

## جلسه نهم — سیگنال‌ها

در این جلسه با سیگنال‌ها در یونیکس و شیوه‌ی دریافت و ایجاد آنها، و همچنین با مدیریت دسترسی به فایل‌ها آشنا خواهید شد.

### سیگنال‌ها

سیستم عامل می‌تواند با استفاده از سیگنال<sup>1</sup> پردازش‌ها را از رخدادهای خارجی مطلع سازد. سیگنال‌ها برای اهداف مختلفی استفاده می‌شوند: گاهی برای اطلاع از گذشت زمان مشخص (مثل فراخوانی سیستمی `alarm()`)، گاهی برای گزارش اشکال در اجرای پردازش (مثل اشکال در دسترسی به حافظه)، گاهی برای ارتباط بین پردازش‌ها و گاهی برای اطلاع پردازش از درخواست‌های خارجی (مثل درخواست اتمام پردازش).

سیستم عامل برای هر سیگنال (که با یک عدد مشخص می‌شود) به صورت پیش‌فرض عمل خاصی را در هر پردازش انجام می‌دهد (این عملیات پیش‌فرض در صفحه‌ی راهنمای «signal» شرح داده شده‌اند). هر پردازش می‌تواند عملی که باید بعد از رخداد هر سیگنال (با چند استثنا) انجام شود را تغییر دهد. یکی از راه‌های انجام این کار، استفاده از فراخوانی سیستمی `signal()` می‌باشد. فراخوانی سیستمی `signal()` دو ورودی دریافت می‌کند: ورودی اول شماره‌ی سیگنال و ورودی دوم عملی که باید پس از رخداد سیگنال انجام شود را مشخص می‌کنند.

```
signal(SIGINT, SIG_DFL); /* انجام عمل پیش فرض */
signal(SIGINT, SIG_IGN); /* دور انداختن سیگنال (پس از سیگنال عملی انجام نمی‌شود) */
signal(SIGINT, func); /* در صورت بروز سیگنال تابع func صدا زده می‌شود */

void func(int signo) /* تابع «func» باید به این صورت تعریف شده باشد */
{
    printf("Signal %d\n", signo);
}
```

---

1 Signal

همان طور که در این مثال مشاهده می‌شود، شماره‌ی سیگنال‌ها توسط ماکروهایی<sup>1</sup> (که در فایل «signal.h» تعریف شده‌اند) مشخص می‌شود. عملی که باید پس از رخداد سیگنال انجام شود با چند ماکرو یا یک تابع مشخص می‌شود. ماکروی «SIG\_DFL» عمل پیش‌فرض را مشخص می‌کند و ماکروی «SIG\_IGN» به این مفهوم است که سیگنال باید نادیده گرفته شود. در صورتی که یک تابع به عنوان ورودی دوم به signal() داده شود، در صورت رخداد سیگنال تابع مشخص شده فراخوانی می‌گردد. در مثال قبل این سه حالت نمایش داده شده‌اند.

یکی از سیگنال‌هایی که بسیاری از برنامه‌ها رفتار آن را تغییر می‌دهند سیگنال SIGINT است؛ در صورتی که کاربر (با استفاده از کلیدهای کنترل و C) درخواست فاتمه‌ی یک برنامه را داشته باشد این سیگنال به برنامه فرستاده می‌شود که به صورت پیش‌فرض موجب فاتمه‌ی آن می‌گردد. سیگنال SIGCHLD پس از اتمام هر یک از فرزندان پردازنده به آن ارسال می‌گردد. سیگنال‌های SIGUSR1 و SIGUSR2 برای تعامل بین پردازنده‌ها استفاده می‌شود (یک پردازنده بتواند پردازنده‌ی دیگری را از اتفاقی آگاه کند). یک پردازنده نیز می‌تواند از سیستم عامل درخواست کند تا سیگنالی به پردازنده‌ی دیگری فرستاده شود. این کار توسط فراخوانی سیستمی kill() انجام می‌شود. این فراخوانی سیستمی دو ورودی دریافت می‌کند: ورودی اول شماره‌ی پردازنده‌ی دریافت‌کننده‌ی سیگنال و ورودی دوم شماره‌ی سیگنال می‌باشد.

/\* ارسال سیگنال «SIGUSR1» به پردازنده‌ی با شناسه‌ی «pid» \*/  
kill(pid, SIGUSR1);

## مدیریت پردازنده‌ها در پوسته

با استفاده از دستور «ps» می‌توان فهرست پردازنده‌های در حال اجرا را مشاهده نمود. دستور «ps aux» فهرست همه‌ی پردازنده‌های در حال اجرا، صاحب هر پردازنده و شماره‌ی آن را نمایش می‌دهد. همچنین دستور «pstree» سافت‌آر درختی پردازنده‌ها را به صورت گرافیکی نمایش می‌دهد.

