمایش 11

کامپایل هسته لینوکس

اهداف:

- درک مفاهیم و انجام کامپایل هسته لینوکس

ابزار و مفاهیم مورد نیاز:

- سیستم عامل لینوکس
- کاربر روت یا sudoer
- سورس کد هسته لینوکس
- ابزارهای کامپایل،gcc ،make، libncurses-dev
- 12 گیگابایت فضای آزاد روی هارد

محتوا:

- معرفی هسته لینوکس
  - مفاهیم مونولیتیک و ماژولار
- دلایل کامپایل مجدد هسته
  - بهینه‌سازی کارایی، اهداف آموزشی، اضافه کردن موارد جدید به هسته، آزمودن patchهای آزمایشی، فعال یا غیرفعال کردن گزینه‌ها و درایورها
- تنظیمات کرنل (make xconfig یا make menuconfig)
- پروسه کامپایل

مفاهیم و شرح دستورکار

1-11- هسته لینوکس

هسته لینوکس جزء مرکزی سیستم عامل گنو/لینوکس است که مسئول مدیریت منابع سخت افزاری و ارائه خدمات اساسی به برنامه های کاربردی سطح کاربر است. هسته لینوکس که در سال 1991 توسط لینوس توروالدز توسعه یافت، به یک پلتفرم قوی و همه کاره تبدیل شده است که در طیف گسترده ای از دستگاه‌ها، از رایانه های شخصی گرفته تا سرورها و سیستم‌های embedded استفاده می‌شود. ماهیت منبع باز آن باعث ایجاد یک محیط توسعه مشترک شده است، که مشارکت توسعه دهندگان در سراسر جهان را به خود جلب کرده و منجر به بهبود مستمر در کارایی، امنیت و عملکرد آن شده است.

مدیریت فرآیند یکی از جنبه‌های مهم هسته لینوکس است. هسته لینوکس، ایجاد، زمان‌بندی و خاتمه فرآیندها را مدیریت می‌کند و استفاده کارآمد از منابع CPU را تضمین می‌نماید. هسته از الگوریتم‌های زمان‌بندی مختلفی برای مدیریت اجرای فرآیند استفاده می‌کند، اما برنامه‌ریزی کاملاً منصفانه(CFS) ، الگوریتم پیش‌فرض در توزیع‌های لینوکس مدرن است. هدف CFS ارائه یک تخصیص متعادل و منصفانه از زمان CPU به فرآیندها، بهبود پاسخگویی و کارایی کلی سیستم است. علاوه بر این، هسته از چند وظیفه‌ای و چند رشته‌ای پشتیبانی می‌کند و به چندین فرآیند و رشته اجازه می‌دهد تا بطور همزمان اجرا شوند.

با توجه به استفاده گسترده از لینوکس در سیستم های حیاتی، امنیت یک نگرانی اساسی برای هسته است. هسته دارای چندین مکانیسم امنیتی برای محافظت در برابر آسیب پذیری‌ها و حملات است که شامل چارچوب‌های کنترل دسترسی مانند SELinux (Security-Enhanced Linux) است که کنترل‌های اجباری دسترسی را اعمال می‌کند و AppArmorکه امنیت در سطح برنامه‌ها را فراهم می‌کند. هسته همچنین از الگوریتم‌های رمزنگاری مختلف و پروتکل‌های ارتباطی امن برای محافظت از یکپارچگی و محرمانه بودن داده‌ها پشتیبانی می‌کند. ممیزی (audit) امنیتی و به روز رسانی مداوم برای مقابله با تهدیدات نوظهور و حفظ انعطاف‌پذیری هسته ضروری است.

در ابتدا، هسته لینوکس به عنوان یک هسته یکپارچه(monolithic) طراحی شد، به این معنی که تمام عملکردهای اصلی، از جمله درایور دستگاه‌ها، فایل سیستم و پروتکل‌های شبکه، در یک فایل اجرایی بزرگ که در مد هسته اجرا می‌شد، ادغام شده بودند. این طراحی مزیت بهره‌وری و کارایی را فراهم می‌کرد، زیرا همه اجزا می‌توانستند مستقیماً در فضای آدرس یکسان با هم ارتباط برقرار کنند و هزینه‌های سربار مرتبط با ارتباطات بین فرآیندی کاهش می‌یافت. اما در این حالت، هسته پیچیده‌تر و نگهداری از آن سخت‌تر می‌شد، زیرا هر تغییری مستلزم کامپایل مجدد کل هسته بود. با گذشت زمان، نیاز به انعطاف‌پذیری و ماژولار بودن بیشتر منجر به معرفی ماژول‌های هسته قابل بارگیری شد. بطوریکه امروزه یکی از نقاط قوت اصلی هسته لینوکس معماری ماژولار آن است که امکان load و unload پویای ماژول‌های هسته را فراهم می‌کند. در تصویر 11-1 می‌توانید مراحل گذار از حالت مونولتیک به ماژولار را ببینید.

بطور کلی انگیزه‌های تغییر به سمت هسته ماژولار را می‌توان عوامل زیر دانست:

- انعطاف‌پذیری بیشتر: زیرا ماژولار بودن کاربران را قادر می سازد بدون کامپایل مجدد کل هسته، ویژگی-ها(features) را حذف یا اضافه کنند. ماژول‌های هسته می‌توانند شامل درایورهای دستگاه، فایل سیستم و پروتکل‌های شبکه و سایر اجزا باشند. این طراحی علاوه بر اینکه تعمیر و نگهداری هسته را ساده می‌کند، در هر زمان، تنها به ماژول های لازم اجازه بارگذاری می‌دهد، و باعث افزایش ثبات و کارایی سیستم می‌شود.
- قابلیت نگهداری بهتر هسته: با جداسازی عملکردهای مختلف در ماژول های جداگانه، توسعه دهندگان می توانند بر روی اجزای جداگانه کار کنند بدون اینکه کل سیستم را تحت تأثیر قرار دهند. بنابراین اشکال زدایی و به روز رسانی ساده‌تر می‌شوند.
- افزایش امنیت: هسته های ماژولار می‌توانند مقدار کد در حال اجرا در فضای هسته را به حداقل برسانند، به این ترتیب سطح حمله کاهش یافته و امنیت سیستم افزایش می‌یابد.

انتقال از هسته یکپارچه به یک هسته ماژولار در لینوکس تدریجی و شامل چندین پیشرفت کلیدی بود:

- ماژول‌های هسته قابل بارگیری :(Loadable Kernel Modules (LKM)) که در اوایل دهه 1990 معرفی شدند، اجازه load و unload بخش‌هایی از هسته را در زمان اجرا می‌دادند. این نوآوری امکان افزودن قابلیت‌های جدید مانند درایورهای دستگاه را بدون راه‌اندازی مجدد سیستم فراهم کرد.

- ابزارهای پیکربندی هسته: ابزارهایی مانند make menuconfig و make xconfig رابط های کاربرپسندی را برای پیکربندی هسته ارائه دادند که به کاربران امکان می‌داد بر اساس نیاز خود انتخاب کنند که کدام ماژول ها را اضافه یا حذف کنند.
- بهبود مدیریت ماژول: با گذشت زمان، هسته لینوکس مکانیسم‌های بهتری را برای مدیریت وابستگی بین ماژول‌ها گنجانده است و اطمینان حاصل می‌کند که ماژول‌ها می‌توانند با خیال راحت و بطور کارآمد load و unload شوند.

