جلسه‌ی ششم — انتقال داده با لوله

در این جلسه با شیوه‌ی مدیریت فایل‌های باز پردازه‌ها در یونیکس و استفاده از لوله برای انتقال اطلاعات بین آنها آشنا خواهید شد.

فراخوانی‌های سیستمی مربوط به فایل‌ها

در یونیکس علاوه بر فایل‌های ذخیره شده در دیسک، بسیاری از از منابع موجود در سیستم عامل (از جمله اتصالات شبکه، لوله‌ها و بسیاری از Device-ها از جمله کارت‌های صوتی، دیسک‌ها، حافظه و حافظه‌ی کارت‌های گرافیکی) نیز توسط فایل قابل دسترسی هستند. استفاده از فایل برای این کاربردها از یک سو موجب سادگی رابط هسته برای کنترل و دسترسی به این منابع گشته است و سوی دیگر موجب شده است بسیاری از برنامه‌ها بدون وابستگی به نوع فایل‌ها، برای همه‌ی این انواع فایل قابل استفاده باشند. در این بخش برخی از فراخوانی‌های سیستمی موجود در یونیکس برای دسترسی به فایل‌ها معرفی می‌گردند.

در یونیکس هر فایل باز[1] در یک پردازه با یک عدد که در این مستند شناسه‌ی فایل[2] نامیده می‌شود، مشخص می‌شود. به صورت قراردادی، فایل شماره‌ی صفر به ورودی استاندارد (stdin در کتابخانه‌ی استاندارد زبان C)، فایل شماره‌ی یک به خروجی استاندارد (stdout) و فایل شماره‌ی دو به خروجی خطا (stderr) اختصاص می‌یابد. پردازه‌ها می‌توانند با استفاده فراخوانی‌های سیستمی مناسب، شناسه‌های فایل جدیدی را ایجاد نمایند (برای مثال با فراخوانی سیستمی open() یا dup()) یا آنها را ببندند (با فراخوانی سیستمی close()).

فراخوانی سیستمی read() با گرفتن یک شناسه‌ی فایل، یک آرایه‌ی کارکتری و اندازه‌ی آن، از فایل مشخص شده می‌خواند. در مثال زیر، استفاده از این فراخوانی سیستمی نشان داده شده است.

```
/* declares: ssize_t read(int fd, void *buf, size_t n) */
#include <unistd.h>

/* an example for using read() system call */
char buf[128];
```

۱ Open file
۲ File descriptor

```
/* nr bytes has been read into buf from FD 0 */
ssize_t nr = read(0, buf, 128);
```

مقدار برگشت داده شده توسط این تابع تعداد بایت‌های خوانده شده از شناسه‌ی فایلی که با ورودی اول داده می‌شود را بر می‌گرداند. در صورتی که خطایی در این فراخوانی سیستمی رخ دهد (مشابه بسیاری از فراخوانی‌های سیستمی دیگر) یک عدد منفی برگشت داده می‌شود و عدد صفر به این معنی است که همه‌ی محتوای فایل خوانده شده است.

فراخوانی سیستمی write() بایت‌های داده شده را (که توسط یک اشاره‌گر و تعداد بایت‌ها مشخص می‌شود) را در یک فایل می‌نویسد. عدد برگردانده شده توسط فراخوانی سیستمی write تعداد بایت‌های نوشته شده در شناسه‌ی فایل داده شده را مشخص می‌کند. در صورتی که خطایی رخ دهد، عددی منفی از این فراخوانی سیستمی برگردانده خواهد شد.

```
/* declares: ssize_t write(int fd, void *buf, size_t n) */
#include <unistd.h>

/* an example for using write() system call */
char buf[] = "Hello World!\n";
/* nw bytes has been written from buf into FD 1 */
ssize_t nw = write(1, buf, 12);
```

فراخوانی سیستمی open() یک فایل در فایل سیستم را باز می‌کند و به آن یک شناسه‌ی فایل آزاد (که در حال استفاده نیست) تخصیص می‌دهد. فراخوانی سیستمی close() یک شناسه‌ی فایل را می‌بندد و سپس، شناسه‌ی فرستاده شده به این فراخوانی سیستمی آزاد می‌شود. برای جزئیات بیشتر به صفحه‌ی راهنمای این فراخوانی‌ها مراجعه شود.

