

جلسه پنجم — نوشتن و ترجمه برنامه‌ها

در این جلسه با امکاناتی که معمولاً در یونیکس برای نوشتن برنامه‌ها و ترجمه‌ی آنها موجود هستند، آشنا خواهید شد. محیط یونیکس ابزارهای زیادی را برای نوشتن، ترجمه و مدیریت کد برنامه‌ها در اختیار برنامه‌نویسان قرار می‌دهد. پس از نوشتن برنامه‌ها، می‌توان با استفاده از یکی از مترجم‌های موجود در توزیع‌های لینوکس برنامه‌ها را ترجمه نمود.

ترجمه برنامه‌ها در محیط یونیکس

۱ فایل test.c را با محتویات زیر ایجاد کنید.

```
#include <stdio.h>
int main(void)
{
    printf("Hello world!\n");
    return 0;
}
```

۲ برای ترجمه این فایل و اجرای آن از دستورهای زیر استفاده کنید. دستور «cc» یک مترجم (Compiler) زبان «C» است که فایلی که آدرس آن به عنوان پارامتر داده می‌شود را ترجمه می‌کند. برای ترجمه برنامه‌هایی که در زبان «C++» نوشته شده‌اند نیز می‌توان از دستور «C++» استفاده نمود. در بیشتر توزیع‌های لینوکس، معمولاً به صورت پیش‌فرض از مترجم «GCC» (GNU Compiler Collection) برای ترجمه برنامه‌ها استفاده می‌شود و معمولاً دستور «cc» معادل دستور «gcc» و «C++» معادل دستور «g++» می‌باشند.

```
$ cc test.c
```

ترجمه‌ی فایل test.c

```
$ ./a.out
```

اجرای فایل خروجی (a.out)

```
Hello world!
```

۳ کد برنامه‌های نسبتاً بزرگ به تعدادی فایل شکسته می‌شود. در صورتی که تعداد فایل‌های کد برنامه زیاد باشد (یا در صورتی که زبان‌های متفاوتی در آن استفاده شده باشند)، می‌توان تولید فایل اجرایی را در دو گام انجام داد. در گام اول فایل‌های «Object» تولید می‌شوند؛ این فایل‌ها خروجی گام ترجمه‌ی مترجم هستند. در گام دوم این فایل‌ها با هم ترکیب می‌شوند تا یک فایل اجرایی حاصل شود. به عملی که در گام اول انجام می‌شود، ترجمه و به عملی که در گام دوم انجام می‌شود، لینک کردن (Linking) گفته می‌شود. پیچونگی انجام این دو گام در ادامه نشان داده می‌شود (فایل‌های «src1.c»، «src2.c» و «src3.c» را قبل از اجرای دستورات زیر بسازید):

```
$ cc -c src1.c
```

ساختن یک فایل «Object» با نام «src1.o» برای فایل «src1.c»

```
$ cc -c src2.c
```

ساختن یک فایل «Object» با نام «src2.o» برای فایل «src2.c»

```
$ cc -c src3.c
```

ساختن یک فایل «Object» با نام «src3.o» برای فایل «src3.c»

```
$ ls
```

```
src1.c src1.o src2.c src2.o src3.c src3.o
```

```
$ cc -o out src1.o src2.o src3.o
```

انجام عمل «Link» برای ساختن فایل اجرایی

```
$ ./out
```

اجرای فایل اجرایی حاصل

۴ یکی از مزیت‌های تولید فایل اجرایی در این دو گام، در هنگام تغییر کد است: اگر فقط یکی از فایل‌ها تغییر کند، لازم نیست سایر فایل‌ها دوباره ترجمه شوند و ترجمه‌ی فایل تغییر یافته و لینک کردن فایل‌های «Object» کافی است.

