

**CS2106 Operating Systems**  
Semester 2 2020/2021

Week of 22<sup>nd</sup> March, 2021  
Tutorial 8 **Suggested Solutions**  
**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 1 <sup>st</sup> Instruction		
Load the 2 <sup>nd</sup> Data word		
Load the 3 <sup>rd</sup> Data word		
Load the 6 <sup>th</sup> 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 1 <sup>st</sup> Instruction		
Load the 2 <sup>nd</sup> Data word		
Load the 3 <sup>rd</sup> Data word		
Load the 6 <sup>th</sup> 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 1 <sup>st</sup> Instruction		
Load the 2 <sup>nd</sup> Data word		
Load the 3 <sup>rd</sup> Data word		
Load the 6 <sup>th</sup> Data word (This is intentionally outside of the range)		

**ANS:**

- a.  
Memory layout:

Page 0	0	1 <sup>st</sup> Instruction
	1	2 <sup>nd</sup> Instruction
	2	3 <sup>rd</sup> Instruction
	3	4 <sup>th</sup> Instruction
Page 1	4	5 <sup>th</sup> Instruction
	5	6 <sup>th</sup> Instruction
	6	1 <sup>st</sup> Data word
	7	2 <sup>nd</sup> Data word
Page 2	8	3 <sup>rd</sup> Data word
	9	4 <sup>th</sup> Data word
	10	5 <sup>th</sup> Data word
	11	<empty>
Page 3	12	<empty>
	13	<empty>
	14	<empty>
	15	<empty>

Page#	Frame#	Valid
0	5	T
1	2	T
2	10	T
3	--	F

Processor Action	Logical Address	Physical Address
Fetch the 1 <sup>st</sup> Instruction	0	$5 \times 4 + 0 = 20$
Load the 2 <sup>nd</sup> Data word	7	$2 \times 4 + 3 = 11$
Load the 3 <sup>rd</sup> Data word	8	$10 \times 4 + 0 = 40$
Load the 6 <sup>th</sup> Data word (This is intentionally outside of the range)	11	$10 \times 4 + 3 = 43$

Note 1: Observe that the consecutive logical addresses may not be consecutive in physical memory (e.g. 2<sup>nd</sup> and 3<sup>rd</sup> Data word).

Note 2: Incorrect memory access is not catchable if it is still within valid page boundary.

b.

Segment#	Base Address	Limit
0	50	6
1	23	5

Processor Action	Logical Address	Physical Address
Fetch the 1 <sup>st</sup> Instruction	<0, 0>	$50 + 0 = 50$
Load the 2 <sup>nd</sup> Data word	<1, 1>	$23 + 1 = 24$
Load the 3 <sup>rd</sup> Data word	<1, 2>	$23 + 2 = 25$

Load the 6 <sup>th</sup> Data word (This is intentionally outside of the range)	<1, 5>	Triggers memory addressing error
--	--------	----------------------------------

- c. Memory layout (code segment on the left, data segment on the right; empty pages not shown):

Page 0	0	1 <sup>st</sup> Instruction	Page 0	0	1 <sup>st</sup> Data Word
	1	2 <sup>nd</sup> Instruction		1	2 <sup>nd</sup> Data Word
	2	3 <sup>rd</sup> Instruction		2	3 <sup>rd</sup> Data Word
	3	4 <sup>th</sup> Instruction		3	4 <sup>th</sup> Data Word
Page 1	4	5 <sup>th</sup> Instruction	Page 1	4	5 <sup>th</sup> Data Word
	5	6 <sup>th</sup> Instruction		5	<empty>
	6	<empty>		6	<empty>
	7	<empty>		7	<empty>

Segment#	Page Limit	Page Table Base
0	2	
1	2	

  

Page#	Frame#	Valid
0	7	T
1	4	T
2	--	F
3	--	F

  

