Assignment #8

Problem 8.1: *address spaces in a paging system* (1+1+1 = 3 points)

Consider a operating system that uses paging for memory management with a page size of 2048 B. The logical address space of processes is limited to a maximum of 32 pages. The physical memory has a size of 1024 KiB.

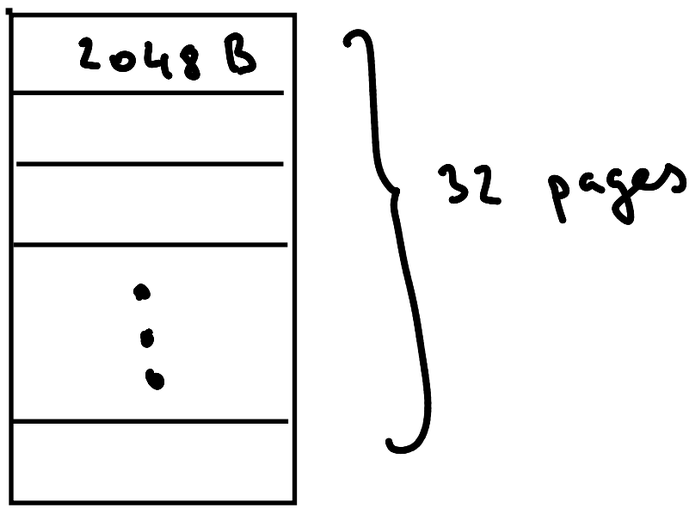
a) How many frames has the physical memory?

b) How many bits has an address in the logical address space and how many bits has an address in the physical address space?

c) How many bits are used for the page number and how many bits are used for the offset within a page?

a) $\text{frames} = \frac{\text{phys. mem.}}{\text{page size}} = \frac{1024 \times 2^{10}\ \text{B}}{2048\ \text{B}} = \underline{\underline{512}}$

b)



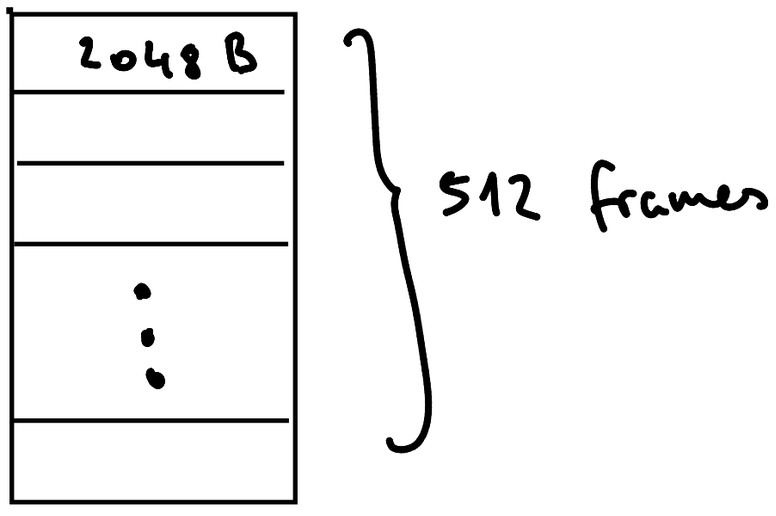
Finding right page:

$32 = 2^5 \Rightarrow 5$ bits

Finding right byte location:

$2048 = 2^{11} \Rightarrow 11$ bits

$\therefore$ Total bits in logical address space = $5 + 11 = \underline{\underline{16\ \text{bits}}}$



Finding frame: $512 = 2^9 \Rightarrow 9$ bits

Finding byte: $2048 = 2^{11} \Rightarrow 11$ bits

$\therefore$ Total bits in physical address space = $9 + 11 = \underline{\underline{20\ \text{bits}}}$

c) Since logical address space of a process is $32 = 2^5$ pages, the logical address needs 5 bits.

Page size is $2048\,B = 2^{11}$, so 11 bits for the offset.

Problem 8.2: *paging and page tables* (1+1+1 = 3 points)

Consider a tiny computer system with a physical memory space of 16 frames. Each 12-bit logical address uses 4 bits for the page number and 8 bits for the offset within a page. There are two processes P_1 and P_2 with the logical address spaces shown below.

Process P_1		
Page	Logical Addresses	Segment
$p_{1,0}$	0x000-0x0FF	text
$p_{1,1}$	0x100-0x1FF	text
$p_{1,2}$	0x200-0x2FF	data
$p_{1,5}$	0x500-0x5FF	heap
$p_{1,6}$	0x600-0x6FF	stack
$p_{1,8}$	0x800-0x8FF	stack

page number

Process P_2		
Page	Logical Addresses	Segment
$p_{2,0}$	0x000-0x0FF	text
$p_{2,1}$	0x100-0x1FF	text
$p_{2,4}$	0x400-0x4FF	data
$p_{2,5}$	0x500-0x5FF	data
$p_{2,6}$	0x600-0x6FF	heap
$p_{2,8}$	0x800-0x8FF	stack

↳ offset

Some pages reside in physical memory as shown in the table below. The notation $p_{i,n}$ refers to page n of the logical address space of process P_i. The OS pages are used by the operating system, unused frames are marked with a dash.

