شروع برنامه‌نویسی در گنو/ لینوکس

سیاوش صفی
siavash@tehlug.org

در این مقاله با مراحل ساخت یک برنامه در لینوکس با استفاده از C++/C آشنا خواهید شد.

- ایجاد و ویرایش کدها منبع با استفاده از ویرایشگرها
- کامپایل کد با GCC
- انجام پروسه به صورت خودکار با استفاده از GNU Make
- اشکال‌زدایی با استفاده از GNU Debugger یا GDB
- پیدا کردن اطلاعات بیشتر

ایجاد و ویرایش کدهای منبع با استفاده از ویراشگرها

ویرایشگرهای زیادی وجود دارند که می‌توان از آن‌ها برای ویرایش کد منبع استفاده کرد. برای نمونه ویرایشگر متنی Vim و ویراشگر گرافیکی Gedit.

برای ایجاد یا ویرایش یک فایل کد منبع کافی است تا به این صورت عمل کنید:

```
$ vim source.c
```

اگر می‌خواهید کد منبع C ایجاد کنید باید از نامی استفاده کنید که به .c یا .h ختم می‌شود.
برای C++ از نامی که به

.cpp, .hpp, .cxx, .hxx, .C, .H

ختم می‌شود استفاده کنید.
پس از اینکه فایل توسط ویرایشگر باز شد با زدن Insert می‌توانید مانند یک واژه‌پرداز معمولی در آن تایپ کنید و در پایان Esc را زده و با وارد کردن :wq فایل را ذخیره کرده و از Vim خارج شود.

```c
#include <stdio.h>

int main()
{
        printf("Hello Tehlug!\n");

        return 0;
}
```

```
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --                                                    9,1           All
```

کامپایل کد با GCC

یک کامپایلر کد منبعی را که برای انسان قابل خواندن است به کد شیئی که برای ماشین قابل خواندن است تبدیل می‌کند که قابل اجرا می‌باشد. بهترین کامپایلر که در گنو/لینوکس از آن استفاده می‌شود مجموعه کامپایلر گنو (GNU Compiler Collection) که معمولا با نام GCC شناخته می‌شود. GCC شامل کامپایلرهایی برای C,C++,Obective-C,Fortran, Chill است. فرض کنید که پروژه‌ای دارید که شامل یک فایل منبع C++ با نام reciprocal.cpp و یک فایل منبع C به نام main.c است. این دو فایل کامپایل شده و سپس به هم متصل (link) می‌شوند تا برنامه‌ای به نام reciprocal را بسازند.

main.c C source file

```
#include <stdlib.h>
#include <stdio.h>
#include “reciprocal.hpp”

int main (int argc, char **argv)
{
    int i;

    I = atoi (argv[1]);
    printf (“The reciprocal of %d is %g\n”, i, reciprocal (i));
    return 0;
}
```

reciprocal.cpp C++ Source file

```
#include <cassert>
#include “reciprocal.hpp”

double reciprocal (int i)
{
   // I should be non-zero.
   assert (i != 0);
    return 1.0/i;
}
```

reciprocal.hpp Header file

```
#ifdef __cplusplus
extern “C” {
#endif

extern double reciprocal (int i);

ifdef __cplusplus
}
#endif
```

اولین مرحله تبدیل کد منبع C و C++ به کد شیئ است.

کامپایل یک فایل منبع

gcc کامپایلر C است. برای کامپایل یک فایل منبع C از c- استفاده می‌کنیم. به عنوان مثال اجرای این دستور main.c را کامپایل می‌کند:

```
$ gcc -c main.c
```

شیئ حاصل main.o نام دارد.
g++ کامپایلر C++ است که عملکردش خیلی شبیه به gcc است، کامپایل reciprocal.cpp با وارد کردن این دستور انجام می‌شود:

```
$ g++ -c reciprocal.cpp
```

c- به g++ می‌گوید که برنامه را تنها به یک فایل شیئ کامپایل کند. بدون آن، g++ سعی خواهد کرد تا برنامه را لینک کند تا یک فایل اجرایی بسازد. بعد از وارد کردن دستور، شما یک فایل شیئ به نام reciprocal.o خواهید داشت.
برای اطلاع از سایر امکانات gcc و g++ از info استفده کنید:

```
$ info gcc
```

متصل کردن فایل‌های شیئ

حالا که main.c و reciprocal.cpp را کامپایل کرده‌اید، باید آن‌ها را متصل (link) کنید. شما باید همیشه از g++ برای متصل‌کردن برنامه‌ای که شامل کد C++ است استفاده کنید، حتی اگر شامل کد C نیز باشد. اگر برنامه شما فقط شامل کد C است شما باید از gcc استفاده کنید. چون این برنامه شامل هر دو کد است، باید از g++ استفاده کنیم، به این صورت:

```
$ g++ -o reciprocal main.o reciprocal.o
```

o- نام فایلی را که از خروجی دستور ساخته می‌شود تعیین می‌کند.
حالا می‌توانید reciptocal را به این صورت اجرا کنید:

```
$ ./reciprocal 7
The reciprocal of 7 is 0.142857
```

همانطور که می‌بینید، g++ به صورت خودکار به کتابخانه استاندارد C که شامل پیاده‌سازی printf است متصل کرده است. اگر نیاز داشتید که به یک کتابخانه دیگر (مثل کتابخانه‌های گرافیکی) باید کتابخانه را با l- مشخص می‌کردید.

