TTK4147 - Real-time Systems
Exercise 2 - Memory

September 2021

Remember to check in at the lab, https://innsida.ntnu.no/checkin/room/3066, to allow for easier infection control in the event of a Covid-19 infection on campus.

In this exercise you will be introduced to some concepts that are important for C programming and necessary for this course, even if it is not directly related to real time. Only parts of the information you need in the exercises is given in the exercise text. You will also need to find information on the internet and from other sources, or ask the student assistants, they are there to help you.

If you have questions or want to get your assignment approved, please apply for a spot (using your NTNU-user) in the queue here: https://s.ntnu.no/rts.

1 Memory

Dynamic memory allows memory to be allocated while the program is running. In C dynamic memory allocation uses the functions `malloc()` and `free()`. This is typically used when the required memory size for a program is not known when it is compiled. It is important to free all the memory you have allocated, or you can get a memory leak. This means that as the program runs it takes more memory than it returns, which over time will crash the system when there is no more memory available. Dynamic memory is often avoided for real-time systems because of the danger of memory leaks and the increase in code complexity. But it is still often needed, especially in communication handling.

TASK A

The program below allocates memory for a very large 8GB matrix of 64-bit integers:

```
long xy_size   = 1000*1000*1000;     // 8 GB (sizeof(long) = 8 bytes)
long x_dim     = 100;
long y_dim     = xy_size/x_dim;

long** matrix  = malloc(y_dim*sizeof(long*));

for(long y = 0; y < y_dim; y++){
    matrix[y] = malloc(x_dim*sizeof(long));
}

printf("Allocation complete (press any key to continue...)\n");
getchar();
```

The computers at the lab have 8GB of RAM, which means there shouldn't be enough space to allocate this matrix. Open the System Monitor from Dash (aka the "start menu"), and go to the resources tab. From here you can see the total memory used by the system. Run the program, and observe the effects on the memory usage. Don't worry if the computer becomes unresponsive, it's just old and needs some time.

What happens?

What is the difference between "Memory" and "Swap"?

Now change the shape of the matrix by increasing `x_dim` first to 1000, then 10000, and run the program again. What happens?

Run the program with `x_dim` = 10000 again, but this time write zeros to all the memory:

```
memset(matrix[y], 0, x_dim*sizeof(long));
```

Explain why something different happens when the memory is also cleared.

TASK B

A common application of dynamic memory is to have a list that will grow and shrink in size as the program runs. The beginnings of the code for a simple dynamic array is available from the same folder where you downloaded this exercise text on Blackboard. Implement `array_insertBack()`, but without implementing a way to grow the size of the array if there isn't enough capacity (we'll get back to that later). Create an array with some small capacity, then `insertBack` more elements than there is capacity for, and run the program. What happens?

Compile and run the program again, but now add the compiler flags `-g -fsanitize=address`. What happens now?

AddressSanitizer is an invaluable tool for making sure you don't get your pointers and memory all tangled up. If you are writing code that uses pointers in any non-trivial way, you should probably enable this memory error detector – as long as the platform supports it. And if it doesn't, see if you can copy the relevant code to a platform that does while you work on it, then port it back after it has been tested.

TASK C

Implement `array_reserve()`, such that it can grow the capacity of the array. You will have to relocate the existing data by first allocating new space, moving the existing data, and then freeing

the old data.

Fix `array_insertBack()` so that it detects a lack of capacity and reserves more if necessary. Run and verify that the code works.

TASK D

Typically when increasing the capacity of a dynamic array, you would increase the capacity by some factor, say 2x or 1.5x. Let's say our array currently has a capacity of 2 elements. From here, we continuously insert new elements at the back, increasing the capacity as necessary. Draw out (or otherwise visualize) what would happen to the available memory if each new relocation requires a contiguous section of memory that is 2x the size of the previous section. Then do the same thing for 1.5x.

Devise and implement a test to verify your hypothesis.

Hint 1: You can print pointers with the format specifier `%p`.

Hint 2 (spoiler): It probably won't work as you expect.