ICS EXE 9

May 11, 2022

1 Symbol

The following program consists of two modules: **main** and **foo**. Their corresponding source code files are shown below. (All the process of linking runs on an x86-64 machine.)

```c
/* main.c */
#include <stdio.h>

extern char *names[];
static int id;
int foo(int n);
void main(void) {
    id = 103;
        char *str = names[foo(id)];
        printf("%s %d\n", str, id);
}
```

```c
/* foo.c */
char *names[] = {"Ayaka", "Klee",
                    "Ganyu", "Link"};
int id = 102;
int foo(int n) {
        int res = 0;
        switch(n) {
                case 100:
                        res = 1; break;
                case 103:
                        res = 2; break;
                case 104:
                        res = 3; break;
                default:
                        res = 0;
        }
        id = 233;
        return res;
}
```

1. For symbols that are defined and referenced in **main.o** and **foo.o**, please complete the symbol tables. The format of them are the same as ones in **section 7.5** of your ICS book.

Module	Name	Type	Bind	Value(Hex)	Size	NDX
main.o	id			00000000		4
main.o	main			00000000	88	
main.o	foo		GLOBAL	00000000		
foo.o	id	OBJECT		00000020		

2. Please explain why the **Value** of **id** in **foo.o** is 0x00000020.

3. Please write down the output of **main.c**.

2. Dynamic Linking

Given the PLT table:

```
PLT[1] <free>
400450: jmpq *0x20055a(%rip)
400456: pushq $0x0
40045b: jmpq 0x400440
PLT[2] <printf>
400460: jmpq *0x200552(%rip)
400466: pushq $0x1
40046b: jmpq 0x400440 ...
PLT[5] <malloc>
400490: jmpq *0x20053a(%rip)
400496: pushq $0x4
40049b: jmpq 0x400440
```

1. Please fill in the GOT table before the execution of main

Address	Entry	Contents	Description
0x600998	GOT[0]	--	
0x6009a0	GOT[1]	--	
0x6009a8	GOT[2]	--	
	GOT[3]		
	GOT[4]		
	GOT[7]		

2. We have the following code:

```c
1: #include <stdlib.h>
2: void main(){
3:   int a = 1;
4:   printf("%d\n", a);
5:   char *c;
6:   c = (char*)malloc(4);
7:   printf("%d\n",a);
8:   free(c);
9: }
```

The addresses of functions are given:

printf	0x00007ffff7a81cf0
malloc	0x00007ffff7aacfc0
free	0x00007ffff7aad600

Which entry of the GOT table will change and what will it be after the execution of:

line 4:

line 6:

line 7:

line 8:

3 Concurrent1

The following code implements a simple stack.

```c
typedef struct Node {
        struct Node *next;
        int value;
} Node;

void push(Node **top_ptr, Node *n) {
        n->next = *top_ptr;
        *top_ptr = n;
}

Node *pop(Node **top_ptr) {
        if (*top_ptr == NULL)
                return NULL;
        Node *p = *top;
        *top_ptr = (*top_ptr)->next;
        return p;
}
```

3.1

Is this implementation thread-safe?

3.2

Use semaphore to protect the operations towards the stack.

4 Concurrent2

Initially, counting semaphore S is initialized with value 2. What is the maximum possible value of x after all the processes complete execution.

Process 1	Process 2	Process 3	Process 4
`P(S)`	`P(S)`	`P(S)`	`P(S)`
`x = *addr`	`x = *addr`	`x = *addr`	`x = *addr`
`x = x + 1`	`x = x + 1`	`x = x - 2`	`x = x - 2`
`*addr = x`	`*addr = x`	`*addr = x`	`*addr = x`
`V(S)`	`V(S)`	`V(S)`	`V(S)`