CS 33

More Libraries
Pipes
Locking Files
A Problem

• You've put together a library of useful functions
  – libgoodstuff.so

• Lots of people are using it

• It occurs to you that you can make it even better by adding an extra argument to a few of the functions
  – doing so will break all programs that currently use these functions

• You need a means so that old code will continue to use the old version, but new code will use the new version
A Solution

• The two versions of your program coexist
  – libgoodstuff.so.1
  – libgoodstuff.so.2

• You arrange so that old code uses the old version, new code uses the new

• Most users of your code don’t really want to have to care about version numbers
  – they want always to link with libgoodstuff.so
  – and get the version that was current when they wrote their programs
Versioning

```
$ gcc -fPIC -c goodstuff.c
$ ld -shared -soname libgoodstuff.so.1 \ 
  -o libgoodstuff.so.1 goodstuff.o
$ ln -s libgoodstuff.so.1.1 libgoodstuff.so
$ gcc -o prog1 prog1.c -L. -lgoodstuff \ 
  -Wl,-rpath .
$ vi goodstuff.c
$ gcc -fPIC -c goodstuff.c
$ ld -shared -soname libgoodstuff.so.2 \ 
  -o libgoodstuff.so.2 goodstuff.o
$ rm -f libgoodstuff.so
$ ln -s libgoodstuff.so.2.2 libgoodstuff.so
$ gcc -o prog2 prog2.c -L. -lgoodstuff \ 
  -Wl,-rpath .
```
Interpositioning

prog

wrapper

puts
How To …

```c
int __wrap_puts(const char *s) {
    int __real_puts(const char *);

    write(2, "calling myputs: ", 16);
    return __real_puts(s);
}
```
$ cat tputs.c

```c
int main() {
    puts("This is a boring message.");
    return 0;
}
```

$ gcc -o tputs -Wl,--wrap=puts tputs.c myputs.c

$ ./tputs
calling myputs: This is a boring message.

$
How To (Alternative Approach) …

#include <dlfcn.h>

int puts(const char *s) {
    int (*pptr)(const char *);

    pptr = (int(*)())dlsym(RTLD_NEXT, "puts");

    write(2, "calling myputs: ", 16);
    return (*pptr)(s);
}

What’s Going On …

• gcc/ld
  – compiles code
  – does static linking
    » searches list of libraries
    » adds references to shared objects
• runtime
  – program invokes *ld-linux.so* to finish linking
    » maps in shared objects
    » does relocation and procedure linking as required
  – *dlsym* invokes *ld-linux.so* to do more linking
    » RTLD_NEXT says to use the next (second) occurrence of the symbol
Delayed Wrapping

• **LD_PRELOAD**
  - environment variable checked by *ld-linux.so*
  - specifies additional shared objects to search (first) when program is started
Environment Variables

• Another form of exec
  – `int execve(const char *filename, char *const argv[], char *const envp[]);`

• `envp` is an array of strings, of the form
  – `key=value`

• Programs can search for values, given a key

• Example
  – `PATH=~/.bin:/bin:/usr/bin:/course/cs0330/bin`
Example

$ gcc -o tputs tputs.c
$ ./tputs
This is a boring message.
$ LD_PRELOAD=./libmyputs.so.1; export LD_PRELOAD
$ ./tputs
calling myputs: This is a boring message.
$
MMapping Libraries

available for mmap

stack

dynamic

bss

data

text

my lib

C library

Copyright © 2019 Thomas W. Doeppner. All rights reserved.
Problem

• How is relocation handled?
Pre-Relocation

C library

stdfiles: 1,200,600
&stdfiles

printf: 1,000,400

call printf

math library

call printf

1,000,000

3,000,000
But …

mary’s library

my library

5,000,000

5,500,000
But …

my library
Mary’s library

5,000,000
5,500,000
8,000,000
We need to relocate all references to Mary’s library in *my library*. What option should we give to *mmap* when we map *my library* into our address space?

