OPERATING SYSTEMS

Virtual Memory Policies

Operating Systems

School of Data & Computer Science
Sun Yat-sen University

Lecture Notes: os_sysu@163.com
Instructor: Guoyang Cai
email: isscgy@mail.sysu.edu.cn

中山大學
SUN YAT-SEN UNIVERSITY

Contents

- Virtual Memory Policies Need to decide on:
  - Fetch policy
  - Placement policy
  - Replacement policy
  - Resident Set Management
  - Load Control

Memory Management Software

- Memory management software depends on whether the hardware supports paging or segmentation or both.
- Pure segmentation systems are rare. Segments are usually paged -- memory management issues are then those of paging.
- We shall thus concentrate on issues associated with paging.
- To achieve good performance we need a low page fault rate.

Fetch policy

- Determines when a page should be brought into main memory. Two common policies:
  - Demand Paging only brings pages into main memory when a reference is made to a location on the page (i.e., paging on demand only).
    - many page faults when process first starts but should decrease as more pages are brought in.
  - Prepaging brings in pages whose use is anticipated:
    - locality of references suggest that it is more efficient to bring in pages that reside contiguously on the disk.
    - efficiency not definitely established: the extra pages brought in are “often” not referenced.

Placement Policy

- Determines where in real memory a process piece resides.
- For paging (and paged segmentation):
  - the hardware decides where to place the page: the chosen frame location is irrelevant since all memory frames are equivalent (not an issue).
- For pure segmentation systems:
  - first-fit, next fit... are possible choices (a real issue).

Replacement Policy

- Deals with the selection of a page in main memory to be replaced when a new page is brought in.
- This occurs whenever main memory is full (no free frame available).
- Occurs often since the OS tries to bring into main memory as many processes (pages) as it can to increase the multiprogramming level.
- Not all pages in main memory can be selected for replacement.
- Some frames are locked (cannot be paged out):
    - much of the kernel is held on locked frames as well as key control structures and I/O buffers.
- The OS might decide that the set of pages considered for replacement should be:
    - limited to those of the process that has suffered the page fault.
    - the set of all pages in unlocked frames.

Replacement Policy

- The decision for the set of pages to be considered for replacement is related to the resident set management strategy:
  - how many page frames are to be allocated to each process?
- No matter what is the set of pages considered for replacement, the replacement policy deals with algorithms that will choose the page within that set.

Allocation of Frames

- Resident Set Management
  - The OS must decide how many page frames to allocate to a process:
    - large page fault rate if too few frames are allocated.
    - low multiprogramming level if too many frames are allocated.
- Resident Set Size
  - Fixed-allocation policy:
    - allocates a fixed number of frames that remains constant over time:
      - the number is determined at load time and depends on the type of the application.
  - Variable-allocation policy:
    - the number of frames allocated to a process may vary over time:
      - may increase if page fault rate is high.
      - may decrease if page fault rate is very low.
    - requires more OS overhead to assess behavior of active processes.

Allocation of Frames

- Fixed Allocation
  - Equal Allocation
    - for example, if there are 100 frames and 5 processes, give each process 20 frames.
    - keep some as free frame buffer pool
  - Proportional Allocation
    - allocate according to the size of process.
    - dynamic as degree of multiprogramming, process sizes change

      Let s_i = size of process P_i , m = total number of frames

      Then $S = \Sigma s_i$

      a_i = allocation for $p_i = (s_i / S) \times m.$
    - Example

      $s_1 = 10$, $s_2 = 127$, $m = 64$

      $S = \Sigma s_i = 10 + 127 = 137$

      $a_1 = (10 / 137) \times 64 \approx 5$

      $a_2 = (127 / 137) \times 64 \approx 59$

Allocation of Frames

- Fixed Allocation
  - 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.

Allocation of Frames

- Replacement Scope
  - Replacement scope is the set of frames to be considered for replacement when a page fault occurs.
  - *Global replacement* – process selects a replacement frame from the set of all frames; one process can take a frame from another
    - But then process execution time can vary greatly
    - But greater throughput so more common
  - *Local replacement* – each process selects from only its own set of allocated frames
    - More consistent per-process performance
    - But possibly underutilized memory
  - We will consider the possible combinations of replacement scope and resident set size policy.

Allocation of Frames

- Fixed Allocation + Local Scope
  - Each process is allocated a fixed number of pages:
    - determined at load time; depends on application type.
  - When a page fault occurs, page frames considered for replacement are local to the page-fault process:
    - the number of frames allocated is thus constant.
    - previous replacement algorithms can be used.
  - Problem: difficult to determine ahead of time a good number for the allocated frames:
    - if too low: page fault rate will be high.
    - if too large: multiprogramming level will be too low.
- Fixed Allocation + Global Scope
  - Impossible to achieve:
    - if all unlocked frames are candidate for replacement, the number of frames allocated to a process will necessary vary over time.

