دانشگاه صنعتی شریف

دستور کار آزمایشگاه سیستم‌عامل

تابستان ۱۳۹۸

فهرست مطالب

آزمایش ۱

آشنایی با سیستم عامل لینوکس

در این جلسه از آزمایشگاه نحوه نصب سیستم عامل لینوکس دستورات اولیه و پرکاربرد این سیستم عامل و همچنین آشنایی با روش های اعمال تغییرات در هسته ی این سیستم عامل مورد بررسی قرار خواهد گرفت.

۱.۱ اهداف

انتظار می رود در پایان این جلسه دانشجویان مطالب زیر را فرا گرفته باشند:
آشنایی با نحوه نصب یک توزیع لینوکس به صورت مجازی آشنایی با دستورات اولیه سیستم عامل لینوکس و کار با فایل‌ها کامپایل و اجرای کد در محیط لینوکس آشنایی با نحوه اعمال تغییرات در هسته لینوکس کامپایل مجدد و نصب آن

۲.۱ پیش‌نیازها

انتظار می‌رود که دانشجویان با موارد زیر از پیش آشنا باشند:
برنامه نویسی به زبان C/C++ همچنین نرم‌افزارهای زیر برای انجام آزمایشات این دستور کار الزامی هستند:
یک نرم‌افزار برای نصب سیستم‌عامل مجازی مانند VirtualBox, Parallels Desktop, VMware و ... فایل مورد نیاز برای نصب سیستم‌عامل Debian 8

۳.۱ شرح آزمایش

۱.۳.۱ نصب سیستم عامل‌لینوکس

به دلیل ساده سازی فرایند اعمال تغییرات در سیستم عامل و همچنین توانایی بازیابی در مقابل خطاهای احتمالی که در این جریان ممکن است روی دهد، از نسخه مجازی استفاده می‌کنیم. مناسب است که همواره یک نسخه پشتیبان از سیستم‌عامل مجازی خود داشته باشید. در تمامی مراحل آزمایش از رابطه متنی سیستم‌عامل لینوکس استفاده خواهد شد.

۱. یک نسخه از سیستم‌عامل Debian را با تنظیمات پیش‌فرض به صورت مجازی نصب کنید. توجه داشته باشید که برای کامپایل هسته نیاز به حداقل ۲۰ گیگابایت فضا خواهید داشت، بنابراین در هنگام ایجاد سیستم‌عامل مجازی آن را در نظر بگیرید. حداقل حافظه مورد نیاز نیز ۵۱۲ مگابایت خواهد بود.

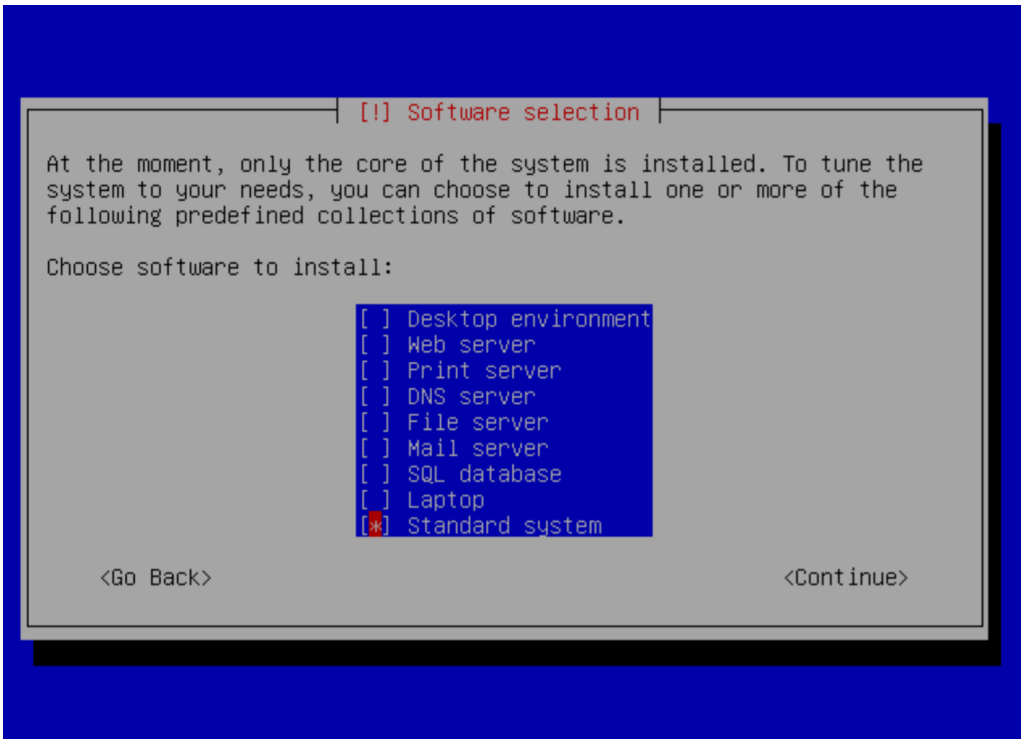
۲. نصب سیستم‌عامل را به صورت کمینه (minimal) انجام دهید. به این ترتیب تنها بسته‌های ضروری نصب خواهند شد. برای این کار کافی است در هنگام نصب در گام Software Selection تنها گزینه‌ی Standard System را انتخاب کنید.
در صورت نیاز به جزئیات و راهنمایی‌هایی بیشتر به [۱] مراجعه کنید.

۲.۳.۱ آشنایی با دستورات پایه‌ی لینوکس

برای دریافت راهنمایی در مورد هرکدام از دستورات ارائه شده در ادامه می‌توانید از دستور man [command] استفاده کنید. در گزارش خود، دستورات مورد استفاده در هریک از مراحل زیر را بیاورید.

۱. به کمک دستور pwd آدرس دایرکتوری جاری را نمایش دهید.

۲. به کمک دستور cd به داخل دایرکتوری tmp/ رفته و به کمک دستور mkdir یک پوشه به نام oslab1 ایجاد کنید.



شکل ۱.۱: مرحله انتخاب بسته‌های مورد نیاز برای نصب سیستم‌عامل

۳. به کمک ویراشگر nano یک فایل متنی با محتوای نام و شماره‌ی دانشجویی خود به اسم information.txt ایجاد کنید و در نهایت از ویراشگر خارج شوید.

۴. به کمک دستور mv نام فایل را به myinformation.txt تغییر دهید.

۵. به کمک دستور cp یک کپی از این فایل به اسم backupinfo.txt را در همان شاخه ایجاد کنید.

۶. محتوای فایل myinformation.txt را به کمک دستور cat نشان دهید.

۷. دستورات زیر را اجرا کنید:

برنامه ۱.۱: نمونه دستورات

```
1 # echo "Hello There!" > myinformation.txt
2 # echo "Hello World!" > > myinformation.txt
```

تفاوت این دو دستور را شرح دهید.

۸. یک فایل متنی جدید با محتوای دلخواه را به کمک دستور cat (بدون استفاده از nano) به نام testfile.txt ایجاد کنید.

۹. لیست پردازه‌های در حال اجرا را به کمک دستور ps aux نمایش دهید.

۱۰. به کمک دستور grep لیست پردازه‌هایی را نشان دهید که در نام آن‌ها حرف a وجود دارد.

۱۱. به کمک دستور cd به داخل شاخه‌ی /usr/bin رفته و به کمک دستور ls لیست فایل‌های موجود در آن را نمایش دهید. فایل‌های موجود در این پوشه بخشی از دستورات قابل اجرا در سیستم هستند.

۱۲. به کمک دستور ls و استفاده از پارامترهای مناسب، علاوه بر نام فایل‌ها، حجم آن‌ها را نمایش دهید.

۱۳. به کمک دستور grep لیست فایل‌های در این پوشه را نشان دهید که در آن‌ها کلمه fs یا ld وجود دارد.

۴.۱ فعالیت‌ها

- کاربرد دستورات زیر را به اختصار بیان کنید:

```
cut / find / head / tail / touch / wc / kill
```

- با کمک دستوراتی که فراگرفته‌اید، فرمان‌هایی برای اعمال زیر بنویسید:
  - پیدا کردن تعداد خطوط در یک فایل متنی به نام mybook.txt
  - پیدا کردن تعداد فایل‌هایی که با حرف A شروع می‌شوند.
  - پیدا کردن حجم فایل mybook.txt

۱.۴.۱ اعمال تغییرات و کامپایل مجدد هسته‌ی سیستم‌عامل

۱. ابتدا کد منبع هسته را دریافت کنید. برای این کار از دستور زیر استفاده کنید:

```
# apt-get install linux-source- 3 . 2
```

۲. ابزارهای لازم برای کامپایل و نصب هسته را دریافت کنید:

```
# apt-get install build-essential fakeroot
# apt-get install build-dep linux
```

۳. به کمک دستور زیر، کدهای هسته را در یک پوشه مشخص بازگشایی کنید:

```
# apt-get source linux
```

۴. یک پوشه با نام linux-source-3.2 ایجاد شده که حاوی کد هسته‌ی لینوکس می‌باشد.

۲.۴.۱ فعالیت‌ها

- به کمک [۲] نحوه کامپایل کردن هسته و نصب آن را به اختصار بیان کنید. سپس هسته‌ی سیستم‌عامل را یک بار کامپایل نمایید. در دفعه اول این کار زمان‌گیر خواهد بود، ولی عملیات را برای دفعات بعد تسریع خواهد کرد.

کتابنامه

[1] http://tuxonomy.wordpress.com/2010/04/15/debian-minimal-install-of-a-base-system-lenny- aka-5-0/
[2] http://kernel-handbook.alioth.debian.org/

آزمایش ۲

آشنایی با فراخوانی‌های سیستمی

۱.۲ مقدمه

در این جلسه از آزمایشگاه با برخی از مهمترین فراخوانیهای سیستمی در سیستم‌عامل لینوکس آشنا خواهیم شد و به کمک آنها چند برنامه خواهیم نوشت. همچنین روش اضافه کردن فراخوانی‌های سیستمی به هسته لینوکس را خواهیم آموخت.

۱.۱.۲ اهداف

انظار می‌رود که در پایان این جلسه دانشجویان مطالب زیر را فراگرفته باشند

- آشنایی با مفهوم فراخوانی سیستمی.
- نحوه‌ی اجرای فراخوانی‌های سیستمی.
- فراخوانی‌های سیستمی مهم و پرکاربرد در سیستم‌عامل لینوکس.
- نحوه ایجاد فراخوانی‌های سیستمی جدید.