# نمایش فهرست پردازنده‌های موجود در سیستم عامل  
\$ ps aux  
# نمایش درخت پردازنده‌ها  
\$ pstree -phCU

برای فرستادن یک سیگنال از پوسته، می‌توان از دستور «kill» استفاده نمود. در دستور «kill»، نام یا شماره‌ی سیگنال بعد از علامت «-» مشخص می‌شود.

1 Macros

## مدیریت دسترسی‌ها

در سیستم‌های عامل چند کاربره<sup>۱</sup> لازم است مکانیزمی برای محافظت از فایل‌ها و به اشتراک گذاشتن آنها وجود داشته باشد. همان طور که در درس سیستم عامل اشاره می‌گردد، برای هر فایل (یا شافه) در یونیکس یک کاربر به عنوان صاحب آن مشخص می‌گردد. صاحب هر فایل می‌تواند گروه و میزان دسترسی افراد مختلف به آن فایل را مشخص نماید. برای دسترسی به فایل‌ها، افراد به سه دسته تقسیم می‌شوند: صاحب فایل، اعضای گروه فایل و سایر افراد. به ازای هر یک از این دسته‌ها، صاحب فایل می‌تواند مشخص کند که این افراد اجازه‌ی خواندن، نوشتن یا اجرای فایل را دارند یا فیر. معمولاً اجازه‌ی دسترسی‌ها به یک فایل را با یک عدد سه رقمی در مبنای هشت (مثل ۶۴۴) نمایش می‌دهند. هر رقم در این نمایش دسترسی یکی از این دسته‌ها را مشخص می‌کند. در «XYZ»، X دسترسی صاحب فایل، Y دسترسی اعضای گروه فایل و Z دسترسی سایر افراد را مشخص می‌نماید. در هر یک از این رقم‌ها، بیت کم‌ارزش توانایی اجرا، بیت بعدی توانایی نوشتن و پر ارزش‌ترین بیت توانایی خواندن را نشان می‌دهد. برای مثال، در صورتی که دسترسی یک فایل «۶۴۰» باشد یعنی صاحب فایل می‌تواند از فایل بخواند و بر روی آن بنویسد، اعضای گروه فایل فقط می‌توانند آن را بخوانند و سایر افراد اجازه‌ی هیچ یک از این عملیات را ندارند. پارامتر «ls» صاحب، گروه و دسترسی افراد را برای فایل‌ها نمایش می‌دهد.

```
$ ls -l
-rw-r--r-- 1 user users 143 Dec 4 14:10 Makefile
-rwxr-xr-x 1 user users 7609 Nov 9 23:57 ex8
-rw----- 1 user users 412 Nov 9 23:57 ex8.c
-rw-r--r-- 1 user users 2232 Nov 9 23:57 ex8.o
```

صاحب، گروه و دسترسی افراد را می‌توان با دستورات زیر تعیین نمود (راه‌های فاصله‌تری برای مشخص کردن اجازه‌ی دسترسی افراد برای دستور «chmod» وجود دارد؛ به صفحه‌ی راهنمای آن مراجعه شود).

```
$ chown user path # تغییر صاحب یک فایل
$ chgrp users path # تغییر گروه یک فایل
$ chmod 644 path # تعیین میزان دسترسی افراد به یک فایل
```

1 Multi-user

## تمرین نهم

برای انجام این تمرین فایل «ex9.c» را دریافت کنید. این برنامه یک ریسمان می‌سازد و در آن مقدار متغیر «found» را به روز می‌رساند. این برنامه را به صورتی تغییر دهید که پس از دریافت سیگنال «SIGUSR1»، آفرین مقدار متغیر «found» را چاپ کند و در صورت دریافت سیگنال «SIGINT»، پس از چاپ مقدار «found» فاصله یابد. دقت کنید که باید در این تمرین دسترسی‌های همزمان به متغیر «found» را با استفاده از قفل‌های معرفی شده در جلسه گذشته مدیریت نمایید.

گام‌های پیشنهادی برای انجام این تمرین:

- ۱ دریافت و ترجمه‌ی فایل «ex9.c»
- ۲ نوشتن تابعی برای دریافت سیگنال «SIGUSR1» و فراخوانی «signal()» برای ثبت آن
- ۳ آزمایش درستی دریافت سیگنال «SIGUSR1» با استفاده از دستور «kill» از پوسته
- ۴ تکرار مرحله‌ی دو برای سیگنال «SIGINT» و فروغ پس از دریافت آن
- ۵ آزمایش درستی دریافت سیگنال «SIGINT» با فشار دادن دکمه‌های کنترل و «C»
- ۶ مدیریت دسترسی‌های همزمان به متغیر «found» با «Mutex»