# Homework 2

# Lectures 3, 4, 5, 6, 7, 8

## Operating Systems

Dr. Javadi

Spring 2023

۱- به هنگام برنامه‌نویسی، حتی برنامه Hello, World که به هر زبانی بنویسید وارد رزومه خود می‌کنید :) از فراخوانی‌های سیستمی بسیاری استفاده می‌کنید. اغلب سیستم‌عامل‌های به‌روز از چندصد فراخوان سیستمی استفاده می‌کنند (حتی ممکن است برای عملیات‌های خیلی خاص مانند حساب کردن لگاریتم نیز فراخوانی سیستم وجود داشته باشند. با این حال اکثر برنامه نویسان هیچوقت این سطح از جزئیات را نمی‌بینند. به طور معمول، برنامه نویسان با کمک رابط برنامه نویسی یا API به پیاده‌سازی نرم‌افزارهای پیچیده می‌پردازند. این روابط برنامه نویسی خود فراخوانی‌های سیستمی را استفاده می‌کنند.

الف) در ابتدا توضیح دهید چرا برنامه نویسان بجای استفاده مستقیم فراخوانی‌های سیستمی از رابط برنامه نویسی استفاده می‌کنند.

ب) یکی از عوامل مهم در رسیدگی به فراخوانی‌های سیستمی، محیط زمان اجرا یا Runtime Environment است. محیط زمان اجرا یا به طور اختصار RTE مجموعه کامل از نرم‌افزار مورد نیاز برای اجرای برنامه‌های کاربردی نوشته شده در یک زبان برنامه‌نویسی خاص، از جمله کامپایلرها یا مفسرهای آن و همچنین نرم‌افزارهای دیگر مانند کتابخانه‌ها و لودرها است. تحقیق کنید و توضیح دهید، وجود RTE چگونه باعث می‌شود تا استفاده از فراخوانی‌های سیستمی راحت‌تر بشود؟

الف) هر سیستم‌عامل رابط‌های برنامه‌نویسی (API - Application Prorgrmming Interface) را در اختیار برنامه‌نویسان قرار می‌دهد تا با استفاده از آن‌ها فراخوانی‌های سیستمی را انجام دهند. به جای برنامه‌نویسان، APIها در پشت صحنه فراخوانی‌های سیستمی را انجام می‌دهند. دلایلی که برنامه‌نویسان از این رابط‌ها استفاده می‌کنند عبارت است از:

۱. کد نوشته شده در هر سیستم دیگری که از آن APIها پشتیبانی کند کامپایل و اجرا می‌شود اما فراخوانی‌های سیستمی در هر سیستمی متفاوت هستند. (البته استفاده از APIها هم لزوما تضمین نمی‌کند که یک برنامه قادر به کامپایل و اجرا به روی هر سیستمی باشد)

۲. ساختار و روش استفاده از APIها ساده‌تر از فراخوانی مستقیم system callهاست و ورودی دادن و فراخوانی sys call می‌تواند برای برنامه‌نویس سختی و پیچیدگی ایجاد کند.

ب) یکی از مهم‌ترین عوامل در انجام فراخوانی سیستمی، محیط اجرایی (RTE) است. RTE مجموعه کاملی از نرم‌افزارهای لازم برای اجرای برنامه‌های نوشته شده با یک زبان برنامه‌نویسی خاص است. این مجموعه شامل کامپایلرها یا مفسرها و نرم‌افزارهای دیگر مانند کتابخانه‌ها و لودرها می‌شود. محیط اجرایی یک رابط فراخوانی سیستمی (system-call interface) را فراهم می‌کند که به عنوان پیوندی میان برنامه و فراخوانی‌های سیستمی که توسط سیستم عامل ارائه شده‌اند، عمل می‌کند. رابط فراخوانی سیستم، فراخوانی‌های APIها را رهگیری می‌کند و فراخوانی‌های سیستمی مورد نیاز را در سیستم عامل اجرا می‌کند. به طور معمول، یک شماره به هر فراخوانی سیستمی اختصاص داده می‌شود و رابط فراخوانی سیستم یک جدول با شماره‌های مربوطه در نظر می‌گیرد. پس از آن، رابط فراخوانی سیستم، فراخوانی مورد نظر را در هسته سیستم عامل اجرا کرده و وضعیت فراخوانی سیستمی را برمی‌گرداند. به این ترتیب جزئیات فراخوانی‌های سیستمی به وسیله API از برنامه‌نویسان مخفی می‌ماند و مدیریت APIها نیز توسط RTE انجام می‌شود.

