Exercise B: Memory management

One of the things us programmers use provided the operating system is memory. Today we are going to take a lot at memory and see how we can use it.

1. Memory allocation

First we are going to write a program that uses up some memory.

- Create a file “memory.c” with an empty program in it.
- Declare a void* variable and initialize it with NULL.

```
void *ptr = NULL;
```

- Allocate 10 megabytes of memory. Use the function malloc() for this job. Store the returned value in the variable created before. It will act as handle to the memory. We will come to the * later.
- To prevent the program from closing, ask the user for some input right before the end of the main function. We don’t really need the input, this is just a workaround.

Hints

These functions may be helpful to use. Use the manpage to get further information:

- malloc()

Includes

```
#include <stdio.h>
#include <stdlib.h>
```

Operating Systems

2. Memory leaks

Now we have allocated some memory. But so far we have only taken some memory and never given any thought on returning it back to the operating system. Kind of unfair. This behavior is called a memory leak. Let’s fix it.

Before we end our program, we always have to free every memory we allocated manually with malloc().

- Use the free()-function to free the allocated memory at the end of the main function. Pass it the handle to the memory.

Hints

These functions may be helpful to use. Use the manpage to get further information:

- free()

Includes

```
#include <stdio.h>
#include <stdlib.h>
```

3. Taking a closer look

We have now allocated some memory, but we have never actually seen it. So the next step is to inspect our programs memory use.

Change the program, so it allocates 500 MB of memory. This is quite a large amount. Then start the program.

- Run the ps command below in another terminal. This should print you a snapshot of the process.

```
$ ps aux | grep memory | grep –v grep
```

- Take a look at the output. It should look something like the following. What are the VSZ and RSS values?

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
xy123     9933  0.0  0.0 516204   792 pts/4    S+   17:05   0:00 ./memory
```

- Include the string-library by adding "`#include <string.h>`" to your program.
- Add the following line to our program and run it again.

```
memset(ptr, 0, 500*1024*1024);
```

- The output of ps should have changed dramatically. Why is that so? What part does the operating system play in it?

Operating Systems

4. Pointers

Pointers are basically a handle to raw memory. In order to access the memory, you always need a pointer to it. Same way as with references to objects in Java.

- Create a file “pointer.c” with an empty program in it.
- Create an int variable called “var” and assign it some random value.

Now we are going to create a pointer to that variable. Therefore, we need an int-pointer.

- Create a pointer to an int variable and initialize it with NULL.

```
int *ptr = NULL;
```

- Assign a target to the pointer. Therefore use the &-Operator.

```
ptr = &var;
```

- Print the pointer to the command line. What does it contain? How is it able to point to a variable? What does the &-Operator do?

Hints

Includes

```
#include <stdio.h>
```

5. CbV vs CbR

The content of a pointer is nothing more than a memory address. This is why pointers can point to raw memory and variables. They simply contain the address where the pointee is located.

But how do you change the content of the pointee? You have to dereference the pointer with the *-operator.

- Deference the pointer with the *-operator and assign the pointee some value.

```
*ptr = 42;
```

- Add the functions listed below to the program. Insert them before your main function.

```
void func1(int val) {
}

void func2(int *ref){
}
```

- In the functions, access the variables passed as arguments and try to change their values.
- Call both functions from your main function. Call func1 with your int variable as argument, func2 with the address to your int variable as argument.
- Print the content of the int variable in your main-function before and after each function call. Where is the difference between the two calls? Does the content of the variables change? Why is that so?

Operating Systems

6. Time measurement

In this exercise we are going to write a function for measuring time. We will need this function again later in the lecture to measure the performance of some programs.

- Create a file called “time_measurement.c” with an empty program in it.
- Create two variables of the type “struct timeval”. These will be used as the start and end point of the measurement.
- Create a time stamp by calling gettimeofday(). You will find the necessary parameters in the manpage (HINT: pay close attention to the second parameter).
- Now add something to your program that consumes time. You can create a loop that counts to a very high number like 1000000.
- Take the second time stamp for our measurement.
- Add the following function to your file and implement the necessary logic. The function should return the time elapsed between the two time stamps in seconds.

```
double sec_between(struct timeval *start, struct timeval *end) {
    // code goes here
}
```

  For better understanding the timeval structure is displayed below. The tv_sec field is the number of seconds since the 1st Jan. 1970. The tv_usec field contains the number of microseconds passed since the beginning of that second (1 second = 1 000 000 microseconds).

```
struct timeval {
    time_t      tv_sec;     /* seconds */
    suseconds_t tv_usec;    /* microseconds */
};
```

  Note: Convert microseconds to seconds by dividing them by 1000000.0. The “.0” at the end tells the C compiler that it's a double value so the result will be seconds stored as double.

- Print the elapsed time to the console. Use “%lf” to print the value.
- Clean up the function and add all the necessary comments you need to understand it. We will use this function later in the course of the lecture.

Hints

These functions may be helpful to use. Use the manpage to get further information:

- gettimeofday()

Includes

```
#include <stdio.h>
#include <sys/time.h>
```