
REMINDING C: MEMORY ASPECTS

Sebastien Varrette, Frederic Pinel and Pascal Bouvry

<Firstname.Lastname@uni.lu>

Version 1.1

Exercise 1 *Memory allocation*

The objective of this exercise is to expose the cost of the system calls `malloc` and `free`. You will show this with an example program which stores the fibonacci sequence and iterates through it. This program will also serve for exercise 2.

1. Generate and store the fibonacci sequence use a doubly linked list, the one you wrote for TP1.
 - a) Write a function `list *fibo (int n)` which returns the fibonacci sequence of `n` elements.
 - b) Write a function which frees the memory used by the fibonacci sequence.
 - c) Use these 2 functions to write a program that will generate the fibonacci sequence, print the fibonacci sequence, and the total number of even numbers in the fibonacci sequence. This count should not be efficient, but serves as a processing step with the data.
 - d) Test your fibonacci program with `n = 10, 50, 100`. Do you notice any problems ? Correct this problem when `n ≤ 50`.
2. Write another version of the doubly linked list which allocates memory in batch. These batches are called memory pools.
 - a) Define a data structure `mem_pool` for a simple memory pool. This structure must keep track of how many elements are allocated in the pool, etc.
 - b) Write a function which creates a memory pool, `mem_pool *create_pool (int size)`.
 - c) Write a function which "allocates" an element from the memory pool, `elm *pool_malloc (mem_pool *mp)`. This function returns `NULL` when there is not any space left in the pool.
 - d) Modify `add_tail()`, `del_list()` to use the memory pool.
3. Measure the execution time of both versions. You may need to use a large value for `n`, to be able to measure differences. Which version is faster ? Why ?
4. Explain how you would manage single element deletes in the memory pool ? (you do not need to provide code)

Exercise 2 *Discovering the cache line size*

The objective of this exercise is to understand the performance impact of cache memory. It is quite easy to expose the benefits of cache memory, since its mechanism is well documented. Yet, to benefit from it, the programmer must organize his/her memory access correctly. This exercise is meant to highlight this.

Simple programs can be used to discover the cache characteristics of your processor. One can write a program to discover the size of cache memory. Here, we will focus on another characteristic of cache memory: the cache line.

1. Write a program which allows you to infer the size of the cache line for your processor. Verify your findings with the specification from the processor vendor, or some software tool.

Hints:

- allocate a large contiguous memory area (such as an array)
- read this array in sequential mode, cell by cell, so as to maximize the cache hit ratio
- introduce gaps in the sequential read (make sure that you read the same amount of memory, so as to compare equivalent loads)
- by increasing the gap, discover the cache line size.
- the `valgrind` tool supports cache monitoring with the tool `cachegrind`.

Exercise 3 *Cache*

This exercise is an extension of the previous one and is meant again to expose the impact of the spatial locality of reference. To do so, we must compare a program which stores and accesses data in a cache efficient way, and another that does not. Assume the memory pool version of the fibonacci generator is the first program. If we allocate one big chunk of memory, store consecutive elements in consecutive memory location, then iterating through this list achieves good spatial locality of reference.

1. Write a function `void dump (list *)` which displays each element of the list, including the pointer values, for previous, next and current elements. Use this function to display the memory layout of the memory pool version of the fibonacci sequence.
2. Modify the fibonacci program such that consecutive calls to `pool_malloc` do not return consecutive memory addresses, but leave a gap.
3. Compare the execution times of the memory pool version with the different gap values (including without a gap). Discuss the results, in relation to the caches of your processor. What size would you recommend the gap to be, in order to decrease the cache hit rate ?