a) the MAP_SHARED option
b) the MAP_PRIVATE option
c) mmap can’t be used in this situation
Relocation Revisited

- Modify shared code to effect relocation
  - result is no longer shared!
- Separate shared code from (unshared) addresses
  - position-independent code (PIC)
  - code can be placed anywhere
  - addresses in separate private section
    » pointed to by a register
Mapping Shared Objects

Process A

`printf( )`

stdio

`printf( )`

Process B

`printf( )`
Mapping printf into the Address Space

- **Printf’s text**
  - read-only
  - can it be shared?
    » yes: use MAP_SHARED

- **Printf’s data**
  - read-write
  - not shared with other processes
  - initial values come from file
  - can mmap be used?
    » MAP_SHARED wouldn’t work
      • changes made to data by one process would be seen by others
    » MAP_PRIVATE does work!
      • mapped region is initialized from file
      • changes are private
Mapping printf

- printf text
  - page 6
  - page 7
- printf data
  - page 31
  - page 32

Process 1

- printf text
  - page 3
  - page 4
- printf data
  - page 41
  - page 42

Process 2

Real Memory

- printf page 0
- P1's printf page 2
- P1's printf page 3
- P2's printf page 2
- printf page 1

Disk

- page 0
- page 1
- page 2
- page 3
Position-Independent Code

• Produced by gcc when given the –fPIC flag
• Processor-dependent; x86-64:
  – each dynamic executable and shared object has:
    » procedure-linkage table
      • shared, read-only executable code
      • essentially stubs for calling functions
    » global-offset table
      • private, read-write data
      • relocated dynamically for each process
    » relocation table
      • shared, read-only data
      • contains relocation info and symbol table
Global-Offset Table: Data References

Global Offset Table

errno

myglob

errno address

myglob address
Functions in Shared Objects

- Lots of them
- Many are never used
- Fix up linkages on demand
An Example

```c
int main( ) {
    puts("Hello world\n");
    ...

    return 0;
}
```

```
00000000000006b0 <main>:  
  6b0: 55  push   %rbp 
  6b1: 48 89 e5  mov   %rsp,%rbp 
  6b4: 48 8d 3d 99 00 00 00  lea   0x99(%rip),%rdi 
  6bb: e8 a0 fe ff ff  callq  560 <puts@plt> 
  ...
```
Before Calling puts

.PLT0:
    pushq GOT+8(%rip)
    jmp   *GOT+16(%rip)
    nop;  nop
    nop;  nop
.puts:
    jmp   *puts@GOT(%rip)
.putsnext
    pushq $putsRelOffset
    jmp   .PLT0

.PLT2:
    jmp   *name2@GOT(%rip)
.PLT2next
    pushq $name2RelOffset
    jmp   .PLT0

Procedure-Linkage Table

GOT:
    .quad _DYNAMIC
    .quad identification
    .quad ld-linux.so

.puts:
    .quad .putsnext
.name2:
    .quad .PLT2next

Relocation Table

GOT_offset(puts), symx(puts)
GOT_offset(name2), symx(name2)
After Calling puts

```assembly
.PLTO:
    pushq  GOT+8(%rip)
    jmp   *GOT+16(%rip)
    nop;  nop
    nop;  nop
.puts:
    jmp   *puts@GOT(%rip)
.putsnext
    pushq  $putsRelOffset
    jmp   .PLT0
.PLT2:
    jmp   *name2@GOT(%rip)
.PLT2next
    pushq  $name2RelOffset
    jmp   .PLT0

Procedure-Linkage Table

GOT:
    .quad  _DYNAMIC
    .quad  identification
    .quad  ld-linux.so
.puts:
    .quad  puts
.name2:
    .quad  .PLT2next

Relocation Table:

GOT_offset(puts), symx(puts)
GOT_offset(name2), symx(name2)
```

Relocation info:

GOT_offset(puts), symx(puts)
GOT_offset(name2), symx(name2)
Interprocess Communication (IPC): Pipes
Interprocess Communication: Same Machine I

