Thrashing

- A situation in which paging becomes the *major* system activity (so that it appears to the users that the system is not progressing—an observable degradation of performance), is called *thrashing*
- Thrashing happens because there are very few pages left in the system, so the frequently used pages need to be replaced to maintain the progress. As the result, CPU spends more time paging than executing processes
- One solution to thrashing is to swap out all low-priority processes and thus decrease the degree of multiprogramming

But the Memory is Getting Cheaper and Cheaper!

So, would increasing the memory, say by a third, help?

Let Us See...

- We are executing a program with *five virtual memory pages*, which is producing the reference string

  0 1 2 3 0 1 4 0 1 2 3 4

- We have the memory with three *frames* (initially empty), and we are using FIFO:

	0	1	2	3	0	1	4	0	1	2	3	4
	0	1	2	3	0	1	4	4	4	2	3	3
		0	1	2	3	0	1	1	1	4	2	2
			0	1	2	3	0	0	0	1	4	4
	F	F	F	F	F	F	F			F	F	

9 page faults

Let us Add One More Frame

- We are executing a program with five virtual memory pages, which is producing the reference string

0 1 2 3 0 1 4 0 1 2 3 4

- We have the memory with ~~three~~ **four** page frames (initially empty), and we are using FIFO:

	0	1	2	3	0	1	4	0	1	2	3	4
	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
				0	0	0	1	2	3	4	0	1
	F	F	F	F			F	F	F	F	F	F

10 page faults!

Huh?

Belady's Anomaly and the World After Its Discovery

- The same program that causes nine page faults in the three-frame memory may cause ten page faults in the four-frame memory when FIFO is used!
- After *Belady*'s discovery (in 1969), a whole theory of paging was built. A class of paging algorithms whose behavior does *not* deteriorate on the same reference string after a frame is added was determined
- The algorithms in the class are called *Stack Algorithms*. *LRU* is a stack algorithm; *FIFO* is not (of course).

Stack Algorithms: Formal Definition

- Let
  - $\omega = (s_1, s_2, .. s_n)$ be a reference string processed in the memory containing *m* frames; and
  - *M(m, ω)* denote the state of the memory after ω has been processed.
- A page replacement algorithm is called a *stack* algorithm if

$$M(m, \omega) \subseteq M(m+1, \omega).$$

Another way to look at it

- Given ω, there exists such a permutation

$\pi(\omega) = \{\pi_1(\omega), \pi_2(\omega), \cdots, \pi_n(\omega)\}$ of virtual pages *{1, 2, …, n}* that for all m = 1, 2, 3,…,

$$M(m, \omega) = \{\pi_1(\omega), \pi_2(\omega), \cdots, \pi_m(\omega)\}$$

- In other words, the contents of real memory are always determined by the first *m* pages of $\pi(\omega)$ called the *stack*.

- For LRU, *M(m, ω) = {m most recently visited pages} = M (m+1, ω)*

Adaptation to *phase transition*

- Consider the string $\omega = (1,2,3)^k (4,5,6)^l$
- With three frames, MRU will result in *3(l+1)* page faults, and LFU will result in *3[min(k,l)+1]* page faults, but FIFO and LRU will result only in six page faults.

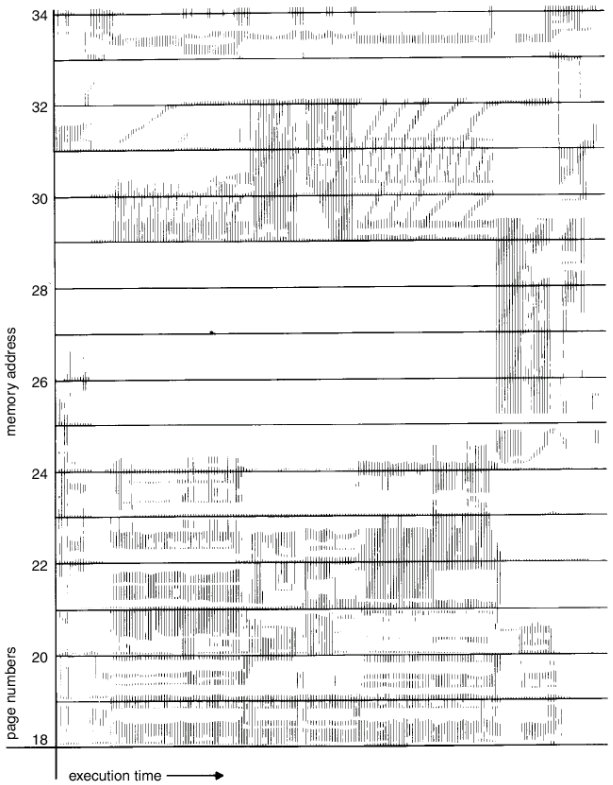
Homework assignment: Prove this!

The *Working Set* Model

- It is easy to write a program that would make any specific paging algorithm to work badly—such a program could generate a page fault on any reference
- But processes that execute real programs exhibit *locality reference*—during the execution, a process uses a relatively small set of pages for a long time. Such stable set is called the *working set* of a process

Locality in Memory Refences

From the Book



References are localized to a few pages, and stay this way for a while

In the beginning, pages 18-24 and 30-34 make a working set, then pages 25-29 make another one

Working with Working Sets

- The operating system uses heuristics to determine a working set of each process and then maintains the frame allocation so as to ensure the working set is always in memory
  1. First, the number of references to determine a stable set is guessed (typically, *10,000* references)
  2. Then the timer value is chosen. The timer causes an interrupt, on which the system clears the *dirty* and *reference* bits, so that it can determine on the next interrupt whether the set has changed
- If it happens that the overall number of free frames falls short of the working sets' requirements, the execution of the low priority processes is postponed

Frame Allocation and Replacement

- Each process can be given either the same number of frames that any other process gets (*equal allocation*) or have the number of frames allocated in proportion to its size
- Typically, a number of frames are given for the stack, text, and heap segments each
- On page fault, either the whole memory is searched for a free (or evicted) page—this is called *global replacement*, or only the memory of the executing process (*local replacement*)
- The local replacement is good in localizing thrashing to just one process—not the whole system

Selecting the Page Size

- Determining the optimum page size requires analysis of several competing factors
  - The smaller the page is, the less the internal fragmentation; for the same reason, smaller pages result in less unused memory
  - But small pages—as we saw in the last lecture—require larger page tables, so large memory address registers dictate large page size
  - Because most of the disk time is spent on *rotational* and *seek* procedures, the transfer of a small page takes about the same time as the transfer of a large page. Then much more memory is transferred in the same amount of time, if large page size is used
  - The space occupied by the page table increases as the page size decreases

Selecting the Page Size (cont.)

- Let
  - *s* be the average process size,
  - *p* be the page size
  - e be the size of a page table entry
- Then the average number of pages per process is *s/p*, and the average page table size is *se/p*
- Given that the average internal fragmentation overhead is *p/2*, the overall overhead is

$$overhead(p) = \frac{se}{p} + \frac{p}{2}$$

Selecting the Page Size (cont.)

$$overhead(p) = \frac{se}{p} + \frac{p}{2}$$

Optimal points, if any, are the roots of the equation:

$$overhead'(p) = -\frac{se}{p^2} + \frac{1}{2} = 0;$$

The only meaningful solution of the above (and thus the optimal point—but please check that it *is* optimal) is

$$p = \sqrt{2se}.$$