This chapter deals virtual memory, which is a technique to map a potentially larger logical address space onto a smaller physical memory. This enables us to run extremely large process, thus to increase the degree of multiprogramming and CPU utilization. In addition, it frees application programmers from worrying about memory availability when writing programs.

Virtual Memory

- The separation of user *logical memory space* from physical memory. This allows only part of program (code and data) of a user to reside in memory for execution. The arguments for this are: 1) in many cases there is no need to load the entire program into physical memory since some of it might never be used; 2) users or programmers not constrained by the amount of physical memory on a machine.
- **Benefit:** since each process occupies less amount of memory, this significantly increases the degree of multi-programming, and improves the CPU utilization.

Demand Paging

- **Pure demand paging** scheme only brings a page into memory when the page is actually referenced. In another word, a page not referenced during execution will never be brought into the memory.
- **Page fault** has to be handled under a demand paging scheme. The performance of demand page scheme is primarily determined by the **page-fault rate**.

Page Replacement Policies

- The page replacement policies discussed here are under the assumption that each process is allocated with a *fixed* number of frames with local replacement. When a page fault occurs and if all frames allocated to this process are occupied, one of the pages (frames) needs to be selected as a *victim* to be replaced. The difference in page replacement policies lies in which page to select as the victim.
- **FIFO replacement**: replace the page that was brought into the memory earliest. This suffers from the *Belady's Anomaly*, in which a higher page fault rate might occur when extra frames are allocated to a process.
- **Optimal (OPT) replacement:** replace the page that will not be used for the longest period of time in the future. This guarantees the lowest page-fault rate for a given fixed number of frames; however, this is practically not possible as it requires future knowledge of page reference pattern.
- **LRU replacement**: replace the page that has not been used for the longest time (as an approximation for OPT). This can be implemented by a **stack** algorithm, which requires hardware assistance and updates for every memory reference. The update is too frequent, which makes LRU not practical.
- **Second-chance algorithm:** One bit used as a *reference bit* for each page that is set with a reference to this page. FIFO replacement order is followed for each page with the reference bit (0). If the reference bit is 1, reset to 0 (second chance). It finds the next page with zero reference bit according to FIFO. A **Clock** algorithm can be used to implement the second-chance algorithm.

The Working-Set Model

- **Assumption**: a working-set model assumes there is a locality in the memory access pattern by each process (this is usually true). The number of frames allocated to a process is a variable.
- **Working-set window**: this specifies the window size (usually fixed) in term of the number of memory references it will keep track of. The selection of the working-set window size is non-trivial as it can not be too small (does not capture the current locality), nor can it be too big (beyond the current locality).
- **The basic idea:** suppose we have selected an adequate working-set window size, the working-set or WSS_i (working-set for a process i) would roughly capture the current locality, i.e., the minimum number of pages a process needs at the moment. This allows the OS to dynamically adjust the number of frames allocated to each process based on the current working-set.
- **Thrashing:** since WSS_i is a good approximation of the number of pages needed for a process i at a given time, the total memory needed at the time by all processes is D = \sum WSS_i. If D > m (the actual physical memory size), thrashing will occur, in which at least one process is constantly in short of memory, thus its pages are swapped in and out.
- **The difficulty:** This requires keeping track of the working set for each process. The working-set window is a moving window with each new memory reference. A page is in the working-set if it has been referenced anywhere in this working-set window. This has too much overhead, esp. with a large working-set window. Noticing what this requires is that each memory reference by any process, there must be updates for its working-set (even if there is no change)

Other Design Issues

- **Pre-paging** is used at the startup time of a process execution to preload certain number of pages into the memory to reduce the initial page fault rate. This is useful when the cost of the saved page faults is greater than the cost of pre-paging unnecessary pages (the pages that will not be used)
- **Page size**: a conflicting set of factors affect the selection of page size, internal fragmentation, page table size, I/O overhead, reference locality. For instance, large page size results in more internal fragmentation; small page size results in large page table.
- **TLB reach**: the amount of memory that the TLB can access, determined by the number of entries in TLB and page size. This calls for multiple page sizes in some cases.