۲.۱.۲ پیش‌نیازها

انتظار می‌رود که دانشجویان با موارد زیر از پیش آشنا باشند:

- برنامه‌نویسی به زبان c/c++

۲.۲ فراخوانی سیستمی چیست؟

فراخوانی سیستمی یا call system تابعی است که در هسته‌ی سیستم‌عامل پیاده‌سازی شده است و هنگامی که یک برنامه یک فراخوانی سیستمی انجام می‌دهد، کنترل اجرا از آن برنامه به هسته منتقل می‌شود تا عملیات درخواست شده صورت پذیرد. فراخوانی‌های سیستمی برای اعمال مختلفی مانند دسترسی به منابع، تخصیص آنها، خاموش کردن یا راه‌اندازی مجدد سیستم‌عامل و ... مورد استفاده قرار می‌گیرد. برخی از این فراخوانی‌های سیستمی تنها در پروسه‌هایی قابل استفاده است که توسط super-user اجرا شده باشند.

هر فراخوانی سیستمی با یک شماره ثابت شناخته می‌شود که این شماره پیش از کامپایل شدن هسته باید مشخص گردد. به همین دلیل د سیستم‌عاملللینوکس افزودن فراخوانیهای سیستمی تنها با کامپایل و نصب مجدد هسته امکان‌پذیر است.

برای اطلاعات بیشتر در مورد فراخوانی‌های سیستمی در لینوکس به [۱] مراجعه کنید.

۳.۲ شرح آزمایش

۱.۳.۲ مشاهده فراخوانی‌های سیستمی تعریف شده

۱. وارد سیستم عامل مجازی ایجاد شده در جلسه قبل شوید.

۲. سیستم عامل لینوکس در حال حاضر شامل بیش از ۳۰۰ فراخوانی سیستمی است. فایل زیر را به کمک یک ویرایشگر باز کنید؛ در این فایل می توانید لیست فراخوانی‌های سیستمی به همراه شماره‌ی آنها را بیابید

```
/usr/include/i386-linux-gnu/asm/unistd_32.h
```

۲.۳.۲ اجرای یک فراخوانی سیستمی

۱. در پوشه‌ی خانه خود یک فایل testsyscall.cpp ایجاد کنید.

۲. کد زیر با استفاده از فراخوانی سیستمی mkdir یک پوشه جدید ایجاد می کند. آن را در فایلی که در مرحله قبل لیجاد کرده‌اید وارد کنید:

برنامه ۱.۲: برنامه نمونه برای ایجاد یک پوشه

```
#include <stdio.h>
#include <unistd.h>
#include <sys/syscall.h>
int main () {
  long result;
  result = syscall(__NR_mkdir, "testdir", 0777);
  printf("The result is %ld.\n", result);
  return 0;
}
```

۳. کد را کامپایل کنید و سپس اجرا نمایید.

۴. نتیجه اجرای آن را شرح دهید.

تمرین ۱.۲ در مثال بالا، نقش *__NR_mkdir* چیست؟

تمرین ۲.۲ در مورد نحوه استفاده از دستور syscall و ورودی‌ها و خروجی‌های آن توضیح دهید.

۳.۳.۲ اجرای ساده‌تر فراخوانی سیستمی

برای کاربرد ساده‌تر فراخوانی‌های سیستم بدون نیاز به شماره آن‌ها، می‌توان از توابعی استفاده کرد که از پیش به عنوان wrapper برای آن‌ها نوشته شده‌اند. برای مثال برای فراخوانی سیستمی بخش قبلی می‌توان از تابع mkdir() که در <sys/stat.h> قرار دارد استفاده کرد. به دلیل خوانایی بالاتر سادگی کاربرد، معمولاً ترجیح بر استفاده از این توابع به جای استفاده مستقیم از دستور syscall است.

تمرین ۳.۲

- کد بخش قبل را به کمک تابع mkdir() بازنویسی کنید و در فایل testsyscall۲.cpp ذخیره نمایید.

۴.۳.۲ آشنایی با چند فراخوانی سیستمی پرکاربرد

در هر کدام از فعالیت‌های این بخش، یک فراخوانی سیستمی معروف می‌شود؛ به کمک این فراخوانی سیستمی برنامه‌های خواسته شده را بنویسید. برای دریافت راهنمایی در مورد هر کدام از این فراخوانی‌های سیستمی می‌توانید از دستور man ۲ [syscall_name] استفاده کنید.

- برای دیدن امکان دسترسی به فایل‌ها، فراخوانی سیستمی access مورد استفاده قرار می‌گیرد. برنامه‌ای بنویسید که به عنوان آرگومان ورودی یک آدرس را دریافت کند و ببیند که آیا اولاً آن آدرس وجود دارد یا خیر و ثانیاً آیا دسترسی به آن برای پروسه‌ی اجرا شده امکان‌پذیر است؟
- به کمک فراخوانی‌های سیستمی close open، و write برنامه‌ای بنویسید که یک فایل با اسم oslab۲.txt ایجاد کرده و نامتان را در آن فایل بنویسد.
- به کمک فراخوانی سیستمی sysinfo برنامه‌ای بنویسید که میزان حافظه RAM کل و همچنین حافظه‌ی خالی را در خروجی چاپ کند.
- به کمک فراخوانی سیستمی getrusage برنامه‌ای بنویسید که میزان حافظه مصرفی خود را چاپ کند.

۵.۳.۲ اضافه کردن یک فراخوانی سیستمی به سیستم‌عامل

همان‌طور که در ابتدا بیان شد، برای اضافه کردن فراخوان‌های سیستمی به هسته‌ی لینوکس نیازمند آن هستیم که هسته را مجدداً کامپایل و نصب کنیم. برای اضافه کردن یک فراخوانی سیستمی سه گام اصلی باید انجام شود:

۱. اضافه کردن تابع جدید،

۲. به روزرسانی فایل‌های سرآیند،

۳. به روزرسانی جدول فراخوانی‌های سیستمی.

در اینجا قصد داریم که یک فراخوانی سیستمی ساده را به سیستم‌عامل اضافه کنیم.

۱. مطمئن شوید که با دسترسی root به سیستم‌عامل وارد شده‌اید.

۲. وارد پوشه‌ی کد منبع هسته سیستم‌عامل که در جلسه قبل ایجاد کردیم شوید.

۳. دستور oldconfig make را اجرا کنید. این دستور هسته‌ی جدید را مطابق با ویژگی‌های هسته‌ی فعلی که بر روی که بر روی سیستم نصب شده است تنظیم می‌کند.

۴. با دستور make هسته را کامپایل کنید و مطمئن شوید که این عملیات به درستی صورت می‌گیرد.

۵. به کمک دستور install install modules_ make هسته‌ی جدید را نصب نمایید.

۶. سیستم‌عامل را مجدداً راه‌اندازی کنید. دقت کنید که در منوی بوت، هسته جدید را انتخاب کنید.
حال نوبت به نوشتن فراخوانی سیستمی می‌رسد.

۷. یک پوشه خالی با نام hello در شاخه اصلی کد منبع هسته ایجاد کنید.

۸. در این پوشه یک فایل hello.c شامل کد فراخوانی سیستمی با محتوای زیر ایجاد کنید:

```c
#include <linux/kernel.h>
asmlinkage long sys_hello(void) {
  printk("Hello World\n");
  return 0;
}
```

۹. یک فایل با نام Makefile در همین شاخه با محتوای زیر ایجاد کنید:

```
obj-y := hello.o
```

۱۰. فایل Makefile موجود در ریشه‌ی کد منبع را باز کنید. در حوالی خط ۸۰۰ این فایل متنی مشابه زیر وجود دارد که به انتهای آن /hello را اضافه نمایید.

```
Core-y += kernrl/ mm/ fs/ ipc/ security/ crypto/ block/
```

۱۱. حال جداول مربوط به فراخوانی سیستمی را به روزرسانی می‌کنیم.
فایل

```
./arch/x86/syscalls/syscall\_ 32.tbl
```

را باز کرده و خط زیر را در انتهای آن اضافه نمایید.

```
357 i386 hello sys_hello
```

۱۲. در فایل

```
./include/linux/syscalls.h
```

خطی به صورت زیر اضافه نمایید.

```
asmlinkage long sys_hello(void);
```

۱۳. هسته را مجدداً کامپایل و نصب کنید و سیستم را دوباره راه‌اندازی نمایید.

تمرین ۴.۲ دو برنامه با کارکرد زیر بنویسید.

- برنامه‌ای بنویسید که از فراخوانی سیستمی hello استفاده کند. برای مشاهده‌ی خروجی چاپ شده آن از دستور dmesg استفاده کنید.
- یک فراخوانی سیستمی با نام adder بنویسید که دو عدد را با یکدیگر جمع کند.

کتاب‌نامه

[1] http://www.advancedlinuxprogramming.com/alp-folder/alp-ch08-linux-system-calls.pdf

[2] http://seshagiriprabhu.wordpress.com/2012/11/15/adding-a-simple-system-call-to-the-linux-3-2- 0-kernel-from-scratch/

آزمایش ۳

مشاهده رفتار هسته و سیستم عامل

۱.۳ مقدمه

در این جلسه از آزمایشگاه خواهیم آموخت که چگونه می‌توان در سیستم عامل لینوکس رفتار هسته را مشاهده کرد و اطلاعات مربوط به پردازه‌ها و هسته را استخراج نمود.

۱.۱.۳ پیش‌نیازها

انتظار می‌رود که دانشجویان با موارد زیر از پیش آشنا باشند:

- برنامه‌نویسی به زبان c/c++
- دستورات پوسته‌ی لینوکس که در جلسات قبل فرا گرفته شده‌اند.

۲.۳ فایل سیستم proc/

در سیستم عامل لینوکس برای بررسی وضعیت هسته، مشاهده پردازه‌های در حال اجرا و دریافت اطلاعاتی از این دست، روشی پیش‌بینی شده است که proc/ system file نامیده می‌شود. در حقیقت proc/ به عنوان یک فایل سیستم عادی نیست، بلکه واسطی است برای دسترسی به فضای آدرس پردازه‌های در حال اجرا. این کار با ع می‌شود تا بتوان به صورت عادی به کمک فراخوانی‌های سیستمی ،read open و write در مورد پردازه‌های اطلاعات مورد نیاز را استخراج کرد یا تغییراتی در آن‌ها ایجاد نمود.

۳.۳ شرح آزمایش

۱.۳.۳ مشاهده‌ی فایل سیستم proc/

۱. وارد سیستم عامل مجازی ایجاد شده در جلسه قبل شوید.

