링킹 (LD LINKER)

안양대학교 컴퓨터공학과 하은용

정적 라이브러리와 공유 라이브러리

내용

- 정적 라이브러리와 공유라이브러리
- 정적 라이브러리 만들기
- 공유 라이브러리 만들기

라이브러리

- 라이브러리(library)란
  - 미리 컴파일된 목적파일 들의 집합
  - 자주 사용되는 함수들을 미리 목적파일로 만들어 놓아 재사용 및 용이한 링킹을 가능하게 함
- 라이브러리의 종류
  - 정적(static) 라이브러리
    - 컴파일 시점에 링킹에 의해 목적 코드가 실행파일에 추가됨
  - 공유(shared) 라이브러리
    - 실행 시점에 동적 링킹에 의해 요구된 라이브러리가 적재

정적(static) 라이브러리

- 정적 라이브러리란
  - 컴파일 되어있는 목적파일 들을 하나로 묶은 형태의 파일
- 정적 라이브러리 파일명
  - lib{라이브러리명}.a
  - 예)
    - libc.a: C 라이브러리 'c', libm.a: 수학 라이브러리 'm', libfl.a: flex 라이브러리 'fl', ...
- 정적 라이브러리 생성과 연관된 유틸리티
  - ar: 파일 묶기 도구
  - ranlib: 색인정보 생성 유틸리티

ar

ar 유틸리티

- 아카이브를 만들거나 변경하고, 아카이브로부터 파일을 추출하는 도구
- 아카이브(archive)란
  - 여러 개의 파일을 묶어 가지고 있는 하나의 파일
  - 이 파일은 파일 외에 원래의 개별 파일을 찾을 수 있도록 하는 구조정보(색인정보)를 함께 가지고 있음
- 라이브러리 파일을 생성하고 유지하는데 주로 사용됨

ar 명령어 간단 형식

- ar command[mod] archive [member ...]

ar commands (1)

- 아카이브(라이브러리) 파일 생성하기
  - r: 새로운 파일을 아카이브에 삽입, 이미 존재한다면 교체
- 생성과 연관한 수식
  - u: 파일이 이미 존재하고 더 최신 파일일 경우에만 교체
  - s: 아카이브 내의 심볼에 대한 색인(index)정보 추가 및 갱신
    - 색인정보는 라이브러리 링킹 속도를 빠르게 함, 아카이브 내에서의 모듈 위치와 관계없이 서로 호출 가능하게 함
    - 만약 ‘s’가 주어지지 않았을 경우 별도의 명령을 통하여 추가 가능: $ ranlib 아카이브파일명
    - 색인정보가 일단 추가되면 이후 아카이브가 변경될 때마다 갱신
  - o: 추가될 파일의 원래 날짜 유지, 다른 정보는 기본적으로 유지
  - v: verbose로 동작 과정이 표준 출력

ar commands (2)

- 목록 보기
  - t: 아카이브 파일에 추가된 파일의 목록 보기
- 삭제
  - d: 아카이브 파일로부터 지정된 파일 삭제하기
- 추출하기
  - x: 아카이브 파일에 포함된 파일 추출
- 내용보기
  - p: 아카이브 파일에 포함된 파일의 내용 보기, 표준 출력되므로 아카이브에 포함된 파일이 이진 파일일 경우 무의미

예제: 정적 라이브러리 생성(1)

file1.c

```
#include <stdio.h>
void func1()
{
    printf("hello func1\n");
}
void func2()
{
    printf("hello func2\n");
}
```

file2.c

```
#include <stdio.h>
void func3()
{
    printf("hello func3\n");
}
```

my.h

```
void func1();
void func2();
void func3();
```

```
$ ls
file1.c file2.c my.h
$ gcc -c file1.c file2.c
$ ar rsv libmy.a file1.o file2.o
ar: creating libmy.a
a - file1.o
a - file2.o
$ ar t libmy.a
file1.o
file2.o
$ ar d libmy.a file1.o
$ ar t libmy.a
file2.o
$ ar rv libmy.a file2.o
r - file2.o
$ ar t libmy.a
file2.o
$ ar rv libmy.a file1.o
a - file1.o
$ ar t libmy.a
file2.o
file1.o
```

예제: 정적 라이브러리 이용 (2)

