

Menambahkan Kemampuan Plugin ke dalam Kode Anda

By [Tom Bradley](#)

0. Perkenalan

Hari dimana sebuah program hidup sebagai satu kesatuan telah berakhir. Program-program pada masa sekarang ini harus lebih serbaguna dan dapat diperluas/dikembangkan lebih lanjut. Cara yang paling sederhana untuk menambahkan fleksibilitas dan kemampuan untuk dapat dikembangkan lebih lanjut adalah dengan menggunakan module atau biasa disebut sebagai plugin. Web browser dan player musik adalah dua contoh program yang bagus yang mengijinkan penggunaan plugin. Browser mempergunakan plugin untuk menambahkan dukungan terhadap halaman web seperti misalnya Java, Flash dan QuickTime sehingga anda akan mendapatkan pengalaman surfing yang lebih kaya. Player musik seperti XMMS menggunakan plugin untuk mendukung encoding yang berbeda, sebagaimana visual plugin yang bisa menampilkan 'tarian' musik pada layar. Artikel ini akan menunjukkan bagaimana cara untuk memberikan dukungan terhadap plugin pada program anda. Catatan: saya terkadang mempergunakan istilah module atau plugin, yang pada artikel ini mereka kita anggap sama.

1. Bagaimana Kita Bekerja Dengan Plugin

Hanya ada empat fungsi yang dibutuhkan untuk dapat bekerja dengan plugin. Keempat fungsi tersebut adalah bagian dari dl (Dynamic Loader) library. Saya hanya akan memberikan pengenalan secara singkat mengenai mereka. Anda dapat membaca sendiri halaman info dari tiap-tiap fungsi tersebut untuk mendapatkan keterangan lebih lanjut.

dlopen

Fungsi ini digunakan untuk memuat module ke dalam memory.

dlclose

Fungsi ini digunakan untuk membuang module dari memory.

dlsym

Fungsi ini digunakan untuk melihat dan memberikan alamat dari suatu fungsi yang ada di dalam module.

dlerror

Fungsi ini akan memberikan pesan kesalahan kepada anda.

2. Sebuah Program Loader Sederhana untuk Plugin

Ini adalah kode untuk program loader sederhana yang mengambil nama plugin sebagai sebuah argumen pada command line.

main.c

```
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <dlfcn.h>

#define PATH_LENGTH 256

int main(int argc, char * argv[])
{
    char path[PATH_LENGTH], * msg = NULL;
    int (*my_entry)();
    void * module;

    /* build the pathname for the module */
```

```
getcwd(path, PATH_LENGTH);
strcat(path, "/");
strcat(path, argv[1]);

/* load the module, and resolve symbols
now */
module = dlopen(path, RTLD_NOW);
if(!module) {
    msg = dlerror();
    if(msg != NULL) {
        dlclose(module);
        exit(1);
    }
}

/* retrieve address of entry point */
my_entry = dlsym(module, "entry");
msg = dlerror();
if(msg != NULL) {
    perror(msg);
    dlclose(module);
    exit(1);
}

/* call module entry point */
my_entry();

/* close module */
if(dlclose(module)) {
    perror("error");
    exit(1);
}

return 0;
}
```

Kode ini sangat sederhana. Setelah loader tersebut memuat plugin, ia akan melihat ke dalam tabel simbol plugin dengan menggunakan perintah `dlsym` untuk mendapatkan alamat dari fungsi 'entry.' Sekali saya mendapatkan alamat dari fungsi ini, saya dapat memanggil fungsi ini, dan menempatkannya ke dalam function pointer yang telah saya buat. Kemudian plugin tersebut di-unload. Baris function pointer di atas mungkin membutuhkan sedikit penjelasan.

`int (*my_entry)()`

digunakan sebagai sebuah pointer kepada fungsi yang tidak mengambil argumen apapun dan memberikan sebuah `int`. Yang mana bisa saya gunakan untuk menunjuk fungsi 'entry' di dalam plugin.

`int entry()`

Perintah di bawah ini digunakan untuk meng-compile program loader:

```
$ gcc -o loader main.c -ldl
```

3. Dua Plugin yang Sederhana

Setelah kita memiliki sebuah loader, kita membutuhkan beberapa plugin yang akan dimuat. Tidak ada prototipe yang disepakati untuk entry point pada module; anda bisa menggunakan apapun yang anda sukai. Pada contoh di atas, entry point akan memberikan sebuah `int` dan tidak mengambil argumen apapun. Anda bisa mengeset entry point anda untuk mengambil argumen apapun yang mereka butuhkan dan mengembalikan apapun yang anda mau. Ia tidak harus disebut

sebagai 'entry'. Saya menggunakannya hanya agar tujuan dari fungsi tersebut lebih mudah dimengerti. Sebagai tambahan, anda bisa memiliki lebih dari satu entry point dalam satu plugin. Di bawah ini adalah dua contoh dari sebuah module, semuanya dengan entry point yang sama: module1.c

```
int entry()  
{  
    printf("I am module one!\n");  
    return 0;  
}
```

module2.c

```
int entry()  
{  
    printf("I am module two!\n");  
    return 0;  
}
```