۲. با وارد کردن دستور مناسب وارد شاخه proc/ شوید.

۳. به کمک دستور ls لیست فایل‌های موجود در این شاخه را ببینید.

۴. همانطور که ملاحظه می‌کنید، تعدادی فایل در این شاخه وجود دارد که اسامی آنها به صورت عدد می‌باشد. این اسامی در واقع proccess ID پردازه‌های در حال اجرا در سیستم می‌باشند. دقت کنید که این فایل‌ها در واقع به شکل فایل‌های سنتی وجود ندارند، بلکه واسط‌هایی هستند که توسط هسته برای دسترسی به اطلاعات پردازه‌ها ایجاد شده‌اند.

۲.۳.۳ مشاهده‌ی محتویات یک فایل در شاخه proc/

۱. همانطور که در قبل اشاره شد. فایل‌های موجود در شاخه proc/ به شکل فایل‌های عادی دیده می‌شوند. اما در واقع هرکدام از این فایل‌ها یا زیرشاخه‌ها موجود در این بخش، برنامه‌هایی هستند که متغیرهایی را از هسته خوانده و آن‌ها را به صورت ASCII برمی‌گردانند.

۲. به کمک دستور cat محتویات مربوط به فایل proc/version/ در خروجی چاپ کنید. چه چیزی در خروجی مشاهده می‌کنید؟

۳. محتویات چند فایل دیگر را (فایل‌هایی با نام غیر عددی) در این شاخه چاپ کنید. هر کدام از این فایل‌ها چه چیزی را نشان می‌دهد؟

۴. یک برنامه ساده به زبان c++ بنویسید که به کمک توابع <fstream> فایل proc/version/ را خوانده و محتویات آن را در فایل LinuxVersion.txt بنویسد. همان‌طور که مشاهده خواهید کرد، به کمک توابع کار با فایل به راحتی می‌توان با این فایل‌ها کار کرد.

۵. سعی کنید در فایل proc/version/ یک جمله دلخواه را بنویسید. چه اتفاقی می‌افتد؟

۳.۳.۳ مشاهده‌ی وضعیت پردازه‌ها

۱. به ازای هر کدام از پردازه‌ها، یک پوشه با شماره‌ی آن پردازه در proc/ وجو دارد. وارد یکی از این پوشه‌ها به دلخواه شوید و سپس با دستور ls فایل‌های موجود در آن را ملاحظه کنید.

۲. هر کدام از فایل‌ها اطلاعات خاصی را در مورد این پردازه در اختیار ما قرار می‌دهند. محتویات هر کدام از فایل‌های زیر را در این شاخه به کمک cat نشان دهید و بررسی نمایید که کدام از این پوشه‌ها حاوی چه چیزی هستند؟ برای اطلاعات بیشتر در مورد هر کدام از این موارد از دستور proc man استفاده کنید. statm / root / exe / cwd / statm / status / stat / environ / cmdline

۳. یک اسکریپت ساده بنویسید که لیست شماره‌ی پردازه های در حال اجرا به همراه نام آنها را در خروجی چاپ کند.

تمرین ۱.۳ • به کمک ممطالبی که در بالا آموخته‌اید، برنامه‌ای بنویسید که شماره یک پردازه را دریافت و در خروجی اطلاعاتی اعم از نام فایل اجرایی آن، مقدار حافظه مصرفی (به بایت)، پارامترهای اجرا و متغیرهای محیطی مربوط به آن در خروجی چاپ کند.

۴.۳.۳ مشاهده اطلاعات مربوط به هسته

- مشابه روشی که اطلاعات مربوط به پردازه‌ها را می‌توان مشاهده کرد، فایل سیستم proc/ این امکان را در اختیار شما قرار می‌دهد تا اطلاعات را در ارتباط با هسته مشاهده کنید. از جمله‌ی این اطلاعات می‌توان به اطلاعات دستگاه‌های I/O، وضعیت وقفه‌ها، اطلاعات پردازنده و ... اشاره کرد. این فایل‌ها در شاخه‌ی اصلی proc/ قرار دارند (فایل‌هایی که نام آن‌ها عدد نمی‌باشد). وارد پوشه proc/ شوید.
- به کمک دستور دستور ls بار دیگر لیستی از فایلموجود در این پوشه را ببینید.
- هر کدام از فایل‌ها یا پوشه‌های زیر را بررسی و ملاحظه کنید که هر کدام چه اطلاعاتی را در اختیار ما قرار می‌دهند:
cmdline / cpuinfo / filesystems / ioports / interrupts / loadavg / net / mount / stat / uptime / version / meminfo
- برنامه‌ای بنویسید که نام مدل پردازنده، فرکانس آن و مقدار حافظه نهان آن را در خروجی چاپ کند.
- برنامه‌ای بنویسید که مقدار حافظه کل، حافظه استفاده شده و حافظه آزاد را در خروجی چاپ کند.

تمرین ۲.۳ به پرسش‌های زیر پاسخ دهید.

- در باره پنج مورد از مهمترین فایل‌های موجود در proc/sys/kernel/ تحقیق کنید و کاربرد آن‌ها را بیان نمایید.
- در مورد self در شاخه proc/ و کاربرد آن توضیح دهید.

آزمایش ۴

ایجاد و اجرای پردازه‌ها

۱.۴ مقدمه

در این جلسه از آزمایش خواهیم آموخت که چگونه در سیستم عامل لینوکس می‌توان پردازه‌ها را ایجاد و اجرا نمود.

۲.۴ پیش‌نیازها

انتظار می‌رود که دانشجویان با موارد زیر از پیش آشنا باشند:

- برنامه نویسی به زبان C/C++
- دستورات پوسته لینوکس که در جلسات قبل فراگرفته شدند

۳.۴ پردازه چیست؟

به عنوان یک تعریف غیر رسمی، پردازه را می‌توان یک تک برنامه در حال اجرا دانست. ممکن است پردازه متعلق به سیستم باشد (مثلاً login) یا توسط کاربر اجرا شده باشد (مثلاً ls یا vim).

هنگامی که در سیستم‌عامل لینوکس یک پردازه ایجاد می‌شود، سیستم‌عامل یک عدد یکتا به آن پردازه می‌دهد. این عدد یکتا را Process ID یا PID می‌نامند. برای دریافت لیست پردازه‌ها به همراه PID آن‌ها از دستور ps استفاده می‌شود.

نکته‌ی مهمی که باید در مورد پردازه‌ها بدانید آن است که پردازه‌ها در سیستم‌عامل لینوکس به عنوان واحدهای اولیه‌ی اختصاص منابع به شمار می‌روند. هر پردازه فضای آدرس خاص خود و یک یا چند ریسه در کنترل خود دارد. هر پردازه، یک «برنامه» را اجرا می‌کند. چند پردازه می‌توانند یک برنامه یکسان را اجرا کنند ولی هرکدام از پردازه‌ها یک کپی جداگانه از آن برنامه را در فضای آدرس خود و مستقل از پردازه‌های دیگر اجرا می‌کنند.

پردازه‌ها در یک ساختار سلسله‌مراتبی قرار می‌گیرند. به جز پردازه init، هر پردازه یک والد دارد. هر پردازه می‌تواند با ایجاد پردازه‌های جدید، پردازه‌های فرزند به وجود بیاورد. ممکن است والد یک پردازه، لزوماً ایجاد کننده‌ی آن نباشد. چرا که پس از قطع شدن اجرای پردازه‌ی والد اصلی (برای مثال در صورت پایان یافتن آن)، والدی جدید برای پردازه‌های فرزند در حال اجرا، در نظر گرفته می‌شود.

۴.۴ شرح آزمایش

۱.۴.۴ مشاهده‌ی پردازه‌های سیستم و PID آن‌ها

۱. به کمک دستور ps لیست پردازه‌ها و PID آنها را مشاهده می‌کنید.

۲. چه پردازه‌ای دارای PID برابر ۱ است؟ به کمک دستور man [process_name] اطلاعاتی در مورد آن کسب کرده و به طور خلاصه وظیفه‌ی این پردازه و نحوه‌ی ساخته‌شدن آن را شرح دهید.

۳. به کمک تابع getpid برنامه‌ای بنویسید که PID خود را در خروجی چاپ کند.

۲.۴.۴ ایجاد یک پردازه‌ی جدید

تنها راه ایجاد یک پردازه‌ی جدید در سیستم‌عامل لینوکس، تکثیر کردن یک پردازه موجود در سیستم است. همان‌طور که در بخش قبل دیدید، ابتدا تنها یک پردازه‌ی init در سیستم وجود داد و در واقع این پردازه جد تمام پردازه‌های دیگر در سیستم است.

هنگامی که یک پردازه تکثیر می‌شود، پردازه‌ی فرزند و والد دقیقاً مانند هم خواهند بود؛ به غیر از اینکه مقدار PID آن‌ها با هم متفاوت است. کد، داده‌ها و پشته‌ی فرزند، دقیقاً از روی والد کپی می‌شود و حتی فرزند از همان نقطه‌ای که والد در حال اجرا بود، اجرای خود را ادامه می‌دهد. با این وجود، پردازه‌ی فرزند می‌تواند کد خود را با یک کد یک برنامه‌ی اجرای دیگر جایگزین نماید و به این صورت برنامه‌ای غیر از والد خود را اجرا نماید.

۱. به کمک تابع getppid برنامه‌ای بنویسید که PID پردازه‌ی والد خود را چاپ کند. برنامه‌ی نوشته شده را در ترمینال اجرا کنید؛ پردازه‌ی والد چه پردازه‌ای است؟ نام آن را همراه با توضیح کوتاهی بیان کنید.