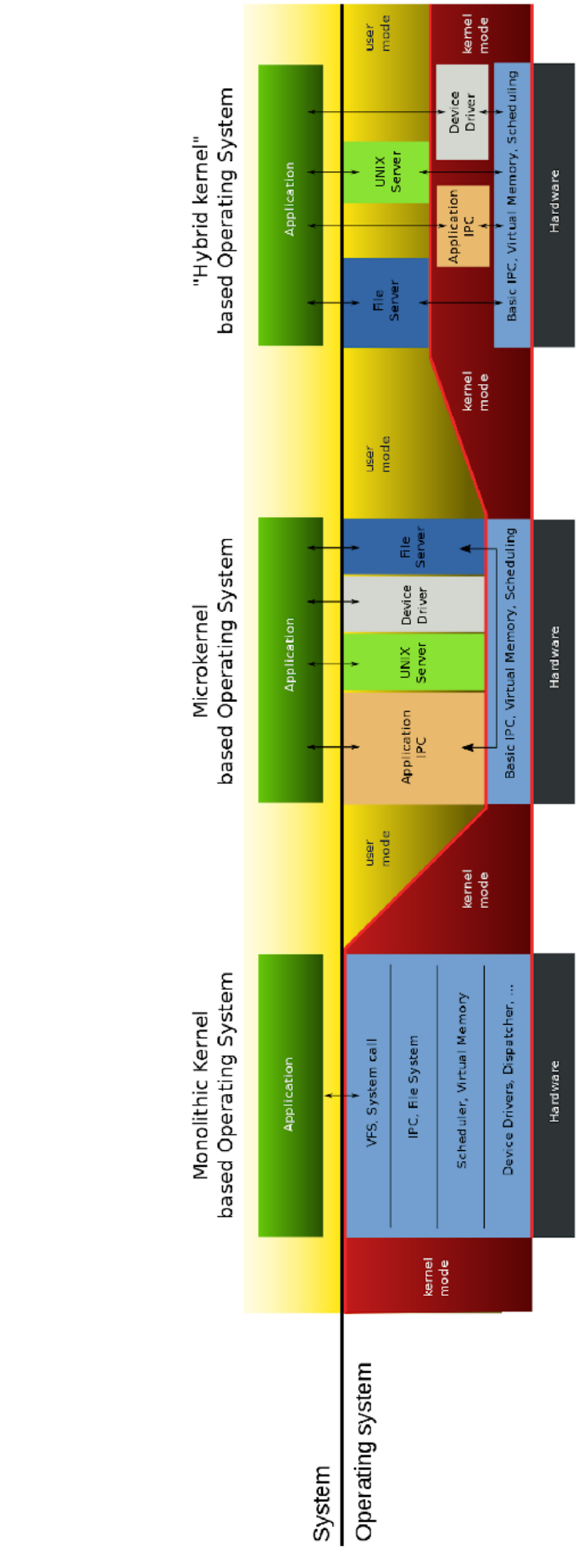
11-1-1- مزایای هسته ماژولار

- کارایی بهبودیافته: هسته ماژولار فقط ماژول های لازم را بارگذاری می‌کند و به این ترتیب، سیستم می تواند استفاده از منابع را بهینه کرده و کارایی را بهبود بخشد، به خصوص در سیستم هایی با منابع سخت افزاری محدود.
- مقیاس پذیری: هسته های ماژولار می توانند به راحتی مقیاس شوند تا طیف وسیعی از دستگاه‌ها و معماری‌ها را، از سرورهای با کارایی بالا گرفته تا سیستم‌های توکار، پشتیبانی کنند.
- مشارکت جامعه: طراحی ماژولار، مشارکت جامعه open-source را تسهیل کرده است، زیرا توسعه‌دهندگان می توانند بدون نیاز به درک کل هسته بر روی ماژول های خاصی تمرکز کنند.

11-1-2- چالش‌های هسته ماژولار

رویکرد ماژولار علیرغم مزایایش، چالش‌هایی را نیز به همراه دارد:

- پیچیدگی: مدیریت وابستگی‌ها و اطمینان از سازگاری بین ماژول‌ها می تواند پیچیده بوده و به ابزارها و فرآیندهای قوی نیاز دارشته باشد.
- سربار کارایی: در حالی که ماژولار بودن می تواند کارایی را افزایش دهد، به دلیل نیاز به ارتباط بین ماژول ها می تواند سربار نیز ایجاد کند.
- خطرات امنیتی: بارگذاری پویای ماژول‌ها، به طور بالقوه می تواند آسیب‌پذیری‌های امنیتی را در صورت عدم مدیریت صحیح، ایجاد کند.

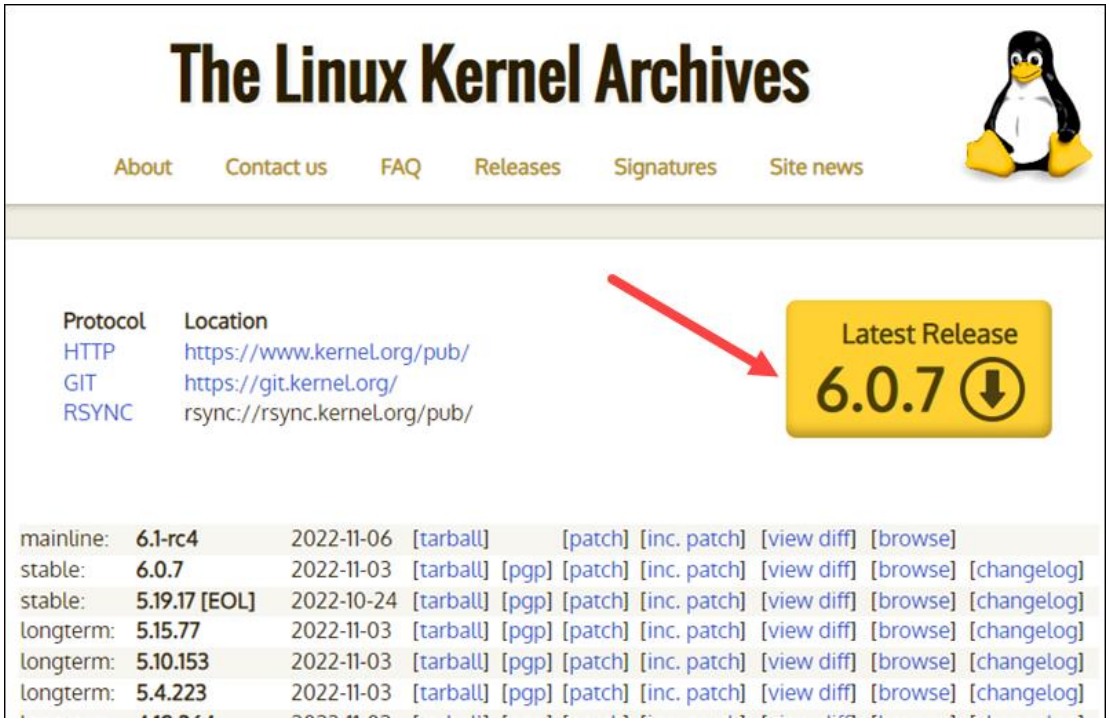


تصویر 11-1- مراحل گذار هسته لینوکس از مونولیتی به ماژولاریتی

در نهایت، هسته لینوکس پایه‌ای قدرتمند و انعطاف‌پذیر برای طیف گسترده‌ای از برنامه‌ها ارائه می‌دهد. تکامل آن از یک معماری یکپارچه به یک معماری ماژولار به طور قابل توجهی انعطاف‌پذیری و قابلیت نگهداری آن را افزایش داده است. معماری ماژولار هسته، مدیریت فرآیند کارآمد و ویژگی‌های امنیتی قوی، آن را به انتخابی ارجح برای توسعه‌دهندگان در سراسر جهان تبدیل کرده است. ماهیت مشارکتی توسعه آن، تضمین می‌کند که هسته به تکامل خود ادامه می‌دهد، با فناوری‌های جدید سازگار می‌شود و سعی می‌کند به چشم‌انداز دائماً در حال تغییر نیازهای محاسباتی پاسخ بدهد.

11-2-ساخت و کامپایل هسته لینوکس

1. از وب سایت رسمی کرنل آخرین نسخه کرنل[11] را دانلود کنید. فایل دانلود شده حاوی سورس کد فشرده است.



[11] اگر آخرین نسخه هسته در وب سایت رسمی با نسخه ذکر شده در راهنما مطابقت ندارد، شماره نسخه موجود در دستورات را با آخرین نسخه فعلی جایگزین کنید.

