جلسه‌ی هفتم — 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.

فایل‌ها در یونیکس

در یونیکس علاوه بر فایل‌های ذخیره شده در دیسک، بسیاری 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 مدیریت این منابع و دسترسی به آنها 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 توابع موجود در یونیکس برای دسترسی به فایل‌ها معرفی می‌گردند؛ این توابع در بیشتر سیستم‌های عامل مبتنی بر یونیکس فراخوانی‌های سیستمی هستند.

شناسه‌های فایل

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 توابع مناسب، شناسه‌های فایل جدیدی را ایجاد نمایند (برای مثال با فراخوانی تابع open() یا dup()) یا آنها را ببندند (با فراخوانی تابع close()).

خواندن از و نوشتن به فایل‌ها

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

1 Open file
2 File descriptor

```c
#include <unistd.h>

char buf[128];                  /*      خروجی تابع read() در این آرایه ریخته می‌شود */
ssize_t nr = read(0, buf, 128); /*                یک مثال از فراخوانی تابع read() */
```

مقدار برگشت داده شده توسط این تابع (متغیر «nr» در مثال قبل) تعداد بایت‌های خوانده شده از شناسه‌ی فایلی که با ورودی اول داده می‌شود را نشان می‌دهد. در صورتی که خطایی در فراخوانی این تابع 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.

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

```c
#include <unistd.h>

char buf[] = "Hello World!\n";  /*                رشته‌ای که نوشته می‌شود */
ssize_t nw = write(1, buf, 12); /*  تعداد بایت‌های نوشته شده در «nw» قرار می‌گیرد */
```

تابع open() یک فایل در فایل سیستم را باز می‌کند و به آن یک شناسه‌ی فایل آزاد (که در حال استفاده نیست) تخصیص می‌دهد. تابع 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.

استفاده از توابع کتابخانه‌ای زبان C برای دسترسی به شناسه‌های فایل

استفاده‌ی مستقیم از توابع read() و write() در برخی شرایط دشوار است؛ این توابع فقط رشته‌ها را می‌پذیرند (برای مثال، اعداد را نمی‌توان مستقیما توسط این دو تابع چاپ کرد) و همچنین باید خروجی این توابع بررسی شود تا تعداد بایت‌های نوشته شده یا خوانده شده (که می‌تواند کمتر از مقدار درخواست شده باشد) مشخص گردد. برای رفع این مشکل، می‌توان برای این شناسه‌ها یک داده از نوع «FILE» ایجاد نمود و سپس با استفاده از توابع کتابخانه‌ی استاندارد زبان C مثل fprintf() و fscanf() به آنها به صورت غیر مستقیم دسترسی داشت. برای ساخت یک «FILE» از یک شناسه‌ی فایل می‌توان از تابع fdopen() استفاده نمود. در مثال زیر، فراخوانی این تابع نشان داده شده است.

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

پارامتر دوم تابع ()fdopen (مشابه تابع ()fopen) عمل قابل انجام روی فایل را مشخص می‌کند: «w» برای نوشتن به فایل و «r» برای خواندن از آن.

ایجاد لوله

لوله‌ها[1] (که در جلسه‌های گذشته معرفی شدند) در یونیکس با استفاده از فراخوانی سیستمی ()pipe ساخته 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. داده‌هایی که به سر نوشتن لوله نوشته می‌شوند، از سر خواندن آن خوانده می‌شوند. با شناسه‌ی نوشتن یک لوله، می‌توان داده‌ها را به لوله انتقال داد (تابع ()write). به صورت مشابه، با استفاده از شناسه‌ی خواندن یک لوله، می‌توان داده‌های نوشته شده به یک لوله را توسط تابع ()read خواند.

تابع ()pipe 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.

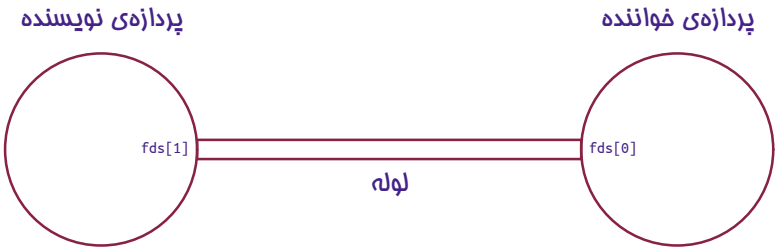
```
int fds[2];          /*  شناسه‌های خواندن و نوشتن در این آرایه قرار می‌گیرند  */
pipe(fds);           /*  fds[0] سر خواندن و fds[1] سر نوشتن هستند  */
```

با استفاده از یک پایپ می‌توان داده‌هایی را بین دو پردازه انتقال داد و معمولا پس از این فراخوانی، با تابع ()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()) {                       /*  پردازه‌ی پدر: نویسنده  */
      close(fds[0]);                /*  احتیاجی به سر خواندن در نویسنده نیست  */
      write(fds[1], "Hello\n", 6);  /*  نوشتن رشته‌ای در لوله  */
} else {                            /*  پردازه‌ی فرزند: خواننده  */
      close(fds[1]);                /*  احتیاجی به سر نوشتن در خواننده نیست  */
      read(fds[0], buf, 100);       /*  خواندن از لوله  */
}
```

1 Pipe
2 Buffer

در شکل زیر، استفاده از لوله بین دو پردازه‌ی مثال قبل نشان داده شده است.



تمرین هفتم

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 «ex7.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 تابع ()pipe یک لوله ایجاد نمایید و سپس با تابع ()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 «ex7.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