۲. برای تکثیر پردازه از تابع fork استفاده می‌شود. کد زیر به زبان C نوشته شده است. خروجی آن را مشاهده کنید. در مورد اینکه این کد چه کاری انجام می‌دهد توضیح دهید:

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/syscall.h>
int main () {
  long result;
  result = syscall(__NR_mkdir, "testdir", 0777);
  printf("The result is %ld.\n", result);
  return 0;
}
```

۳. برنامه بالا را به گونه‌ای تغییر دهید که نشان دهد حافظه‌ی والد و فرزند از هم مستقل هستند.

۴. برنامه قسمت (۲) را به گونه‌ای تغییر دهید که برای والد و فرزند هرکدام پیام‌های جداگانه‌ای نمایش دهد؛ برای مثال برای فرزند I am the child و برای والد I am the parent را در خروجی چاپ کند (راهنمایی: از خروجی تابع fork استفاده کنید).

۵. به برنامه‌ی قسمت (۲) دو تابع fork دیگر نیز اضافه کنید و بین هرکدام از هاfork یک خروجی (مثلاً After first fork) چاپ کنید و نتیجه را ملاحظه کنید. کد خود را به همراه توضیح خروجی در گزارش بیاورید.

۳.۴.۴ اتمام کار پردازه‌ها

گاهی اوقات نیاز است که پردازه‌ی والد تا پایان اجرای پردازه‌ی فرزند منتظر بماند و سپس کار خود را ادامه دهد. برای این کار تابع wait مورد استفاده قرار می‌گیرد. جزئیات این تابه را می‌توانید با دستور man wait ملاحظه کنید. همچنین تابع exit برای خاتمه‌ی اجرای برنامه کاربرد دارد.

۱. برنامه‌ای بنویسید که پردازه‌ی فرزندی را ایجاد کند که این پردازه‌ی فرزند اعداد ۱ تا ۱۰۰ را در خروجی چاپ کند. بعد از پایان کار فرزند، پردازه‌ی والد باید با چاپ پیامی پایان کار فرزند را اعلام کند. برای این کار از تابع wait(NULL) استفاده کنید.

۲. در صورتی که پیش از پایان کار فرزند، والد به اتمام برسد، والد پردازه‌ی فرزند به init تغییر پیدا می‌کند (اصطلاحاً گفته می‌شود که پردازه‌ی فرزند توسط آن «adopt» می‌شود). به کمک استفاده از دستور sleep در فرزند برنامه‌ای بنویسید که این اتفاق را نشان دهد؛ یعنی PID والد را قبل و بعد از اتمام والد در خروجی به همراه پیامی جهت پایان اجرای والد چاپ کند (راهنمایی: از sleep در بدنه‌ی پردازه‌ی فرزند استفاده کنید).

۴.۴.۴ اجرای فایل

برای اینکه پردازه‌ی فرزند برنامه‌ی دیگری غیر از والد را اجرا کند، از دستورات execv، execl، execvp و execlp استفاده می‌شود.

۱. تفاوت‌های این دستورات را بیان کنید.

۲. برنامه‌ای بنویسید که یک پردازه‌ی فرزند ایجاد کند که این پردازه‌ی فرزند دستور ls g h را اجرا کند (راهنمایی: آرگومان صفرم یا دستور اجرا کننده‌ی برنامه را نیز باید در لیست آرگومان‌ها قرار بدهید).

۵.۴ فعالیت‌ها

- در مورد گروه‌های پردازه‌ای و دستورات getpgrp و setpgid تحقیق کنید و توضیح مختصری در مورد آن‌ها ارائه دهید.
- برنامه‌ی ساده‌ی زیر را در نظر بگیرید. درخت پردازه‌هایی که این برنامه ایجاد می‌کند را رسم نمایید و خروجی آن را نیز بیان کنید:

```c
#include <stdio.h>
#include <unistd.h>

int main() {
  fork();
  fork();
  printf("Parent Process ID is %d\n", getppid());
  return 0;
}
```

- برنامه‌ی زیر را چند بار اجرا کنید. این برنامه چه چیزی را در سیستم‌عامل نشان می‌دهد؟

```c
#include <stdio.h>
#include <unistd.h>

int main() {
  int i = 0, j = 0, pid, k, x;
  pid = fork();
  if(pid == 0) {
    for (i = 0; i < 20; i++) {
      for (k = 0; k < 10000; k++);
      printf("Child %d\n", i);
    }
  } else {
    for (j = 0; j < 20; j++) {
      for (x = 0; x < 10000; x++);
      printf("Parent %d\n", j);
    }
  }
}
```

- پردازه‌ی Zombie چیست؟

آزمایش ۵

ارتباط بین پردازه‌ای

۱.۵ مقدمه

در این جلسه از آزمایشگاه مکانیزم‌های مربوط به ارتباط و تبادل پیام بین پردازه‌ها در سیستم‌عامل لینوکس را خواهیم آموخت.

۱.۱.۵ پیش‌نیازها

انتظار می‌رود که دانشجویان با موارد زیر از پیش آشنا باشند:

۱. .نحوه ی ایجاد پردازه‌ها در سیستم‌عامل لینوکس (مطالب جلسه‌ی چهارم)

۲. .برنامه‌نویسی به زبان C/C++

۳. .دستورات پوسته‌ی لینوکس که در جلسات قبل فرا گرفته شده‌اند.

۲.۵ ارتباط بین پردازه‌ها

در جلسات قبل نحوه‌ی ایجاد پردازه‌های جدید را آموختیم. در این جلسه سعی داریم روش های ارتباط میان این پردازه‌ها را بررسی کنیم. مکانیزم‌های متعددی برای تبادل پیام بین پردازه ها وجود دارد که در این جلسه دو روش استفاده از Pipe و Signal را بررسی خواهیم کرد. از جمله کاربردهای ارتباط بین پردازه‌ای می‌توان به همگام‌سازی و انتقال اطلاعات اشاره کرد.

Pipeها، برای کاربران پوسته‌ی لینوکس آشنا هستند. برای مثال شما می‌توانید برای مشاهده لیست پردازه‌هایی که در آن‌ها کلمه‌ی init وجود دارد، از دستور ps aux l grep init استفاده کنید. در اینجا دو پردازه به کمک یک pipe به هم متصل شده‌اند. نکته‌ای که در اینجا مهم است آن است که این pipe ایجاد شده تنها در یک جهت (از پردازه‌ی اول به پردازه‌ی دوم) اطلاعات را جابه‌جا می‌کند. به کمک فراخوانی‌های سیستمی می توان Pipe های دو سویه و حلقوی نیز ایجاد کرد.

۳.۵ شرح آزمایش

۱. ایجاد یک Pipe یک‌سویه

(آ) برای ایجاد Pipe یک‌سویه در سیستم‌عامل لینوکس از فراخوانی سیستمی Pipe استفاده می‌شود. به کمک دستور man 2 pipe خلاصه ای از نحوه‌ی کار آن ملاحظه کنید.

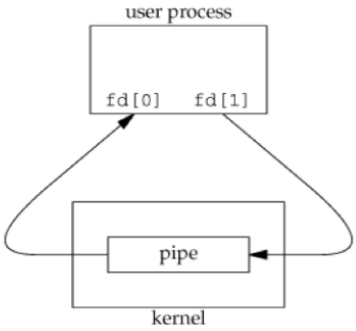
(ب) به کمک کد زیر یک Pipe ایجاد کنید:

```
1 int   fd[2];
2 int  res = pipe(fd);
```

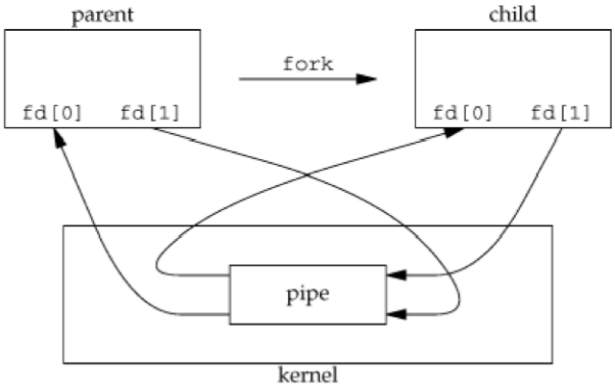
دستور Pipe در اینجا دو File Descriptor ایجاد می کند (آرایه‌ی fd). یکی از آن ها برای خواندن و دیگری برای نوشتن مورد استفاده قرار خواهد گرفت. fd[0] برای خواندن و fd[1] برای نوشتن خواهد بود.

(ج) تا اینجا که تنها یک پردازه داریم ، می توان شمای کلی های fd ایجاد شده را در شکل ۱.۵ نشان داد:

هر چیزی که بر روی fd[1] نوشته شود، قابل خواندن با fd[0] خواهد بود. حال توجه کنید که در صورتی که عملیات fork را صورت دهیم، پردازه‌ی فرزند، File Descriptorهای پدر را به ارث خواهد برد. بنابراین بعد از انجام شدن عملیات fork و ایجاد



شکل ۱.۵: Pipe یک سویه



شکل ۲.۵: Pipe دو سویه

پردازه‌ی فرزند، ساختار بالا به شکل زیر (شکل ۱.۵) در خواهد آمد: مشکل مهمی که در این جا با آن مواجه هستیم آن است که در صورتی هر دو پردازه بخواهند بر روی Pipe بنویسند و یا از آن بخوانند، به دلیل اینکه تنها یک بافر مشترک داریم، رفتار قابل پیش بینی نخواهد بود. در این حالت یک پردازه ممکن است داده‌ای که خودش بر روی Pipe قرار داده است را بخواند! بنابراین نیاز است که یک طرف تنها بر روی Pipe بنویسد و یک طرف تنها از آن بخواند. برای مثال فرض کنید پردازه‌ی فرزند قصد خواندن از Pipe و پردازه‌ی پدر قصد نوشتن بر روی آن را دارد. به کمک فراخوانی سیستمی close،پردازه‌ی پدر fd[0] خود را می‌بندد (زیرا قصد خواندن ندارد) و پردازه‌ي فرزند نیز fd[1] را خواهد بست. به این ترتیب یک ارتباط Half-Duplex بین این دو پردازه ایجاد می‌شود.

به کمک توضیحات بالا و استفاده از فراخوانی های سیستمی write و read،جمله‌ی Hello World را از سمت پردازه‌ی پدر به پردازه‌ی فرزند منتقل کرده و در پردازه‌ی فرزند آن را چاپ کنید.

فعالیت‌ها همان طور که در جلسه‌ی پیش آموختیم، به کمک دستورات خانواده exec،بعد از انجام fork می توان یک پردازه، مثلا ls، را اجرا نمود. به کمک دستورات dup/dup2 برنامه‌ای بنویسید که پردازه‌ی پدر دستور ls و پردازه‌ی فرزند دستور wc را اجرا کند و خروجی پردازه‌ی پدر (که دستور lsاست) به عنوان ورودی به پردازه‌ی فرزند داده شود. راهنمایی:یک Pipe ایجاد کنید و به نحوی خروجی که در حالت عادی Standard Output نوشته می شود را به آن منتقل کنید. پردازه‌ی فرزند نیز باید به جای خواندن از ورودی استاندارد از Pipe ورودی خود را بخواند. این کار به کمک دستورات dup/dup2 امکان‌پذیر است .
چگونه ارتباطات تمام دوطرفه بین پردازه ها داشته باشیم؟

۲. سیگنال‌ها

بعضی اوقات نیاز است که برنامه‌ها بتوانند با برخی از شرایط غیر قابل پیش بینی مواجه شده و آن‌ها را کنترل کنند؛ برای مثال:

(آ) در خواست بستن برنامه توسط کاربر به وسیله‌ی Ctrl+C

(ب) رخ دادن یک خطا در محاسبات Floting Point

(ج) مرگ پردازه ی فرزند

این رخدادها توسط سیستم‌عامل لینوکس شناخته می‌شوند و سیستم‌عامل با ارسال یک ((سیگنال)) ، پردازه را از وقوع آن‌ها آگاه می‌سازد. برنامه‌نویس می‌تواند این سیگنال‌ها را نادیده بگیرد، یا در عوض با نوشتن کد مخصوص آن‌ها را مدیریت و کنترل نماید. به این ترتیب، برنامه‌نویس می‌تواند برنامه خود را به نحوی تغییر دهد که مثلا با دریافت کلیدهای Crtl+C بسته نشود.

(آ) به کمک دستور man 7 signal لیستی از سیگنال‌های موجود در سیستم‌عامل لینوکس را ملاحظه کنید. برخی از آن‌ها را به دلخواه انتخاب و در گزارش خود با توضیح مختصری .بیاورید

(ب) یک سیگنال ساده، سیگنال Alarm است. به کمک man در مورد آن توضیح کوتاهی ارائه دهید.

(ج) کد زیر به کمک این سیگنال نوشته شده است؛ آن را اجرا کرده و در مورد کارکرد آن توضیح دهید:

```
1 #include <stdio.h>
2 static int main (){
3    Alarm (5);
4    printf "(Looping forever . . . \ n ");
5    while (1);
6    printf"(This line should never be executed.\"n);
7    return  0;
8 }
```

(د) به طور پیش فرض پردازه بعد از دریافت یکی از سیگنال‌های تعریف شده ، کشته می‌شود. به کمک فراخوانی سیستمی signal می توان این رفتار را تغییر داد و کد مورد نظر برنامه‌نویس را اجرا کرد. همچنین یک فراخوانی سیستمی دیگر به نام pause وجود دارد که پردازه را تا زمانی که یک سیگنال دریافت کند، متوقف می‌سازد. به کمک این دو تابع، برنامه‌ی بالا را به گونه‌ای ویرایش کنید که بعد از دریافت سیگنال alarm که با SIGALRM شناخته می‌شود، از توقف خارج شود و خط آخر را در خروجی چاپ کند.

تمرین ۱.۵ برنامه‌ای بنویسید که در صورتی که کاربر کلیدهای $Crtl+C$ را فشار دهد، برای بار اول خارج نشود ولی در دفعه‌ی دوم برنامه به پایان برسد.

آزمایش ۶

مدیریت حافظه

۱.۶ مقدمه

در این جلسه از آزمایشگاه با ساختار حافظه‌ی پردازه‌ها آشنا خواهیم شد.

۲.۶ پیش‌نیازها

انتظار می‌رود که دانشجویان با موارد زیر ازپیش آشنا باشند:

- برنامه‌نویسی به زبان C/C++
- دستورات پوسته‌ی لینوکس که در جلسات قبل فرا گرفته شده‌اند.

۳.۶ مدیریت حافظه

همان‌طور که می‌دانید، در زبان‌های برنامه‌نویسی سطح بالا، مانندC، هنگامی که یک متغیر تعریف می‌کنید، کامپایلر فضای مورد نیاز برای آن‌ها را در نظر می‌گیرد و نیازی به تخصیص فضا به صورت دستی ندارید. متغیرهای سراسری در Data Segment پردازه و متغیرهای محلی در Segment Stack قرار می‌گیرند.

همچنین در برخی از شرایط نیاز است که حافظه به صورت پویا اختصاص یابد؛ برای مثال برای ایجاد یک داده‌ساختار مانند درخت یا لیست پیوندی. در زبان برنامه‌نویسی C فراخوانی‌های سیستمی malloc و free برای این منظور وجود دارند.

۴.۶ شرح آزمایش

آ) استفاده از فراخوانی‌های سیستمی malloc و free

- ساختار زیر را فرض کنید

```
1 struct MyStruct {
2   int a;
3   int b;
4   char name[20];
5 };
```

- به کمک malloc حافظه‌ی مورد نیاز برای یک instance از این ساختار را تخصیص دهید. خروجی دستور malloc چیست؟
- برای فیلدهای instance ایجاد شده مقادیری را اختصاص دهید و آن‌ها را چاپ کنید.
- به کمک free حافظه‌ی گرفته شده را آزاد کنید.

ب) مشاهده وضعیت حافظه‌ی پردازه‌ها

- به کمک دستور زیر وضعیت حافظه‌ی پردازه‌ها را مشاهده کنید:

```
1 ps -o user,vsz,rss,pmem,fname -e
```

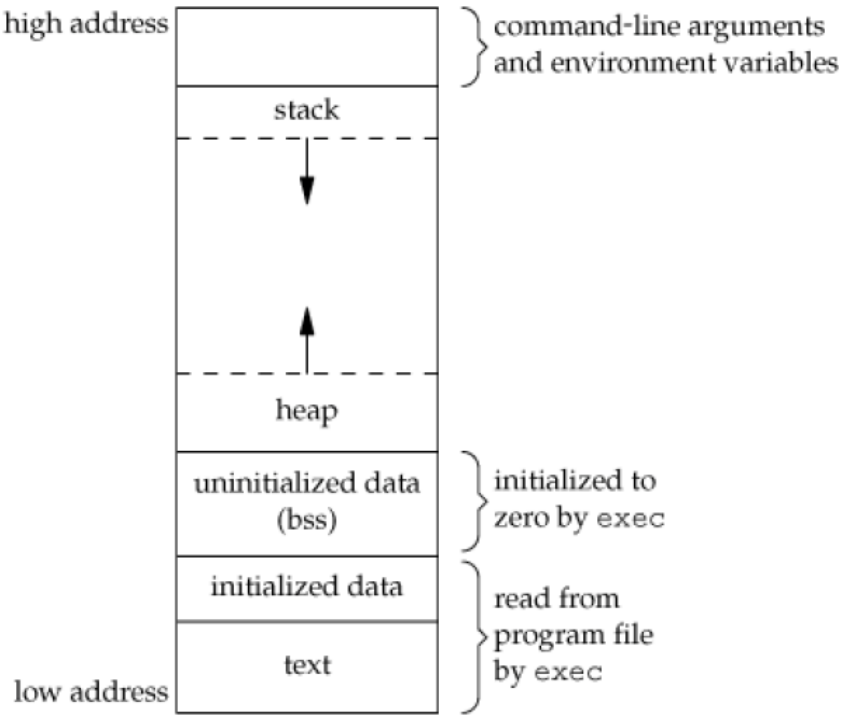
- به کمک دستورps map توضیح دهید که هر کدام از ستون‌ها چه اطلاعاتی را نشان می‌دهند.

ج) اجزای حافظه‌ی یک پردازه

حافظه‌ی یک پردازه در سیستم‌عامل لینوکس به اجزای زیر تقسیم‌بندی می‌شود:

- بخش text که کد زبان ماشین پردازه را نگه‌داری می‌کند؛
- بخش data که متغیرهای سراسری پردازه در آن قرار می‌گیرد، برخی از متغیرهای دارای مقدار اولیه هستند و برخی خیر. دسته‌ی دوم در بخشی به نام bss واقع می‌شوند؛
- بخش heap که شامل حافظه‌هایی است که به صورت پویا اختصاص داده می‌شود و
- بخش stack که متغیرهای محلی، پارامترهای توابع و مقادیر بازگشتی آن‌ها در آن قرار دارد.

تصویر زیر این ساختار را نمایش می‌دهد (آدرس‌های کوچک‌تر در پایین تصویر قرار دارند):



شکل ۱.۶: ساختارحافظه پردازه

- به کمک دستور which محل قرارگیری دستور ls را بر روی دیسک پیدا کنید.
- با استفاده از دستور size مشاهده کنید که چه مقدار از کد ماشین این دستور به بخش‌هایی که در بالا اشاره شد اختصاص دارد. این دستور کدام یک از بخش‌های بالا را نشان نمی‌دهد؟

د) اشتراک حافظه

در سیستم عامل لینوکس، معماری حافظه به صورت صفحه‌بندی شده است؛ به این معنا که حافظه در تکه‌هایی (معمولا ۸۱۹۲ یا ۴۰۹۶ بایتی) به پردازه‌ها اختصاص می‌یابد. سیستم عامل می‌تواند بعضی از این تکه‌ها را بین پردازه‌های مختلف به اشتراک بگذارد. برای مثال تمام پردازه‌هایی که یک کد یکسان را اجرا می‌کنند، بخش text مشترک دارند. کاربرد دیگر اشتراک این صفحات برای کتابخانه‌های مشترک است. برای مثال، اکثر برنامه‌ها از تابع printf استفاده می‌کنند؛ بنابراین منطقی است که تنها یکبار آن را در حافظه آورده و بین تمام پردازه‌هایی که از آن استفاده می‌کنند، حافظه را به اشتراک بگذاریم.

- به کمک دستور ldd کتابخانه‌های مشترکی که توسط دستور ls استفاده شده است را مشاهده کنید.
- این کار را برای چند برنامه‌ی دیگر (برای مثال nano) امتحان کنید و نتیجه را در گزارش کار خود بیاورید.

ه) آدرس‌های بخش‌های مختلف حافظه‌ی پردازه

به کمک استفاده از متغیرهای extern خاصی که در سیستم‌عامل تعریف شده‌اند، قادر هستیم که آدرس پایان code segment ، آدرس پایان global segment و همچنین آدرس پایان بخش متغیرهای سراسری دارای مقدار اولیه را مشاهده کنیم.

برای این کار، سه متغیر زیر در دسترس هستند:

- etext: اولین آدرس بعد از پایان text در این متغیر قرار دارد.
- edata: اولین آدرس بعد از پایان متغیرهای سراسری دارای مقدار اولیه در این متغیر قرار دارد.
- end: اولین آدرس بعد از پایان بخش bss در این متغیر قرار می‌گیرد.

- به کمک دستور man etext ، جزئیات بیشتری در مورد نحوه‌ی استفاده از این متغیرها ببینید و کدی که به عنوان مثال در انتهای این راهنما آمده است را نوشته و اجرا کنید.
- آیا خروجی این برنامه با ساختاری که در تصویر قبل آمده است تطابق دارد؟
- برای مشاهده‌ی آدرس انتهای heap از فراخوانی سیستمی sbrk استفاده می‌شود. به کمک این فراخوانی سیستمی نشان دهید که با اجرا کردن malloc و اختصاص حافظه به صورت پویا، این آدرس انتهای heap هربار تغییر می‌کند.
- همان طور که گفته شد stack در جهت عکس heap رشد می‌کند. همچنین، متغیرهای محلی و آرگومان‌های توابع در stack قرار می‌گیرند. یک تابع بازگشتی بنویسید که ابتدا یک متغیر محلی i ایجاد کند؛ سپس آدرس i را چاپ کرده و دوباره خودش را فراخوانی کند. روند تغییر آدرس متغیر i چگونه است؟ (برنامه را به گونه‌ای محدود کنید تا عملیات بازگشت مثلا ۱۰۰ بار انجام شود).

آزمایش ۷

آشنایی با ریسه‌ها

۱.۷ مقدمه

هدف اصلی این آزمایش، بررسی جنبه‌های مختلف ریسه‌ها و چندپردازشی (و چند ریسه‌ای) است. از اهداف اصلی این آزمایش پیاده‌سازی توابع مدیریت ریسه‌ها است:

- ساخت ریسه‌ها
- پایان بخشیدن به اجرای ریسه
- پاس دادن متغیر به ریسه‌ها
- شناسه‌های ریسه‌ها
- متصل شدن ریسه‌ها

۱.۱.۷ پیش‌نیازها

انتظار می‌رود که دانشجویان با موارد زیر از پیش آشنا باشند:

- برنامه‌نویسی به زبان C/C++
- دستورات پوسته‌ی لینوکس که در جلسات قبل فرا گرفته‌اند.

۲.۷ ریسه چیست؟

یک ریسه، شبه پردازه‌ای است که پشته‌ی خاص خود را در اختیار دارد و کد مربوط به خود را اجرا می‌کند. برخلاف پردازه، یک ریسه، معمولاً حافظه‌ی خود را با دیگر ریسه‌ها به اشتراک می‌گذارد. یک گروه از ریسه‌ها، یک مجموعه از ریسه‌ها است که در یک پردازه‌ی یکسان اجرا می‌شوند. بنابراین آنها یک حافظه‌ی یکسان را به اشتراک می‌گذارند و می‌توانند به متغیرهای عمومی یکسان، حافظه‌ی heap یکسان و ... دسترسی داشته باشند. همه‌ی ریسه‌ها می‌توانند به صورت موازی (استفاده از برش زمانی، یا اگر چندین پردازه وجود داشته باشد، به معنای واقعی موازی) اجرا شوند.

۳.۷ pthread

بر اساس تاریخ، سازندگان سخت‌افزار نسخه‌ی مناسبی از ریسه‌ها را براي خود پیاده‌سازی کردند. از آنجا که این پیاده‌سازی‌ها با هم تفاوت می‌کرد، پس کار را برای برنامه‌نویسان، برای نگارش یک برنامه‌ی قابل حمل دشوار می‌کرد. بنابراین نیاز به داشتن یک واسط یکسان برای بهره بردن از فواید ریسه‌ها احساس می‌شد. برای سیستم‌های Unix این واسط با نام POSIX IEEE ۱۰۰۳ ۱.c مشخص می‌شد و به پیاده‌سازی مرتبط با آن POSIX THREADS یا pthread گفته می‌شود. اکثر سازندگان سخت‌افزار، علاوه بر نسخه‌ی مناسب با خودشان، استفاده از pthread را نیز پیشنهاد می‌کنند. هاpthread در یک کتابخانه‌ی C تعریف شده‌اند که شما می‌توانید با برنامه‌ی خود link کنید.
توابع موجود در این کتابخانه به صورت غیررسمی به سه دسته تقسیم می‌شوند:

- مدیریت ریسه‌ها: دسته‌ی اول از این توابع به صورت مستقیم با ریسه‌ها کار می‌کنند. همانند ایجاد، متصل کردن و ...
- Mutex: دسته‌ی دوم از این توابع برای کار با هاmutex ایجاد شده‌اند. توابع مربوط به mutex ابزار مناسب برای ایجاد، تخریب، قفل و بازکردن هاmutex را در اختیار قرار می‌دهند.

- متغیرهای شرطی (Condition Variables): این دسته از توابع، براي كار با متغیرهای شرطی و استفاده از مفهوم همزمانی در سطح بالاتر در اختیار قرار می‌گیرند. این دسته از توابع براي ایجاد، تخریب، wait و signal بر اساس مقادیر معین متغیرها استفاده می‌شوند.

نکته ۱ در این جلسه قصد آن را داریم تا با دسته‌ی اول از توابع آشنا شویم. توابع مربوط به این دسته به طور خلاصه در جدول زیر مشاهده می‌شود: برای آشنایی با جزئیات می‌توانید از دستور man و یا اینترنت استفاده کنید.

جدول ۱.۷: توابع مربوط به مدیریت ریسه‌ها

نام تابع	کاربرد
pthread_create	از کتابخانه‌ی pthread، درخواست ساخت یک ریسه‌ی جدید را می‌کند.
pthread_exit	این تابع توسط ریسه استفاده شده تا پایان بپذیرد.
pthread_join	این تابع، براي ریسه‌ی مشخص شده صبر می‌کند تا پایان بپذیرد.
pthread_cancel	درخواست کنسل شدن ریسه‌ی مشخص شده را ارسال می‌کند.
pthread_attr_int	مقدار های attribute پاس داده شده به خود را با مقادیر پیش‌فرض پر می‌کند.
pthread_self	شماره‌ی ریسه را بر می‌گرداند.

۴.۷ شرح آزمایش

۱.۴.۷ آشنایی اولیه

۱. وارد سیستم عامل مجازی ایجاد شده در جلسات قبل شوید.

۲. با استفاده از تکه کد زیر، یک ریسه ایجاد کنید.

```c
#include<pthread.h>

int main()
{
   pthread_t the_thread;
   return 0;
}
```

دقت کنید در هنگام کامپایل، lpthread- را به پرچم‌های linker اضافه کنید:

```
# gcc thread.c -o threads -lpthread
```

دقت کنید در هنگام کامپایل، lpthread- را به پرچم‌های linker اضافه کنید:

۳. با استفاده از توابع pthread_create و pthread_join یک ریسه درست کنید که در آن شماره‌ی پردازه را چاپ کنید. همچنین در پردازه‌ی اصلی نیز شماره‌ی پردازه را چاپ کنید. دقت کنید پردازه‌ی اصلی بعد از پایان یافتن ریسه‌ها تمام شود. آیا شماره پردازه‌های چاپ شده یکسان می‌باشند؟

۴. برنامه‌ی بالا را در یک فایل جدید کپی کنید. حال، متغیر oslab را به این تکه کد به صورت عمومی اضافه کنید. حال، یک بار این متغیر را در ریسه‌ی اصلی و یک بار در ریسه‌ی فرزند تغییر دهید. بعد از تغییر در ریسه‌ی فرزند، بار دیگر در ریسه‌ی اصلی چاپ کنید. آیا ریسه‌ها کپی‌های جداگانه‌ای از متغیر را دارند؟

۵. با استفاده از تابع pthread_attr_init و تنظیم کردن های attribute ریسه به صورت پیش‌فرض، کدی بنویسید که در آن با گرفتن عدد n از ورودی، حاصل‌جمع اعداد ۲ تا n را چاپ کند.

۲.۴.۷ ریسه‌های چندتایی

در این قسمت قصد آن را داریم تا در یک کد، چند ریسه داشته باشیم.

- با استفاده از تابع pthread_create، تعدادي ریسه به تعداد دلخواه ایجاد کنید (حداقل پنج تا) و پیام World Hello را در آن چاپ کنید. سپس ریسه‌ها را با استفاده از تابع pthread_exit خاتمه دهید.

۳.۴.۷ تفاوت بین پردازه‌ها و ریسه‌ها

در این قسمت، قصد آن را داریم تا تفاوت میان پردازه‌ها و ریسه‌ها را بهتر متوجه بشویم.

۱. تکه کد زیر را به عنوان تابع ریسه در فایلی بنویسید:

دقت کنید در هنگام کامپایل، lpthread- را به پرچم‌های linker اضافه کنید

```
1 void kid(void* param)
2 {
3    int local_param;
4    printf("Thread %d, pid %d, addresses: &global: %X, &local: &X \n",
        pthread_self(), getpid(), &global_param, &local_param);
5    global_param++;
6    printf("In Thread %d, incremented global parameter=%d\n",
        pthread_self(), global_param);
7    pthread_exit(0);
8 }
```

دقت کنید در هنگام کامپایل، lpthread- را به پرچم‌های linker اضافه کنید:

۲. حال، در ریسه‌ی اصلی، یک متغیر عمومی به عنوان global_param تعریف کرده، مقداردهی کنید و دو ریسه‌ی فرزند با تابع kid ایجاد و اجرا کنید. در پایان ریسه‌ها نیز مقدار متغیر global_param را چاپ کنید.

۳. حال، مقدار متغیر عمومی را بار دیگر تغییر داده و یک متغیر محلی دیگر در تابع اصلی تعریف کنید. آن را نیز مقداردهی کنید. حال، با استفاده از تابع fork که در جلسات پیش یاد گرفته‌اید، یک پردازه‌ی فرزند ایجاد کرده و متغیرها را در آن دوباره مقداردهی کنید. تغییرات را با استفاده از تابع printf نمایش دهید.

۴.۴.۷ پاس دادن متغیرها به ریسه

تابع pthread_create تنها اجازه می‌دهد که یک متغیر به عنوان ورودی به ریسه داده شود. برای حالاتی که چند پارامتر می‌بایست به ریسه داده شود، این محدودیت به راحتی با استفاده از ساختار (structure) حل می‌شود. تمامی متغیرها می‌بایست به وسیله‌ی reference و تبدیل به void* پاس داده شوند.

- ساختار زیر را در فایلی قرار دهید:

```
1 typedef struct thdata {
2    int thread_no;
3    char message[100];
4 } stdata;
```

حال، در ریسهٔ اصلی، دو متغیر از ساختار معرفی شده ایجاد کنید و مقادیر آن را به صورت دلخواه تنظیم کنید. سپس، متغیرها را به دو ریسهٔ جداگانه پاس بدهید. در ریسه‌ها نیز عدد و پیام ذخیره شده در ساختار را نمایش بدهید.

آزمایش ۸

آشنایی با توابع سیستمی

۱.۸ مقدمه

یک برنامه در حین اجرا لازم است به منابعی همچون یک فایل دسترسی داشته باشد. درخواست دسترسی به این منابع توسط توابع سیستمی، به سیستم عامل ارائه می‌گردد.توابع سیستمی فراخوانی شده توسط یک برنامه نشان‌دهنده نحوه عملکردآن برنامه است.

هدف این آزمایش آشنایی دانشجویان با توابع سیستمی و نحوه عملکرد آن‌ها است. به عبارت دیگر، اهداف این آزمایش عبارتند از:

۱. آشنایی با توابع سیستمی

۲. آشنایی با عملکرد بدافزارهای سطح هسته بویژه روتکیت‌ها

۳. آشنایی با نحوه نظارت ابزارهای مانیتورینگ و ضدبدافزار بر روی رفتار نرم‌افزارهای مخرب

۴. آشنایی با قابلیت محافظت از خود (Self-protection) در ابزارها و نرم‌افزارهای امنیتی

۲.۸ پیش‌نیاز نظری

سیستم عامل رابطی بین کاربر و سخت افزار است که هدف آن آسان‌سازی استفادهِ کاربر از منابع سیستمی است. به عبارت دیگر سیستم‌عامل وظیفه مدیریت منابع سیستمی همانند حافظه، پردازنده و پورت‌های شبکه را بر عهده دارد. بطور معمول سیستم‌های عامل مدیریت این منابع را بوسیله‌ی توابع خاصی که بدین منظور طراحی و ساخته شده‌اند انجام می‌دهد، که به این توابع، توابع سیستمی و یا رابط برنامه‌نویسی کاربردی می‌گویند.

درخواست منبع می‌تواند درخواست سرویس سخت‌افزاری همانند دسترسی به دیسک سخت و یا سرویس‌های نرم‌افزاری همانند درخواست ایجاد یک فرآیند جدید باشد.

یک نرم افزار در حین اجرای خود در سیستم عامل منابعی را درخواست می‌کند. این منابع و یا درخواست‌های ارسالی توسط نرم افزار عموما در پنج کلاس دسته بندی می‌شود که عبارتند از:

۱. درخواست‌های مرتبط با فایل: این درخواست‌ها شامل ایجاد، کپی، خواندن و یا نوشتن اطلاعات در/از یک فایل است.

۲. درخواست‌های مرتبط با پردازه: این درخواست‌ها شامل ایجاد، حذف و و تغییر خصوصیات یک پردازه و یا ریسه است. همچنین درخواست‌های مرتبط با دریافت لیست پردازه‌های در حال اجرا، تزریق داده در فضای آدرس فرآیندها، ایجاد نخ‌های از راه دور جزء این دسته هستند.

۳. درخواست‌های مرتبط با پنجره‌ها: این درخواست‌ها شامل ایجاد پنجره، دریافت کلیدهای فشرده شده در صفحه کلید و یا نمایش اطلاعات در صفحه نمایش است.

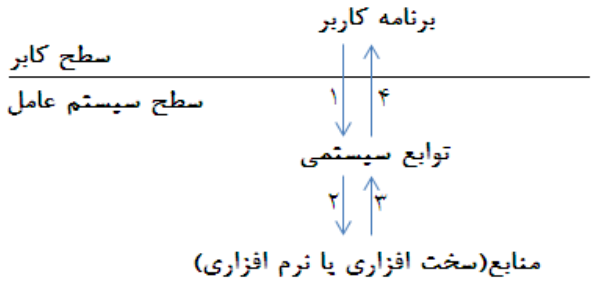
۴. درخواست‌های مربوط به شبکه: این درخواست‌ها شامل اتصال و گوش دادن به درگاه‌های شبکه و ارسال اطلاعات از طریق شبکه است.

۵. سایر درخواست‌ها: درخواست‌هایی همانند درخواست زمان سیستم و زمانبندی پردازه‌ها جز این دسته هستند.

در نهایت می‌توان گفت که فراخوانی سیستمی، رابطی بین یک پردازه (برنامه کاربر) و سیستم عامل است، بدین طریق که برنامه کاربر درخواست منابع خود را از طریق این توابع به سیستم‌عامل ارائه می‌نماید.

بر اساس شکل ۱.۸ درخواست یک فراخوانی سیستمی توسط برنامه شامل موارد زیر است:

۱. ابتدا برنامه سطح کاربر با توجه به منبع مورد نیاز، تابع سیستمی مربوطه را فراخوانی می‌نماید. این امر باعث تغییر سطح اجرایی (از سطح کاربر به سطح سیستم عامل) می‌گردد.



شکل ۱.۸: روال فراخوانی توابع سیستمی در سیستم عامل

۲. تحویل منابع درخواست شده از تابع سیستمی به سخت افزار و انجام عملیات ورودی/خروجی . البته باید به این نکته توجه نمود که درخواست دسترسی به منابع تنها محدود به منابع سخت افزاری نمی‌شود و منبع می‌تواند یک منبع نرم افزاری باشد.

۳. بازگرداندن نتیجه عملیات به تابع سیستمی فراخوانی شده.

۴. ارسال نتیجه عملیات از تابع سیستمی به برنامه کاربر جهت بررسی صحت نتایج و تغییر سطح اجرایی از سطح سیستم عامل به سطح کاربر.

در تمامی سیستم‌های عامل داده ساختاری بصورت جدول وجود دارد که آدرس تمام توابع سیستمی در آن ذخیره می‌گردد که به این جدول، « جدول توابع سیستمی» می‌گویند. به عنوان مثال بخشی از توابع سیستمی موجود در هسسته سیستم عامل لینوکس نسخه ۶.۴ در جدول زیر نشان داده شده است. لازم به ذکر است که در این هسته حدود ۳۲۸ تابع سیستمی وجود دارد.

Table ۱.۸: لینوکس سیستم‌عامل سیستمی توابع جدول از بخشی

rax	System call	rdi	rsi	rdx
۰	sys_read	unsigned int fd	char *buf	size_t count
۱	sys_read	unsigned int fd	const char *buf	size_t count
۲	sys_open	const char *filename	int flags	int mode
۳	sys_close	unsigned int fd		
۴	sys_stat	const char *filename	struct stat *statbuf	
۵	sys_fstat	unsigned int fd	struct stat *statbuf	
۶	sys_lstat	fconst char *filename	struct stat *statbuf	
...	...	...	...	...

۳.۸ آزمایش ۱

یک ماژول سطح هسته بنویسید که آدرس تمام توابع سیستمی موجود در هسته سیستم عامل لینوکس را چاپ کند.

راهنمایی: لیست توابع سیستمی و تعداد آنها در فایل سرآیند unistd.h از هسته سیستم عامل وجود دارد. در سیستم عامل Ubuntu ۰۴.۱۶ این فایل در مسیر زیر قرار دارد:

/usr/src/linux-headers-۲۸-۰.۱۰.۴/include/uapi/asm-generic

انتظار می رود خروجی این آزمایش شامل موارد زیر باشد.

۱. یک فایل اجرایی که همان کد پیاده‌سازی شده توسط دانشجو است.

۲. یک مستند که حاوی موارد زیر است.

(آ) توضیحاتی در خصوص تمام کدهای موجود در فایل اجرایی.

(ب) تصاویری از خروجی فایل اجرایی.

(ج) نیازمندی‌های لازم جهت اجرای فایل اجرایی.

۴.۸ آزمایش ۲

با تغییر آدرس توابع موجود در جدول توابع سیستمی و جایگزینی آدرس تابع دلخواه به جای تابع اصلی می‌توان عملکرد توابع سیستمی سیستم عامل را تغییر داد. به این عمل اصطلاحا هوک کردن یا Hooking گفته می‌شود.با طراحی یک ماژول سطح هسته در سیستم عامل لینوکس، فایل‌های موجود در دایرکتوری home سیستم عامل خود را پنهان نمایید. ابزارهای ضدبدافزار و یا بدافزارهای پیشرفته از این روش چه بهره‌ای می‌برند؟ انتظار می رود خروجی این آزمایش شامل موارد زیر باشد.

۱. یک فایل اجرایی که همان کد پیاده‌سازی شده توسط دانشجو است.

۲. یک مستند که حاوی موارد زیر است.

(آ) توضیحاتی در خصوص تمام کدهای موجود در فایل اجرایی.

(ب) تصاویری از خروجی فایل اجرایی.

(ج) نیازمندی‌های لازم جهت اجرای فایل اجرایی.

آزمایش ۹

آشنایی با وقفه ها

۱.۹ مقدمه

وقفه یک سیگنال به ریزپردازنده است که به توجه و پاسخ سریع پردازنده نیاز دارد. هنگامی که یک وقفه رخ می‌دهد، پردازنده عملیات جاری خود را متوقف می‌کند تا به درخواست وقفه رسیدگی کند. ریزپردازنده‌های خانواده ۸۰۸۶ به وقفه‌های تولید شده به وسیله سخت‌افزار و نرم‌افزار پاسخ می‌دهند که به ترتیب به آن‌ها وقفه‌های سخت‌افزاری، و وقفه‌های نرم‌افزاری گفته می‌شود.
هدف این آزمایش آشنایی با انواع وقفه‌ها است. به عبارت دیگر اهداف این آزمایش عبارتند از:

۱. آشنایی با وقفه‌ها و انواع آن‌ها

۲. آشنایی با وقفه‌های نرم‌افزاری

۳. آشنایی با سطح دسترسی به وقفه‌ها

۴. آشنایی با نحوه عملکرد بدافزارهای سوء استفاده کننده از وقفه

۲.۹ پیش‌نیاز نظری

در سیستم های عامل داده ساختارهای متفاوتی وجود دارد. یکی از این داده ساختارها جدول توصفیگر وقفه های سیستم است.
در سیستم عامل می توان وقفه ها را به سه گروه کلی تقسیم کرد:

۱. وقفه‌های داخلی سخت‌افزاری: وقفه‌های داخلی سخت‌افزاری بدلیل رخ دادن وضعیت معینی که درحین اجرای یک برنامه پیش آمده تولید می‌شوند (مانند تقسیم بر صفر). وقفه‌هایی که در اثر خطا بوجود می‌آید تله (trap) هم نامیده می‌شود. تله باعث سقط برنامه می‌شوند. این وقفه‌ها توسط سخت‌افزار اداره می‌شوند و امکان تغییر آنها وجود ندارد. اما با وجودیکه نمی‌توان آنها را مستقیما مدیریت کرد، این امکان وجود دارد که از اثر آن روی کامپیوتر به نحو مفیدی استفاده شود. به عنوان مثال سخت‌افزار، وقفه شمارنده ساعت کامپیوتر را چندبار در ثانیه فراخوانی می‌کند، تا زمان را نگه دارد. می‌توان برنامه‌ای نوشت که مقدار شمارنده ساعت را خوانده، آنرا به شکل قابل درک کاربر به صورت ساعت و دقیقه تبدیل کند.

۲. وقفه‌های خارجی سخت‌افزاری: وقفه‌های خارجی سخت‌افزاری خارج از پردازنده و توسط دستگاه‌های جانبی، مانند صفحه کلید، چاپگر، کارت‌های ارتباطی و یا کمک پردازنده تولید می‌شوند. دستگاه‌های جانبی با ارسال وقفه به پردازنده خواستار قطع اجرای برنامه فعلی شده و پردازته را متوجه خود می‌کنند.

۳. وقفه‌های نرم‌افزاری: وقفه‌های نرم‌افزاری در نتیجه دستورالعمل int در یک برنامه درحال اجرا تولید می‌شوند. برنامه نویس می‌تواند با دادن دستور int یک وقفه نرم‌افزاری تولید کند. بدین طریق بلافاصله اجرای برنامه فعلی را متوقف می‌کند و پردازنده را به روتین وقفه هدایت می‌کند. برنامه نویس از طریق وقفه‌ها می‌تواند در برنامه با وسایل جانبی ارتباط برقرار کند. استفاده از وقفه‌ها باعث کوتاهتر شدن کد برنامه و درک آسانتر و اجرای بهترآن می‌شود.

۳.۹ آزمایش ۱

یک برنامه بنویسید که تمام وقفه های موجود در سیستم عامل لینوکس را نمایش دهد. انتظار می رود خروجی این آزمایش شامل موارد زیر باشد.

۱. یک فایل اجرایی که همان کد پیاده‌سازی شده توسط دانشجو است.

۲. یک مستند که حاوی موارد زیر است.

(آ) توضیحاتی در خصوص تمام کدهای موجود در فایل اجرایی.
(ب) تصاویری از خروجی فایل اجرایی.
(ج) نیازمندی‌های لازم جهت اجرای فایل اجرایی.

۴.۹ آزمایش ۲

یک وقفه نرم افزاری به سیستم عامل لینوکس اضافه نمایید. آیا امکان فراخوانی این وقفه جدید توسط یک برنامه سطح کاربر وجود دارد؟ چرا؟ انتظار می رود خروجی این آزمایش شامل موارد زیر باشد.

۱. یک فایل اجرایی که همان کد پیاده‌سازی شده توسط دانشجو است.

۲. یک مستند که حاوی موارد زیر است.

(آ) توضیحاتی در خصوص تمام کدهای موجود در فایل اجرایی.
(ب) تصاویری از خروجی فایل اجرایی.
(ج) نیازمندی‌های لازم جهت اجرای فایل اجرایی.

آزمایش ۱۰

آشنایی با درایورها

۱.۱۰ مقدمه

درایور یا راه‌انداز قطعات سخت‌افزاری عبارتست از نرم‌افزاری که رایانه را برای کار با دستگاه خاصی توانا می‌سازد. به‌عبارت دیگر اگر دستگاهی به رایانه متصل شود و گرداننده مرتبط با دستگاه روی رایانه نصب نشده باشد، سیستم‌عامل قادر به شناسایی دستگاه نبوده و نمی‌تواند از آن دستگاه استفاده کند. همچنین در برخی موارد نیز سیستم‌عامل دستگاه را به‌طور ابتدایی با حداقل توانایی‌ها شناسایی کرده و نمی‌تواند از تمام امکانات و قابلیت‌های قطعه به‌طور کامل استفاده کند.

هدف این آزمایش آشنایی با درایورها و نحوه عملکرد آن‌هاست. به عبارت دیگر اهداف این آزمایش عبارتند از:

۱. آشنایی با درایورها و انواع آن‌ها

۲. آشنایی با ابزارهای شنود ترافیک شبکه و ابزارهای نظارتی

۳. آشنایی با محدودیت‌های درایورها

۲.۱۰ پیش‌نیاز نظری

شناساندن صحیح گرداننده‌های قطعات به رایانه از اهمیت بسیار زیادی برخوردار است، به‌طوری‌که گرداننده می‌تواند نحوه استفاده از دستگاه را نیز برای شما تغییر دهد. به‌عنوان مثال فرض کنید گرداننده دستگاه مودم را به‌طور کامل نصب نکرده‌اید، در این حالت شاید مودم رایانه شما قادر به شماره‌گیری و اتصال به اینترنت باشد اما قطعی‌های مکرر، کاهش سرعت اتصال و عدم‌پشتیبانی از امکاناتی همچون انتظار مکالمه، برخی از اشکالاتی است که هنگام شناسایی ناقص دستگاه توسط رایانه مشاهده می‌شود.

همچنین فرض کنید رایانه شما مجهز به یک کارت گرافیک است اما پس از گذشت چند سال، از عملکرد کارت گرافیک خود راضی نیستید یا نمی‌توانید روی سیستم‌عامل‌های جدید از آن استفاده کنید. در چنین مواردی می‌توانید با مراجعه به سایت سازنده دستگاه، نسخه جدید درایور کارت گرافیک را دانلود کنید تا علاوه بر پشتیبانی توسط سیستم‌عامل‌های جدیدتر، بتوانید از امکانات بیشتری که برای کارت گرافیکی شما در نظر گرفته شده است نیز استفاده کنید.

۳.۱۰ آزمایش ۱

یک نمونه درایور بنویسید که پیام «Hello» را پس از بارگذاری در فضای هسته سیستم عامل چاپ کند. انتظار می رود خروجی این آزمایش شامل موارد زیر باشد.

۱. یک فایل اجرایی که همان کد پیاده‌سازی شده توسط دانشجو است.

۲. یک مستند که حاوی موارد زیر است.

(آ) توضیحاتی در خصوص تمام کدهای موجود در فایل اجرایی.

(ب) تصاویری از خروجی فایل اجرایی.

(ج) نیازمندی‌های لازم جهت اجرای فایل اجرایی.

۴.۱۰ آزمایش ۲

یک درایور مجازی کنار درایور کارت شبکه‌ی خود ایجاد نمایید و سپس به تمام اتصالات شبکه گوش دهید و اطلاعات را روی یک فایل ذخیره نمایید. انتظار می رود خروجی این آزمایش شامل موارد زیر باشد.

۱. یک فایل اجرایی که همان کد پیاده‌سازی شده توسط دانشجو است.

۲. یک مستند که حاوی موارد زیر است.

(آ) توضیحاتی در خصوص تمام کدهای موجود در فایل اجرایی.

(ب) تصاویری از خروجی فایل اجرایی.

(ج) نیازمندی‌های لازم جهت اجرای فایل اجرایی.

آزمایش ۱۱

آشنایی با صفحه بندی حافظه

۱.۱۱ مقدمه

در سیستم های عامل، صفحه‌بندی یکی از روشهای مدیریت حافظه است که در آن یک کامپیوتر می‌تواند اطلاعات را برای استفاده در حافظه اصلی، در یک رسانه ذخیره‌سازی ثانویه ذخیره و بازیابی کند. در این روش مدیریت حافظه، سیستم‌عامل اطلاعات را در قالب بلاک‌های هم اندازه‌ای که صفحه نامیده می‌شوند، از رسانه ذخیره‌سازی ثانویه (مانند دیسک سخت) بازیابی می‌کند. مزیت اصلی صفحه‌بندی نسبت به روش قطعه بندی این است که در روش صفحه‌بندی، فضای آدرس‌دهی فیزیکی یک فرایند می‌تواند غیر پیوسته باشد. یعنی مثلاً یک فرایند می‌تواند در قسمت‌های مختلف حافظه قرار بگیرد. تا قبل از ابداع این روش، سیستم‌ها مجبور بودند کل یک برنامه را به صورت پیوسته در حافظه ذخیره کنند که این امر مشکلاتی را برای مدیریت حافظه بوجود می‌آورد.

هدف این آزمایش آشنایی با مفهوم صفحه‌بندی است. به عبارت دیگر اهداف این آزمایش عبارتند از:

۱. آشنایی با مفهوم صفحه‌بندی

۲. آشنایی با مفهوم فقدان صفحه یا Fault Page

۳. آشنایی با انواع فقدان صفحه

۲.۱۱ پیش‌نیاز نظری

کارکرد اصلی صفحه‌بندی زمانی است که برنامه‌ای سعی می‌کند به صفحاتی دسترسی پیدا کند که در حال حاضر در حافظه اصلی وجود ندارند. به این حالت نقص صفحه می‌گویند. سیستم‌عامل باید کنترل را در دست گرفته و نقص صفحه را مدیریت کند. مدیریت نقص صفحه باید از دید برنامه مخفی باشد و برنامه متوجه آن نشود.

۳.۱۱ آزمایش ۱

یک برنامه سطح کاربر و یک برنامه سطح هسته در سیستم عامل لینوکس اجرا کرده و مشخص کنید که این برنامه ها در چه آدرسی از حافظه قرار دارند؟ انتظار می رود خروجی این آزمایش شامل موارد زیر باشد.

۱. یک فایل اجرایی که همان کد پیاده‌سازی شده توسط دانشجو است.

۲. یک مستند که حاوی موارد زیر است.

(آ) توضیحاتی در خصوص تمام کدهای موجود در فایل اجرایی.

(ب) تصاویری از خروجی فایل اجرایی.

(ج) نیازمندی‌های لازم جهت اجرای فایل اجرایی.

۴.۱۱ آزمایش ۲

در سیستم‌عامل لینوکس چند نوع fault Page وجود دارد؟ برنامه‌ای که در آزمایش پیشین استفاده نموده اید در حین اجرا ممکن است چندین بار با نقص صفحه مواجه شده باشد. مشخص کنید این برنامه در حین اجرا چند بار با خطای Fault Page مواجه شده است. انتظار می رود خروجی این آزمایش شامل موارد زیر باشد.

۱. یک فایل اجرایی که همان کد پیاده‌سازی شده توسط دانشجو است.

۲. یک مستند که حاوی موارد زیر است.

(آ) توضیحاتی در خصوص تمام کدهای موجود در فایل اجرایی.

(ب) تصاویری از خروجی فایل اجرایی.

(ج) نیازمندی‌های لازم جهت اجرای فایل اجرایی.