정적 라이브러리를 이용한 실제 컴파일

main.c

```
#include "my.h"
int main()
{
    func1();
    func2();
    func3();
    return 0;
}
```

```
$ ls
libmy.a main.c my.h
$ gcc -o main main.c -L. -lmy
$ ls -F
libmy.a main* main.c my.h
$ main
hello func1
hello func2
hello func3
$ ar xv libmy.a
x - file2.o
x - file1.o
$ ls -F
file1.o file2.o libmy.a main* main.c my.h
```

공유 라이브러리

- 공유 라이브러리
  - 프로그램이 실행될 때 적재되는 라이브러리
  - 한 번 적재된 공유라이브러리는 이를 필요로 하는 또 다른 프로그램이 실행될 때 다시 적재하지 않고 공유하여 같이 사용함
  - 단순한 목적 파일의 모음이 아니라 공유라이브러리가 사용하는 또 다른 라이브러리들이 링크되어있는 파일
  - 컴파일시 특별한 옵션을 사용하여 링킹까지 완료한 바이너리 파일
- 정적 라이브러리와 비교
  - 정적 라이브러리는 실행 속도가 빠르며 바이너리 배포시 제한이 없는 반면 바이너리의 크기가 큼
  - 공유라이브러리는 크기가 작은 대신 정적 라이브러리에 비해 실행 속도가 느림
  - 바이너리의 배포시 컴파일 시에 사용한 공유 라이브러리와 **같은 메이저 버전**의 공유 라이브러리가 있는 시스템에서만 동작

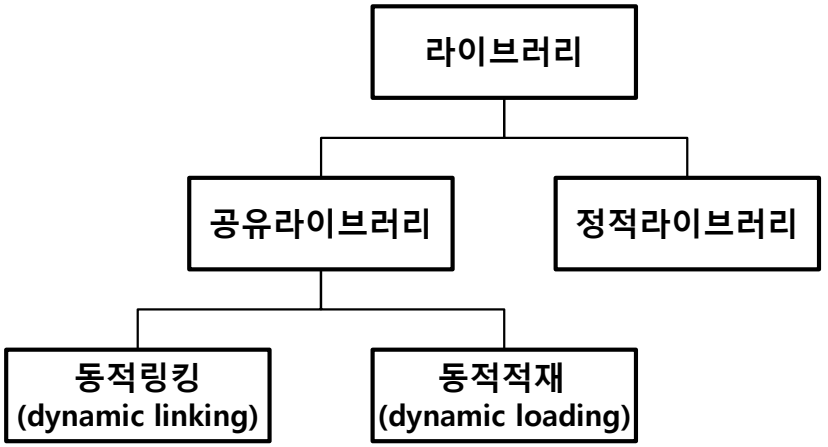
라이브러리의 중요성과 종류

공유 라이브러리의 중요성

- 사용하는 정적 라이브러리에 중대한 결함이 있어 교체해야 할 경우 정적 라이브러리를 사용하는 프로그램은 모두 다시 컴파일
- 공유 라이브러리를 사용하는 프로그램에서는 라이브러리만 교체

공유 라이브러리를 다루는 두 가지 방식

- 동적 링킹: 프로그램 실행과 함께 적재
- 동적 적재: 프로그램에서 함수를 사용하여 필요한 시점에 적재
  - dlopen(), dlerror(), dlsym(), dlclose()



공유라이브러리 생성/사용

기초적인 방법

- 공유 라이브러리 생성을 위한 링킹시 '-shared' 옵션을 사용

file1.c

```
#include <stdio.h>
void hello_linux()
{printf("Hello Linux");}
```

file2.c

```
#include <stdio.h>
void hello_fedora()
{printf("Hello Fedora");}
```

libmy.h

```
void hello_linux();
void hello_fedora();
```

main.c

```
#include "libmy.h"
int main()
{
    hello_linux();
    hello_fedora();
    return 0;
}
```

```
$ gcc -fPIC -c file1.c file2.c
$ gcc -shared -o libmy.so file1.o file2.o
$ ls
file1.c file1.o file2.c file2.o libmy.h
libmy.so main.c
$ gcc -o main main.c -L. -lmy
$ main
main: error while loading shared libraries:
libmy.so: cannot open shared object file: No
such file or directory
$
```

오류 발생 !!

