WORKING SET

Peter J. Denning, Naval Postgraduate School, Monterey, California

January 2008
Rev 5/26/08

Abstract: The working set is a dynamic subset of a process's address space that must be loaded in main memory to ensure acceptable processing efficiency. Working set policies can be tuned for close-to-optimal throughput and response time. They prevent thrashing. The working set is the reference standard for all virtual memory management policies.

Keywords: working set, locality, memory management, replacement policy, virtual memory, locality principle

The working set is a dynamic subset of a process's address space that must be loaded in main memory to ensure acceptable processing efficiency. In the early days of computing, this intuitive idea enabled programmers to plan their memory usage over time in a constrained main memory. Later it turned into a formal definition of the ideal subset of address space to be loaded in main memory. Eventually the formal definition became the reference standard for all virtual memory replacement policies.

In early computing systems, programmers built overlay strategies for their programs. They made maps showing computational phases on the time axis and instruction or data block on the vertical axis. They checked off the blocks that needed to be loaded in main memory for each phase. They adjusted computational strategies and block contents until the blocks needed for each phase would fit. Then at the phase transitions, they programmed "down" and "up" commands. Down commands moved blocks from main to secondary memory when they were no longer needed in the next phase. Up commands moved blocks from secondary and main memory in time for the next phase. The set of blocks checked in the map was called the "working set" of a phase. The memory usage of the program could be characterized as a sequence

(L1,T1) (L2,T2) (L3,T3) ...

in which Li is the set of blocks loaded in phase i and Ti is the duration of the phase.

System designers were concerned that this process, already tedious and time consuming for small programs, would become unmanageable for large programs. In the late 1950s, the designers of the Atlas Computer at the University of Manchester invented virtual memory to automate this process. Their system broke program code and data into fixed size pages. It issued "up" commands at the moments that the program attempted to access a page not loaded. They invented a replacement algorithm that decided which loaded page to move "down" to make way for incoming "up" pages. Their algorithm assumed that each page was in a cycle of use and nonuse; by measuring the most recent use and nonuse periods, they predicted when each page would be used again. They selected for replacement the page not needed for the longest time.

Many people were attracted to the idea of virtual memory because of its big boost for programming productivity. But they were put off by the unpredictability of the replacement algorithms, which worked well for some programs and poorly for others. There were numerous contradictory experimental studies, but no one found a replacement algorithm that worked consistently well for all programs. In 1966 Les Belady (1) published an extensive study of replacement algorithms in which he demonstrated that replacement policies with usage bits performed better than those without. He suggested that this is due to "program locality", a tendency of programs to cluster references into subsets of their pages. He suggested that under multiprogramming a program should be given enough space to hold its "parachor", which was roughly the space at which the replacement algorithm's mean time between page faults equaled the page fault service time from the secondary memory.

In 1967, Denning (2) offered a precise definition of a working set. He defined it as the set of pages referenced in a virtual time window looking backwards for time T into the past. The working set dynamically varied as more or fewer pages appeared in the window. It was important to measure in virtual time -- that is, not counting any interruptions -- so as to get an intrinsic measure of the program's favored pages. He was able to show the somewhat surprising property that the paging rate and mean working set size could be computed easily from a histogram of the times between repeated references to the same page. It was then a simple matter to choose the window size T so that the mean time between page faults would always be larger than the mean page fault service time -- that is, the CPU efficiency of the program would be at least 0.5.

Denning also showed that a multiprogrammed memory managed by a working set policy could not thrash. In later experiments with students and others, he established that the working set policy would produce system throughput within 5% to 10% of optimal, where optimal was defined in terms of perfect knowledge of the future (3). Thus, the working set policy became an ideal for other memory management policies.

The true working set would require measurements in a sliding window looking backwards from the current time. Although it worked perfectly (4), the cost of the mechanism was high. Many simpler software approximations were tried and tested, the most successful being the "WS Clock" (5).

Denning interpreted this definition of working set as a measure of the program's intrinsic memory demand. He hypothesized that programs have inherent tendencies to cluster their references into small subsets for extended periods, an idea he called "locality" after Belady. In numerous experiments with students and others, he concluded that the dynamic locality processes of programs consisted of phases and transitions; phases were periods of stability, with all references concentrated in a "locality set", and transitions were short periods of instability. In other words, every program has a natural sequence of locality sets and phases,

(L1,T1) (L2,T2) (L3,T3) ...

A memory policy that loads exactly the locality set for each phase will achieve optimal paging behavior. As long as most phases are longer than the working set window T, the working set will be a very close measurement of these actual locality sets of varying sizes. Locality is the reason working sets work.

The locality behavior so painstakingly planned by early programmers is a natural property of programs anyway! It arises from the way that the human brain solves problems and pays attention.

In some systems, working sets can be deduced from a program's structure rather than by measurement of usage bits. For example, on machines using block-structured programming languages such as Ada, the working set can be defined as the current procedure segment, the stack, and all other data structures accessible from activated procedures.

In paging systems, it can be advantageous to "restructure" a program by clustering small, logical segments of the same locality on large pages. By preserving in the page references the locality originally present in the segment references, this strategy can yield the small working sets and efficient performance in systems with large page size. Restructuring is less important in systems with smaller page sizes.

BIBLIOGRAPHY

1. L. A. Belady, A study of replacement algorithms for virtual storage computers. IBM *Systems J.* **5**, 2: 78-101, 1966.

2. P. J. Denning, The working set model for program behavior. *Commun. ACM* **11**, 5 (May): 323-333, 1968. First published in *Proc. ACM Symp. on Operating System Principles*, Gatlinburg, TN, 1967.

3. P. J. Denning, Working sets past and present. *IEEE Trans. Software Eng.* **SE-6**, 1 (January): 64-84, 1980.

4. J. Rodriguez-Rosell and J. P. Dupuy, The design implementation, and evaluation of a working set dispatcher. *Commun. of ACM* **16**, 4 (April), 1973.

5. R. Carr and J. Hennessey, WSCLOCK -- a simple and effective algorithm for virtual memory management. ACM *SIGOPS Review* **18** (December): 87-95, 1981.

FURTHER READING

A. Tanenbaum. *Modern Operating Systems*, 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 2007.