CI583: Data Structures and Operating Systems
Memory management part 3

Outline

Virtual memory in practice

The aim of virtual memory is to allow the OS to address a larger amount of memory than the available primary storage.

The job of the OS is to make sure that using the (expensive) techniques of virtual memory can be done as efficiently as possible.

Virtual memory in practice

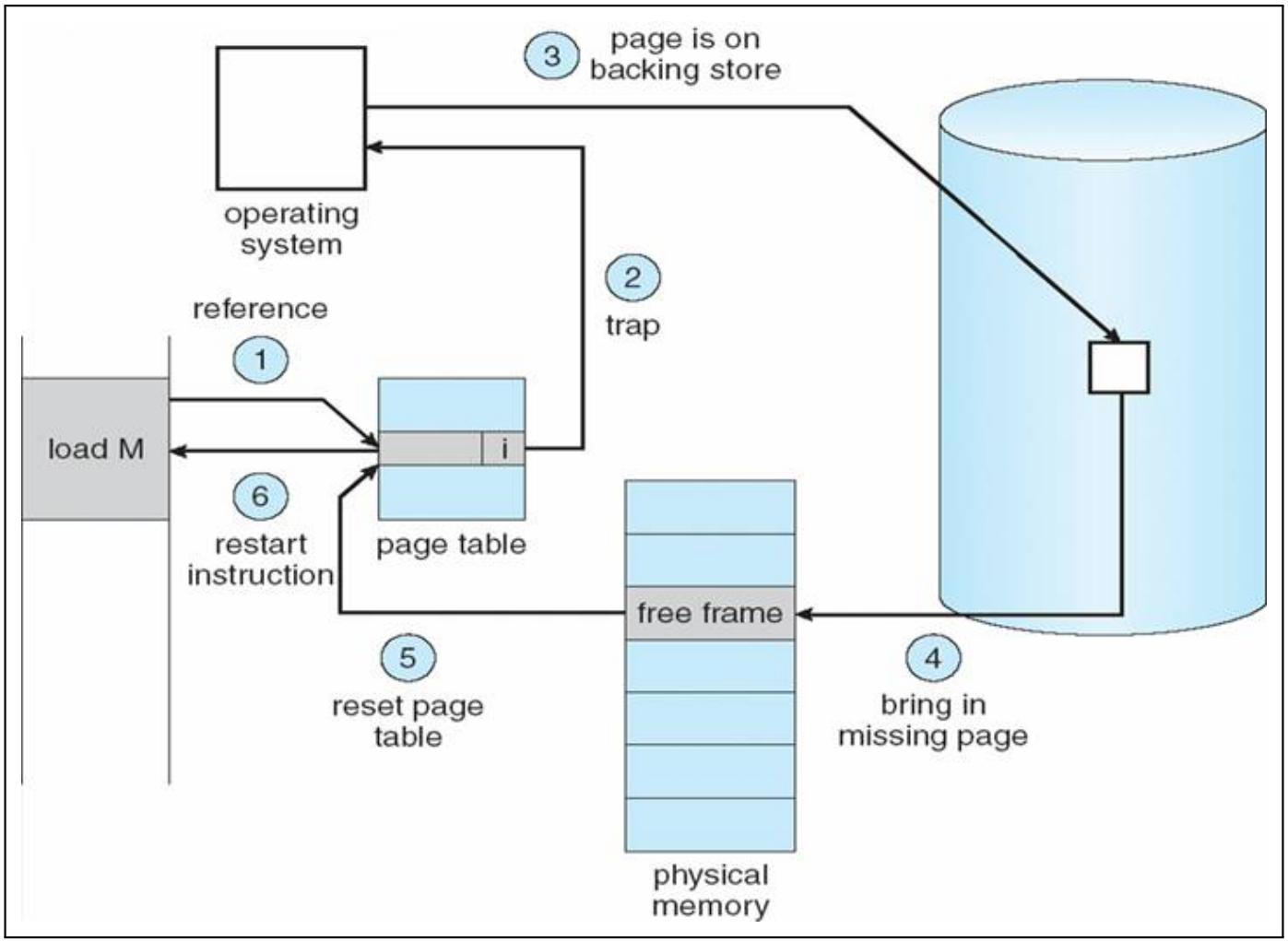

Image ©http://www.cs.odu.edu/~cs471w

Virtual memory in practice

Consider the actions required when a page fault occurs:

1. Hardware address translation facility raises an interrupt.
2. Find a free page frame.
3. If there are no free frames, decide which existing frame to reuse.
4. If the frame we're reusing contains a modified page, write it out to secondary storage.
5. Fetch the required page from secondary storage and modify the page table.
6. Return from interrupt.

Virtual memory in practice

This is an expensive process, especially the IO involved, which could take milliseconds to complete.

We have seen how to map virtual memory locations to page frames in primary storage, and we have seen how to move pages from secondary to primary storage when a page fault occurs.

Now we need to know which pages to hold in primary storage and which to get rid of.

Virtual memory in practice

We can break this down into three areas:

1. The fetch policy: when to bring pages in from secondary storage and which ones to bring.
2. The placement policy: where to put the pages in primary storage (i.e. how to allocate page frames).
3. The replacement policy: which pages to remove from primary storage and when to do it.

Fetch policies

Probably the simplest fetch policy we can imagine is demand paging: fetch a page (only) when a thread references a location in a page that is not in primary storage.

Thus, the execution of a program, P, begins by loading a single page that contains the initial instructions for P. Each subsequent page is retrieved as needed.

If the cost of fetching n pages is $n\times$ the cost of fetching one page, this is the best we can do! It isn't though – why not?

Fetch policies

As we have said, we want to minimise the IO in all of this and reduce the number of faults that occur.

One way to do this is by prepaging: fetching pages before they are actually required (i.e. without having to go through a whole page fault in order to get it).

Fetch policies

How do we know which pages a process will require, before it actually requires them?

Most of the time there is no way to know this, but it is a fair bet that if a process requests a given page, it might well request its subsequent pages next.

Fetching 2 or 3 pages in one go is not much more expensive than fetching one, since the file system, disk controller etc are already engaged. This is called readahead.

Fetch policies

Another way in which the fetch policy can have a big impact is in choosing a page size that reflects the type of files in the system and the way they are accessed.

For instance the Google File System (GFS) is a distributed file system that makes a number of design decisions based on the assumption that a file which is less than 10GB in size is a relatively small one.

`http://research.google.com/archive/gfs.html`

Next time

Policies for **placement** and **replacement**.