참고: -fPIC는 "Position Independent Code" 생성을 활성화하는 옵션으로 공유라이브러리 검색시 속도를 빠르게 해줌

오류 원인 파악

공유 라이브러리의 의존성 확인 명령어: ldd

- $ ldd 파일명
  - 파일명은 실행권한 설정된 파일명: 실행파일, 공유라이브러리의 파일명

```
$ ldd main
        linux-gate.so.1 =>  (0x008c2000)
        libmy.so => not found
        libc.so.6 => /lib/libc.so.6 (0x0035d000)
        /lib/ld-linux.so.2 (0x0033b000)
$ ldd libmy.so
        linux-gate.so.1 =>  (0x0046d000)
        libc.so.6 => /lib/libc.so.6 (0x0075a000)
        /lib/ld-linux.so.2 (0x0033b000)
$
```

linux-gate.so.1: 시스템콜을 다루기 위한 virtual dynamic shared objects
libc.so.6: c공유라이브러리
/lib/ld-linux.so.2: 적재기

적재과정

- 적재기(dynamic linker or loader)
  - 적재기는 프로그램에 의해 사용되는 공유 라이브러리를 찾아 적재
  - 적재기는 공유 라이브러리를 사용하는 프로그램이 실행됨으로써 실행
  - /lib/ld-linux.so.N (N은 버전번호)
- 적재기가 찾는 기본 디렉터리
  - /lib, /usr/lib, /etc/ld.so.conf 파일에 명시된 디렉터리
    - 페도라는 /etc/ld.so.conf 파일에 디렉터리를 명시하는 대신 /etc/ld.so.conf에서 "include ld.so.conf.d/*.conf"
- 디렉터리 캐쉬 파일
  - /etc/ld.so.cache
  - 캐쉬 파일의 갱신을 위한 명령어: /sbin/ldconfig
    - 기본 경로로부터 공유 라이브러리를 찾아 검색을 빠르게 하는 구조정보와 함께 캐쉬에 저장

검색 경로의 사용자 추가 (1)

- ldconfig 의한 경로 추가의 문제점
  - 설정 파일(/etc/ld.so.conf)과 캐쉬파일(/etc/ld.so.cache)의 소유자는 root이며, 접근권한은 rw-r--r--이므로 일반 사용자는 변경 불가
- 일반 사용자의 경로 추가
  - ld-linux.so의 환경 변수 LD_LIBRARY_PATH에 추가하되, 전역적으로 지정
  - 예) export LD_LIBRARY_PATH=/home/mylinux/ex1

```
$ export LD_LIBRARY_PATH=/home/mylinux/ex1
$ main
Hello Linux
Hello Fedora
$
```

검색 경로의 사용자 추가 (2)

또 다른 방법

- 컴파일시 –rpath 옵션(링커 옵션) 사용

```
$ gcc -o main main.c -rpath=/home/mylinux/ex1 -L. -lmy
gcc: unrecognized option '-
rpath,/home/mylinux/class/linux/ch3/ex2-1'
$
```

- 이 옵션은 구동기인 gcc에 의해 처리되지 않는 옵션
- 따라서 링커에 직접 전달될 필요 있음, 직접 전달 방법은 링커를 별도로 수행하던가 다음과 같이 –Wl 옵션을 사용하여 전달

```
$ gcc -o main main.c -Wl,-rpath,/home/mylinux/ex1 -L. -lmy
$ main
Hello Linux
Hello Fedora
```

공유라이브러리 이름 규약 (1)

- 공유 라이브러리 장점 중 하나
  - 공유 라이브러리가 업그레이드 되어 새로운 버전의 공유 라이브러리가 설치되더라도
  - 구버전을 사용했던 모든 프로그램이 새로운 버전을 사용해야 할 필요가 없음
  - 그러나 앞선 방법으로는 이 장점을 살릴 수 없음, 버전 번호가 붙은 라이브러리명을 사용
- 공유 라이브러리 이름 종류
  - real name: 공유 라이브러리 실제 파일명
  - soname: /etc/ld-linux.so가 찾는 공유 라이브러리 파일명
  - linker name: 컴파일(링킹)시 참조하는 공유 라이브러리 파일명

공유 라이브러리 이름 규약 (2)

