Tut 7

Memory management

Hongfan Mu

1. Describe two differences between logical and physical addresses.

Logical and physical addresses

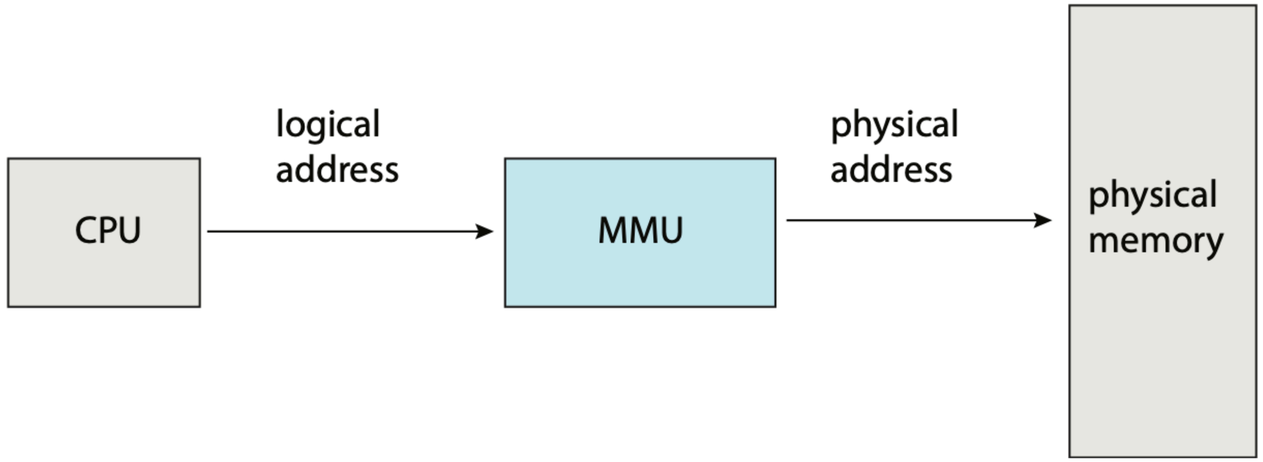


Figure 9.4 Memory management unit (MMU).

Logical address: An address generated by the CPU.

Physical address: An address seen by the memory unit—that is, the one loaded into the memory-address register of the memory.

1. **Describe two differences between logical and physical addresses.**

- A logical address does not refer to an actual existing address; rather, it refers to an abstract address in an abstract address space. Contrast this with a physical address that refers to an actual physical address in memory.
- A logical address is generated by the CPU and is translated in to a physical address by the memory management unit (MMU) (when address binding occurs at execution time). Therefore, physical addresses are generated by the MMU.

2. **Consider a system in which a program can be separated into two parts: code and data. The CPU knows whether it wants an instruction (instruction fetch) or data(data fetch or store). Therefore, two base–limit register pairs are provided: one for instructions and one for data. The instruction base–limit register pair is automatically read-only, so programs can be shared among different users. Discuss the advantages and disadvantages of this scheme.**

- The major advantage of this scheme is that it is an effective mechanism for code and data sharing. For example, only one copy of an editor or a compiler code needs to be kept in memory, and this code can be shared by all processes needing access to the editor or compiler code.
- Another advantage is protection of code against erroneous modification.
- The only disadvantage is that the code and data must be separated, which is usually adhered to in a compiler-generated code.

.