۵ در صورتی که سرعت ترجمه اهمیت نداشته باشد، بسیاری از مترجم‌ها دو گام ترجمه و لینک را با یک دستور انجام می‌دهند:

```
cc -o out src1.c src2.c src3.c
```

تولید فایل اجرایی در یک مرحله

متغیرهای محیطی

۶ به هر پردازش در یونیکس، از جمله پوسته، تعدادی متغیر محیطی (Environment Variable) اختصاص می‌یابد. این متغیرهای محیطی پس از فراخوانی سیستمی «fork()» در پردازش فرزند باقی می‌مانند و از این رو برای انتقال داده‌های رشته‌ای کوتاه به پردازش‌ها استفاده می‌شوند. متغیرهای محیطی پوسته را می‌توان به صورت زیر تعریف کرد یا مقدار آنها را فوند (متغیرهای محیطی مشابه متغیرهای پوسته فوند می‌شوند).

```
$ export MYENV="my def"
```

تعریف متغیر محیطی MYENV با مقدار «my env»

```
$ echo $MYENV
```

چاپ مقدار متغیر محیطی MYENV

```
my def
```

```
$ env
```

چاپ همه‌ی متغیرهای محیطی پوسته و مقدارشان

```
...
```

```
MYENV=my env
```

```
...
```

۷ یکی از متغیرهای محیطی مهم در یونیکس، متغیر «PATH» می‌باشد. این متغیر، فهرستی از شافه‌هایی که حاوی فایل‌های اجرایی هستند و با علامت «:» جدا می‌شوند را در خود نگه می‌دارد. برای اجرای فایل‌هایی که در این شافه‌ها قرار دارند، مشخص کردن آدرس آنها لازم نیست (برای اجرای سایر فایل‌ها باید آدرس آنها مشخص شود).

| | |
|-------------------------------|--|
| \$ echo \$PATH | متغیر ممیطی «PATH» شامل شافه‌هایی مثل «/bin» |
| /sbin:/usr/sbin:/bin:/usr/bin | |
| \$ ls /bin/u* | شافه‌ی «/bin» شامل فایل اجرایی «uname» است |
| uname umount | |
| \$ uname | بنابراین «uname» معادل «/bin/uname» است |
| Linux | |

۸ با استفاده از تابع «getenv()» در کتابخانه‌ی استاندارد زبان «C» می‌توان مقدار یک متغیر ممیطی را در زبان «C» فوآند. این تابع که در فایل «stdlib.h» معرفی می‌شود، در صورتی که متغیر ممیطی داده شده تعریف نشده باشد، مقدار «NULL» را بر می‌گرداند؛ برای جزئیات بیشتر به صفحه‌ی راهنمای تابع «getenv()» مراجعه کنید.

پارامترهای ورودی برنامه

۹ همان طور که در جلسه‌های گذشته مشاهده کرده‌اید، دستوراتی که در پوسته اجرا می‌شوند تعدادی پارامتر می‌پذیرند. در صورتی که برنامه‌ی اجرا شونده در زبان «C» نوشته شده باشد، این پارامترها به تابع «main()» آن برنامه فرستاده می‌شوند. برای دسترسی به پارامترها، شکل تابع «main()» می‌تواند به صورت زیر باشد.

```
int main(int argc, char *argv[])
```

۱۰ در زمان اجرای برنامه، متغیر «argc» تعداد پارامترها و متغیر «argv» پارامترهای داده شده فوآهند بود. به صورت قراردادی پارامتر اول (argv[0]) همواره نام خود برنامه‌ی اجرا شونده است. بنابراین در صورتی که برنامه‌ای در پوسته به صورت زیر صدا زده شود، مقدار «argc» برابر ۳، مقدار argv[0] برابر «cmd»، مقدار argv[1] برابر «hello» و مقدار argv[2] برابر «world» فوآهد بود.

```
$ cmd hello world
```

برنامه‌ی آزمایشی

11 در شافهی «~/ex5» برنامه‌ای به نام «procinfo.c» بنویسید که پیغامی به صورت زیر چاپ کند. در این فروجی، عبارت پس از «user» مقدار متغیر ممیطی «USER»، عبارت بعد از «home» مقدار متغیر ممیطی «HOME»، مقدار پس از «pid» شماره‌ی پردازهی ایجاد شده (تابع «getpid()» را فراخوانی کنید)، مقدار پس از «uid» شماره‌ی کاربری (تابع «getuid()» را فراخوانی کنید)، مقدار پس از عبارت «path» شافهی جاری پردازه (تابع «getcwd()» را فراخوانی کنید)، عبارت پس از «prog» نام برنامه‌ای که نوشته‌اید هستند. در قسمت پایانی فروجی، پارامترهایی که به برنامه فرستاده شده‌اند باید نمایش داده شوند. در نمونه‌ی بالا فرض شده است برنامه به صورت «./ex5 hello world» فراخوانی شده است. برای اطلاعات بیشتر در مورد توابع مورد نیاز، به صفحه‌های راهنمای آنها مراجعه نمایید.

```
user      me
home      /home/me
pid       111
uid       1001
path      /home/me/ex5
prog      ./ex5
arguments:
          1      hello
          2      world
```