Allocation of Frames

- Variable Allocation + Global Scope
  - Simple to implement -- adopted by many OSs (like Unix SVR4).
  - A list of free frames is maintained:
    - when a process issues a page fault, a free frame (from this list) is allocated to it.
    - Hence the number of frames allocated to a page fault process increases.
    - The choice for the process that will loose a frame is arbitrary; It's far from optimal.
  - Page buffering can alleviate this problem since a page may be reclaimed if it is referenced again soon.

Allocation of Frames

- Variable Allocation + Local Scope
  - May be the best combination (used by Windows NT).
  - Allocate at load time a certain number of frames to a new process based on application type:
    - use either pre-paging or demand paging to fill up the allocation.
  - When a page fault occurs, select the page to replace from the resident set of the process that suffered the page fault.
  - Reevaluate periodically the allocation provided and increase or decrease it to improve the overall performance.

Allocation of Frames

- The Working Set Strategy
  - The working set strategy is a variable-allocation method with local scope based on the assumption of locality of references.
  - The working set for a process at time t, W(D, t), is the set of pages that have been referenced in the last D virtual time units:
    - virtual time = time elapsed while the process was in execution.
    - D is a window of time.
    - W(D, t) is an approximation of the program's locality.
  - The working set of a process first grows when it starts executing.
  - Then stabilizes by the principle of locality.
  - It grows again when the process enters a new locality (transition period):
    - up to a point where the working set contains pages from two localities.
  - Then decreases after a sufficient long time spent in the new locality.

Allocation of Frames

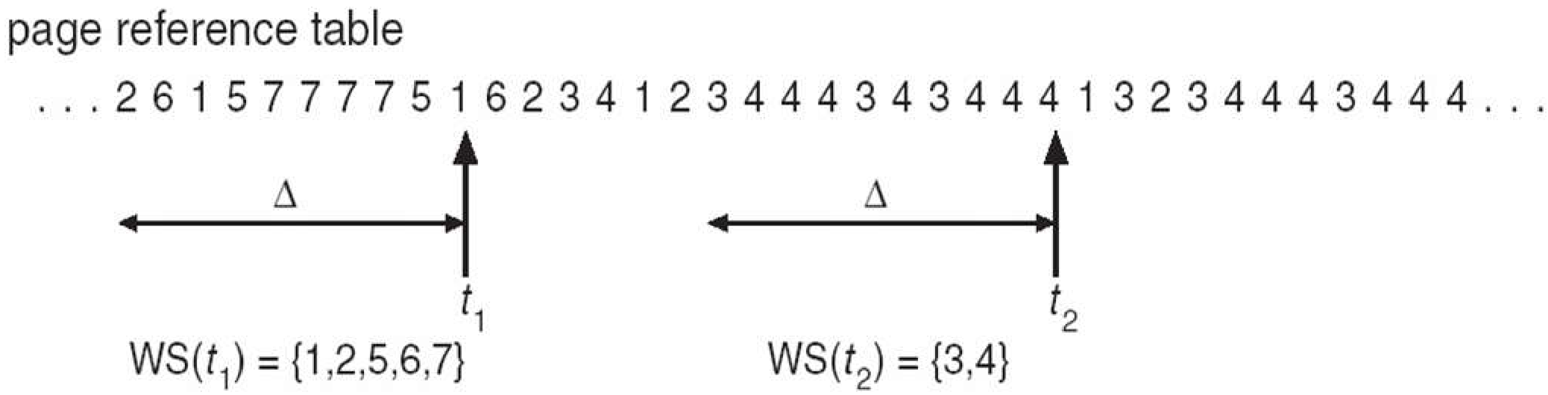
- The Working Set Strategy
  - The working set concept suggests the following strategy to determine the resident set size:
    - Monitor the working set for each process.
    - Periodically remove from the resident set of a process those pages that are not in the working set.
    - When the resident set of a process is smaller than its working set, allocate more frames to it:
      - If not enough free frames are available, suspend the process (until more frames are available), i.e., a process may execute only if its working set is in main memory.

