1. Linkers and Executables

Due Jan 27 by 11:59pm **Points** 7 **Submitting** a file upload **File types** pdf and zip

Submission

Turned in!
Jan 27 at 11:50pm
Submission details
Download Assignment1.1.pdf

Grade: 7 (7 pts possible)
Assigned peer reviews
YATIN DANDI 170825
SUNIDHI DHANDHANIA 180799

Comments: No comments

Re-submit assignment

CS 335: Compiler Design

Assignment 1

This assignment has 7 questions.

Q1: Objdump 1

Use **gcc** with appropriate options to generate object files and a.out for the given C program divided in 3 files. Use the **objdump** command to answer the following

```
(a) For each object file, list all the locations that need
relocation.

(b) Examine the a.out file to find out how the relocation has
actually taken place. For each location in part (a), list the
final address.
```

Use **objdump** as **objdump** -D -S -r modulename.o to see the relocation information along with the disassembled code.

```
-----------------------------------------------
/*main.c*/
#include <stdio.h>
void swap (); /* declaration */
int buf [] = {0x72,0x56}; /* initialised global */
int main () /* definition main */
{
    swap ();
    printf("buf[0]= %d buf[1]= %d\n", buf[0], buf[1]);
    return 0;
}
-----------------------------------------------
/*swap.c*/
extern int buf []; /*declaration buf*/
int *bufp0 = &buf[0]; /* initialized global */
int *bufp1;  /* uninitialized global */
void swap () /* definition swap */
{
    static int temp = 100; /* static variable */
    bufp1 = &buf[1];
    temp = *bufp0;
    *bufp0 = *bufp1;
    *bufp1 = temp;
}
------------------------------------------------
/*other.c*/
int buf[2]; /* uninitialized global */
static void sf(); /* static function declaration */
void f() {
    sf();
}
static void sf() /* static function definition */
{
    buf[0] = 3;
    buf[1] = 4;
}
-----------------------------------------------
```

Q2: Objdump 2

For the following program, use **gcc** to generate object code. Then, do the following:

```
(a) Annotate fragments of target code with the source
statemets that they correspond to.

(b) Annotate each local variable and parameter with its relative address.
```

Compile your programs as:

```
   gcc -static -fno-asynchronous-unwind-tables programname.c
```

[Optional question, not to be submitted: What happens if you remove -fno-asynchronous-unwind-tables flag?]

```
int main()
{
    int a=1,b=1;
    while(a<=10)
    {
        b=b*a;
        a++;
    }
    return b;
}
```

Q3: Objdump 3

For the following program, use **gcc** to generate object code. Then, do the following:

```
(a) Annotate fragments of target code with the source
statemets that they correspond to.

(b) Annotate each local variable and parameter with its (relative address)
```

Compile your programs as:

```
   gcc -static -fno-asynchronous-unwind-tables programname.c
```

[Optional question, not to be submitted: What happens if you remove -fno-asynchronous-unwind-tables flag?]

```
struct data {
    int sum;
    int b[5];
};

int main()
{
    struct data rec1;
    rec1.sum=0;
    rec1.b[0]=2;
    rec1.sum=rec1.sum+rec1.b[0];
    return rec1.sum;
}
```

Q4: Static Linking

This question assumes static linking. In each of the pairs of modules shown below, indicate how the multiply defined symbol main would be resolved. Your answer should be of the form "The use of the symbol main in module X will resolve to the declaration of main in module Y". You may also mention if the linker will give an error or will arbitrarily choose a declaration. Verify your answer by using **objdump** and **readelf** on the .o and the a.out files.

```
****(a)****
```

```
/* Module a1 */
#include <stdio.h>
int main()
{
    printf("%p\n", &main);
    p();
}

/* Module a2 */
#include <stdio.h>
int main;
int p ()
{
    printf("%p\n", &main);
}
```

****(b)****

```
/* Module b1 */
#include <stdio.h>
int main()
{
    printf("%p\n", &main);
    p();
}

/* Module b2 */
#include <stdio.h>
int main=1;
int p ()
{
    printf("%p\n", &main);
}
```

