NTNU
Kunnskap for en bedre verden

Department of Computer Science

TDT4186 - Operating Systems

Exercise 3

Memory Management

Professor:
Di Liu
Teaching Assistant:
Roman K. Brunner
Student Assistants:
Eik Hvattum Røgeberg
Halvor Linder Henriksen
Ole Ludvig Hozman

Handout: 06.02.2023

Introduction

This week's exercise is all about memory management. We are going to focus on different aspects, such as the methods the OS has to ensure security through isolation, the illusion the OS creates and how this all looks from the perspective of a user process.

1 Address Space of a Process

1. Memry Organization in a User-Space Process:
   (a) How is memory usually organized in a user-space process?
   (b) Does memory organisation within a process change if we run on a uniprogramming or a multiprogramming OS?
   (c) Why do we organize memory within processes like that? What are the advantages or disadvantages of organizing memory and address spaces in that particular way?
2. Given the following virtual addresses, select to which of the three parts of the process memory they most likely belong to. **Note:** Knowing that can be very useful when debugging, as we know that we might need to look at a local variable going out of scope or checking variables that were allocated or if we are trying to write to a constant value.

   There is no exact solution to that question, so don't expect such a question in the exam.

   - 0x0000000100003f98
     - ☐ Code
     - ☐ Heap
     - ☐ Stack
     - ☐ Stack or Code
     - ☐ Heap or Code
     - ☐ Stack or Heap
   - 0x7fffc7e18c2bcf98
     - ☐ Code
     - ☐ Heap
     - ☐ Stack
     - ☐ Stack or Code
     - ☐ Heap or Code
     - ☐ Stack or Heap
   - 0x000055e66a841004
     - ☐ Code
     - ☐ Heap
     - ☐ Stack
     - ☐ Stack or Code
     - ☐ Heap or Code
     - ☐ Stack or Heap
3. Let's see the stack at work: Write a small program that calls a function recursively, allocates a variable (remember: this variable will be put on the stack) and prints out the address of the variable. How do you expect the addresses that you print out to change, the deeper in the callstack we go?
4. Which type of addresses are seen by the user-space program? Do the addresses that are seen by the kernel differ from the addresses by the user-space program?

2 Address Space Organization by the OS

For the following questions, take the position of the OS. So you might have to handle multiple processes, with multiple address spaces, and the physical addresses. Also you are responsible to handle the physical address space and ensure all the guarantees that we expect from the OS with respect to process isolation and similar.

1. We discussed that the addresses within a single process follow a certain inner logic. When we look at the corresponding physical addresses, do they also have to follow the same logic? Please explain your solution
2. When we start a process, we need to load the code and other static content into memory. How does the compiler know which addresses it needs to put down for the instructions, e.g. at which address a function starts?
3. Given the different different methods (fixed and variable partition) to allocate memory for a process, which problems do arise? How can they be circumvented?
4. Since modern hardware offers memory management units, capable of checking memory accesses, and fast address translation, why don't we just stitch together larger virtual blocks out of smaller physical blocks? Give at least three reasons why this might not be desireable.
5. Does Segmentation help with external fragmentation?
6. Can you suggest a solution in between variable and fixed partition? E.g. trying to combine the flexibility of the variable partitioning approach with the predictability of the fixed approach?