

جلسه‌ی ششم — مدیریت پردازشها

در این جلسه با توابعی برای مدیریت پردازشها در سیستم عامل‌های مشابه یونیکس آشنا خواهید شد. این جلسه به سه بخش تقسیم شده است. بخش اول ساختن یک پردازش جدید را شرح می‌دهد، بخش دوم به اجرای یک برنامه که در فایل سیستم قرار دارد، می‌پردازد و بخش سوم شیوه‌ی انتظار در یک پردازش برای پردازش‌های فرزند آن را توصیف می‌نماید.

ایجاد یک پردازش

۱ با فرافروانی تابع `fork()` سیستم عامل پردازش جدیدی ایجاد می‌کند. با توجه به فروجی کد زیر، توضیح دهید فروجی این تابع در پردازش پدر و فرزند چه مقداری است.

```
printf("PID: %d\n", getpid());           چاپ شناسه‌ی پردازش  
ret = fork();                            ایجاد یک پردازش جدید  
printf("PID: %d - %d\n", getpid(), ret);  شناسه‌ی پردازش و فروجی fork()
```

۲ در مثال زیر، هر یک از پیغامها در چه پردازش‌های (فرزند یا پدر) چاپ می‌شود؟

```
if (fork() > 0)  
    printf("A\n");                       در صورتی که فروجی fork() بزرگ‌تر از صفر باشد  
else  
    printf("B\n");                       در صورتی که فروجی fork() صفر باشد
```

اجرای یک برنامه

۳ با تابع `execvp()` (مثل سایر توابع خانواده `exec()`) سیستم عامل یک برنامه را اجرا می‌کند. ورودی اول این تابع، آدرس برنامه‌ی مورد نظر است. پس از این فراخوانی، قسمت‌های کد و داده‌ی پردازش از بین می‌روند و با مقدار مناسب برای پردازش جدید جایگزین می‌گردند.

```
char *argv[] = {"ls", NULL};           آرایی پارامترها  
execvp("ls", argv);                   اجرای برنامه با پارامترهای داده شده
```

۴ ورودی دوم `execvp()` آرایی است که پارامترهایی که به پردازش ایجاد شده فرستاده می‌شوند (ورودی‌های فرستاده شده به تابع `main()` در یک برنامه)، را مشخص می‌کند. این آرایه باید با یک عنصر `NULL` خاتمه پذیرد. به صورت قراردادی، در درایی صفرم این آرایه، آدرس برنامه تکرار می‌شود. در مثال زیر، پارامترهای ورودی برنامه‌ی `ls` چه هستند؟

```
char *argv[] = {"ls", "/", NULL};  
execvp("ls", argv);
```

۵ در صورت موفقیت‌آمیز بودن این فراخوانی، هیچ یک از عبارت‌های پس از این فراخوانی اجرا نمی‌شوند. در چه صورت دستور بعد از `execvp()` اجرا می‌شود؟

```
execvp("nonexistent/file", argv);     آدرس یک فایل ناموجود  
printf("After execvp()\n");
```

انتظار برای اتمام پردازشها

۶ فرافوانی سیستمی `wait()` منتظر می‌ماند تا یکی از پردازش‌های فرزند پردازش فرافوانی کننده فایده یابد. مقدار برگشت داده شده از این تابع، شماره‌ی PID پردازش فایده یافته است و اطلاعاتی در مورد فایده‌ی این پردازش (از جمله مقدار کد برگشتی آن) در متغیری که آدرس آن به این تابع فرستاده می‌شود قرار می‌گیرد. در مثال زیر، شیوه‌ی استفاده از `wait()` نمایش داده شده است.

```
int pid, status;
pid = wait(&status);
```

انتظار برای فایده‌ی یک پردازش فرزند

۷ با استفاده از ماکروی `WEXITSTATUS` می‌توان کد برگشتی یک برنامه را از مقداری که `execvp()` در متغیر `status` قرار می‌دهد، استخراج نمود.

```
printf("pid %d exited with return code %d\n",
      pid, WEXITSTATUS(status));
```

تمرین‌ها

۸ در قطعه کد زیر چند پردازش ساخته می‌شوند؟ درخت پردازش‌ها را بکشید و مشخص کنید هر پیغام توسط کدام پردازش چاپ می‌شود.

```
fork();
if (fork())
    printf("A\n");
else
    printf("B\n");
```

۹ برنامه‌ای به نام ex6.c بنویسید که دو پردازش تولید کند: پردازش جدید اول پس از یک ثانیه مرف A را چاپ کند، پردازش جدید دوم پس از دو ثانیه مرف B را چاپ کند. برای انتظار می‌توانید تابع sleep() را فراخوانی کنید.

A	در ثانیه‌ی اول توسط پردازش اول
B	در ثانیه‌ی دوم توسط پردازش دوم

انجام در جلسه‌ی آینده در صورت کمبود وقت

۱۰ برنامه‌ی ex6.c را به شکلی تغییر دهید که پس از فاصله‌ی هر پردازش، پردازش اصلی مرف C را چاپ کند (چگونه می‌توان متناظر فاصله‌ی یک پردازش در پردازش اصلی شد؟).

A	در ثانیه‌ی اول توسط پردازش اول
C	توسط پردازش اصلی
B	در ثانیه‌ی دوم توسط پردازش دوم
C	توسط پردازش اصلی

۱۱ برنامه‌ی ex6.c را به شکلی تغییر دهید که به جای فراخوانی تابع sleep()، برنامه‌ی sleep را با تابع execvp() اجرا کند.