Kernel buffer
Interprocess Communication: Same Machine II

Shared Memory

process 1

process 2
Interprocess Communication: Different Machines

[Diagram showing two machines communicating through the internet]
Intramachine IPC

\$\text{cslab2e \ who | wc -l}
Intramachine IPC

$\texttt{cslab2e \ who \ | \ wc \ -l}$

```c
int fd[2];
pipe(fd);
if (fork() == 0) {
    close(fd[0]);
    close(1);
    dup(fd[1]); close(fd[1]);
    execl("/usr/bin/who", "who", 0); // who sends output to pipe
}
if (fork() == 0) {
    close(fd[1]);
    close(0);
    dup(fd[0]); close(fd[0]);
    execl("/usr/bin/wc", "wc", "-l", 0); // wc’s input is from pipe
}
close(fd[1]); close(fd[0]);
// ...
```
Intermachine Communication

• Can pipes be made to work across multiple machines?
  – covered soon ...
    » what happens when you type
      who | ssh cslab3a wc -l
      ?
Sharing Files

• You’re doing a project with a partner
• You code it as one 15,000-line file
  – the first 7,500 lines are yours
  – the second 7,500 lines are your partner’s
• You edit the file, changing 6,000 lines
  – it’s now 5am
• Your partner completes her changes at 5:01am
• At 5:02am you look at the file
  – your partner’s changes are there
  – yours are not
Lessons

- Never work with a partner
- Use more than one file
- Read up on git
- Use an editor and file system that support file locking
What We Want ...

I want to just read the file.

Me too.

I want to modify the file.

Me too.
Types of Locks

• **Shared (readers) locks**
  – any number may have them at same time
  – may not be held when an exclusive lock is held

• **Exclusive (writers) locks**
  – only one at a time
  – may not be held when a shared lock is held
What We Want ...

I’ve got a shared lock.

Me too.

My exclusive lock request was rejected.

Mine too.
What We Want ...

I’ve got an exclusive lock.

My shared request was rejected.

My exclusive request was rejected.

Mine too.
Locking Files

- Early Unix didn’t support file locking
- How did people survive?
  - `open("file.lck", O_RDWR|O_CREAT|O_EXCL, 0666);`
    » operation fails if *file.lck* exists, succeeds (and creates *file.lck*) otherwise
    » requires cooperative programs
Locking Files (continued)

• How it’s done in “modern” Unix
  – “advisory locks” may be placed on files
    » may request shared (readers) or exclusive (writers) lock
      • `fcntl` system call
    » either succeeds or fails
    » `open, read, write` always work, regardless of locks
    » a lock applies to a specified range of bytes, not necessarily the whole file
    » requires cooperative programs
Locking Files (still continued)

- How to:

```c
struct flock fl;
fl.l_type = F_RDLCK;  // read lock
// fl.l_type = F_WRLCK;  // write lock
// fl.l_type = F_UNLCK;  // unlock
fl.l_whence = SEEK_SET;  // starting where
fl.l_start = 0;        // offset
fl.l_len = 0;          // how much? (0 = whole file)
fd = open("file", O_RDWR);
if (fcntl(fd, F_SETLK, &fl) == -1)
    if ((errno == EACCES) || (errno == EAGAIN))
        // didn’t get lock
    else
        // something else is wrong
else
    // got the lock!
```
Locking Files  (yet still continued)

• Making locks mandatory:
  – if the file’s permissions have group execute permission off and set-group-ID on, then locking is enforced
    » read, write fail if file is locked by someone other than the caller
  – however …
    » difficult to implement on distributed file systems (such as used at Brown CS)
Quiz 2

• Your program currently has a shared lock on a portion of a file. It would like to “upgrade” the lock to be an exclusive lock. Would there be any problems with adding an option to `fcntl` that would allow the holder of a shared lock to wait until it’s possible to upgrade to an exclusive lock, then do the upgrade?

  a) at least one major problem
  b) either no problems whatsoever or some easy-to-deal-with problems