Some pages reside in physical memory as shown in the table below. The notation $p_{i,n}$ refers to page n of the logical address space of process P_i. The OS pages are used by the operating system, unused frames are marked with a dash.

Frame	Physical Addresses	Loaded Page
0	0x000-0x0FF	OS
1	0x100-0x1FF	$p_{2,5}$
2	0x200-0x2FF	-
3	0x300-0x3FF	$p_{1,8}$
4	0x400-0x4FF	$p_{2,4}$
5	0x500-0x5FF	-
6	0x600-0x6FF	$p_{1,1}$
7	0x700-0x7FF	-
8	0x800-0x8FF	$p_{1,0}$
9	0x900-0x9FF	$p_{2,0}$
10	0xA00-0xAFF	-
11	0xB00-0xBFF	$p_{1,6}$
12	0xC00-0xCFF	$p_{2,1}$
13	0xD00-0xDFF	-
14	0xE00-0xEFF	-
15	0xF00-0xFFF	-

a) Write down the page tables for both processes P_1 and P_2. Each page table entry maintains the following additional bits: r = read access, w = write access, x = execute access, d = dirty, v = valid, commonly written in the form `rwxdv` if all bits are set or as `rw--v` if only the r, w, and v bits are set. Assume that all writable pages are dirty.

b) The CPU executes process P_1 and the machine instructions modify a global variable and a dynamically allocated string. A context switch occurs and the CPU executes process P_2, which performs a function call that allocates and initializes 16 bytes on the heap.

   Write down the content of the page tables after all write operations and initializations have been performed. If a page fault occurs use the first free physical frame to load the page.

c) The processes P_1 and P_2 establish a shared memory page to exchange data. The shared page appears as $p_{1,4}$ in the logical address space of P_1 and as $p_{2,2}$ in the logical address space of P_2. Show the resulting use of the physical memory frames (i.e., update the table shown above).

a)

P1:

Logical Addresses	Physical Address	Flags
0x000 - 0x0FF	0x800 - 0x8FF	--x-v
0x100 - 0x1FF	0x600 - 0x6FF	--x-v
0x200 - 0x2FF	NULL	rw---
0x500 - 0x5FF	NULL	rw--- ← cannot be dirty (since invalid)
0x600 - 0x6FF	0xB00-0xBFF	rw-dv
0x800 - 0x8FF	0x300-0x3FF	rw-dv

P2:

Logical Addresses	Physical Address	Flags
0x000 - 0x0FF	0x900 - 0x9FF	--x-v
0x100 - 0x1FF	0xC00 - 0xCFF	--x-v
0x400 - 0x4FF	0x400 - 0x4FF	rw-dv
0x500 - 0x5FF	0x100 - 0x1FF	rw-dv
0x600 - 0x6FF	NULL	rw---
0x800 - 0x8FF	NULL	rw---

b) P1:

- All text segments are valid
- Global variable ⇒ data segment e.g. $P_{1,6}$ (valid)
- Heap ⇒ heap segment $P_{1,5}$ (invalid)
  ↳ load into Frame #2

P2:

- All text segments are valid
- Function call requires a stack $P_{2,8}$ (invalid)
  ↳ load into Frame #5
- Heap ⇒ heap segment $P_{2,6}$ (invalid)
  ↳ load into Frame #7

P1:

Logical Addresses	Physical Address	Flags
0x000 - 0x0FF	0x800 - 0x8FF	--x-v
0x100 - 0x1FF	0x600 - 0x6FF	--x-v
0x200 - 0x2FF	NULL	rw---
0x500 - 0x5FF	0x200 - 0x2FF	rw-dv
0x600 - 0x6FF	0xB00 - 0xBFF	rw-dv
0x800 - 0x8FF	0x300 - 0x3FF	rw-dv

P2:

Logical Addresses	Physical Address	Flags
0x000 - 0x0FF	0x900 - 0x9FF	--x-v
0x100 - 0x1FF	0xC00 - 0xCFF	--x-v
0x400 - 0x4FF	0x400 - 0x4FF	rw-dv
0x500 - 0x5FF	0x100 - 0x1FF	rw-dv
0x600 - 0x6FF	0x700 - 0x7FF	rw-dv
0x800 - 0x8FF	0x500 - 0x5FF	rw-dv

Physicall Address Space (changes only):

Frame	Physicel Address	Loaded Page
⋮	⋮	⋮
2	0x200 - 0x2FF	$P_{1,5}$
⋮	⋮	⋮
5	0x500 - 0x5FF	$P_{2,8}$
⋮	⋮	⋮
7	0x700 - 0x7FF	$P_{2,6}$
⋮	⋮	⋮

c) Assuming this is a continuation of b):

Next available frame = 10, so:

Physicall Address Space (changes only):

Frame	Physicel Address	Loaded Page
⋮	⋮	⋮
2	0x200 - 0x2FF	$P_{1,5}$
⋮	⋮	⋮
5	0x500 - 0x5FF	$P_{2,8}$
⋮	⋮	⋮
7	0x700 - 0x7FF	$P_{2,6}$
⋮	⋮	⋮
10	0xA00 - 0xAFF	$P_{1,4}$ and $P_{2,2}$
⋮	⋮	