3. Why are page/frame sizes always powers of 2?

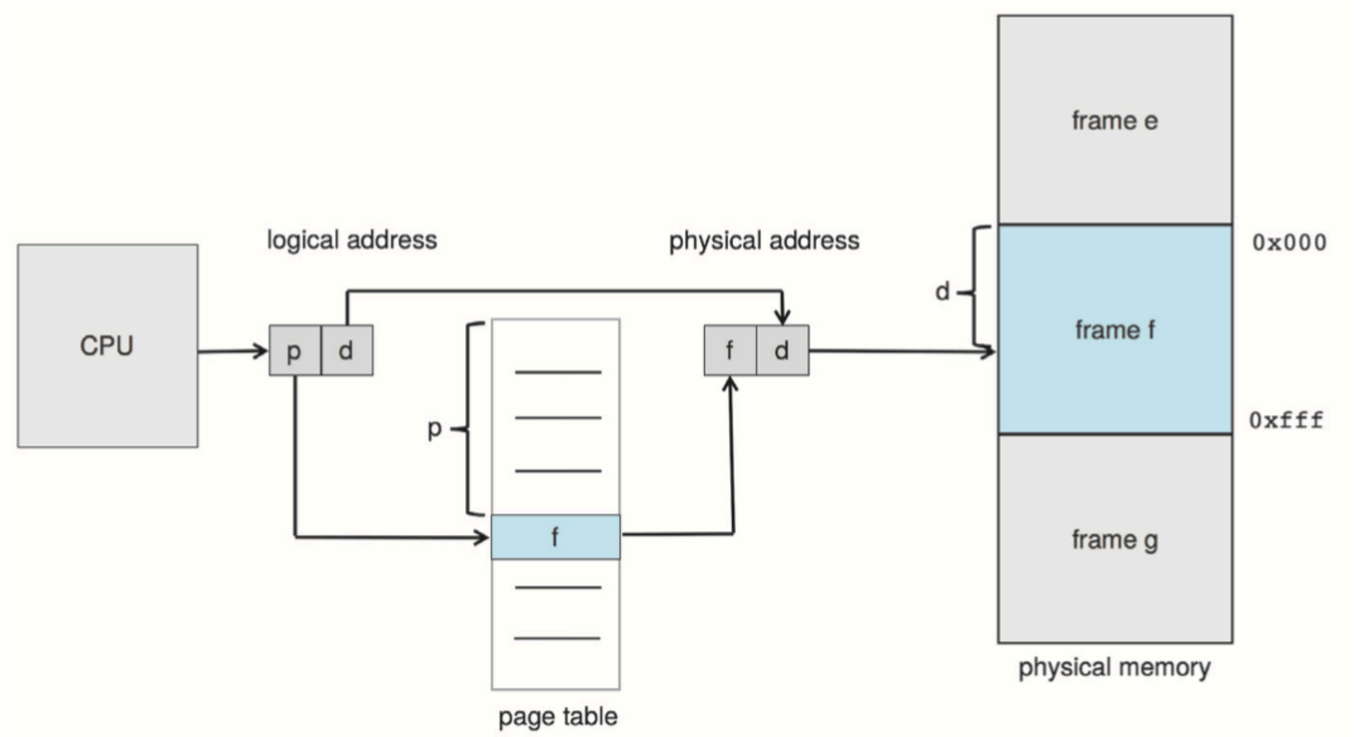


Figure 9.8 Paging hardware.

- Recall that paging is implemented by breaking up an address into a page and offset number.
- It is most efficient to break the address into X page bits and Y offset bits, the offset represents the position of an octet from the start of the page. Since both the page size and frame size are the same, translating a logical address to a physical address only requires the replacement of the page number by the frame number (a concatenation operation).
- Because each bit position represents a power of 2, splitting an address between bits results in a page size that is a power of 2.

4. **Consider a logical address space of eight pages of 1024 bytes each, mapped onto a physical memory of 32 frames.**

- How many bits are there in the logical address? Identify in the logical address the bits used as an offset and the bits used as the page number.

  Logical address: 13 bits (8 pages X 1024 bytes/page = 8196 bytes –> requires 2^{13} addresses)

  Bits for offset: 3 bits identify page number (2^3= 8 pages), bits 0 to 9 (10 bits (13-3) bits for offset into 1024, 2^{10} byte page)

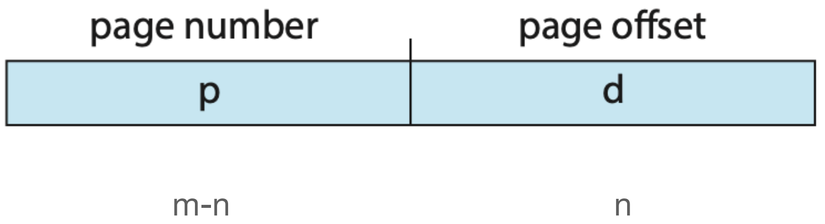
- How many bits are there in the physical address? Identify in the physical address the bits
  used as an offset and the bits used as the frame number.

  Physical address: 15 bits (32 frames X 1024 byes/frame = 32768 bytes, requires 2^{15} addresses).