می‌توانید از طریق ترمینال و اجرای دستور wget نیز کد منبع هسته لینوکس را دانلود کنید(نام و شماره نسخه را باید درست وارد کنید)

```
wget    https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.0.7.tar.xz
```

پس از اتمام دانلود، پیام "saved " در خروجی نمایش داده می‌شود.

2. وقتی فایل دانلود شد، دستور tar را برای استخراج کد منبع اجرا کنید:

```
tar    xvf  linux-6.0.7.tar.xz
```

فایل‌های سورس‌کد استخراج شده، در خروجی نمایش داده می‌شود.

3. قبل از شروع به ساخت کرنل، باید پکیج‌های مورد نیاز را نصب کنید:

```
sudo apt-get  install git fakeroot build-essential ncurses-dev xz-utils libssl-dev bc flex libelf-dev bison
```

این دستور، بسته‌های زیر را نصب خواهد کرد:

git : رهگیری و ثبت تمام تغییرات کد منبع در حین توسعه. همچنین امکان برگرداندن تغییرات را فراهم می کند.

fakeroot : یک محیط ریشه جعلی ایجاد می‌کند.

build-essential : ابزارهای توسعه مانند C، C++، gcc و g++ را نصب می کند.

ncurses-dev : برای ترمینال‌های متنی، API ارائه می‌دهد.

xz-utils : فشرده سازی و خروج از حالت فشرده سریع را برای فایل‌ها فراهم می‌کند.

libssl-dev : از SSL و TSL پشتیبانی می‌کند که داده ها را رمزگذاری کرده و اتصال اینترنت را ایمن می‌کند.

bc (Basic Calculator) : از اجرای تعاملی statements پشتیبانی می‌کند.

flex (Fast Lexical Analyzer Generator) : تحلیلگرهای واژگانی تولید می‌کند که کاراکترها را به توکن تبدیل می‌کند.

libelf-dev : یک کتابخانه مشترک برای مدیریت فایل‌های ELF (فایل‌های اجرایی، فایل‌های خالی هسته و کد اشیا) ایجاد می‌کند.

bison : توضیحات دستور زبان را به یک برنامه C تبدیل می‌کند.

4. کد منبع هسته لینوکس با پیکربندی پیش فرض ارائه می شود. اما می‌توان آن را با نیازهای خود تنظیم کرد.

با استفاده از دستور cd به دایرکتوری linux-6.0.7 بروید(شماره نسخه را بر اساس نسخه دانلودی خودتان تصحیح کنید):

```
cd    linux-6.0.7
```

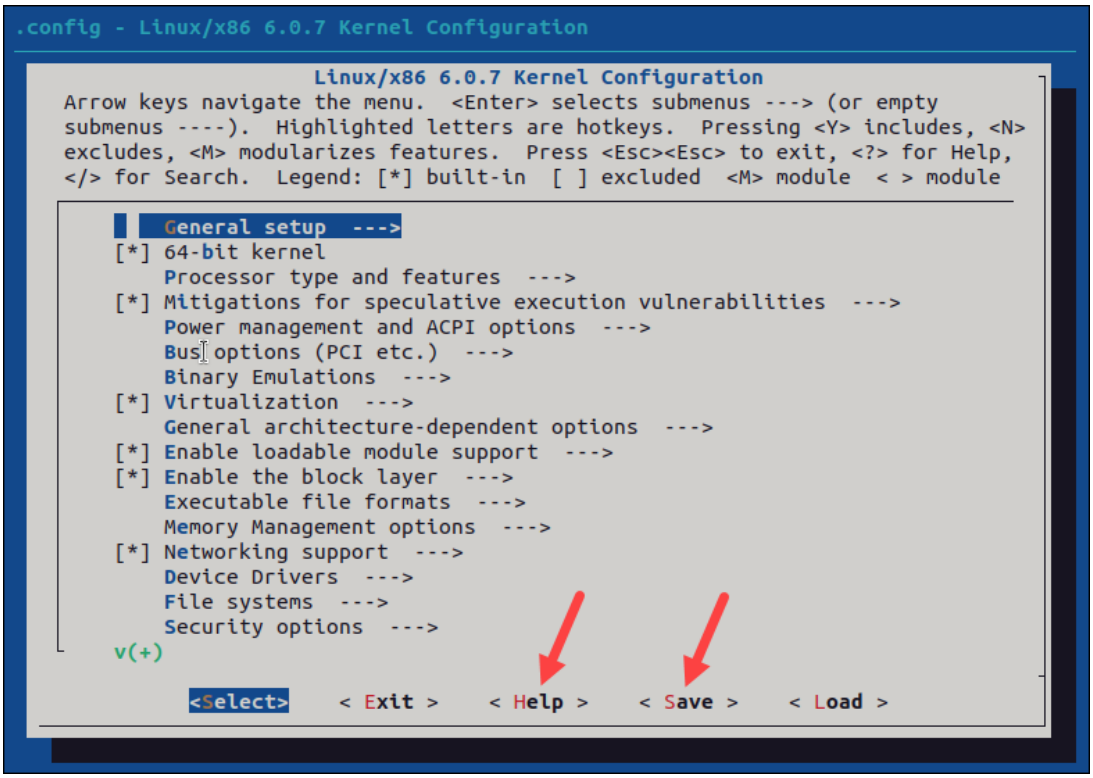
فایل پیکربندی لینوکس موجود را با استفاده از دستور cp کپی کنید:

```
cp     -v    /boot/config-$(uname -r)   .config
```

برای ایجاد تغییرات در فایل پیکربندی، دستور make را اجرا کنید:

```
make  menuconfig
```

این فرمان چندین اسکریپت را اجرا می‌کند که منوی پیکربندی را باز می‌کنند. این منو شامل گزینه‌هایی مانند تنظیمات سیستم عامل، سیستم فایل، شبکه و حافظه است. از کلیدهای جهت کیبورد برای حرکت روی گزینه‌ها استفاده کنید. همچنین می‌توانید راهنما را انتخاب کنید تا درباره گزینه‌های موجود، اطلاعات بیشتری به دست بیاورید. پس از اتمام تغییرات، ذخیره را انتخاب کنید و سپس از منو خارج شوید.



توجه: تغییر تنظیمات برای برخی از گزینه ها می تواند منجر به یک هسته غیر کاربردی شود. اگر مطمئن نیستید که چه چیزی را تغییر می‌دهید، تنظیمات را به حالت پیش فرض رها کنید.

5. با اجرای دستور زیر شروع به ساخت هسته کنید:

```
make
```

فرآیند ساخت و کامپایل هسته لینوکس مدتی طول می کشد تا کامل شود. ترمینال تمام اجزای هسته لینوکس را فهرست می کند: مدیریت حافظه، درایورهای دستگاه سخت افزار، درایورهای سیستم فایل، درایورهای شبکه و مدیریت فرآیند.

اگر در حال کامپایل کردن هسته در اوبونتو هستید، ممکن است خطای زیر را دریافت کنید که فرآیند ساخت را مختل می کند:

```
No rule to make target 'debian/canonical-certs.pem
```

در صورت مواجهه با چنین خطایی، با اجرای دو دستور زیر، گواهینامه‌های امنیتی را که منجر به ایجاد تداخل شده‌اند، غیرفعال کنید:

```
scripts/config --disable SYSTEM_TRUSTED_KEYS
scripts/config --disable SYSTEM_REVOCATION_KEYS
```

اجرای دستورات مذکور هیچ خروجی ندارد. فرآیند ساخت را دوباره با make شروع کنید و برای تأیید گزینه‌های پیش‌فرض برای تولید گواهی‌های جدید، مکرراً Enter را فشار دهید.

ماژول های مورد نیاز را با این دستور نصب کنید:

```
sudo  make  modules_install
```

در نهایت، هسته را با دستور زیر نصب کنید:

```
sudo  make  install
```

پس از اتمام کار، عبارت done در خروجی نمایش داده می‌شود.

