Exercise sheet 12 – Memory management

Goals:

- Memory management

Exercise 12.1: Memory allocation strategies

Consider a main memory that contains the following free partitions: 10 KiB, 4 KiB, 20 KiB, 18 KiB, 7 KiB, 9 KiB, 12 KiB, and 15 KiB. Between the free partitions there are used partitions of an unknown size.

(a) Visualise the situation: Draw a sketch of the memory view.

Now, the following subsequent requests for memory partitions occur: 12 KiB, 10 KiB, and 9 KiB.

(b) Show the results within your memory sketch when *first fit* is used.

(c) Show the results within your memory sketch when *best fit* is used.

(d) Show the results within your memory sketch when *next fit* is used.

(e) Show the results within your memory sketch when *worst fit* is used.

Proposal for solution:

OS exercise: Memory allocation strategies

	a) Memory	b) first fit	c) best fit	d) next fit	e) worst fit
15 KiB	free				9 KiB
	acquired				
12 KiB	free		12 KiB		
	acquired				
9 KiB	free		9 KiB	9 KiB	
	acquired				
7 KiB	free				
	acquired				
18 KiB	free	9 KiB		10 KiB	10 KiB
	acquired				
20 KiB	free	12 KiB		12 KiB	12 KiB
	acquired				
4 KiB	free				
	acquired				
10 KiB	free	10 KiB	10 KiB		
	acquired				

Exercise 12.2: Memory management programming and OS memory mechanism

(a) How and where can a process acquire (allocate) main memory in C (there are two possibilities)?

Proposal for solution:

- Stack memory: by declaring variables inside functions.
- Heap memory: by using `malloc()` or `calloc()`

(b) How can a process release memory (distinct two possibilities)?

Proposal for solution:

- Stack memory: when the function goes out of scope, all variables that are declared inside the function goes out of scope and are automatically released.
- Heap memory: by using `free()`.

(c) Write a small C program that shows how the main memory acquire (allocation) and release works (distinct two possibilities).

Proposal for solution:

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

void function_with_stack_data(int num_bytes)
{
    char stack_array[num_bytes];

    //when the function is left, stack_array is automatically deleted from stack.
}

void function_with_heap_data(int num_bytes)
{
    // acquire (allocate) heap memory
    char* heap_array = (char*)malloc(sizeof(char)*num_bytes);

    //free the heap memory
    free(heap_array);
    heap_array = NULL;
}

int main(int argc, char** argv)
{
    if(argc < 2){
        printf("Error: no size is specified\n");
        printf("Usage: %s <size>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    int num_bytes = atoi(argv[1]);
    if(num_bytes <= 0){
        printf("Error: size has to be > 0\n");
        exit(EXIT_FAILURE);
    }

```

```c
    function_with_stack_data(num_bytes);
    function_with_heap_data(num_bytes);

    printf("The very end.\n");


    return 0;
}
```

(d) Is the operating system involved when acquire (allocation) and release of main memory is done by a process (distinct two possibilities)?

Proposal for solution:

- Stack memory: No, this is done by manipulating the stack pointer: decrease means acquire memory, increase means release memory.
- Heap memory: Yes, `malloc()` and `free()` involves SVCs and are handled by the OS.

(e) Is the operating system involved when the process writes data into the main memory (distinct two possibilities)?

Proposal for solution:

- Stack memory: No, the process (machine code) directly writes into the main memory using virtual addresses.
- Heap memory: No, the process (machine code) directly writes into the main memory using virtual addresses.

Exercise 12.3: Memory management

(a) What is a cache?

Proposal for solution: A cache is an associative memory, which stores data of a request or computation. The next time this data is needed, it is read from the cache instead, which is much faster.

(b) In the context of caching: What is a hit and what is a fault?

Proposal for solution: A hit occurs if the requested data is already in the cache. A fault occurs if the requested data is not in the cache (usually at the first access).

(c) What is position independent code (PIC)?

Proposal for solution: It means, that the code can be loaded on any location on the main memory.

(d) Can fragmentation problems be solved by variable partition sizes (each process can choose its own required partition size)?

Proposal for solution: No, the problem is somewhat alleviated but it still exists, because the number of processes is limited by the partitions and there will be gaps formed automatically.

(e) Can the operating system protect the memory of a process against others without the help of the CPU?

Proposal for solution: No, not really. Because memory access is done directly with CPU instructions.

(f) What is swapping?

Proposal for solution: Swap out the memory or parts of the memory (with virtual memory) of a process that is not actively running to the disk.

(g) Does swapping improve the performance?

Proposal for solution: It depends! Because transferring data from memory to the disk and vice versa takes a lot of time. The overall performance may go down. But it helps to execute a lot of processes where some can be swapped. And usually, a lot of processes are required to fully utilise the CPU.

(h) What happens on a page fault and how is the operating system involved?

Proposal for solution: The operating system has to perform a swap in first. More details: See `os_12_memory_management_handout.pdf` slide 35.

(i) Does thrashing help to improve the systems performance?

Proposal for solution: No, thrashing happens when the operating system is almost fully busy with swap in/swap out.

(j) Is it right, that the virtual memory has to be smaller than the real memory, because the operating systems also needs some memory?

Proposal for solution: No, the virtual memory has the theoretically available size and is usually greater than the real memory.

(k) What is a virtual address space?

Proposal for solution: The virtual address space consists of all virtual addresses.

(l) What is a page table and how is a virtual address transformed into a real address?

Proposal for solution: A page table contains the information which page (virtual memory) is mapped on which frame (real memory).

(m) Consider a system with virtual memory, MMU, and swapping. Is it required that the code of the executables (ELFs) is build with position independent code?

Proposal for solution: No, because every process has its own virtual memory where it can always use the same virtual addresses.

(n) If you have a system without a MMU, is it possible to have threads?

Proposal for solution: Yes, it is possible to have threads without a MMU at Linux. Systems like Arduinos (microcontrollers) with an Atmega or Cortex M processors also can have threads—but these small microcontrollers does not have an MMU.

(o) If you have a system without a MMU, is it possible to have “real” processes with all that security?

Proposal for solution: No, without a MMU it is not really possible to have processes that are very fast and very secure, because no MMU means no virtual memory.