- 예: gdbm(GNU Database Manager)

```
$ ls -l /usr/lib/libgdbm.so*
/usr/lib/libgdbm.so.4 -> libgdbm.so.4.0.0
/usr/lib/libgdbm.so.4.0.0
```

- 버전의 의미
  - Major Number(첫 번째 숫자):
    - 중대한 부분이 변경되어 이전 버전과의 호환성을 유지 할 수 없는 경우
    - 인터페이스 함수가 없어지거나 추가되고, 함수의 매개변수가 변경되는 등
  - Minor Number(두 번째 자리 숫자):
    - 호환성을 유지할 수 있는 수준의 중대한 변경
    - 인터페이스 함수의 변경은 없으나 기능이 변경된 경우
  - Release Number(마지막 자리 숫자, 선택사항):
    - 호환성을 유지할 수 있을 뿐만 아니라 기능 변경도 없는 사소한 변경

공유라이브러리의 업데이트 (1)

Release Number or Minor Number 변경

- 단순히 soname과 linker name의 심볼릭 링크 변경
  - 이후에 실행되는 모든 응용 프로그램은 새로운 공유라이브러리로 동작
- 보통 installer가 이를 수행

```
$ ls -l /usr/lib/libgdbm.so*
/usr/lib/libgdbm.so.4 -> libgdbm.so.4.0.0
/usr/lib/libgdbm.so.4.0.0
```

공유라이브러리의 업데이트 (2)

Major Number 변경

- linker name의 링크만 새로운 공유 라이브러리 real name로 연결
  - 이후에 컴파일되는 응용 프로그램은 모두 새로운 버전으로 컴파일 될 것임을 의미
- 새로운 공유 라이브러리를 연결하는 새로운 soname을 symbolic link로 생성
  - 이후에 실행되는 모든 응용프로그램은 여전히 예전 공유라이브러리로 동작

```
$ ls -l /usr/lib/libgdbm.so*
/usr/lib/libgdbm.so   -> libgdbm.so.2.0
/usr/lib/libgdbm.so.1 -> libgdbm.so.1.0
/usr/lib/libgdbm.so.2 -> libgdbm.so.2.0
/usr/lib/libgdbm.so.1.0
/usr/lib/libgdbm.so.2.0
```

컴파일과 사용방법 (1)

이름 규약에 따른 라이브러리 생성

- 라이브러리 생성시 soname과 real name 지정
  - gcc –shared –Wl,-soname,*your_soname* –o *real_name* *file_list* *lib_list*

```
$ ls
file1.c file2.c libmy.h main.c
$ gcc -fPIC -c file1.c file2.c
$ gcc -shared -Wl,-soname,libmy.so.0 \
-o libmy.so.0.0 file1.o file2.o
$ ls
file1.c file1.o file2.c file2.o libmy.h libmy.so.0.0 main.c
```

컴파일과 사용방법 (2)

프로그램코드 컴파일

- 프로그램 코드 컴파일(링킹)시에는 linker name을 사용하므로 linker name의 심볼릭 링크 생성

```
$ ln -s libmy.so.0.0 libmy.so
$ ls
file1.c file1.o file2.c file2.o libmy.h libmy.so
libmy.so.0.0 main.c
$ gcc -o main main.c -L. -lmy
```

실행

- 실행 시에 적재기 /lib/ld-linux.so는 soname으로 라이브러리를 참조하므로 soname의 심볼릭 링크 생성, 또한 실행 전에 당연히 경로 추가

```
$ ln -s libmy.so.0.0 libmy.so.0
$ main
Hello Linux
Hello Fedora
```

요약

- 정적 라이브러리
  - ar 도구로 목적파일 묶어 만듬
  - ar 명령: r, t, d, x, p
  - ar 명령 r과 연관된 수식어: u, s, o, v
- 공유
- 라이브러리
  - 공유라이브러이를 위한 목적파일 컴파일 옵션: -fPIC
  - 공유라이브러리 생성 옵션: -shared
  - 사용자 경로 추가: LD_LIBRARY_PATH, -Wl,-rpath,path
  - 공유라이브러리 이름: linker name, soname, real name
  - soname지정 옵션: -Wl,-soname,name
  - 관련 도구들: ldd, ldconfig, /lib/ld-linux.so.N