Operating Systems 2018/19
Solutions for Assignment 2

Prof. Dr. Michael Beigl
Mathias Gottschlag, M.Sc.
Jonas Kiemel, M.Sc.

T-Question 2.1: Anatomy of a Program

Consider the following C program that does some random computations. Refer to the introductory C slides provided with the lecture in ILIAS if you need help with some of the keywords (e.g., const or static). Download the source code of the program from ILIAS and build it using `gcc` with the following command line:

```
gcc -g main.c func.c -o out
```

You should now have an executable file called `out`.

main.c:

```
#include <stdlib.h>
#include "func.h"

int main()
{
    int *parg, result;

    parg = (int*)malloc(sizeof(int));
    if (parg == NULL) exit(1);
    *parg = 10;

    result = func(parg);
    free(parg);

    return result;
}
```

func.h:

```
int func(int *parg);
```

func.c:

```
const int a = 42;
int b = 1;

int func(int *parg)
{
    static int s = 0;
    int r;

    if (s == 0) {
        r = *parg + a;
        s = 1;
    } else {
        r = *parg + b;
        b++;
    }

    return r;
}
```

a. In which segments of the executable are `a`, `b`, `s`, and `func` stored? Use the command `readelf -hSs out` to verify your solution. Locate each object in the symbol table (`.symtab`) and match the section index given in the Ndx column with the section headers. Hint: The compiler may have renamed `s` to `s.n` with n being some decimal number to prevent name clashes. **2 T-pt**

Solution:
a: .rodata Read-only Data Segment
b: .data Data Segment
s: .bss Block Started by Symbol Segment (because we initialized it to zero!)
func: .text Code Segment

b. In which address space segments do `r` and `*parg` reside, when executing the program? **1 T-pt**

Solution:
r: Stack
**parg: Heap*

c. Where is the return value of `func()` placed? Verify you solution by disassembling the executable with `objdump -Sd out` and finding the epilogue of `func()`. **1 T-pt**

Solution:
Disassembly of the return statement in func:

```
return r;
40065c: 8b 45 fc  mov  -0x4(rbp),eax ; move (r) from stack to eax
40065f: 5d        pop  rbp
400660: c3        retq
```

The return value is placed in the `eax` register.

d. What shared libraries are needed by `out`? Use the tool `ldd` to list all library dependencies. What purpose does each of the libraries serve? **3 T-pt**

Solution:
On an Ubuntu 14.04 (64-bit) the following dependencies exist:

linux-vdso.so.1 *Kernel provided shared library (virtual dynamic shared object) that contains system helper routines such as a way for the C library to perform system calls in a platform-independent manner.*

/lib/x86_64-linux-gnu/libc.so.6 *64-bit C standard library. Contains, besides others, the malloc() and free() functions.*

/lib64/ld-linux-x86-64.so.2 *64-bit ELF dynamic linker/loader. Responsible for resolving library dependencies, loading them into the address space of the process and performing the dynamic linking.*

T-Question 2.2: Processes

a. What is the difference between a program and a process? **1 T-pt**

Solution:

Program: *Passive entity, just a file on disk*

Process: *Active entity, has an execution context (instruction pointer, resources, . . .)*

b. When a process exits, it may become a zombie. What is a zombie and what needs to be done for this not to happen? **1 T-pt**

Solution:
When a child-process terminates the parent may want to know the child's exit status. To make this possible a stub of the child will stay in the process table after termination as a zombie. The parent needs to collect the exit status of the terminated processes with the `wait()` or `waitpid()` system calls to free the zombies.

c. Process A creates process B which in turn creates process C. In a Linux system: What is C's parent after B was killed? **1 T-pt**

Solution:
The init process (PID 1) adopts C.

Note: *Recent Linux kernel versions additionally support* subreapers *(processes which fulfill the role of init for their descendant processes). See* `man 2 prctl` *for more information.*

d. What is the context of a process? Name at least 4 properties. **2 T-pt**

Solution:
The context is made up of properties which identify the current runtime state of a process. Some examples are:

- *General purpose registers, Instruction Pointer, Stack pointer*
- *Process State*
- *Process Priority*
- *Unique Process ID*
- *Open files*
- *Network connections*
- *Security credentials*

Total: 12 T-pt