۲- شبه کد زیر را در نظر بگیرید. با در نظر گرفتن سناریوهای اجرای موفقیت آمیز و اجرایی که با خطا مواجه می‌شود:

```c
int main() {
    char input[100];
    FILE *fp;

    printf("Enter some text: ");
    fgets(input, 100, stdin);

    // Check if input is empty
    if (strlen(input) == 1) {
        printf("Error: Input is empty.\n");
        return 1;
    }

    fp = fopen("example.txt", "w");

    // Check if file was opened successfully
    if (fp == NULL) {
        printf("Error: File not found.\n");
        return 1;
    }

    fprintf(fp, "%s", input);
    fclose(fp);

    printf("Input written to file successfully!\n");

    return 0;
}
```

الف) حداقل ۷ دستور را نام ببرید که نیازمند اجرای فراخوانی سیستمی هستند.

ب) همانطور که می‌دانید توابع مورد استفاده در این کد، فراخوانی سیستمی نیستند بلکه واسطهایی برای این امر هستند. در پیاده‌سازی این توابع از چه روش‌هایی برای پاس دادن نام فایل‌های مورد استفاده به سیستم‌عامل می‌توان استفاده کرد؟

پ) آیا فایل کامپایل شده‌ای از نسخه‌ی کامل این کد را می‌توان در هر سیستم‌عاملی اجرا کرد؟ توضیح دهید.

الف) char input و FILE * و printf و fgets و fopen و fclose و fprintf

ب) سه راه عمده برای پاس دادن متغیرها به سیستم عامل وجود دارد. ساده‌ترین مسیر، پاس دادن متغیرها با registerهاست. در صورتی که تعداد پارامترها از ثبات‌ها بیشتر باشد، در این صورت پارامترها داخل بلوک‌ها یا جدول‌هایی در مموری ذخیره می‌شوند و آدرس آن‌ها در ثبات‌ها قرار داده می‌شود. همچنین استفاده از stack یا پشته نیز راه دیگری برای پاس دادن پارامترها به سیستم عامل است.

پ) خیر. زیرا هر سیستم عامل، فراخوانی‌های سیستمی مختص به خود را داراست که برنامه با استفاده از APIهای تعریف‌شده، آن‌ها را فراخوانی می‌کند و APIها نیز توسط RTEها مدیریت می‌شوند. این interfaceها در زمان کامپایل مشخص می‌شوند. به همین دلیل فایل کامپایل شده روی یک سیستم عامل روی سیستم عامل دیگری قابل اجرا نیست. برای اجرا لازم است که کدهای این برنامه در سیستم عامل مقصد کامپایل شده باشند. (یا آنکه برای کامپایل از یک cross-compiler استفاده شده باشد) علاوه بر این هر کدام از سیستم‌عامل‌ها binary format خاص خود را برای برنامه‌ها تعریف می‌کنند که جایگاه header و دستورات و متغیرها در آن متفاوت است. علاوه بر این‌ها CPUهای متفاوت

مجموعه دستورات متفاوتی دارند. به همین دلایل، برنامه‌ای که روی یک سیستم عامل و با کامپایلر مختص به همان سیستم عامل، کامپایل شده باشد روی سیستم عامل دیگری قابل اجرا نیست.

۳- فراخوانی سیستمی در چه دسته‌بندی‌ای از انواع وقفه قرار می‌گیرد؟ به صورت کلی و خلاصه، مراحل اجرای یک فراخوانی سیستمی به عنوان یک وقفه را توضیح دهید.