با استفاده از تابع fdopen() می‌توان یک FILE از شناسه‌ی فایل داده شده ایجاد نمود تا از توابع ورودی و خروجی کتابخانه‌ی استاندارد C برای خواندن و نوشتن به آن استفاده کرد:

```
FILE *fp = fdopen(fd, "w");
fprintf(fp, "Hello\n");
fclose(fp);
```

استفاده از لوله‌ها در پوسته

لوله‌ها[1] در پوسته (که در جلسه‌های گذشته معرفی شده‌اند) با استفاده از فراخوانی سیستمی pipe() پیاده‌سازی می‌شوند. لوله یک بافر[2] (یعنی حافظه‌ی محدودی که برای انتقال داده‌ها استفاده می‌گردد) در سیستم عامل است که با دو شناسه‌ی فایل قابل دسترسی می‌باشد: یک شناسه‌ی فایل برای سر نوشتن و دیگری برای سر خواندن. با شناسه‌ی نوشتن یک لوله، می‌توان داده‌ها را به لوله انتقال داد (فراخوانی سیستمی write()). به صورت مشابه، با استفاده از شناسه‌ی خواندن یک لوله، می‌توان داده‌های نوشته شده به یک لوله را توسط فراخوانی سیستمی read() خواند.

می‌توان با فراخوانی سیستمی pipe() یک لوله ساخت: این تابع یک لوله می‌سازد و شناسه‌ی فایل دو سر این فایل را در یک آرایه‌ی با طول دو که به عنوان ورودی به آن داده می‌شود می‌نویسد.

```
int fds[2];
pipe(fds);              /* fds[0] for reading and fds[1] for writing */
```

با استفاده از یک پایپ می‌توان داده‌هایی را بین دو پردازه انتقال داد و معمولا پس از این فراخوانی، با فراخوانی سیستمی fork() پردازه‌ی جدیدی ساخته می‌شود. سپس یکی از این پردازه‌ها از سر نوشتن لوله داده‌ها را می‌نویسد و پردازه‌ی دیگر از سر خواندن لوله، داده‌ها را می‌خواند:

```
char buf[100];
pipe(fds);
if (fork()) {           /* the parent process; writing */
      close(fds[0]);
      write(fds[1], "Hello\n", 6)
} else {                /* the child process; reading */
      close(fds[1]);
      read(fds[0], buf, 100);
}
```

---

۱ Pipe
۲ Buffer

تمرین ششم

برنامه‌ای را در نظر بگیرید که پردازش را در دو گام انجام می‌دهد: در گام اول، داده‌هایی را تولید می‌کند و در گام دوم این داده‌ها را پردازش می‌نماید. فرض کنید بیش از دو پردازنده در سیستم عامل برای اجرای این برنامه موجود باشند؛ این برنامه فقط می‌تواند از یکی از این پردازنده‌ها به صورت همزمان استفاده نماید. برای استفاده از دو پردازنده، می‌توان برنامه را تغییر داد تا با استفاده از لوله برای انتقال داده‌ها، پردازش را به دو قسمت تقسیم کند که در دو پردازه‌ی مجزا اجرا شوند. این تغییرات را انجام دهید.

پس از دریافت فایل ex6.c، آن را تغییر دهید. در این برنامه، قسمت اول پردازش در تابع prod() و قسمت دوم در تابع cons() انجام می‌شود (در حلقه‌ی تابع main()، هر خروجی prod() به تابع cons() فرستاده می‌شود). ابتدا با فراخوانی سیستمی pipe() یک لوله ایجاد نمایید و سپس با فراخوانی سیستمی fork() یک پردازه‌ی جدید بسازید. در پردازه‌ی پدر، تابع prod() را صدا بزنید و خروجی آن را به سر نوشتن لوله بنویسید. در پردازه‌ی فرزند، داده‌هایی که توسط پردازه‌ی پدر نوشته می‌شود را از سر خواندن لوله بخوانید و به تابع cons() بفرستید.

گام‌های پیشنهادی برای انجام این تمرین:

۱ دریافت و ترجمه‌ی فایل ex6.c

۲ ایجاد یک پردازه‌ی جدید با فراخوانی fork()

۳ ساختن یک لوله قبل از ایجاد پردازه‌ی جدید با فراخوانی pipe()

۴ ایجاد یک FILE با fdopen() برای سر نوشتن در پردازه‌ی پدر و برای سر خواندن در فرزند

۵ آزمایش درستی عملکرد لوله برای انتقال یک رشته‌ی آزمایشی

۶ نوشتن اعداد محاسبه شده توسط prod() در پردازه‌ی پدر به لوله و خواندن و فرستادن آنها به تابع cons() در پردازه‌ی فرزند