Operating Systems – spring 2022
Assignment 2

Instructor: Hans P. Reiser

Submission Deadline: Monday, January 31, 2022 – 23:59

A new assignment will be published every week, right after the last one was due. It must be completed before its submission deadline.

T-Questions are theory homework assignments. **The answers to the assignments must be uploaded to Canvas.**

P-Questions are programming assignments. Download the provided template from Canvas. Do not fiddle with the compiler flags. Submission instructions can also be found in first assignment.

In this assignment you will get familiar with the process abstraction, the basic address space layout of a process, and the Linux process API.

T-Question 2.1: Anatomy of a Program

Consider the following C program that does some random computations. Refer to the introductory C slides if you need help with some of the keywords (e.g., const or static). Download the source code of the program from Canvas 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**

b. In which address space segments do `r` and `*parg` (the value (int) that `parg` points to) reside, respecively, when executing the program? **1 T-pt**

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**

d. What shared libraries are needed by `out`? Use the tool `ldd` to list all library dependencies. `/lib64/ld-linux-x86-64.so.2` is the 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. What purpose does each of the other libraries serve? **2 T-pt**

T-Question 2.2: Processes

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

b. Explain the terms Zombie and Orphan in the context of processes! What is done to clean up the situation? **2 T-pt**

c. Some 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**

P-Question 2.1: Dynamic Memory Allocation

Download the template **p1** for this assignment from Canvas. You may only modify and upload the file `persistence.c`.
The function `gmtime()` transforms a date and time value (epoch seconds) to a structure in which that value is broken down into year, month, day, hours, etc. by returning a pointer to a global `tm` structure, which contains the converted value. Subsequent calls to `gmtime()` return the same pointer and only update the global structure.

a. Write a function that creates a persistent copy of the supplied `tm` structure on the heap and updates the caller's pointer to point to the copy. **2 P-pt**

```
void make_persistent(struct tm **time);
```

b. Extend your program with a function that releases a copy of the `tm` structure and resets the caller's pointer to `NULL`. **1 P-pt**

```
void free_persistent(struct tm **time);
```

P-Question 2.2: A Simple Program Starter

Download the template **p2** for this assignment from Canvas. You may only modify and upload the file `run_program.c`.
An important feature of every shell is to start external programs.

a. Write a function with the following features: **5 P-pt**

- Starts a program that may be specified by its full path and name (i.e., /usr/-bin/who) or only by its name if it is located in one of the directories contained in the `PATH` environment variable. Do not use `system()`.
- Passes the supplied arguments on to the new process.
- Waits for the newly created process to exit.
- Returns the special error value 127 to report an error condition and 0 to indicate success.

```
int run_program(char *file_path, char *argv[]);
```

- If `file_path` is `NULL`, you should return the special error value. If `argv` is `NULL`, you should run the program without arguments. (Your program starter should not be terminated by an exception in either of the two cases.)

b. Modify the starter to return the exit status of the previously started/exited process. Keep a return value of 127 to indicate error conditions in your own program. **2 P-pt**

Total:
10 T-pt
10 P-pt