جلسه‌ی پنجم — نوشتن و ترجمه‌ی برنامه‌ها

در این جلسه با امکاناتی که معمولا در یونیکس برای نوشتن برنامه‌ها و ترجمه‌ی آنها موجود هستند، آشنا خواهید شد.

ترجمه‌ی برنامه‌ها در محیط یونیکس

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.

```
$ cat >test.c
#include <stdio.h>
int main(void)
{
      printf("Hello world!\n");
      return 0;
}
^D
$ cc test.c                 #      ترجمه‌ی فایل «test.c»؛ نام فایل اجرایی حاصل «a.out» می‌باشد
$ ./a.out                   #                                  اجرای فایل «a.out»
Hello world!
$ cc -o test test.c         #                  مشخص کردن نام فایل خروجی با پارامتر «o-»
$ ./test
Hello world!
```

همان طور که مشاهده می‌شود، دستور «cc» یک مترجم[1] زبان «C» است که فایلی که آدرس آن به عنوان پارامتر به آن داده می‌شود را ترجمه می‌کند. برای ترجمه‌ی برنامه‌هایی که در زبان «C++» نوشته شده‌اند نیز می‌توان از دستور «c++» استفاده نمود. در بیشتر توزیع‌های لینوکس، معمولا به صورت پیش‌فرض از مترجم «GCC»[2] برای ترجمه‌ی برنامه‌ها استفاده می‌شود و معمولا دستور «cc» معادل دستور «gcc» و «c++» معادل دستور «g++» می‌باشد.

1 Compiler
2 GNU Compiler Collection

شکستن کد به تعدادی فایل

کد برنامه‌های نسبتا 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[1] گفته می‌شود. چگونگی انجام این دو گام در ادامه نشان داده می‌شود (فرض کنید فایل‌های «src1.c»، «src2.c» و «src3.c» شامل کد برنامه هستند):

```
$ 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»
$ ./out                           # اجرای فایل اجرایی حاصل
```

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 فقط ترجمه‌ی فایل تغییر یافته و لینک کردن فایل‌های «Object» کافی است. در صورتی که سرعت ترجمه اهمیت نداشته باشد، بسیاری از مترجم‌ها این دو گام را با یک دستور انجام می‌دهند:

```
$ cc -o out src1.c src2.c src3.c # تولید فایل اجرایی در یک مرحله، بدون تولید فایل‌های «Object»
```

متغیرهای محیطی

به هر پردازه در یونیکس، از جمله پوسته، تعدادی متغیر محیطی[2] اختصاص می‌یابد. این متغیرهای محیطی پس از فراخوانی سیستمی «()fork» در پردازه‌ی فرزند باقی می‌مانند و از این رو برای انتقال داده‌های رشته‌ای کوتاه به پردازه‌ها استفاده می‌شوند. متغیرهای محیطی پوسته را می‌توان به صورت زیر تعریف کرد یا مقدار آنها را خواند (متغیرهای محیطی مشابه متغیرهای پوسته خوانده می‌شوند).

1 Linking
2 Environment Variable

```
$ export MYENV="my env"          #          تعریف متغیر محیطی «MYENV» با مقدار «my env»
$ echo $MYENV                    #          چاپ مقدار متغیر محیطی «MYENV»
my env
$ env                            #          چاپ همه‌ی متغیرهای محیطی پوسته و مقدارشان
...
MYENV=my env
...
```

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

```
$ echo $PATH                     #          متغیر محیطی «PATH» شامل شاخه‌هایی مثل «/bin»
/sbin:/usr/sbin:/bin:/usr/bin
$ ls /bin                        #          شاخه‌ی «/bin» شامل فایل اجرایی «uname» است
...
uname
...
$ uname                          #          بنابراین «uname» معادل «/bin/uname» است
Linux
```

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

پارامترهای ورودی برنامه

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[])
```

در زمان اجرای برنامه، متغیر «argc» تعداد پارامترها و متغیر «argv» پارامترهای داده شده خواهند 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:

```
$ cmd hello world
```

مقدار «argc» برابر ۳، مقدار argv[0] برابر «cmd»، مقدار argv[1] برابر «hello» و مقدار argv[2] برابر «world» خواهد بود.

ساختن خودکار فایل‌های خروجی (اختیاری)

برای جلوگیری از تکرار دستورات لازم برای تولید فایل اجرایی یک برنامه، می‌توان ساخت فایل اجرایی را از کد برنامه‌ها به صورت خودکار انجام داد. یک راه برای ترجمه‌ی خودکار 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 خروجی، فایل‌ها را فقط در صورت نیاز ترجمه می‌کند.

برنامه‌ی «make» با خواندن یک فایل ورودی (که معمولا «Makefile» یا «makefile» نامیده می‌شود)، دستورهای لازم برای تولید یک فایل خروجی را یکی پس از دیگری اجرا می‌کند. فایل ورودی «make» به ازای خروجی‌های 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 خطهای بعد مشخص شده‌اند (دستورهای «command 1» و «command 2») اجرا می‌شوند.

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

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 بخش 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. یک «Makefile» نمونه برای ساختن این فایل در ادامه نشان داده می‌شود:

```
$ cat Makefile
src1.o: src1.c                  #       فایل «src1.o» به فایل «src1.c» احتیاج 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» احتیاج دارد
      cc -o out src1.o src2.o src3.o
```

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
```

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
```

در «Makefile»-ها می‌توان متغیر تعریف نمود و دستورات و پیش‌نیازهای فایل‌هایی که به صورت مشابه شاخته می‌شوند را به صورت خلاصه‌تری بیان نمود. برای جزئیات بیشتر، به مستنداتی که با عمق بیشتری به ابزار «make» می‌پردازند مراجعه کنید.

تمرین پنجم

در شاخه‌ی «~/ex5» برنامه‌ای به نام «procinfo.c» بنویسید که پیغامی به صورت زیر چاپ کند:

```
user    me
home    /home/me
pid     111
uid     1001
path    /home/me/ex5
prog    ./ex6
arguments:
        1       hello
        2       world
```

در این خروجی، عبارت پس از «user» مقدار متغیر محیطی «USER»، عبارت بعد از «home» مقدار متغیر محیطی «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» شاخه‌ی جاری پردازه (تابع «getcwd()» را فراخوانی کنید)، عبارت پس از «prog» نام برنامه‌ای که نوشته‌اید هستند. در قسمت پایانی خروجی پارامترهایی که به برنامه فرستاده شده‌اند باید نمایش داده شوند. در نمونه‌ی بالا فرض شده است برنامه به صورت «./ex6 hello world» فراخوانی شده است. برای اطلاعات بیشتر در مورد توابع مورد نیاز، به صفحه‌های راهنمای آنها مراجعه نمایید.