More Fun With Virtual Memory: Paging, Page Replacement Strategies & Analysis, Page Table Management

Backing Store

All pages of the memory must be copied to *backing store*, i.e., disk, so that if their space is needed in memory they can be restored from disk. The analogy to swap space should be pretty obvious: both are copies of a runnable process on disk. The distinction is that paging backing store can contain copies of memory of processes that are currently executing, while swapped processes are always stopped. Paging systems use backing store, and allocate and use it on a per-page basis; swapping systems use swap space and allocate it in units of entire processes' address spaces.[1] That said, people frequently say swap when they mean backing store; UNIX® certainly does.

It's expensive (in time) to move pages onto and off of backing store, so OSes take various steps to avoid it. Some systems page executables and other read-only memory from the filesystem directly, avoiding the copy to backing store. Others overcommit backing store by only copying a page to backing store if it must be removed from memory for some reason. The tradeoffs here are similar to those for swap spaces.

Page Faults

When a page is not present, the OS faces a *page fault.* The following are the basic steps in servicing a page fault:

- The MMU interrupts the CPU (with a paging interrupt or exception depending on the CPU model)
- The paging ISR (usually called a *pager* or *page fault handler*) loads the required page into an available frame. If a frame is not available, the pager must make one available, by discarding a frame. It's generally better to evict a clean page than a dirty one because dirty ones must be written to backing store. The current process may give up the CPU while the relevant pages are moved in and out. (This is basically an I/O request, although not an explicit one).
- When the pages are available and the process is runnable again, the faulting instruction is restarted.

The whole process is rather expensive, which is why adding memory to a paging system generally[2] speeds up a paging system. More memory makes the expensive page faults less frequent.

Page Replacement

Page replacement is another form of scheduling, so these algorithms should look familiar:

- Optimal - page out the page that will be needed the furthest in the future. This is impossible (halting problem), but provides an interesting benchmark.
- FIFO - keep a list of pages in the order that they were loaded, we'll talk at length about why this isn't used, but the oldest page isn't always the best one to be rid of.
- Second Chance - each page includes a *referenced*bit in its page table entry (maintained by the hardware). A FIFO replacement algorithm is run, but pages with the referenced bit set are put at the bottom of the queue and given a second chance. The goal is to keep recently used pages in memory. Note that this is FIFO in the limit.
- Clock - Second Chance with a circular list. (Like next fit). First time a page is considered, its referenced bits are cleared so it will be chosen the second time around.

[1] My favorite exchange on `comp.unix.wizards` consisted of the question "what's the difference between paging and swapping?" answered with "One's slow and the other isn't much faster."

[2] dramatic pause...

- wsclock (see the working set below)
- Least Recently Used (LRU) - pages are ordered by their reference times, and the one referenced the longest time ago is the first choice for eviction. The idea here is that programs exhibit code and data *locality*, that is they frequently reuse the same data or code.

  This isn't really possible, but we do have some approximation methods:

  Use hardware counters, matrix.

  Use software counters, periodic shifting
- Not Recently Used (NRU) - Basically this is LRU with only one bit. Clean pages get preference in being paged out.

Theory of Page Replacement

Paging algorithms are analyzed by studying the effects of the algorithm on reference strings - that is sequences of page references against which the algorithm is studied. Here's an example:

Reference String	1	3	0	2	0	1	2	1
	1	3	0	2	0	1	2	1
		1	3	0	2	0	1	2
			1	3	3	2	0	0
				1	1	3	3	3
Page Faults	P	P	P	P	-	P	P	-
Distance String	∞	∞	∞	∞	2	4	3	2

Belady's Anomaly

A group of researchers, led by a fellow named Belady, discovered a surprising fact about FIFO paging. It's possible (though unlikely) that adding memory to a FIFO paging system **increases** the number of faults. The example is below:

	0	1	2	3	0	1	4	0	1	2	3	4
Youngest Page	0	1	2	3	0	1	4	4	4	2	3	3
		0	1	2	3	0	1	1	1	4	2	2
Oldest Page			0	1	2	3	0	0	0	1	4	4
	P	P	P	P	P	P	P			P	P	

	0	1	2	3	0	1	4	0	1	2	3	4
Youngest Page	0	1	2	3	3	3	4	0	1	2	3	4
		0	1	2	2	2	3	4	0	1	2	3
			0	1	1	1	2	3	4	0	1	2
Oldest Page				0	0	0	1	2	3	4	0	1
	P	P	P	P			P	P	P	P	P	P

This result has been generalized, and the key property is called the stack property: that increasing the size of memory only adds contents to it. In the FIFO case above, there are different contents in the upper 3 page frames in memory for several states. FIFO is not a stack algorithm. Any non-stack algorithm can display Belady's anomaly. LRU is a stack algorithm.

The Working Set

The working set of a process is the set of pages that it will reference in the near future - the pages it's working on. (That's intentionally vague, of course) If this set is in memory, the process will not fault. Consequently, some OSes try to identify and keep the working set in memory.

Just letting a process pull in its pages as it needs them is called *demand paging*, and is a common form of paging. In systems that try to identify the working set will try to prefetch pages that it thinks are or will be in the working set. A simple example is a process that sequentially fills an array from memory. It's a reasonable idea to prefetch pages sequentially.

More complex methods involve keeping the LRU approximations around for each process, and if the counters are non-zero, the page is in the working set. A variant of the clock algorithm that takes the working set counter into account rather than the referenced bit is called *wsclock*.

Nachos questions