Memory Management

Week 09 – Tutorial

Implementation Issues &

Segmentation

Problem 3.38 (1/2)

- Consider the following two-dimensional array:
  int X[64][64];
- Suppose that a system has **four page frames** and each frame is **128 words** (an integer occupies one word). Programs that manipulate the X array fit into **exactly one page** and always occupy **page 0**. The data are swapped in and out of the other three frames. The X array is stored in row-major order (i.e., X[0][1] follows X[0][0] in memory). Which of the two code fragments shown below will generate the lowest number of page faults? Explain and compute the total number of page faults

Problem 3.38 (2/2)

- Fragment A:

```
for (int j = 0; j < 64; j++)
    for (int i = 0; i < 64; i++)
    X[i][j] = 0;
```

- Fragment B:

```
for (int i = 0; i < 64; i++)
    for (int j = 0; j < 64; j++)
    X[i][j] = 0;
```

Problem 3.38 – Solution (1/3)

- Step 1:
  - The fragment B will generate the lowest number of page faults since the code has more spatial locality than Fragment A

Problem 3.38 – Solution (2/3)

- Step 2 (Fragment B):
  - Clearly fragment B initializes the X array elements row-wise: first element X[0][0] is initialized, and then X[0][1] followed by X[0][2] and so on. Thus for each iteration of the outer loop, one page fault occurs for the inner loop.
  - Given that one frame is of 128 words and one row has 64 integers each of which are of one word, the number of rows of the array in one page is 2 (=128/64)
  - As there are 64 rows in total, the number of page faults caused by fragments B would be 32 (=64/2)

Problem 3.38 – Solution (3/3)

- Step 3 (Fragment A):
  - Since a frame is 128 words, one row of the X array occupies half of a page (i.e., 64 words)
  - For each alternate element access of X[i][j], a new page fault will occur as two rows fit in one page
  - The total number of page faults will be 64 × 64/2 = 2,048

Problem 3.41

- A computer provides each process with **65,536 bytes of address space** divided into **pages of 4096 bytes** each. A particular program has **a text size of 32,768 bytes, a data size of 16,386 bytes, and a stack size of 15,870 bytes**. Will this program fit in the machine's address space?
- Suppose that instead of 4096 bytes, the page size were 512 bytes, would it then fit? Each page must contain either text, data, or stack, not a mixture of two or three of them

Problem 3.41 – Solution

- The text is eight pages, the data are five pages, and the stack is four pages. The program does not fit because it needs 17 4096-byte pages
- With a 512-byte page, the situation is different. Here the text is 64 pages, the data are 33 pages, and the stack is 31 pages, for a total of 128 512-byte pages, which fits. With the small page size it is OK, but not with the large one

Problem 3.45

- Explain the difference between internal and external fragmentation
  - Which one occurs in paging systems?
  - Which one occurs in systems using pure segmentation?

Problem 3.45 – Solution (1/5)

- **Internal fragmentation** occurs when the last allocation unit is not full
- **External fragmentation** occurs when space is wasted between two allocation units
- In a paging system, the wasted space in the last page is lost to internal fragmentation
- In a pure segmentation system, some space is invariably lost between the segments. This is due to external fragmentation

Problem 3.45 – Solution (2/5)

Internal Fragmentation	External Fragmentation
When memory is allocated to a process is larger than the memory requested by the process, the amount of memory not used by the process leads to internal fragmentation, this memory cannot be allocated to another process and wasted.	External fragmentation occurs when a process is allocated with memory which is not contagious; sometimes the memory blocks in between theses allocated memory blocks remain unused leading to external fragmentation.

When they occur, …

Problem 3.45 – Solution (3/5)

Internal Fragmentation	External Fragmentation
Occurs in programs or process where the program is divided into same size fixed partitions, in which at least one partition would be smaller than the memory block allocated.	Occurs when programs are allocated exactly the requested amount of memory in multiple variable sized memory blocks.

And to which programs they occur

Problem 3.45 – Solution (4/5)

Internal Fragmentation	External Fragmentation
Can be overcome by allocating multiple variable sized memory blocks, and compact all the blocks into one large block.	Can be reduced by compaction that is moving all the free memory blocks to one place and making it into a large memory block which can be allocated to a process.

How to address

Problem 3.45 – Solution (5/5)

Internal Fragmentation	External Fragmentation
Is usually observed in paging systems where fixed size pages are allocated for a program and the last partition of the program may not require the entire page.	Is seen in segmentation where memory is not allocated contagiously and in fixed partitions, leading to unused memory blocks.

Observed when…

Problem 3.48

- Can you think of any situations where supporting virtual memory would be a bad idea, and what would be gained by not having to support virtual memory? Explain.

Problem 3.48 – Solution (1/2)

- General virtual memory support is not needed when the memory requirements of all applications are well known and controlled. Some examples are:
  - Only a single program that is small enough to fit within the memory is running
  - Multiple programs that fit within the memory and their sizes don't change
  - Smart cards, special-purpose processors (e.g., network processors), and embedded processors

Problem 3.48 – Solution (2/2)

- In these situations, we should always consider the possibility of using more real memory. If the operating system did not have to support virtual memory, the code would be much simpler and smaller
- On the other hand, some ideas from virtual memory may still be profitably exploited, although with different design requirements. For example, program/thread isolation might be paging to flash memory

Problem 3.49

- Virtual memory provides a mechanism for isolating one process from another. What memory management difficulties would be involved in allowing two operating systems to run concurrently?
  - How might these difficulties be addressed?

Problem 3.49 – Solution (1/3)

- This question addresses one aspect of virtual machine support. Recent attempts include Denali, Xen, and VMware. The fundamental hurdle is how to achieve near-native performance, that is, as if the executing operating system had memory to itself

Problem 3.49 – Solution (2/3)

- The problem is how to quickly switch to another operating system and therefore how to deal with the TLB
- Typically, you want to give some number of TLB entries to each kernel and ensure that each kernel operates within its proper virtual memory context

Problem 3.49 – Solution (3/3)

- But sometimes the hardware (e.g., some Intel architectures) wants to handle TLB misses without knowledge of what you are trying to do
- So, you need to either handle the TLB miss in software or provide hardware support for tagging TLB entries with a context ID

End

Week 09 – Tutorial 1