6. به روز رسانی بوت‌لودر (اختیاری)

دستور make install این فرآیند را به صورت خودکار انجام می دهد، اما می‌توانید آن را به صورت دستی نیز انجام دهید.

initramfs را به نسخه هسته نصب شده، به روز کنید:

```
sudo  update-initramfs      -c     -k   6.0.7
```

بوت لودر GRUB را با این دستور به روز کنید:

```
sudo  update-grub
```

در ترمینال فرآیند و پیام تأیید چاپ می‌شود.

7. پس از تکمیل مراحل بالا، دستگاه را مجددا راه اندازی کنید. پس از بوت شدن سیستم، نسخه هسته را با استفاده از دستور uname تأیید کنید:

```
uname  -mrs
```

باید نسخه فعلی هسته لینوکس در ترمینال چاپ شود.

11-3-آپدیت هسته لینوکس:

روش اول: استفاده از فرآیند آپدیت سیستم

فرآیند به‌روزرسانی سیستم راهی ساده برای به‌روزرسانی همه بسته‌ها، از جمله هسته لینوکس، به آخرین نسخه‌های موجود است. قبل از شروع، نسخه هسته فعلی را با دستور uname بررسی کنید:

```
uname   -r
```

خروجی دستور عبارتی مشابه 6.2.0-37-generic خواهد بود که در آن ؛ 6.2.0 : شماره نسخه هسته، 37 : شماره انتشار یا ساخت خاص هسته و عبارت genertic : نشان دهنده نوع هسته است. نوع generic یک انتخاب رایج برای سیستم‌های دسکتاپ و سرورهای معمولی است.

اکنون، با استفاده از sudo apt فهرست بسته‌های محلی را به روز رسانی کنید. این دستور اطلاعات مربوط به آخرین نسخه بسته‌های موجود را از مخازن تنظیم‌شده روی سیستم دریافت می‌کند.

```
 sudo  apt   update
```

اگر نسخه هسته جدیدتری وجود داشته باشد، دستور آن را پیدا کرده و برای دانلود و نصب علامت گذاری می‌کند. با این حال، این دستور هیچ بسته ای را نصب یا ارتقا نمی‌دهد.

برای ارتقاء بسته های نصب شده، از جمله هسته، به آخرین نسخه‌های موجود در مخازن، دستور زیر را اجرا کنید:

```
sudo  apt   upgrade
```

به این ترتیب آخرین نسخه هسته لینوکس روی سیستم نصب می‌شود. تغییرات حاصل را با uname -r بررسی کنید. البته اگر هسته از قبل در آخرین نسخه خود بوده باشد، خروجی دستور uname نسبت به حالت قبل از ارتقا، تغییری نخواهد کرد.

نکته: اگر به غیر از به‌روزرسانی نسخه هسته، سیستم خود را به نسخه جدید اوبونتو نیز ارتقا می‌دهید (به عنوان مثال، از اوبونتو 20.04 به 22.04)، از sudo apt-get dist-upgrade استفاده کنید. این دستور تغییرات در پیش‌نیازها و ارتقاهای عمده سیستم را نیز مدیریت می کند.

روش دوم: به‌روزرسانی دستی هسته

برای ارتقاء به جدیدترین هسته (تست نشده) که خطرات احتمالی نیز به همراه دارد، روش دیگری نیز وجود دارد که از Mainline استفاده می‌کند.

این ابزار کاربرپسند به‌روزرسانی هسته لینوکس را ساده می‌کند، اما در اکثر سیستم‌های لینوکس از پیش نصب نشده است.

بنابراین اگر Mainline روی سیستم خود ندارید، ابتدا PPA نگهداری شده توسط cappelikan را به فهرست منابع یا مخازن نرم افزار اضافه کنید.

```
sudo  add-apt-repository    ppa:cappelikan/ppa
```

این PPA توسط کاربر cappelikan نگهداری می شود و حاوی نصاب هسته Mainline است. افزودن این PPA باعث می‌شود که نرم افزار Mainline برای نصب در سیستم اوبونتو در دسترس باشد.

پایگاه داده را با استفاده از دستور زیر به روز رسانی کنید:

```
sudo  apt   update
```

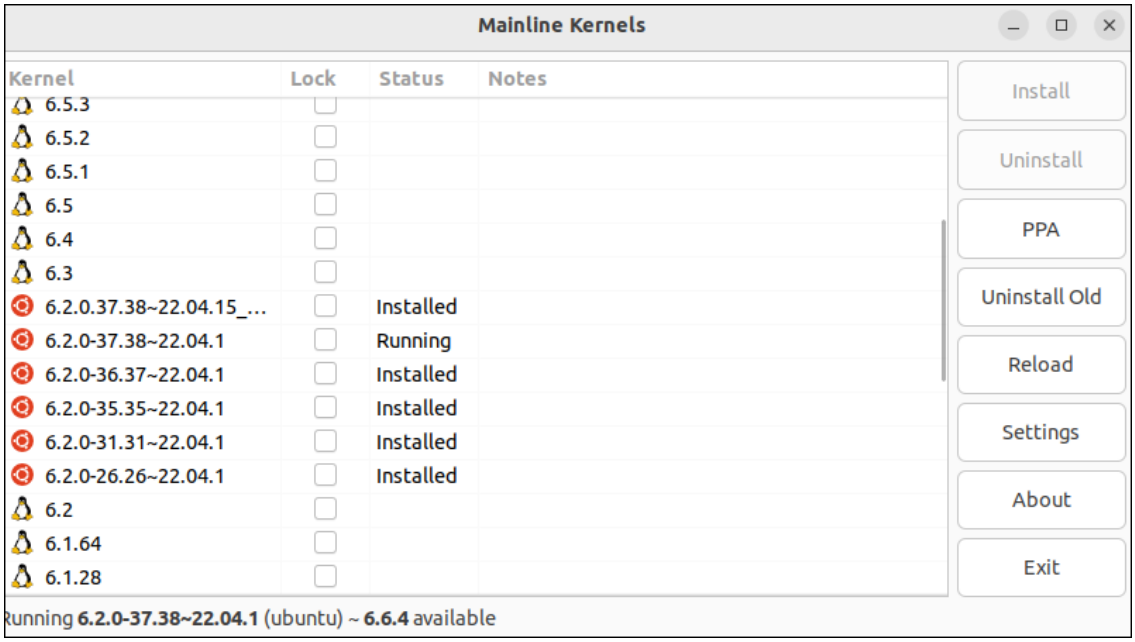
نصب Mainline :

```
sudo  apt   install    mainline
```

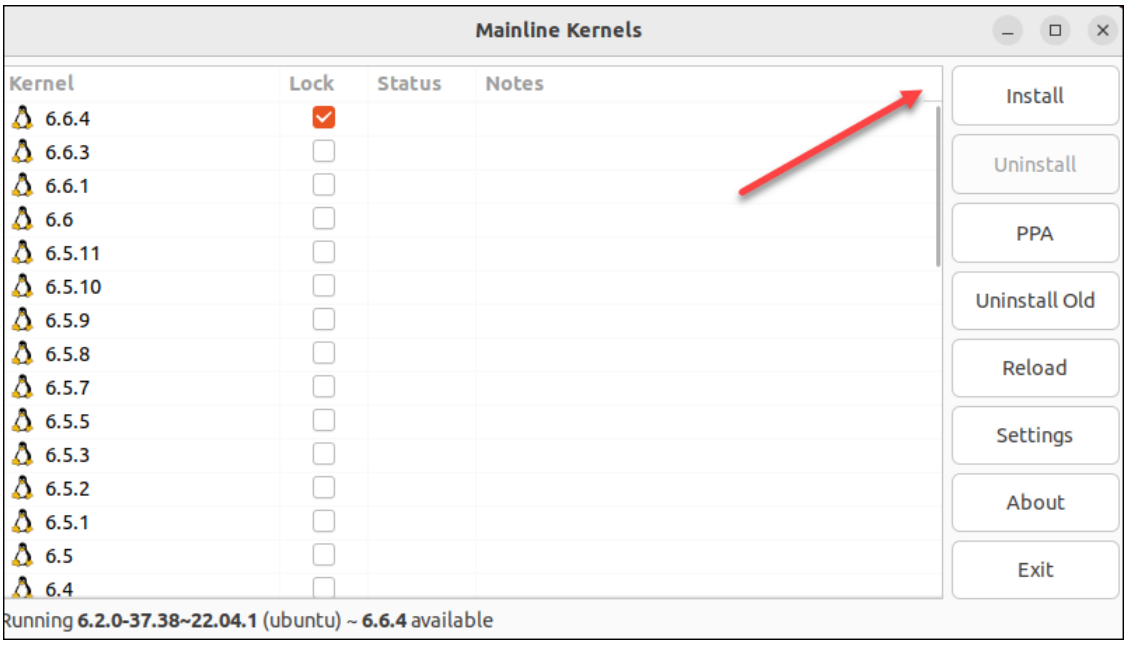
Mainline را اجرا کنید:

```
sudo  mainline
```

