Q: The separation between a Xinu ELF file and the running image leads to a potential problem: if the ELF file is changed (Xinu sources are recompiled) after an image starts to run, symbol table addresses in the ELF file may no longer match the locations of items in the running image. How can you ensure the ELF file read at run time matches the image that is executing?
A: One solution can be every time you run load_program or load_library, the xinu.elf file will be updated and reload into the memory, thus guarantee the loaded xinu.elf file is always up to date
Warning :my implementation does not provide this mechanism, bearing in mind that it would be too much a memory lost if you load xinu.elf file every time you load_program, so for all programs, you will only do one time load xinu.elf, if the xinu.elf is updated, you have to restart the machine.

Q: What is the most difficult aspect of the project? Why?
A: The most difficult aspect of the project is to figure out the relocation process, cause there are so many steps in the process that can go totally wrong. Though the ELF header is well structured, the way it stores information is not direct. To get a symbol value(the address where the info of the symbol is actually stored) for relocation is tricky. For example, for a global function used, you need following steps to get it S value

1. check if it is global, (symbol->st_shndx == SHN_UNDEF)
2. get the symbol name for the corresponding strtab
3. compare the name retrieved to all the symbol names in the symtab of xinu.elf
4. Find match, get it back;

Q: Details of my implementation

load_program: the details of load_program and load_library is hidden in two files, with the elfload.c containing all the functions and reloc.h containing all the predefined structures.
Relocation process (part of it is covered in midway submission) :
> iterate through all the section headers
>find relocation sections, section type being SHT_RELA or SHT_REL
>for every entry in the section, do the relocation, if it is global variable, find its S value from the preloaded xinu.elf file in the memory
> get the main function address of the file
> run the function

load_library: load library is done based on the the completion of full relocation, the cool part about my implementation is load_program and load_library both call elf_load_file to take care of all the business, the only difference is the setloadmode(), with load_library setting loadmode to 2, which signals extra step elf_load_stage() needed to store the function

information to the global extern funrec funrecs[31] structure. Provided neither of the following two rule is violated: function number not exceeding 10, no duplicate of functions

ls: what ls command does is to take into a parament standing for the directory, open the directory file, dirfd = open(RFILESYS, argv[1], "ro");   then keep reading that file block by block (block size is the sizeof(struct rfdirent))to the end, then print it out.
semdump: Similar to the ls command

Testcases :

/*test for load_program*/

case 1: consecutive load same file

```
void* helloworld = load_program("helloworld");
resume(create(helloworld, 4096, 20, "helloworld", 2, 0, NULL));
void* hello2 = load_program("helloworld");
resume(create(helloworld, 4096, 20, "hello2", 2, 0, NULL));
```

output: expected match
Morning!
Morning!

case 2: load a program that does not exist

```
void* helloworld = load_program("hidee");
```

output: error !

case 3: load a program that has uninitialized variables (SHN_COMM)

```
  void* hihi = load_program("hihi");
  resume(create(hihi, 4096, 20, "helloworld", 2, 0, NULL));
```

output: match expected

a=2
b=4
c=2
d=3
e=10
ye = l
xi = x
Hello World!!Hello World!!

/*test for load_iibrary*/

case 1: load a library contains main

output:

sorry, two main not allowed !
Unable to load library
 isdirty: 0 name: (null)
 isdirty: 0 name: (null)
 isdirty: 0 name: (null)
 isdirty: 0 name: (null)
 isdirty: 0 name: (null)
 isdirty: 0 name: (null)
 isdirty: 0 name: (null)
 isdirty: 0 name: (null)
 isdirty: 0 name: (null)

case 2: load 2 libraries with duplicate functions

output:

gotta! , you are already here, id number: 6, name : add6
Conflict function or library
Unable to load library

case 3: successfully loaded the libraries, then call the function

```
int result = load_library("myadd3");
int32 (*add1)(int32) = find_library_function("add10");
```

```
    if((int32)add1 == SYSERR) {
            return SYSERR;
    }

    kprintf("everything done\n");
    kprintf("%d\n", add1(myvalue));
```

output:

```
 everything done
 4
```

case 4: load a library containing more than 10 functions

output:

```
trying to load too many functions
Unable to load library
 isdirty: 0        name: (null)
 isdirty: 0        name: (null)
 isdirty: 0        name: (null)
 isdirty: 0        name: (null)
 isdirty: 0        name: (null)
 isdirty: 0        name: (null)
 isdirty: 0        name: (null)
 isdirty: 0        name: (null)
```

case 5: load more than 3 libraries

output: **Error: Load more than 3 libs**

/*test for semdum and ls*/

case 1: run semdump command in the shell

xsh $ semdump

Output for semdump

Entry	State	Count	Queue
0	S_USED	0	100
1	S_USED	64	102
2	S_USED	16	104
3	S_USED	-1	106
4	S_USED	64	108
5	S_USED	-1	110
6	S_USED	1	112
7	S_USED	1	114
8	S_USED	1	116
9	S_USED	1	118
10	S_USED	1	120

Note: the table is huge, this is just a snippet of the whole output

case 1: run ls command with /without parameter

Output for ls test

xsh $ ls (deleted part of the output)

helloworld.c
test/
rfserver
myadd2.c
test2.c
Not DIR: End of file

xsh $ ls test

a/
c
./
../
Not DIR: End of file