ساختن فودکار فایل‌های فروجی (اختیاری)

۱۲ برای جلوگیری از تکرار دستورات لازم برای تولید فایل اجرایی یک برنامه، می‌توان ساخت فایل اجرایی را از کد برنامه‌ها به صورت فودکار انجام داد. یک راه برای ترجمه‌ی فودکار تعداد زیادی فایل، نوشتن اسکریپتی می‌باشد که دستورات لازم برای تولید فایل اجرایی را نگه دارد. اما راه بهتر استفاده از ابزار make است که با گرفتن رابطه بین فایل‌ها و دستورهای لازم برای تولید فایل‌های فروجی، فایل‌ها را فقط در صورت نیاز ترجمه می‌کند.

۱۳ برنامه‌ی make با خواندن یک فایل ورودی (که معمولاً Makefile یا makefile نامیده می‌شود)، دستورهای لازم برای تولید یک فایل فروجی را یکی پس از دیگری اجرا می‌کند. فایل ورودی make به ازای فروجی‌های میانی و نهایی، پیش‌نیازها و دستورهای لازم برای تولید آنها را مشخص می‌کند. در مثال زیر، الگوی کلی این فایل نمایش داده شده است: برای ساختن فایل «target» فایل‌هایی که پس از آن مشخص می‌شوند («prereq1» و «prereq2») باید ساخته شوند و برای ساختن آن، دستورهایی که در فطهای بعد مشخص شده‌اند (دستورهای «command 1» و «command 2») اجرا می‌شوند.

```
target: [prereq1] [prereq2] ...
    [command 1]
    [command 2]
    ...
```

۱۴ با دستور «make target» در پوسته، فایل «target» تنها وقتی ساخته می‌شود که این فایل وجود نداشته باشد یا موجود باشد و حداقل یکی از پیش‌نیازهای آن جدیدتر از آن باشد. در مثال بفش قبل برای ساخت «out»، در صورتی که پس از دستورهای گفته شده فایل «src3.c» تغییر کند، ترجمه‌ی دوباره‌ی «src1.c» و «src2.c» لازم نیست ولی فایل «out» باید دوباره ساخته شود. محتویات یک Makefile نمونه برای ساختن این فایل در ادامه نشان داده می‌شود.

```

src1.o: src1.c          فایل «src1.o» به فایل «src1.c» امتیاج دارد
    cc -c src1.c        دستور لازم برای ساختن «src1.o»

src2.o: src2.c
    cc -c src2.c

src3.o: src3.c
    cc -c src3.c

out: src1.o src2.o src3.o  فایل «out» به فایل‌های «src[123].o» امتیاج دارد
    cc -o out src1.o src2.o src3.o

```

۱۵ سپس با دستور make می‌توان فایل‌های مشخص شده در Makefile را با استفاده از دستورات معرفی شده به صورت خودکار ساخت.

```

$ make out              ساختن فایل «out»
cc -c src1.c
cc -c src2.c
cc -c src3.c
cc -o out src1.o src2.o src3.o

```

۱۶ اگر یکی از فایل‌های پیش‌نیاز تغییر کند، ساختن دوباره‌ی همه‌ی فایل‌ها لازم نیست؛ در مثال زیر، فایل «src3.c» با استفاده از دستور touch (که زمان تغییر یک فایل را به روز می‌رساند) تغییر داده می‌شود.

```

$ touch src3.c          تغییر فایل «src3.c»
$ make out              فقط فایل‌های «src3.o» و «out» دوباره ساخته می‌شوند
cc -c src3.c
cc -o out src1.o src2.o src3.o

```

۱۷ در Makefile-ها می‌توان متغیر تعریف نمود و دستورات و پیش‌نیازهای فایل‌هایی که به صورت مشابه شناخته می‌شوند را به صورت فاصله‌تری بیان نمود. برای جزئیات بیشتر، به مستندات که با عمق بیشتری به ابزار make می‌پردازند مراجعه کنید.