رابط Mainline نسخه‌های موجود هسته لینوکس را نمایش می‌دهد. اگر این اتفاق نیفتاد، با زدن سوپرکلید (کلید ویندوز) و جستجوی Mainline به رابط دسترسی پیدا کنید.



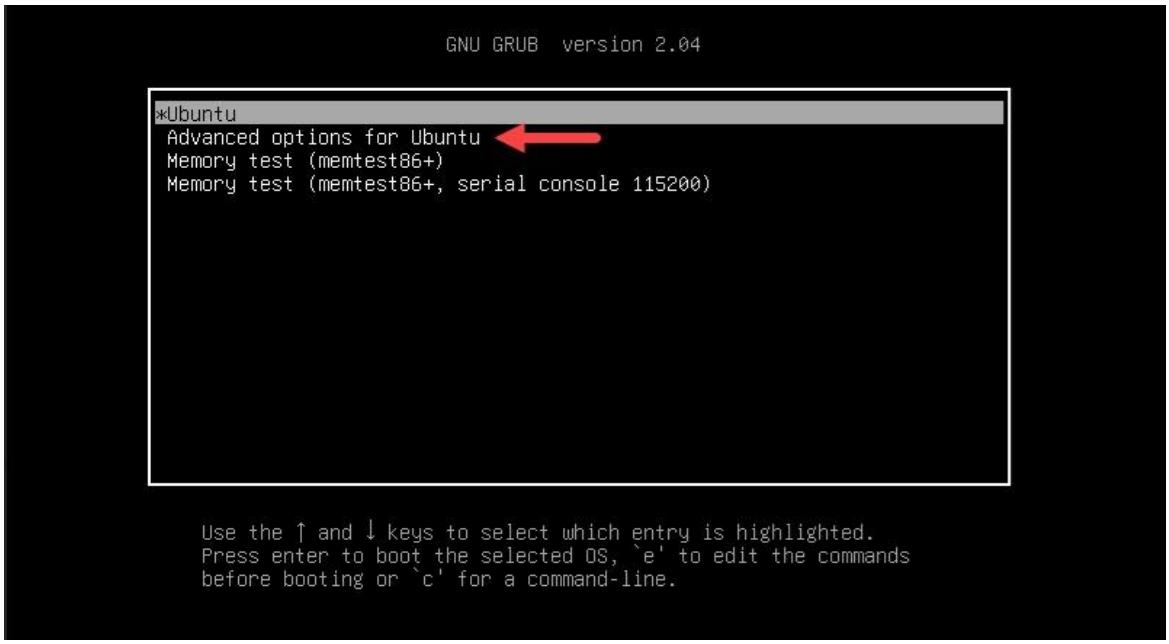
حال برای دریافت یک کرنل خاص، آن را در لیست پیدا کرده و انتخاب کنید. سپس روی دکمه Install در سمت راست کلیک کنید.



پس از اتمام نصب هسته، سیستم را راه اندازی مجدد کنید.

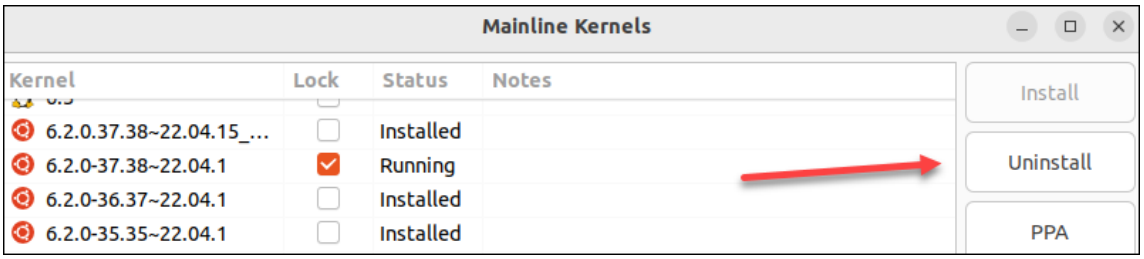
اگر مشکلی وجود دارد، نسخه هسته قبلی را از صفحه بوت انتخاب کنید تا فرآیند برگردانده شود.

- ابتدا سیستم را راه اندازی مجدد کنید و بسته به توزیع لینوکس، کلید Shift یا Esc را نگه دارید.
- سپس گزینه Advanced options for Ubuntu را انتخاب کنید.



سپس هسته قبلی (که با شماره نسخه مشخص شده است) را انتخاب کنید.

روش دیگر برای حذف هسته های قدیمی در اوبونتو، استفاده از Mainline است. کافیست روی نسخه هسته نصب شده قبلی (یا هر نسخه قدیمی تر) کلیک کنید و Uninstall را در سمت راست انتخاب کنید.



روش سوم برای آپدیت هسته از طریق software updater می‌باشد. گاهی اوقات، یک هسته جدیدتر منتشر می شود اما در نسخه جدیدتر اوبونتو قرار نمی‌گیرد. به عنوان مثال، یک کاربر ممکن است از اوبونتو 22.04 استفاده کند درحالیکه اوبونتو 23.10 نسخه هسته جدیدتری دارد. در صورت علاقمندی می‌توانید این روش را از راهنماهای آنلاین، جستجو و دنبال کنید.

تمرین کلاسی:

- انجام مراحل و کامپایل هسته لینوکس

گزارش کار:

- با کمک فایل آموزشی مربوطه یا هر منبع آموزشی دیگر، هسته لینوکس سیستم خود را به روز رسانی کنید.

منابع و مراجع:

- Love R. Linux kernel development. Pearson Education; 2010 Jun 22.
- Salzman PJ, Burian M, Pomerantz O. The Linux Kernel Module Programming Guide. CreateSpace; 2022 Dec 25.

آزمایش 12

مجازی سازی با KVM

اهداف:

- آشنایی با مفاهیم مجازی سازی
- آشنایی و کار با KVM

ابزار و مفاهیم مورد نیاز:

- سیستم عامل لینوکس
- کار با کابران و گروه‌ها
- مدیریت بسته‌ها در لینوکس

محتوا:

- مدیریت کاربران و گروه‌ها در لینوکس
- استفاده از دستور apt
- نصب و تنظیمات KVM
- ایجاد و مدیریت ماشین‌های مجازی با KVM

مفاهیم و شرح دستورکار

12-1- مجازی سازی

فن آوری های مجازی سازی

با ظهور محاسبات سمت سرور به عنوان یک سرویس، ارائه تضمین منابع و ایزوله سازی در محیط چند مستاجری (multi-tenant) از اهمیت بالایی برخوردار شد.

و این ضرورت ایجاد شد که این زیرساخت‌ها، هدف جداسازی (isolation) برنامه‌ها و کارایی منابع را برآورده کنند. برای دستیابی به این اهداف، با توجه به مسائل اقتصادی زیرساخت، باید به سرورها اجازه داده شود که بین چندین کاربر به اشتراک گذاشته شوند و در عین حال جداسازی عملیاتی برنامه‌ها را تضمین کند. مجازی سازی رایج ترین راه حل برای اطمینان از این اهداف است. ادغام (Consolidation) با استفاده از مجازی سازی منجر به جداسازی برنامه، استفاده بهتر از منابع و هزینه‌های عملیاتی کمتر می شود. فن آوری‌های مختلف مجازی سازی در جدول زیر خلاصه شده است.