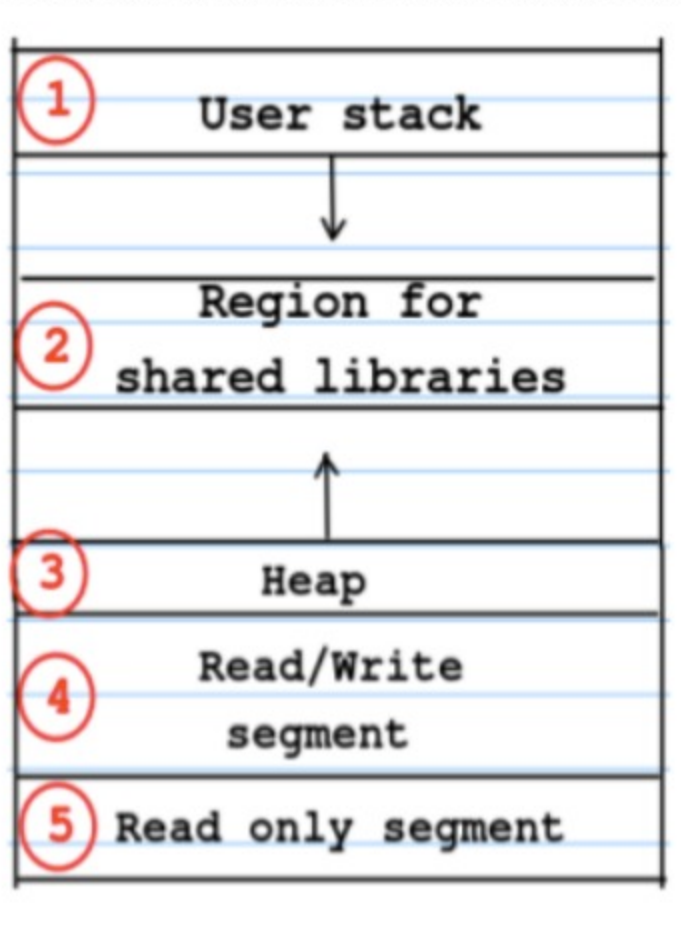
```
****(c)****
```

```
/* Module c1 */
#include <stdio.h>
int main()
{
    printf("%p\n", &main);
    p();
}


/* Module c2 */
#include <stdio.h>
static int main=1;
int p ()
{
    printf("%p\n", &main);
}
```

Q5: Memory Layout

Consider the memory image of a program in Linux. The regions in the memory image have been numbered.



Conside the following two modules in which symbols (variables and function names) have been declared. Consider the executable formed after the two modules are compiled and statically linked under the linux environment. For each symbol mention the region number in which it will be stored.

```
/* Module 1: */
int x = 10;
static int y = 5;
int main ()
{
    int* p;
    p = (int *) malloc(4);
    *p = 5;
    printf("%d\n", *p);
    printf("%d\n", f());
}

/* Module 2: */
int a[10];
int f ()
{
    static int k = 0;
    return k++;
}
```

Q6: Link Issue!

Assume that two files test1.c and test2.c contain the programs shown below.

```
(a) What is the output of the program when the files are
    compiled and linked with the gcc compiler?  Explain your
    answer. You can use the information collected in part (b) and
    (c) below.

(b) For each object file, list all the locations that need
relocation.

(c) Examine the a.out file to find out how the relocation has
actually taken place. For each location in part (b), list the
final address.
```

Use **objdump** as **objdump** -D -S -r modulename.o to see the relocation information along with the disassembled code.

```
/* test1.c: */
#include<stdio.h>
struct {
    int x;
    int y;
} rec = {13,5};
void f1();

int main ()
{
    f1 ();
    printf(" %d %d\n", rec.x, rec.y);
    return 0;
}

/* test2.c: */
#include<stdio.h>
struct {
    int y;
    int x;
} rec;

void f1 ()
{
    printf(" %d %d\n", rec.x, rec.y);
}
```

Q7: Static Linking - 2

This question assumes static linking. Consider the two modules shown below stored in the files f1.c and f2.c.

```
/* Module 1: f1.c */
#include<stdio.h>
int a[5] = {0,1,2,3,4};
extern int b [];
double c;
int *x = &(a[3]);
int *y = b;
extern int fn();
int main ()
{
    c = 100;
    printf("%d %f", 5==fn(), c);
    return 0;
}

/* Module 2: f2.c */
int c = 58; int d = 5;
extern int a[];
int b []= {1,2,3};
int z = a[3];
int fn ()
{
    int e = d;
    return e;
}
```

Now answer the following questions assuming that an address takes 4 bytes, an int takes 4 bytes and a double takes 8 bytes.

```
(a) There is a compilation error in the program involving a
    single line. Point out the line and explain the
    error. For the rest of the question assume that the line
    is not there.

(b) The program outputs the value 0. Assuming the symbol
    resolution policy discussed in the class explain the
    output.

(c) Which of the symbol references in the program are
    relocatable? For each relocatable symbol reference
    mention whether the relocation is PC-relative or
    Absolute. You can use gcc to compile the files, and
    objdump and  readelf to collect this information.
```