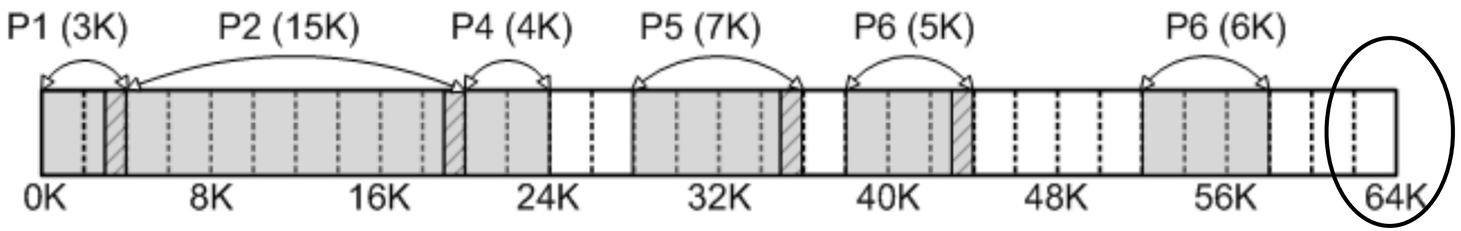
  Bits for offset: 5 bits identify frame number (2^5= 32 frames), bits 0 to 9 (10 bits for offset into 1024, 2^{10} byte frame)

The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called frames and breaking logical memory into blocks of the same size called pages.

Every address generated by the CPU is divided into two parts: a **page number** (p) and a **page offset** (d):



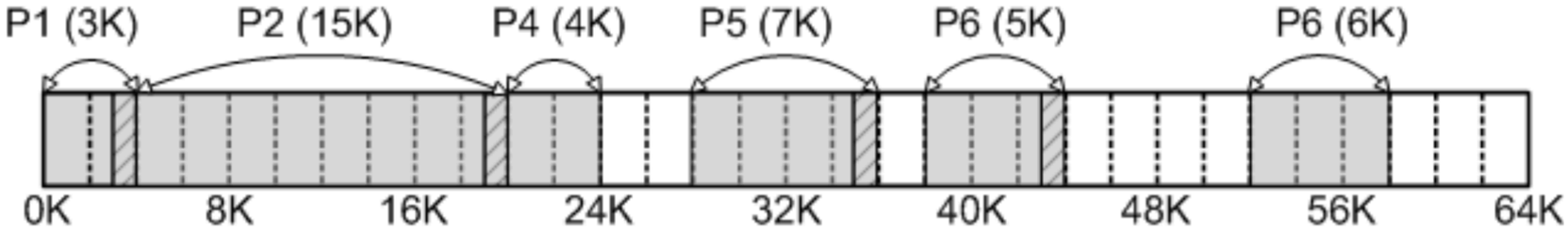
5. Consider a computer using dynamic contiguous allocation, where each allocated block is a multiple of 2kb. Consider the following situation:

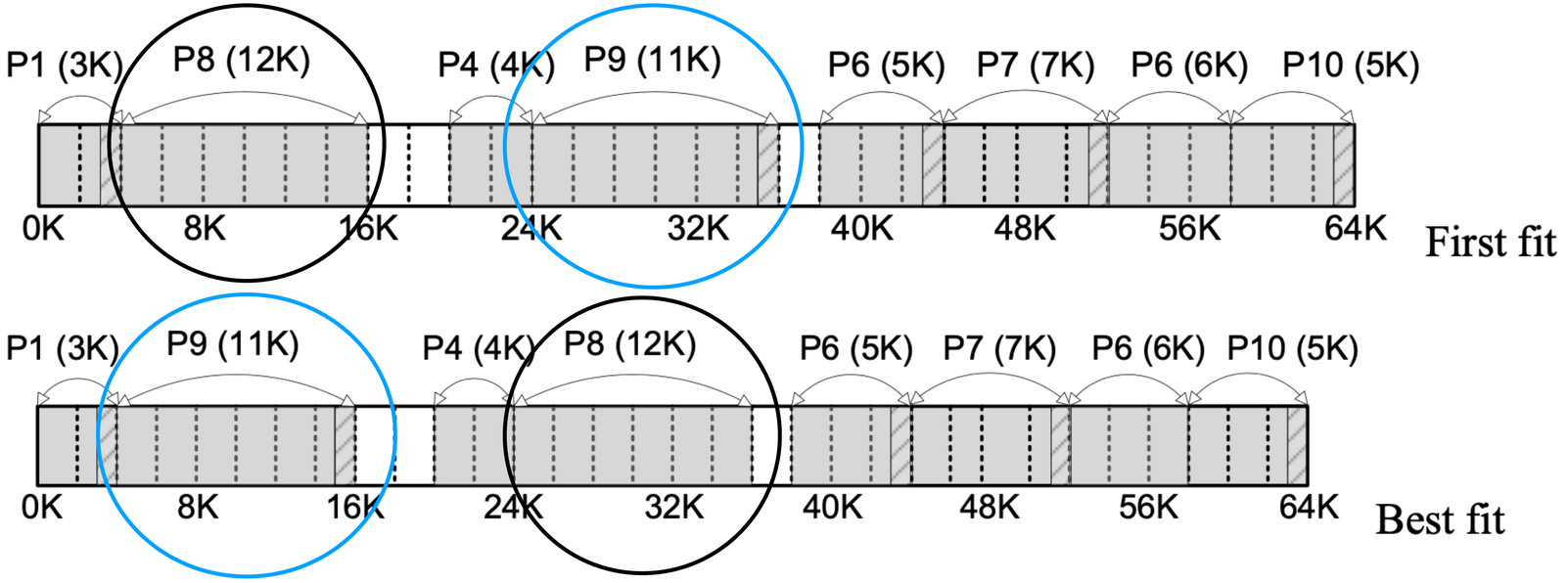


Consider the following sequence of allocating/deallocating

- Allocate process P7, size 7kb
- Deallocate process P2
- Deallocate process P5
- Allocate process P8, size 12 kb
- Allocate process P9, size 11 kb
- Allocate process P10, size 5k

a) Draw pictures describing the final allocation using first-fit and best-fit allocation algorithms.





Consider the following sequence of allocating/deallocating

- Allocate process P7, size 7kb
- Deallocate process P2
- Deallocate process P5
- Allocate process P8, size 12 kb
- Allocate process P9, size 11 kb
- Allocate process P10, size 5k

Memory Allocation

- **First-fit**. Allocate the first hole that is big enough.Searching can start either at the beginning of the set of holes or at the location where the previous first-fit search ended. We can stop searching as soon as we find a free hole that is large enough.

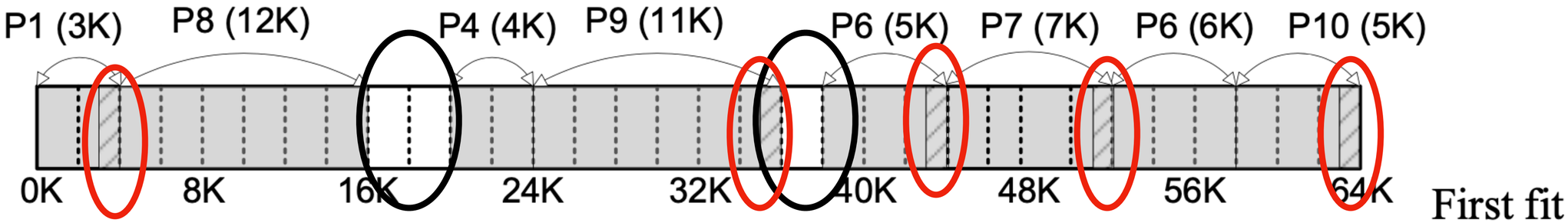
- **Best-fit**. Allocate the smallest hole that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest leftover hole.

- **Worst-fit**. Allocate the largest hole. Again, we must search the entire list, unless it is sorted by size. This strategy produces the largest leftover hole, which may be more useful than the smaller leftover hole from a best-fit approach.

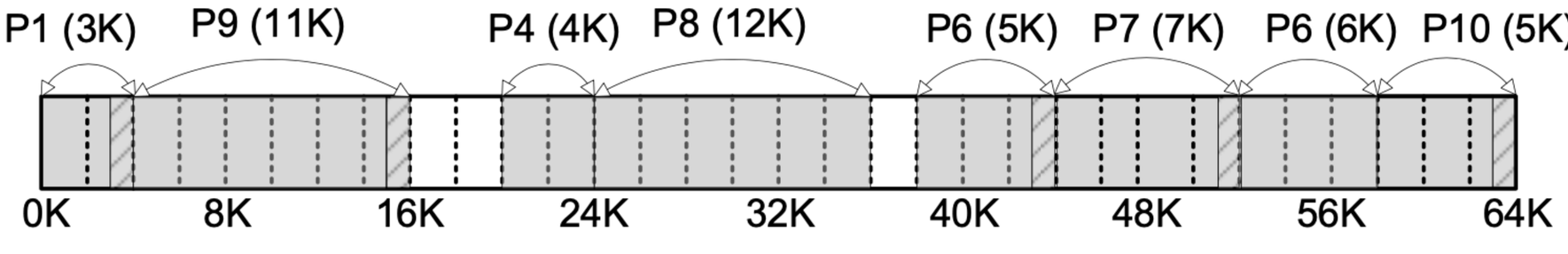
b) What is the internal fragmentation at the end? What about external fragmentation?