1whiteblack Type (Classic Definition)	Layer	Virtual Machine Monitor	Virtualization Technology	Example
Type 1	VMM	ISA(HW)	native virtualization	XenServer
Type 1-modified	ISA(HW) and hypercalls	hosted	para-virtualization	Hyper-V, VMware ESX(i)
Type 2	ISA(HW)	hosted	full virtualization	VMware Workstation, virtual box
Container Virtualization	ABI(OS)	kernel	container virtualization	LXC, Docker

جدول. 12-1-تکنولوژی‌های مجازی سازی

دو نوع مجازی سازی وجود دارد: Type1 و Type2. با این حال، برای اینکه بتوانیم تکنیک های مجازی سازی را بدون همپوشانی در ویژگی‌های آنها طبقه‌بندی کنیم، Type1 را به دو کلاس مختلف تقسیم می کنیم.

علاوه بر این، طبقه بندی کلاسیک شامل فناوری اخیر، یعنی مجازی سازی کانتینر نمی‌شود

Para-Virtualization (Type1 – تغییر داده شده) نوعی مجازی سازی است که در آن سیستم‌عامل مهمان (سیستم عاملی که مجازی شده است) می‌داند که مهمان است و بر این اساس درایورهایی دارد که به جای صدور دستورات

سخت افزاری، دستوراتی را به مانیتور ماشین مجازی صادر می کنند. این شامل مدیریت حافظه و رشته نیز می شود که معمولاً به دستورالعمل های غیرقابل دسترسی در پردازنده نیاز دارد.

از طرفی در مجازی سازی کامل (Type2) سیستم عامل مهمان از قرار گرفتن در محیط مجازی بی اطلاع است. بنابراین، سخت‌افزار توسط سیستم‌عامل میزبان مجازی‌سازی می‌شود تا مهمان بتواند فرمان‌هایی را به آنچه که فکر می‌کند سخت‌افزار واقعی است صادر کند، اما در واقع فقط با دستگاه‌های سخت‌افزاری شبیه‌سازی شده توسط میزبان سروکار دارد.

مجازی سازی بومی (Type1) نوعی مجازی سازی کامل است که در آن معماری ریزپردازنده دستورالعمل‌های ویژه ای برای کمک به مجازی سازی سخت افزار دارد. این دستورالعمل‌ها ممکن است اجازه دهند یک زمینه مجازی تنظیم شود تا مهمان بتواند دستورالعمل‌های ممتاز را مستقیماً روی پردازنده بدون تغییر میزبان اجرا کند. چنین مجموعه‌ای از ویژگی‌ها اغلب مانیتور ماشین مجازی نامیده می‌شود. حتی اگر دستورالعمل‌های مذکور وجود نداشته باشد، مجازی-سازی کامل همچنان امکان پذیر است. با این حال، این کار باید از طریق تکنیک‌های نرم‌افزاری مانند کامپایل مجدد پویا انجام شود که در آن میزبان دستورالعمل‌های ممتاز جاری را در میهمان دوباره کامپایل می‌کند تا بتواند به عنوان دستور غیرممتاز روی میزبان اجرا شود.

راه‌حل‌های مجازی‌سازی مبتنی بر ابر، عمدتاً به مجازی‌سازی مبتنی بر Hypervisor (type1) متکی هستند، زیرا می‌تواند از انواع مختلف سیستم‌عامل مانند لینوکس، ویندوز و غیره پشتیبانی کند. بعلاوه، با گسترش مجازی‌سازی سخت‌افزار در اکثر معماری‌های مدرن، این امر به سیستم‌عامل‌های مهمان اجازه می‌دهد بدون تغییر اجرا شوند. با این حال، ممکن است مجازی‌سازی مبتنی بر Hypervisor از کاهش کارایی رنج ببرد، زیرا در معماری مجموعه دستورالعمل (ISA)، باید متحمل سربار مدیریت VM توسط Hypervisor شود.

به این ترتیب، مجازی‌سازی میزبان (type2) نیز در حال تبدیل شدن به یک جایگزین گسترده است زیرا هر مهمان می‌تواند هسته خود را داشته باشد و میزبان شامل یک هسته اصلاح‌شده با extensionهایی برای مدیریت و اجرای چندین VM است.

از طرف دیگر، از آنجاییکه مجازی‌سازی کانتینر، لایه اضافی مسیریابی ندارد و هسته یکسانی را به اشتراک می‌گذارد، می‌تواند برنامه‌ها را با سرعتی نزدیک به بومی اجرا کند. این نوع مجازی سازی، لایه Application Binary Interface (ABI) را مجازی می کند. با این وجود، از آنجایی که هسته مشترک است، فقط می‌توانند سیستم عامل‌های مهمانی را اجرا کنند که از هسته میزبان پشتیبانی می‌کنند.

LXC (Linux Container) ، یک نمونه نوعی از مجازی سازی کانتینر است. LXC یک روش مجازی سازی در سطح سیستم عامل برای اجرای چندین سیستم لینوکس ایزوله (کانتینر) روی یک میزبان کنترلی با استفاده از یک هسته لینوکس است. داکر نیز می‌تواند از LXC به عنوان یکی از درایورهای اجرایی خود استفاده کند که مدیریت image و ارائه خدمات استقرار را ممکن می‌سازد. محفظه‌های Docker تکه‌ای از یک نرم‌افزار را در یک فایل سیستم کامل می‌پیچند که شامل همه چیزهایی است که برای اجرا نیاز دارد: کد، زمان اجرا، ابزارهای سیستم، کتابخانه‌های سیستم، یعنی هر چیزی که می‌توانید روی یک سرور نصب کنید. این کار تضمین می کند که بدون در نظر گرفتن محیط بومی که در آن اجرا می شود، همیشه اجرای یکسانی را ارائه می‌دهد.

تراکم مجازی سازی به تعداد ماشین های مجازی اشاره دارد که یک میزبان فیزیکی می‌تواند در خود داشته باشد به طوریکه برای هر ماشین مجازی، منابع محاسباتی کافی برای عملکرد خوب فراهم کند. این به عوامل متعددی مانند سخت افزار سرور، نرم افزار مجازی سازی، نوع سرویس و تنوع حجم کاری بستگی دارد. این عوامل مختلف، دستیابی به یک عدد مطلق برای تراکم ماشین مجازی را در تمام سناریوها دشوار می‌کند. افزایش تراکم VM هزینه‌های متحمل شده را کاهش می‌دهد، اما در عین حال چالش های بیشتری را در تضمین کارایی به دلیل مشاجره برای منابع مشترک ایجاد می‌کند. معمولاً گلوگاه در زیرسیستم حافظه و مقدار حافظه دردسترس، آشکار می‌شود. ماشین‌های commodity در چنین منابعی کمیاب هستند و به این ترتیب، سیستم از تدارک محافظه‌کارانه برای تضمین best-effort ، بهره‌مند خواهد شد.

مجازی سازی در زمینه P2P

اگرچه تکنیک‌های مجازی سازی مختلفی وجود دارد، مجازی سازی در زمینه P2P چالش های خاصی دارد که باید مورد توجه قرار گیرد. P2P از ماشین های commodity تشکیل شده است که در درجه سرور نیستند و در نتیجه

قابلیت محاسباتی بالایی ندارند. در مراکز داده، گرچه فناوری مجازی سازی متداول، فناوری مبتنی بر Hypervisor است؛ ولی مجازی سازی میزبانی شده نیز رواج یافته است. سرورهای P2P از نظر منابع دردسترس، محدود شده‌اند، بنابراین حصول اطمینان از اینکه مجازی‌سازی هزینه بالایی را بر روی این ماشین‌های با منابع محدود، تحمیل نمی‌کند، ضروری است.