Page#	Frame#	Valid
0	9	T
1	3	T
2	--	F
3	--	F

Processor Action	Logical Address	Physical Address
Fetch the 1 <sup>st</sup> Instruction	<0, 0, 0>	$7 \times 4 + 0 = 28$
Load the 2 <sup>nd</sup> Data word	<1, 0, 1>	$9 \times 4 + 1 = 37$
Load the 3 <sup>rd</sup> Data word	<1, 0, 2>	$9 \times 4 + 2 = 38$
Load the 6 <sup>th</sup> Data word (This is intentionally outside of the range)	<1, 1, 1>	$3 \times 4 + 1 = 13$

Note: As the limit in segment table now refers to the limit of pages in the corresponding page table, memory error that is in page boundary is now "not catchable".

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:
- Contiguous memory allocation (both fixed and dynamic size partitioning)
  - Pure Paging
  - Pure Segmentation

**ANS:**

- It is simpler to have the heap region to be allocated at the end of the logical memory space. Then, we can enlarge the heap region by enlarging the partition allocated to the process.

**Under fixed partitioning:**

- No extra work is needed, as heap region can simply use up the free space (the internal fragmentation) between the partition size and the actual memory size

**Under dynamic partitioning:**

- If the adjacent partition is free:
    - Simple: Simple modify the partition information, i.e. change the length of current partition and shorten the free partition.
  - If adjacent partitions are occupied:
    - More troublesome: The current partition cannot be enlarged. Relocation is required. OS need to look for a large enough free partition to fit the enlarged partition. Once located, the current partition is moved over.
- b. Due to internal fragmentation of the paging scheme, it is possible that the allocation can use the remaining free space in the page. Suppose the allocation overshoot the page boundary, then OS needs to look for a free physical frame  $f$ . Afterward, update the page table by changing the first invalid page table entry from invalid to valid and fill in frame number  $f$ .
- c. Idea is similar to the dynamic partitioning. If there is free memory at the end of the heap segment, then OS can simply update the limit of the segment and reduce the size of the affected free partition. If there is no free memory, then relocation is required. After relocation, both base and limit of the heap segment needs to be updated.
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:
- 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?

**ANS:**

a.  $50\text{ns (access page table)} + 50\text{ns (access actual item)} = 100\text{ns}$

b.  $0.75 * 50\text{ns} + (1-0.75)*100\text{ns} = 62.5\text{ns}$

c. Total number of page table entries =  $2^{(32-12)} = 2^{20} = 1\text{M}$  entries

If we assume the memory access are **uniformly distributed** between all the possible pages, then we need at least  $1\text{M} * 0.75 = 786,432$  entries in the TLB. Using this reasoning, it seems impossible to have a TLB large enough to guarantee a high hit-rate. You may be surprised that TLB is actually quite small in real-world machines (commonly in the range of 32 to 1024 entries) due to the high hardware complexity (associative search is expensive in term of hardware). However, TLB hit ratio is still very high in actual usage (commonly with 99% hit rate!). The main contributing reason is that memory access are **not uniformly distributed**. This can be understood with the aid of locality principle: it is more likely to have repeated memory accesses to same (temporal locality) or different (spatial locality) parts of the same memory page in a time interval rather than uniformly spread accesses.

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?

**ANS:**

For an allocation request of N bytes: min fragmentation = 0% (exact fit,  $N=2^k$ ), max fragmentation =  $\sim 50\%$  ( $N=2^{k-1} + 1$ ) → on average 25% lost to internal fragmentation. Note that internal fragmentation  $> 50\%$  is not possible.

Yes, the buddy allocator can still suffer from external fragmentation. The basic buddy system (as discussed in lecture) cannot fully utilize memory space with size not equal to power-of-2. e.g. If we use basic buddy system to manage a piece of memory of 1100 bytes, we can only take care of the 1024 ( $2^{10}$ ) bytes, leaving the rest ( $1100 - 1024 = 76$  bytes) as **external fragmentation**.