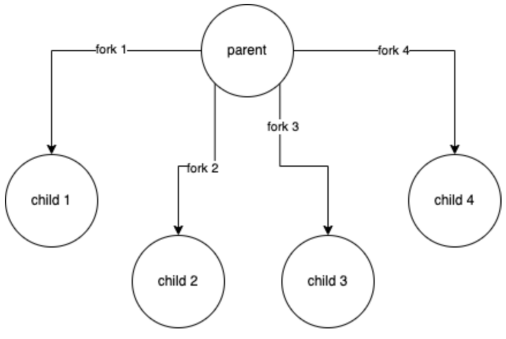
یک وقفهی همگام یا نرمافزاری است به این معنا که توسط پردازهای که داخل cpuدر حال اجراست ایجاد میشود. پردازهی در حال اجرا با فراخوانی یک سیستم کال، indexمربوط به سیستم کال درخواستی را در رجیستر eaxقرار میدهد. کنترل سیستم عامل به حالت کرنل منتقل میشود، مقادیر رجیسترهای پردازهی در حال اجرا در استک کرنل ذخیره میشود. کرنل، ISRمربوط به وقفهی دریافتی را اجرا میکند (سیستم کال درخواستی را اجرا میکند .)سپس حالت رجیسترهای برنامه را از سر میگیرد و سیستمعامل مجددا به حالت کاربر باز میگردد.

۴- خروجی قطعه‌کدهای زیر چیست؟ ضمن رسم درخت پردازه‌های هر برنامه، راه حل خود را توضیح دهید.

الف)

```c
int main() {
   fork() && fork() && fork() && fork();
   printf("+");
   return 0;
}
```

پردازه پدر اولین forkرا اجرا میکند و C۱را میسازد. با توجه به اینکه خروجی forkبرای پردازه پدر مقداری غیر از صفر است، forkدوم و سوم و چهارم را هم اجرا میکند و پردازههای C۲و C۳و C۴هم میسازد. در نهایت با چاپ +، اجرای پدر پایان مییابد. از آنجایی که مقدار خروجی forkبرای پردازهی C۱صفر بوده است به دلیل خاصیت Short-Circuitدر ANDهای متوالی، forkهای بعدی را اجرا نمیکند، +را پرینت میکند و اجرایش پایان مییابد. به همین ترتیب پردازههای C۲تا C۴هم هیچ forkای اجرا نمیکنند و بعد از چاپ +اجرایشان به پایان میرسد. خروجی این قطعه کد، +++++ است و درخت پردازه‌های آن به شکل زیر است:

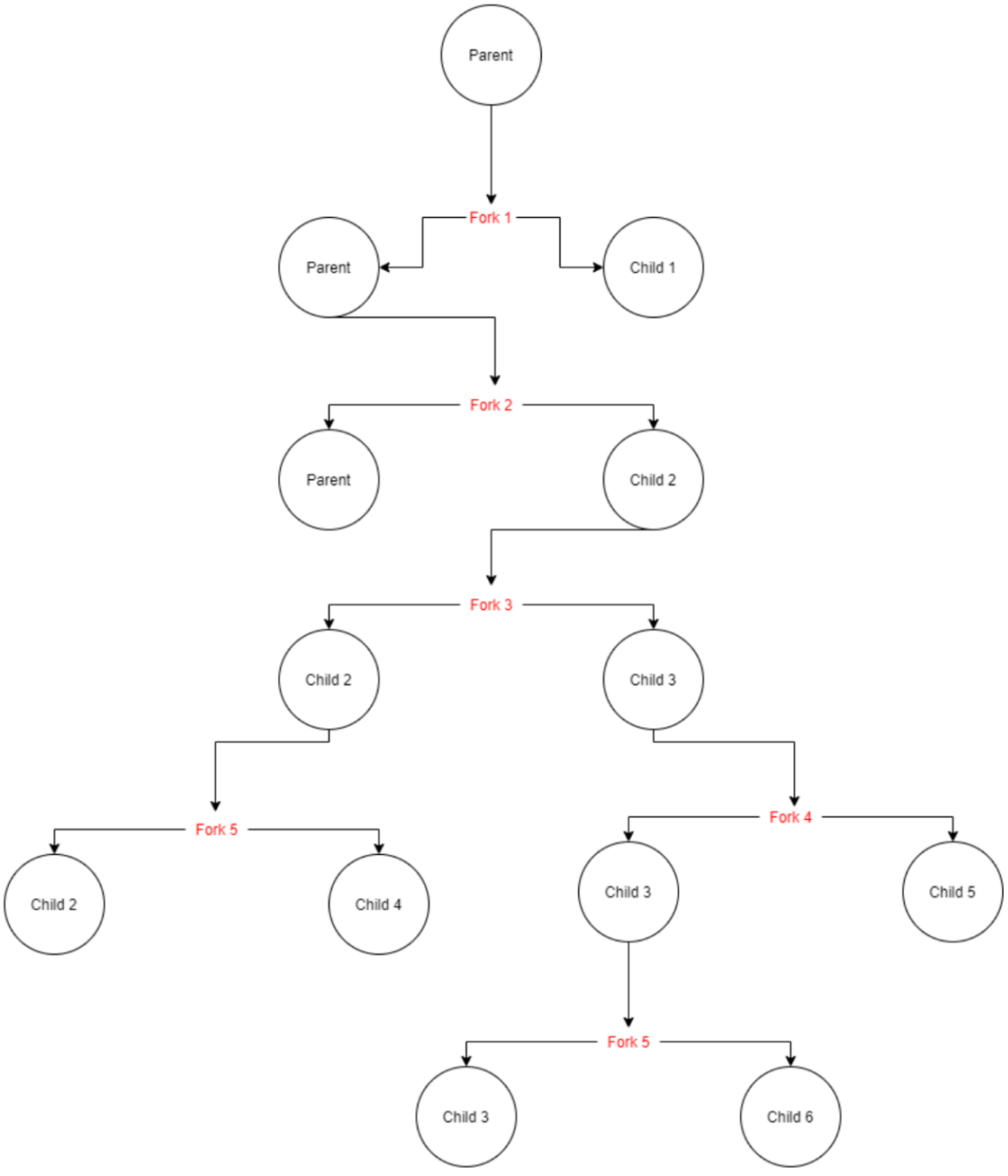


ب)

```c
int main() {
(
  if (fork() && (!fork())) {
    if (fork() || fork()) {
      fork();
    }
  }

  return(0);
}
```

ابتدا پردازه پدر یک فرزند به اسم C۱میسازد. به دلیل وجود اپراتور && و ویژگی Short-Circuitادامه شرط برای C۱اجرا نمیشود و تنها برای پردازه پدر بار دیگر اجرا شده و یک فرزند دیگر به نام C۲ساخته میشود. در نتیجه پردازه C۲وارد شرط میشود و پردازه C۱و پردازه پدر از ifخارج شده و اجرایشان تمام میشود. پردازه C۲وارد شرط دوم شده و در اجرای اولین forkیک فرزند C۳را ایجاد میکند. سپس به دلیل trueبودن مقدار بازگشتی از forkو وجود اپراتور ||وارد شرط میشود C۴وارد بدنه شرط دوم میشود و در نتیجه با اجرای forkداخل شرط یک فرزند دیگر به اسم C۴ساخته و به کار پایان میدهد. پردازه C۳وارد forkدوم از شرط دوم شده و فرزند C۵را ایجاد میکند و خود این بار وارد شرط میشود و با اجرای forkباعث ایجاد C۶میشود. در حالی که C۵از شرط دوم خارج میشود. در ادامه به دستور returnمیرسیم و اجرا پایان مییابد. این برنامه خروجیای ندارد و درخت پردازههای آن به صورت زیر است:

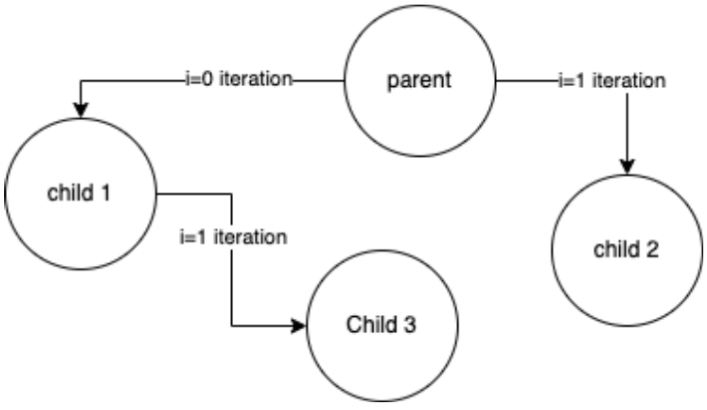

Parent
Fork 1
Parent
Child 1
Fork 2
Parent
Child 2
Fork 3
Child 2
Child 3
Fork 5
Fork 4
Child 2
Child 4
Child 3
Child 5
Fork 5
Child 3
Child 6

