جلسه‌ی ششم — مدیریت پردازه‌ها

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

ایجاد یک پردازه

۱ با فراخوانی تابع fork() سیستم عامل پردازه‌ی جدیدی ایجاد می‌کند. با توجه به خروجی کد زیر، توضیح دهید خروجی این تابع در پردازه‌ی پدر و فرزند چه مقداری است.

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

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

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

اجرای یک برنامه

۳ با تابع execvp() (مثل سایر توابع خانواده‌ی 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 پردازه از بین می‌روند و با مقدار مناسب برای پردازه‌ی جدید جایگزین می‌گردند.

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

۴ ورودی دوم execvp() آرایه‌ای است که پارامترهایی که به پردازه‌ی 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 درایه‌ی 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 پردازه‌ها

۶ فراخوانی سیستمی ()wait منتظر می‌ماند تا یکی از پردازه‌های فرزند پردازه‌ی فراخوانی کننده خاتمه یابد. مقدار برگشت داده شده از این تابع، شماره‌ی PID پردازه‌ی خاتمه یافته است و اطلاعاتی در مورد خاتمه‌ی این پردازه (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 خاتمه‌ی یک پردازه‌ی 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 پردازه ساخته می‌شوند؟ درخت پردازه‌ها را بکشید و مشخص کنید هر پیغام توسط کدام پردازه چاپ می‌شود.

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

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

```
در ثانیه‌ی اول توسط پردازه‌ی اول A
در ثانیه‌ی دوم توسط پردازه‌ی دوم B
```

۱۰ برنامه‌ی ex6.c را به شکلی تغییر دهید که پس از خاتمه‌ی هر پردازه، پردازه‌ی اصلی حرف C را چاپ کند (چگونه می‌توان منتظر خاتمه‌ی یک پردازه در پردازه‌ی اصلی شد؟).

```
در ثانیه‌ی اول توسط پردازه‌ی اول A
توسط پردازه‌ی اصلی C
در ثانیه‌ی دوم توسط پردازه‌ی دوم B
توسط پردازه‌ی اصلی C
```

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