تکنیک‌های مجازی‌سازی مبتنی بر Hypervisor مانند Xen به دلیل وجود یک مرحله مسیریابی اضافی، سربار بالایی را تحمیل می‌کنند، زیرا اجرای هر سیستم عاملی را پشتیبانی می‌نمایند. بنابراین، مجازی‌سازی کانتینر به دلیل سرعت نزدیک به اجرای بومی و همراه با سربار کم، برای چنین محیط‌هایی با منابع محدود مناسب‌تر است. مجازی‌سازی میزبانی‌شده نیز یک جایگزین جذاب برای محیط‌های با منابع محدود است زیرا حداقل هزینه‌های سربار را تحمیل می‌کند. با این حال، هر مهمان می‌تواند هسته خود را داشته باشد که بر کارایی ایجاد یک VM جدید تأثیر دارد. LXC از این منظر سربار کمتری دارد. علیرغم وجود مزایا و معایب نسبی، LXC به نظر می‌رسد انتخاب معقولی از تکنیک مجازی‌سازی برای استفاده در محیط P2P باشد.

12-2-نصب KVM در اوبونتو

مرحله 1: نصب بسته های مورد نیاز

در خط فرمان دستور زیر را برای نصب بسته های مورد نیاز اجرا کنید:

```
$ sudo apt -y install bridge-utils cpu-checker libvirt-clients
libvirt-daemon qemu qemu-kvm
```

مرحله 2: بررسی قابلیت‌های مجازی سازی

برای اطمینان از اینکه پردازنده شما از قابلیت‌های مجازی‌سازی پشتیبانی می‌کند دستور زیر را اجرا کنید:

```
$ kvm-ok
```

خروجی این دستور بسیار ساده و واضح است و نشان می دهد که آیا KVM قابل استفاده است یا خیر:

```
INFO: /dev/kvm exists
```

```
KVM acceleration can be used
```

مرحله 3: راه‌اندازی VM

به منظور راه اندازی اولین نمونه Ubuntu Server خود بر روی KVM دستور زیر را اجرا کنید:

اینجا از ورژن 20.04 اوبونتو استفاده شده است.

```
$ sudo virt-install --name ubuntu-guest --os-variant ubuntu20.04 --
vcpus 2 --ram 2048 -- location
http://ftp.ubuntu.com/ubuntu/dists/focal/ main/installer-amd64/ --
network bridge=virbr0,model=virtio --graphics none -- extra-
args='console=ttyS0,115200n8 serial'
```

با اجرای دستور بالا، یک کنسول تعاملی باز می‌شود که می‌توانید از آن برای نصب اوبونتو مهمان به صورت دستی استفاده کنید.

12-3-نصب KVM (ماشین مجازی مبتنی بر کرنل) و QEMU (Quick Emulator) در اوبونتو

پیش نیازها:

- CPU با پشتیبانی از مجازی سازی سخت افزاری (Intel VT-x یا AMD-V)
- یک سیستم اوبونتو (دستورات زیر برای اوبونتو 22.04/20.04 است.)

مرحله 1: بررسی پشتیبانی از مجازی سازی

ابتدا مطمئن شوید که CPU شما از فناوری مجازی سازی پشتیبانی می کند.یک ترمینال باز کنید و دستور زیر را اجرا نمایید:

```
egrep       -c    '(vmx|svm)'     /proc/cpuinfo
```

اگر خروجی بزرگتر از 0 باشد، مجازی سازی پشتیبانی می شود.

مرحله 2: به روزرسانی سیستم

لیست بسته خود را به روز کنید تا مطمئن شوید آخرین نسخه‌های نرم افزاری را دارید.

```
$ sudo          apt    update
$ sudo          apt    upgrade
```

مرحله 3: نصب KVM و QEMU

بسته های لازم را با اجرای دستورات زیر نصب کنید:

```
$ sudo apt install qemu-kvm libvirt-daemon-system libvirt-clients
bridge-utils virt-manager
```

مرحله 4: افزودن کاربر به گروه‌ها

کاربر خود را به گروه‌های libvirt و kvm اضافه کنید تا بتوانید VMها را بدون امتیاز کاربر روت مدیریت کنید.

```
$ sudo     adduser     `whoami`    libvirt
$sudo      adduser     `whoami`    kvm
```

مرحله 5: تأیید نصب

بررسی کنید که ماژول KVM بارگذاری شده باشد:

```
lsmod | grep kvm
```

مرحله 6: نصب Virt-Manager (اختیاری)

برای داشتن یک رابط کاربری گرافیکی برای مدیریت ماشین‌های مجازی، Virt-Manager را نصب کنید:

```
$ sudo     apt   install    virt-manager
```

مرحله 7: راه اندازی مجدد سیستم

برای اعمال تغییرات، سیستم خود را مجددا راه اندازی کنید

```
$ sudo      reboot
```

پس از راه اندازی مجدد، می توانید از Virt-Manager برای ایجاد و مدیریت ماشین‌های مجازی استفاده کنید.

12-4- نصب و پیکربندی KVM برای اتصال به شبکه

مرحله 1: نصب بسته‌های KVM

ابتدا مطمئن شوید که سیستم شما از مجازی سازی پشتیبانی می کند و بسته‌های ضروری KVM را نصب کنید:

```
$ sudo apt update –y
$ sudo apt install qemu-kvm libvirt-clients libvirt-daemon-system
virtinst bridge-utils -yStep
```

مرحله 2: افزودن کاربر به گروه libvirt

برای مدیریت KVM بدون امتیاز کاربر روت، کاربر خود را به گروه libvirt اضافه کنید.

```
$sudo       adduser     $(whoami)         libvirt
```

مرحله 3: راه اندازی و فعال سازی سرویس libvirt

```
$ sudo      systemctl         start       libvirtd
$ sudo      systemctl         enable      libvirtd
```

مرحله 4: بررسی وضعیت KVM

بررسی کنید که ماژول های KVM بارگذاری شده اند:

```
lsmod | grep kvm
```

مرحله 5: پیکربندی شبکه برای KVM

شبکه را برای محیط KVM خود راه اندازی کنید. می توانید از شبکه پیش‌فرض NAT استفاده کنید یا یک شبکه bridged برای تنظیمات پیچیده‌تر ایجاد نمایید.

برای شبکه NAT

```
$ sudo virsh net-list –all
$ sudo virsh net-start default
$ sudo virsh net-autostart default
```

برای شبکه Bridged

یک واسط از نوع پل در /etc/network/interfaces ایجاد کنید و سرویس شبکه را مجددا راه اندازی نمایید.

```
$ sudo vim /etc/network/interfaces
# Add your bridge configuration here
$ sudo systemctl reload networking.serviceStep
```

مرحله 6: ایجاد و مدیریت ماشین های مجازی

از دستورات virt-manager یا virsh برای ایجاد و مدیریت ماشین های مجازی خود استفاده کنید.

استفاده از virt-manager

virt-manager را نصب و راه اندازی کنید، سپس دستورات رابط کاربری گرافیکی را برای ایجاد یک ماشین مجازی جدید دنبال کنید.

استفاده از :virsh

با استفاده از دستور virt-install یک VM ایجاد کنید:

```
$ virt-install \
--name myvm \
--ram 2048 \
--disk path=/var/lib/libvirt/images/myvm.img,size=10 \
--vcpus 1 \--os-type linux \
--os-variant ubuntu20.04 \
--network bridge=br0 \
--graphics none \
--console pty,target_type=serial \
--location 'http://archive.ubuntu.com/ubuntu/dists/focal/main/
installer-amd64/' \
--extra-args 'console=ttyS0,115200n8 serial'
```

مرحله 7: دسترسی به ماشین های مجازی

با استفاده از virt-manager به VM های خود دسترسی پیدا کنید یا از طریق SSH یا کنسول متصل شوید.