انجام خودکار پروسه با استفاده از GNU Make

ایده‌ای که در make استفاده شده خیلی ساده است. شما به make می‌گویید که قصد دارید چه اهدافی را بسازید و سپس دستوراتی که نحوه ساخت آن‌ها را توضیح می‌دهند را تعیین می‌کنید. همچنین شما وابستگی‌هایی را که نشان می‌دهند که یک هدف خاص باید دوباره ساخته شود را تعیین می‌کنید.

در مثال ما یعنی پروژه reciprocal، سه هدف مشخص وجود دارند: reciprocal.o و main.o و خود reciprocal. شما دستوراتی که برای ساختن اهداف لازم هستند را می‌دانید. وابستگی‌ها نیاز به کمی تفکر دارند. مشخص است که reciprocal وابسته به reciprocal.o و main.o است چون شما نمی‌توانید برنامه کامل را لینک کنید تا زمانی که این دو شیئ ساخته شده باشند. شیئ‌ها باید زمانی که فایل‌های منبع تغییر کردند دوباره ساخته شوند. همچنین تغییر در reciprocal.hpp باید باعث دوباره ساخته شدن هر دو شیئ شود چون هر دو شامل آن هستند.

علاوه بر این اهداف مشخص، باید یک هدف clean نیز وجود داشته باشد. این هدف تمام شیئ‌ها و خود برنامه را حذف می‌کند. دستور این هدف از rm برای حذف فایل‌ها استفاده می‌کند.

شما این اطلاعات را با قراردادن آن در فایلی به نام Makefile به make منتقل می‌کنید:

```
reciprocal: main.o reciprocal.o
   g++ $(CFLAGS) -o reciprocal main.o reciprocal.o

main.o: main.c reciprocal.hpp
   gcc $(CFLAGS) -c main.c

reciprocal.o: reciprocal.c reciprocal.hpp
   g++ $(CFLAGS) -c reciprocal.c

clean:
   rm -f *.o reciprocal
```

اگر شیئ‌هایی را که ساخته‌اید حذف کرده و تایپ کنید:

```
$ make
```

این خروجی را خواهید دید:

```
$ make
gcc -c main.c
g++ -c reciprocal.cpp
g++ -o reciprocal main.o reciprocal.o
```

اشکال‌یابی با استفاده از GDB

یک debugger برنامه‌ای است که شما از آن استفاده می‌کنید تا پی ببرید که چرا برنامه شما آنطور که شما فکر می‌کنید کار نمی کند.
برای استفاده از gdb باید برنامه را با g- کامپایل کنید:

```
$ make CFLAGS=-g
gcc -g -c main.c
g++ -g -c reciprocal.cpp
g++ -g -o reciprocal main.o reciprocal.o
```

اجرای gdb:

```
$ gdb reciprocal
(gdb)
```

اجرای برنامه در gdb:

```
(gdb) run
Starting program: reciprocal

Program received singnal SIGSEGV, Segmentation fault.
__strtol_internal (nptr=0x0, base=10, group=0)
at strtp.c:287
287         strtol.c: No such file or directory.
(gdb)
```

مشکل اینجاست که هیچ کدی برای کنترل اشکال در برنامه وجود ندارد. برنامه به یک آرگومان نیاز دارد ولی در این مورد بدون آرگومان اجرا شده است. پیغام SIGSEGV نشان می‌دهد که برنامه متوقف شده است. GDB می‌داند که توقف دوقتی که تابع __strtol_internal فراخوانده شده است رخ داده. تابع در کتابخانه استاندارد است و منبع آن نصب نشده که پیغام "No such file or directory" به همین دلیل است. شما می‌توانید پشته را با دستور where ببینید:

```
(gdb) where
#0  __strtol_internal (nptr=0x0, endptr=0x0, base=10, group=0)
    at strtol.c:287
#1  0x40096fb6 in atoi (nptr=0x0) at ../stdlib/stdlib.h:251
#2  0x804836e in main (argc=1, argv=0xbffff5e4) at main.c:8
```

می‌توانید ببینید که main تابع atoi را با یک اشاره‌گر NULL فراخوانی کرده که باعث ایجاد مشکل شده است.
شما می‌توانید ۲ سطح در پشته بالا بروید تا به main برسید:

```
(gdb) up 2
#2   0x804863e in main (argc=1, argv=0xbffff5e4) at main.c:8
8           i = atoi (argv[1]);
```

شما می‌توانید مقدار متغیر را ببینید:

```
(gdb) print argv[1]
$2 = 0x0
```

یافتن اطلاعات بیشتر

Man pages:

(1) User commands
(2) System calls
(3) Standard library functions
(8) System/administrative commands

```
$ man sleep
```

که مربوط به دستور sleep می‌شود که در بخش ۱ قرار دارد.

```
$ man 3 sleep
```

که مربوط به تابع کتابخانه‌ای sleep می‌شود.

Info:

```
$ info gcc
```

که شامل اطلاعات جامعی است:

- gcc – The gcc compiler
- libc – The GNU C library, including many system calls
- gdb – The GNU debugger
- info – The info system itself

Header Files:

شما می‌توانید اطلاعاتی در مورد توابع سیستمی و نحوه استفاده از آنها از این فایل‌ها استفاده کنید.

/usr/include
/usr/include/sys

Source Code:

با خواندن کد منبع سیستم می‌توانید پی ببرید که سیستم چطور کار می‌کند.

منبع: Advanced Linux Programming فصل اول