CS2106 Operating Systems
Tutorial 8
Disjoint Memory Allocation

1. [Walkthrough of Paging/Segmentation/Hybrid Schemes] Let us use a tiny example to understand the various disjoint memory schemes. For simplicity, we assume there are only two types of memory usage in a program: `text` (instruction) and `data` (global variables).

   The following questions assume that program **P** has:
   - 6 instructions, each fitting in a processor word (instruction words #1 to #6)
   - 5 data words (data words #1 to #5)

   a. (Paging) Given the following parameters:

      - Page Size = Frame Size = 4 words
      - Largest logical memory size = 16 words
      - Number of physical memory frames = 16

      Assuming that **P**'s data region is placed right after the instruction region in the logical memory space, fill in the following page table. Use frames 5, 2, 10, 9 for pages 0, 1, 2, 3 respectively (note: you may not need all frames). Indicate the value of the valid bit for all page table entries.

Page#	**Frame#**	**Valid**
0		
1		
2		
3		

      Find out the logical address and the corresponding physical address for the following actions taken by the processor.

Processor Action	**Logical Address**	**Physical Address**
Fetch the 1st Instruction		
Load the 2nd Data word		
Load the 3rd Data word		
Load the 6th Data word (This is intentionally outside of the range)		

   b. (Segmentation) Assuming `text` and `data` are stored in segments 0 and 1 respectively, fill in the following segment table. Use addresses 50 and 23 as the starting addresses for the two segments, respectively.

Segment#	**Base Address**	**Limit**

0		
1		

Similar to (a), find out the logical address and physical address for the following processor actions:

Processor Action	Logical Address	Physical Address
Fetch the 1st Instruction		
Load the 2nd Data word		
Load the 3rd Data word		
Load the 6th Data word (This is intentionally outside of the range)		

c. (Segmentation with Paging) Assuming the following parameters:

- Page Size = Frame Size = 4 words
- Number of physical memory frames = 16
- Maximum size of each segment = 4 pages

Furthermore, assume the pages from the code segment are allocated to frames 7, 4, 1, and 2 and data segment allocated to frames 9, 3, 14, and 6 (note that you may not need all of them).

Draw the segment and page tables for this setup, then fill in the processor action table. For the logical addresses, use the notation of <segment id, page number, offset>.

Processor Action	Logical Address	Physical Address
Fetch the 1st Instruction		
Load the 2nd Data word		
Load the 3rd Data word		
Load the 6th Data word (This is intentionally outside of the range)		

2. [Dynamic Allocation, adapted from (SGG)] It is possible for a program to dynamically allocate (i.e., enlarge the memory usage) during runtime. For example, the system call **`malloc()`** in C or **`new`** in Java/C++ can enlarge the **heap region** of process memory. Discuss the OS mechanisms needed to support dynamic allocation in the following schemes:
    a. Contiguous memory allocation (both fixed and dynamic size partitioning)
    b. Pure Paging
    c. Pure Segmentation

3. [Paging and TLB] In this question, we attempt to quantify the benefit of using TLB by looking at the memory access time. Suppose the system uses the paging scheme with the page tables entirely stored in physical memory (DRAM). The page size is 4KB, and the logical addresses are 32-bit long. Answer the following:

    a. If accessing DRAM takes 50ns (nanoseconds), what is the latency of accessing a global variable of type `char`?

    b. Assuming the system uses a TLB and 75% of all page-table references hit in the TLB. What is the average memory access time? You can assume that looking up a page table entry in TLB takes negligible time.

    c. How many entries does a TLB need to have to achieve a hit ratio of 75%? Assume the program generates logical memory addresses uniformly at random. Do you think a TLB in an actual machine is this large? If not, then how is it possible to achieve a high TLB-hit rate?

4. Calculate the average amount of memory capacity lost to internal fragmentation in a system that uses the Buddy allocator. Can the Buddy allocator suffer from external fragmentation?

For your own exploration

1. [Adapted from AY1516 Exam Paper] As discussed in lecture, we can protect a memory page by adding permission bits to the page table entry (PTE). Suppose we add 3 access right bits: {**R**: Readable, **W**: Writable, **X**: Executable} to each PTE. When a processor instruction violates the access permission of a page, OS will be invoked to handle the problem. We can utilize this behavior to implement the **copy-on-write** mechanism. Also discussed in lecture, copy-on-write can be used to reduce the memory usage during the `fork()` system call, by allowing memory pages to be shared between parent and child process until modified. In this question, we will explore these ideas using a simplified example.

   Suppose process P has only 3 valid page table entries:

Page No	Frame No	R	W	X
0	7	1	0	1
1	2	1	1	0
2	5	1	0	0
3 …. N-1	---	---	---	---

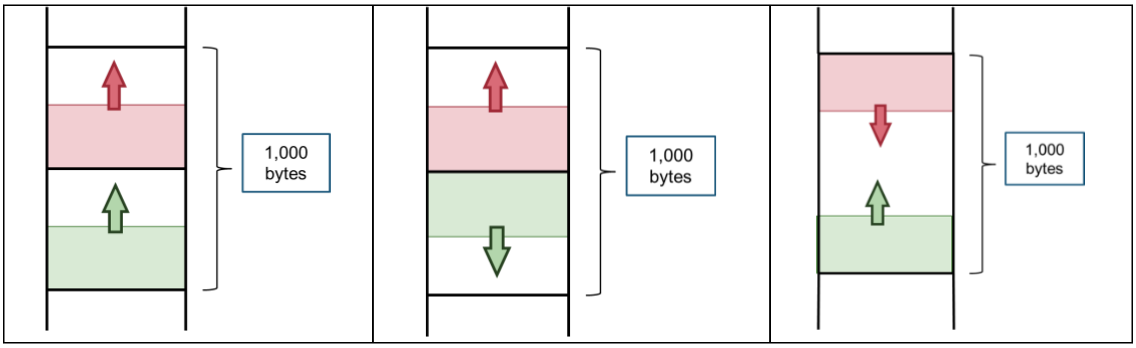
   a. Give the page table entries for the **child process** after P executes **`fork()`**. If you need to use any new frame numbers, use them in this order {6, 0, 3, 4, 1}

   Using the above scenario, explain the following:

   b. How does the OS know that copy-on-write is needed?

   c. What are the steps required to handle copy-on-write? Indicate any additional information that OS need to maintain. Show the affected PTE(s) for the child process afterwards.

2. (Growing/Shrinking of 2 Regions) This question looks at the problem of maximizing the ***logical memory space*** for two growing/shrinking regions (e.g., Stack and Heap regions). Suppose we have only a piece of 1,000 bytes of memory space. Which of the following placements of the stack and heap regions is the best choice? The arrows represent the growing direction of the regions. Briefly justify your choice:



Design One **Design Two** **Design Three**