Allocation of Frames

- How should the OS distribute the frames among the various processes?
- Each process needs *minimum* number of pages - at least the minimum number of pages required for a single assembly instruction to complete
- Example: IBM 370 – 6 pages to handle SS MOVE instruction:
  - instruction is 6 bytes, might span 2 pages
  - 2 pages to handle from
  - 2 pages to handle to
- Two major allocation schemes
  - fixed allocation
  - priority allocation
Fixed Allocation

- Equal allocation – For example, if there are 100 frames and 5 processes, give each process 20 frames.
  \[ s_i = \text{size of process } p_i \]
  \[ S = \sum s_i \]
  \[ m = \text{total number of frames} \]
  \[ a_i = \text{allocation for } p_i = \frac{s_i}{S} \times m \]

- Proportional allocation – Allocate according to the size of process
  \[ m = 64 \]
  \[ s_i = 10 \]
  \[ s_2 = 127 \]
  \[ a_1 = \frac{10}{137} \times 64 \approx 5 \]
  \[ a_2 = \frac{127}{137} \times 64 \approx 59 \]
Priority Allocation

- Use a proportional allocation scheme using priorities rather than size

- If process $P_i$ generates a page fault,
  - select for replacement one of its frames
  - select for replacement a frame from a process with lower priority number
Global vs. Local Allocation

- Global replacement – process selects a replacement frame from the set of all frames; one process can take a frame from another
  - It is possible for processes to suffer page faults through no fault of theirs
  - However, improves system throughput

- Local replacement – each process selects from only its own set of allocated frames
  - May not use free space in the system
Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
  - low CPU utilization
  - operating system thinks that it needs to increase the degree of multiprogramming because of low cpu utilization
  - another process added to the system

- Thrashing ≡ a process is busy swapping pages in and out
Thrashing (Cont.)

![Graph showing CPU utilization vs. degree of multiprogramming with a region labeled "thrashing"]
Demand Paging and Thrashing

- Why does demand paging work?
  Locality model
  - Process migrates from one locality to another
  - Localities may overlap
  - E.g.
    ```
    for (……) {
      computations;
    }
    for (….. ) {
      computations;
    }
    ```

- Why does thrashing occur?
  \[ \sum \text{size of locality} > \text{total memory size} \]
Working-Set Model

- $\Delta \equiv \text{working-set window} \equiv \text{a fixed number of page references}$
  
  Example: 10,000 instruction

- $WSS_i$ (working set of Process $P_i$) = total number of pages referenced in the most recent $\Delta$ (varies in time)
  
  - if $\Delta$ too small will not encompass entire locality
  - if $\Delta$ too large will encompass several localities
  - if $\Delta = \infty \Rightarrow$ will encompass entire program

- $D = \sum WSS_i \equiv \text{total demand frames}$
  
  if $D > m \Rightarrow$ Thrashing

  Policy if $D > m$, then suspend one of the processes
Working-set model

page reference table

\[
\begin{array}{cccccccccccc}
\ldots & 2 & 6 & 1 & 5 & 7 & 7 & 7 & 7 & 5 & 1 & 6 & 2 & 3 & 4 & 1 & 2 & 3 & 4 & 4 & 3 & 4 & 4 & 1 & 3 & 2 & 3 & 4 & 4 & 3 & 4 & 4 & 4 & \ldots \\
\end{array}
\]

\[WS(t_1) = \{1, 2, 5, 6, 7\}\]

\[WS(t_2) = \{3, 4\}\]
Keeping Track of the Working Set

- Approximate with interval timer + a reference bit
- Example: $\Delta = 10,000$
  - Timer interrupts after every 5000 time units
  - Keep in memory 2 bits for each page
  - Whenever a timer interrupts copy and sets the values of all reference bits to 0
  - If one of the bits in memory = 1 $\Rightarrow$ page in working set
- Why is this not completely accurate?
- Improvement = 10 bits and interrupt every 1000 time units
Page-Fault Frequency Scheme

- Establish “acceptable” page-fault rate
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame

![Graph showing relationship between page-fault rate and number of frames. The curve indicates that as the number of frames increases, the page-fault rate decreases. There are two horizontal lines indicating upper and lower bounds.]
Other Issues -- Prepaging

- Prepaging
  - To reduce the large number of page faults that occurs at process startup
  - Prepage all or some of the pages a process will need, before they are referenced
  - But if prepaged pages are unused, I/O and memory was wasted
  - Assume $s$ pages are prepaged and $\alpha$ of the pages is used
    - Is cost of $s \times \alpha$ save pages faults > or < than the cost of prepaging $s \times (1-\alpha)$ unnecessary pages?
    - $\alpha$ near zero $\Rightarrow$ prepaging loses
Other Issues – Page Size

Page size selection must take into consideration:
- fragmentation
- table size
- I/O overhead
- locality
Other Issues – TLB Reach

- TLB Reach - The amount of memory accessible from the TLB
- TLB Reach = (TLB Size) X (Page Size)
- Ideally, the working set of each process is stored in the TLB. Otherwise there is a high degree of page faults.
- Increase the Page Size. This may lead to an increase in fragmentation as not all applications require a large page size
- Provide Multiple Page Sizes. This allows applications that require larger page sizes the opportunity to use them without an increase in fragmentation.
Other Issues – Program Structure

- Program structure
  - `Int[128,128] data;`
  - Each row is stored in one page
  - Program 1
    
    ```
    for (j = 0; j <128; j++)
      for (i = 0; i < 128; i++)
        data[i,j] = 0;
    ```
    
    128 x 128 = 16,384 page faults
  
  - Program 2
    
    ```
    for (i = 0; i < 128; i++)
      for (j = 0; j < 128; j++)
        data[i,j] = 0;
    ```
    
    128 page faults
Wrapup

- **Memory hierarchy:**
  - Speed: L1, L2, L3 caches, main memory, disk etc.
  - Cost: disk, main memory, L3, L2, L1 etc.

- achieve good speed by moving “interesting” objects to higher cache levels while moving “uninteresting” objects to lower cache levels

- Hardware provides reference bit, modify bit, page access counters, page table validity bits

- OS sets them appropriately such that it will be notified via page fault
  - OS provides policies
  - Hardware provides mechanisms

- Implement VM, COW etc. that are tuned to observed workloads