مرحله 8: مدیریت ماشین های مجازی

با استفاده از دستورات virsh، ماشین‌های مجازی را فهرست، شروع، متوقف و حذف کنید.

```
$ virsh list –all
$ virsh    start       myvm
$ virsh    shutdown   myvm
$ virsh    undefine   myvm
```

مرحله 9: پیکربندی پیشرفته

ویژگی‌های پیشرفته KVM مانند پین کردن CPU، تنظیم حافظه، و snapshots را برای تنظیمات بهینه و انعطاف‌پذیرتر VM کاوش کنید.

برای دستورالعمل های دقیق متناسب با توزیع و نیازهای خود، به اسناد رسمی KVM مراجعه کنید.

گزارش کار:

- KVM ار روی سیستم خود نصب کرده و سپس مراحل ایجاد، مدیریت و حذف ماشین مجازی را با یکی از روش‌های ذکر شده در دستورکار انجام دهید.

منابع و مراجع:

- https://www.linux-kvm.org/page/Networking, visited 2024 Aug.
- https://ubuntu.com/blog/kvm-hyphervisor, visited 2024 Aug.
- https://www.tecmint.com/install-qemu-kvm-ubuntu-create-virtual-machines,visited 2024 Aug.
- https://linuxconfig.org/setting-up-virtual-machines-with-qemu-kvm-and-virt-manager-on-debian-ubuntu, visited 2024 Aug.

آزمایش 13

مجازی سازی با ESX

اهداف:

- آشنایی با نصب و مدیریت ماشین مجازی با ESX

ابزار و مفاهیم مورد نیاز:

- سیستم عامل لینوکس
- ssh

محتوا:

- نصب ESX
- مدیریت ماشین مجازی در ESX
  - ایجاد
  - اتصال به هاست
  - پیکربندی
  - رجیستر
  - حذف
  - پاکسازی

13-1- ایجاد VM در ESX

ایجاد یک ماشین مجازی (VM) جدید در ESXi از طریق خط فرمان فرآیند ساده‌ای است کافیست مراحل زیر را به درستی دنبال کنید.

1. اتصال به ESXi Host از طریق SSH : قبل از شروع، مطمئن شوید که به هاست ESXi خود دسترسی SSH دارید. می توانید از یک کلاینت SSH مانند PuTTY یا ترمینال در Linux/Mac استفاده کنید.

2. مسیریابی دایرکتوری: پس از اتصال، به دایرکتوری که می‌خواهید فایل‌های VM خود را در آن ذخیره کنید، بروید. که معمولاً در مسیر /vmfs/volumes/datastore_name/ قرار دارد.

3. ایجاد دایرکتوریVM : با استفاده از دستور mkdir یک دایرکتوری جدید برای VM خود ایجاد کنید. به عنوان مثال:

```
mkdir      /vmfs/volumes/datastore_name/MyNewVM
```

4. ایجاد فایل پیکربندی VM : لازم است یک فایل VMX ایجاد کنید، که فایل پیکربندی ماشین مجازی شما است. برای ایجاد این فایل با پارامترهای پیکربندی لازم می توانید از ویرایشگر vi (یا هر ویرایشگر مناسب دیگر) استفاده کنید. یک نمونه دستور برای ایجاد یک فایل VMX را در زیر می‌بینید:

```
vi     /vmfs/volumes/datastore_name/MyNewVM/MyNewVM.vmx
```

5. ویرایش فایل VMX : در داخل فایل VMX، مشخصات سخت افزاری ماشین مجازی مانند تعداد CPU، مقدار حافظه و پیکربندی دیسک را مشخص خواهید کرد. محتویات این فایل مشابه مواردی است که در ادامه آمده است:

```
.encoding = "UTF-8"
displayName = "MyNewVM"
guestOS = "other-64"
memsize = "4096"
```

```
numvcpus = "2"
scsi0:0.fileName = "MyNewVM.vmdk"
scsi0:0.present = "TRUE"
scsi0.present = "TRUE"
```

6. رجیستر کردن VM : پس از ایجاد فایل VMX، باید VM را در ESXi ثبت کنید. برای این کار از دستور vimcmd استفاده می‌شود:

```
vim-cmd    solo/registervm
/vmfs/volumes/datastore_name/MyNewVM/MyNewVM.vmx
```

7. روشن کردن ماشین مجازی: در نهایت، می توانید VM را با استفاده از یک دستور vim-cmd دیگر روشن کنید:

```
vim-cmd    vmsvc/power.on   vmid
```

توجه: vmid را با شناسه VM واقعی که توسط دستور registervm قبلی برگردانده شده است جایگزین کنید.

13-2- حذف VM

برای حذف ماشین مجازی از طریق خط فرمان در ESXi، می توانید از دستورات ESXCLI استفاده کنید. مرحل این کار در زیر توضیح داده شده است:

1. فعال کردن ESXi Shell یا SSH Access : پوسته ESXi به صورت پیش فرض غیرفعال است. باید آن را به صورت محلی با استفاده از رابط کاربری مستقیم کنسول (DCUI) و یاا از راه دور از طریق SSH فعال کنید.

2. ورود به ESXi Host : اگر از SSH استفاده می کنید، با استفاده از یک کلاینت SSH و با در دست داشتن آدرس IP میزبان ESXi، می‌توانید به هاست ESXi متصل شوید. اگر از کنسول محلی استفاده می کنید، Alt+F1 را بفشارید تا پس از فعال‌سازی به پوسته دسترسی پیدا کنید.

3. لیست کردن ماشین‌های مجازی: از دستور زیر برای لیست کردن همه ماشین‌های مجازی استفاده کنید و VMID ماشین مجازی را که می خواهید حذف کنید یادداشت کنید.

```
vim-cmd    vmsvc/getallvms
```

4. خاموش کردن VM : اگر ماشین مجازی در حال اجرا است، آن را با دستور زیر خاموش کنید.

```
vim-cmd    vmsvc/power.off       VMID
```

5. لغو رجیستر (Unregister) ماشین مجازی: رجیستر ماشین مجازی موردنظرتان را با اجرای دستور زیر از میزبان VMID ESXi لغو کنید.

```
vim-cmd    vmsvc/unregister      VMID
```

6. پاک کردن فایل‌های VM :

در نهایت فایل‌های VM را از datastore حذف کنید. می توانید به دایرکتوری datastore که فایل های VM در آن قرار دارند بروید و از دستور rm برای حذف آنها استفاده کنید. به عنوان مثال:

```
rm    -rf   /vmfs/volumes/datastore_name/VM_directory
```

به یاد داشته باشید که VMID، datastore_name و VM_directory را با مقادیر واقعی مربوط به محیط خود جایگزین کنید. همیشه قبل از انجام چنین عملیاتی از داشتن نسخه پشتیبان اطمینان حاصل کنید، زیرا این اقدامات غیرقابل برگشت هستند.

برای اطلاعات دقیق تر یا در صورت مواجه شدن با هر گونه مشکلی، می توانید به مستندات رسمی VMware یا فروم انجمن مربوطه مراجعه کنید. همچنین هنگام کار با ESXi و ماشین های مجازی همیشه از داشتن مجوزهای لازم اطمینان حاصل کنید و از best practices استفاده کنید.

تمرین کلاسی:

- تمام مراحل نصب، ایجاد، اتصال به هاست و پیکربندی را مطابق دستور کار انجام دهید.

منابع و مراجع:

- Marshall N, Brown M, Fritz GB, Johnson R. Mastering VMware vSphere 6.7. John Wiley & Sons; 2018 Oct 9.
- https://core.vmware.com/esxi, visited 2024 Aug.
- https://community.broadcom.com/vmware-cloud-foundation/communities/community-home?CommunityKey=185ca20b-b56d-41b3-a4b8-aad095f0d970, visited 2024 Aug.
- https://docs.vmware.com/en/VMware-vSphere/index.html, visited 2024 Aug.
- https://forums.servethehome.com/index.php, visited 2024 Aug.
- https://www.nakivo.com/blog/most-useful-esxcli-esxi-shell-commands-vmware-environment, visited 2024 Aug.