Internal Fragmentation: 5K (5 processes have 1 k unused)

External: 6K – Three holes of 2k unused by processes



First fit



Best fit

External fragmentation & Internal fragmentation

External fragmentation: As processes are loaded and removed from memory, the free memory space is broken into little pieces. External fragmentation exists when there is enough total memory space to satisfy a request but the available spaces are not contiguous: storage is fragmented into a large number of small holes.

Internal fragmentation : the memory allocated to a process may be slightly larger than the requested memory. The difference between these two numbers is internal fragmentation —unused memory that is internal to a partition.

6. Consider a memory containing 16 blocks, with the allocation described by the following Inverted Page Table (An entry contains Process number and block number within the process:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P1,4	P2,2	P2,4	P1,1	P1,0	P2,3	P3,1	P1,3	P2,1		P1,5	P3,0		P2,0	P1,2	

From the above Inverted Page Table, complete the Page tables for Processes P1 and P2 below.

	P1	
	Frame #	**Valid (1/0)**
0	**4**	1
1	**3**	1
2	**14**	1
3	**7**	1
4	**0**	1
5	**10**	1

P2	
Frame #	**Valid (1/0)**
13	**1**
8	**1**
1	**1**
5	**1**
2	**1**
-	**0**

Note that P2 does not use as many pages as P1. In fact, if you examine process P3, you will see that only 2 frames have been allocated to this process.

7. Consider a computer with 24 bit logical address space, 64MB of physical memory, 1kb pages, with each page table entry taking 4 bytes. Describe the format of logical address for this computer, by answering the following questions:

- a) **Draw a bar, indicating how many bits define the offset and how many bits define the logical page #.**

1kb = 1024 bytes = 2^10

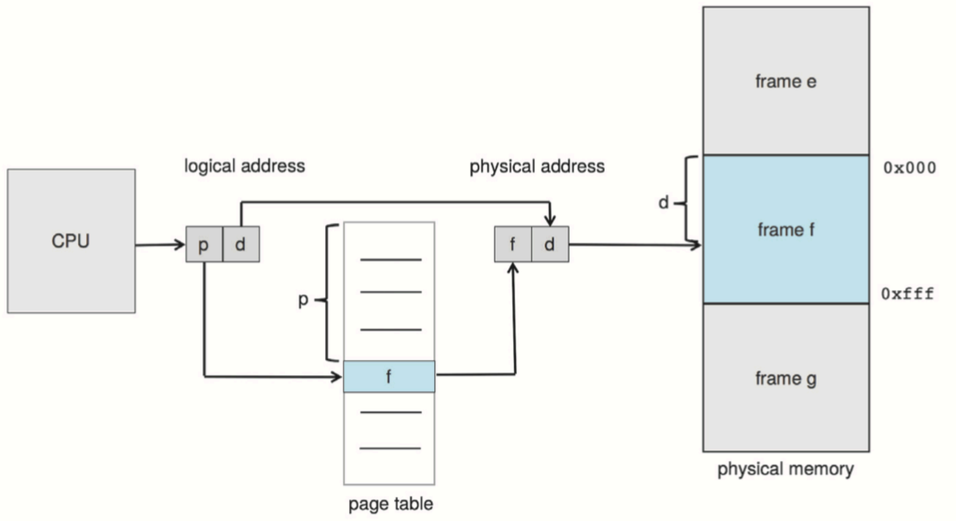


Figure 9.8 Paging hardware.

Total: 24 bit logical address space

Total(24) = #pages * size of each page

= #pages *(size of entries(4) * #entries)

b) Do you need hierarchical page tables? In not, explain why. If yes, separate the bits for the logical page # into the corresponding parts (in the answer for a)). How many levels of hierarchy do you need?