ج)

```c
int main() {
(
  int i = 0;
  while (i<2) {
    printf("%d", i);
    fork();
    i++;
  }

  return(0)
}
```

پردازه پدر وارد whileمیشود، در این لحظه i = ۰ و این عدد را پرینت میکند. سپس با اجرای forkپردازه C۱را اجرا میکند . i++که اجرا شود برای i = ۱هم همین فرایند تکرار میشود. پس تا اینجا حتما ۰ و ۱ در کنسول چاپ میشوند و پردازه C۱و C۲ ایجاد میشوند C۱ .بعد از ایجاد شدن، دستور i++را اجرا میکند و در اینجا i = ۱میشود پس C۱وارد حلقه میشود، ۱ را چاپ میکند و C۳را ایجاد میکند. بعد از آن حلقه و اجرای C۱به پایان میرسد. در زمانی که پردازه پدر C۲را ایجاد کرد، i=۱بود و اولین دستوری که C۲اجرا میکند i++است پس چون i = ۲میشود، C۲دیگر وارد حلقه نمیشود و اجرایش تمام میشود. برای C۳نیز همین اتفاق میافتد .با اینکه نمیتوان دقیقا مطمئن بود که اجرای C۱و ادامهی اجرای پردازهی پدر به چه ترتیبی است ولی خروجی برنامه در هر صورت ۰۱۱ است.



۵- زمانی که فراخوان سیستمی fork صدا زده می‌شود معمولا یکی از دو فرآیند (والد یا فرزند) یک فراخوان سیستمی دیگر به نام exec را صدا می‌زنند. تحقیق کنید و توضیح دهید که این فراخوان سیستمی چه کاری انجام می‌دهد و چرا یکی از دو فرآیند (والد یا فرزند) پس از اجرای دستور fork آن را صدا می‌زنند؟

فراخوانی سیستمی forkبرای ایجاد یک پردازه جدید با تکثیر فرایند کنونی استفاده میشود، به این معنی که محتویات حافظهی پردازهی پدر و باقی اطلاعات ذخیره شدهی مربوط به آن program imageعینا برای پردازهی فرزند ایجاد میشوند. در حالی که فراخوانی سیستمی execبرای جایگزینی program imageفعلی پردازه با یک برنامه جدید استفاده میشود. ترکیب این دو فراخوانی سیستمی، به یک پردازه اجازه میدهد تا یک پردازه جدید با یک برنامه متفاوت ایجاد کند و به طور مؤثر یک پردازه جدید را برای انجام یک کار خاص راهاندازی کند .برنامهنویس میتواند با توجه به خروجی forkکنترل کند که برنامهی جدید روی پردازهی پدر اجرا شود یا پردازهی فرزند.

**سوال عملی**

در سیستم عامل لینوکس و به زبان C کدی بنویسید که یک پردازه فرزند ایجاد کرده و آن را تبدیل به zombie کند. این پردازه باید حداقل به مدت ۱ دقیقه در سیستم باقی بماند. شما می‌توانید با استفاده از دستور ps -l وضعیت پردازه‌های خود را مشاهده کنید. اجزای مختلف خروجی دستور ps -l را توضیح دهید و پردازه zombie ایجاد شده را مشخص کنید. از کد خود به همراه خروجی دستور ps -l اسکرین شات بگیرید و همراه با توضیحات مربوط به کد و همچنین توضیحات دستور ps -l ارسال کنید.

به نکات زیر توجه کنید.

- مهلت ارسال تمرین ساعت ۲۳:۵۹ روز پنجشنبه ۱۸ فروردین ماه می‌باشد.
- در صورت کشف تقلب نمره تمرین ۰ در نظر گرفته می‌شود.
- سوالات خود را می‌توانید از طریق تلگرام از تدریسیارهای گروه خود بپرسید.
- فایل پاسخ تمرین را تنها با قالب **HW?_StudentNumber.pdf** در کورسز بارگزاری کنید.
  - نمونه: HW2_9831072