Untuk mengcompile plugin tersebut :

```
$ gcc -fPIC -c module1.c  
$ gcc -shared -o module1.so module1.o  
$ gcc -fPIC -c module2.c  
$ gcc -shared -o module2.so module2.o
```

Beberapa hal hanyalah menjelaskan bagaimana mereka dicompile. Pertama, flag '-fPIC'. PIC adalah kepanjangan dari "Position Independent Code", yang memberitahu compiler bahwa kode ini harus diset untuk menggunakan ruang alamat 'relatif'. Artinya, kode tersebut dapat diletakkan dimanapun di memory dan loader akan bertanggung jawab dalam menetapkan alamat pada saat dimuat. Flag '-shared' memberitahu compiler bahwa kode ini harus dicompile dengan suatu cara sehingga bisa di-link dengan executable yang lain. Dengan kata lain, .so (shared object) akan bertindak seperti halnya sebuah library; namun, .so anda bukanlah sebuah library dan tidak bisa di-link dengan menggunakan '-l' dengan gcc.

4. Menggunakan Loader

Ini adalah perintah untuk menggunakan dua plugin yang berbeda dan output yang dihasilkannya:

```
$ ./loader module1.so  
I am module one!  
  
$ ./loader module2.so  
I am module two!
```

5. Menambahkan Fungsi Bookkeeping untuk Plugin

Bagian ini mengasumsikan anda menggunakan compiler gcc karena perintah-perintah yang digunakan adalah spesifik untuk gcc. Compiler yang lain mungkin memiliki fitur yang hampir sama, silakan anda cek dokumentasinya untuk mengetahui kompatibilitasnya. Gcc menyediakan sebuah flag '__attribute__' untuk digunakan dengan fungsi. Flag ini memberikan banyak fitur yang berguna untuk fungsi; namun disini, saya hanya akan mendiskusikan dua di antaranya. Lihatlah halaman info dari gcc untuk keterangan lebih lanjut dari atribut yang lain. Kedua fitur yang ingin saya diskusikan di sini adalah 'constructor' dan 'destructor'. Binary ELF (Executable and Linkable Format) menyediakan dua bagian .init dan .fini yang berisi kode yang dieksekusi sebelum dan setelah sebuah module dimuat (dalam sebuah program yang umum, mereka akan dijalankan sebelum dan setelah main() dieksekusi). Dengan meletakkan kode pada bagian ini, anda akan bisa menginisialisasi variabel-variabel atau mengerjakan tanggung jawab bookkeeping

yang mungkin akan dibutuhkan oleh module yang anda buat. Sebagai contoh, anda bisa memiliki variabel 'module read' dari program utama yang akan dibutuhkan untuk bisa berjalan, atau memiliki variabel 'plugin set' di dalam program utama, seperti misalnya tipe interface dari plugin. Tipe interface dari plugin adalah satu set perintah yang disediakan oleh plugin. Dalam contoh, ia hanya menyediakan satu fungsi 'entry'; program anda mungkin akan menyediakan fungsi-fungsi yang lain. Di bawah ini adalah contoh penggunaan atribut-atribut tersebut:

```
__attribute__((constructor)) void init()
{
    /* code here is executed after dlopen() has
    loaded the module */
}

__attribute__((destructor)) void fini()
{
    /* code here is executed just before
    dlclose() unloads the module */
}
```

Nama init() dan fini() tidak begitu penting, saya menggunakannya untuk menjelaskan dimana fungsi ini harus diletakkan agar memudahkan pembacaan. Ada beberapa nama fungsi yang harus anda hindari karena gcc menggunakan nama tersebut. Beberapa diantaranya adalah _init, _fini, _start dan _end. Untuk melihat listing yang lengkap dari fungsi dan variabel yang dibuat oleh gcc anda bisa menjalankan 'nm' pada file binary. Atribut 'constructor' dan 'destructor' memberitahu compiler dimana harus meletakkan kode di dalam file binary. Secara sederhana, 'constructor' memberitahu compiler bahwa fungsi yang berhubungan berada pada bagian .init, dan demikian juga atribut 'destructor' memberitahu compiler letak dari file yang berhubungan pada bagian .fini.

6. Kesimpulan

Dengan menggunakan library dl, akan menjadi lebih mudah untuk menyediakan dukungan plugin pada program yang anda buat, dan akan memudahkan fleksibilitas dan kemampuan untuk dikembangkan lebih lanjut. Walaupun contoh ini hanya mendemonstrasikan cara mengambil satu fungsi dari sebuah plugin, adalah hal yang mudah untuk mengambil banyak fungsi dari sebuah plugin dan menggunakannya sebagaimana mereka menjadi bagian dari program yang sesungguhnya.

Copyright © 2002, Tom Bradley. Copying license <http://www.linuxgazette.com/copying.html>
Published in Issue 84 of *Linux Gazette*, November 2002

Diterjemahkan oleh [Triyan W. Nugroho](#)