No, hierarchical pages are not necessary. A page table will take $2^{14} * 4 = 64$ Kbytes which represents 0.1 % of physical memory space or 0.4 % of the process logical space

.

c) Assume you have decided to use Inverted Page Tables (IPT) instead. How big should the IPT be? (assume that 2 bytes are sufficient to store PID, you have to answer for yourself how many bytes you need to store the logical page #).

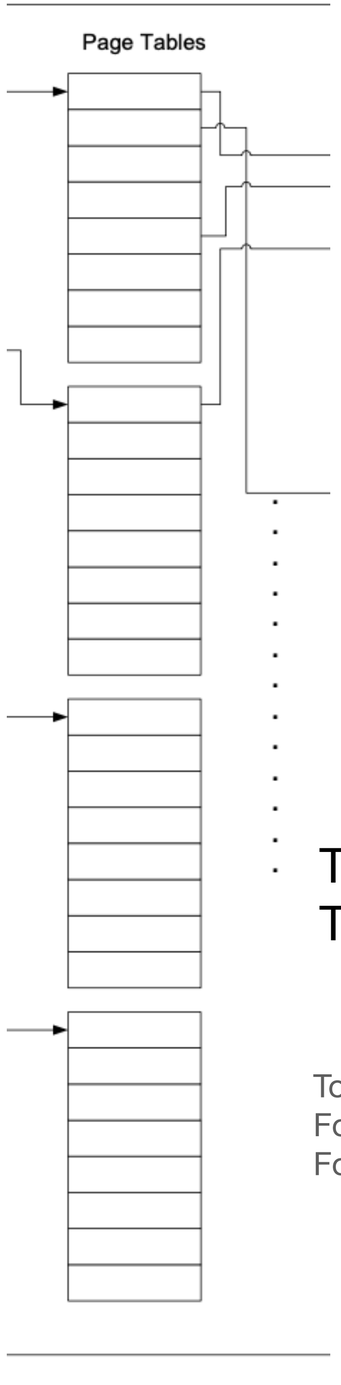
For a 64 Mbytes address and a 1kb page size, 10 bits are used for offset and 16 bits (2 bytes) are used for the frame number, giving 2^{16} = 64 K (64X1024) entries in the IPT.

With a PID of two bytes, and allocating 2 bytes for the logical page number (14 bits, + 2 control bits, such as the dirty bit), i.e. 4 bytes per entry, the size of the IPT would be 64 K X 4 bytes = 256 Kbytes

Alternative: extrapolating from the 4 byte entry in the Page table, two bytes are used for the frame number, and two bytes for control bits (dirty bits, permission bits, etc). Given that the control bits are carried over to the IPT, then an entry in the IPT would contain 2 bytes for the PID, 2 bytes for the logical page number (only 14 bits used), and 2 bytes for control bits, giving 6 bytes for an entry in the IPT. In this case, the size of the IPT would be 64K entries X 6 bytes/entry = 384 Kbytes.

.

8. Consider the following simple memory management system combining segmentation and paging (illustrated in the figure to the right):
A process consists of at most 4 segments, where each segment itself is paged, with pages/frames of size 2kb. The page table of each segment contains 8 entries, each storing reference to the corresponding physical frame. The segment table entries refer to the page tables of the corresponding segments. Because of relatively small size, both segment table and segment page tables are stored directly in the process's PCB (process control block), without occupying another physical frames.



4 page - 8 entries/offset

Total: 24 bit logical address space
Total(24) = #pages * size of each page
= #pages *(size of entries(4) * #entries)

Total 24 bits logical address space - size of pages
For total logical address space:How many pages we have and size of each pages
For each page: how many entries we have and size of each entries

Answer the following questions:

a. What is the maximum size of each segm
16kb (8x 2kb)

b. What is the largest logical address for th
65535 = 2^{16}-1 (16kb x 4 segments, starting

c. Give the format of the logical address.
xx yyy zzzzzzzzzzz
xx – segment #
yyy – page # within the segment
zzzzzzzzzzz(11 bits) – offset within a page

d. Explain how the logical address is transl
physical address.

1. take xx and use it as an index into the s
   get the pointer to the page table of that
2. take yyy and use it as an index in this pa
   frame #fffff
   3. concatenate the offset to the frame n
   physical address fffffzzzzzzzzzzz