Allocation of Frames

- The Working Set Strategy
  - Working-Set Model
    - Δ = working-set window = a fixed number of page references
      - Example: 10,000 instructions
    - WSS_i (working set of Process P_i) =
      total number of pages referenced in the most recent Δ
      (varies in time):
      - if Δ too small it will not encompass entire locality.
      - if Δ too large it will encompass several localities.
      - if $\Delta = \infty \Rightarrow$ will encompass entire program.
    - $D = \Sigma\ WSS_i \equiv$ total demand frames
    - if $D > m \Rightarrow$ Thrashing
    - Policy if D > m, then suspend one of the processes

Allocation of Frames

- The Working Set Strategy
  - Working-Set Model.



Allocation of Frames

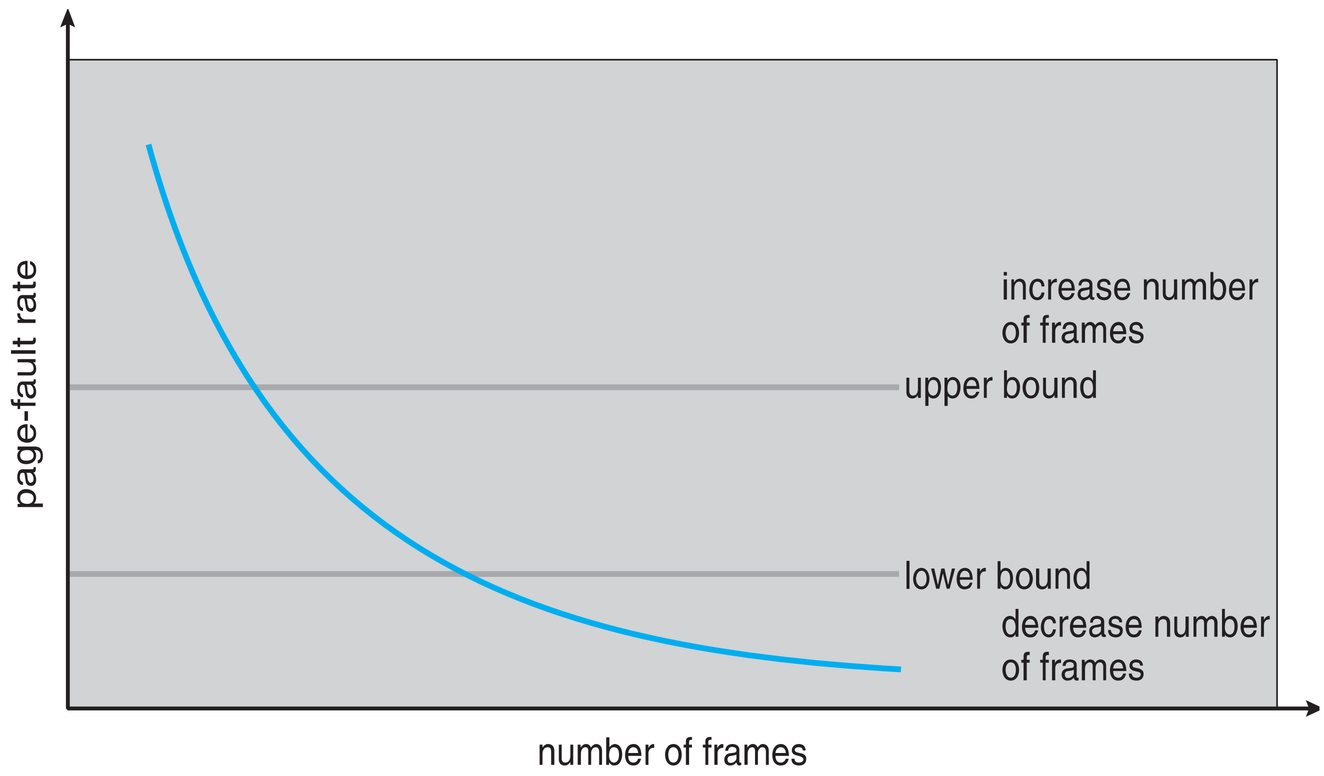
- The Working Set Strategy
  - Keeping Track of the Working Set
    - Approximate with interval timer + a reference bit.
    - Example: $\Delta = 10{,}000$
      - Timer interrupts after every 5000 time units.
      - Keep in memory 2 bits for each page.
      - Whenever a timer interrupts, copy and set 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.

Allocation of Frames

- The Working Set Strategy
  - Practical problems with this working set strategy:
    - measurement of the working set for each process is impractical:
      - necessary to time stamp the referenced page at every memory reference.
      - necessary to maintain a time-ordered queue of referenced pages for each process.
    - the optimal value for D is unknown and time varying.
  - Solution: rather than monitor the working set, monitor the page fault rate!

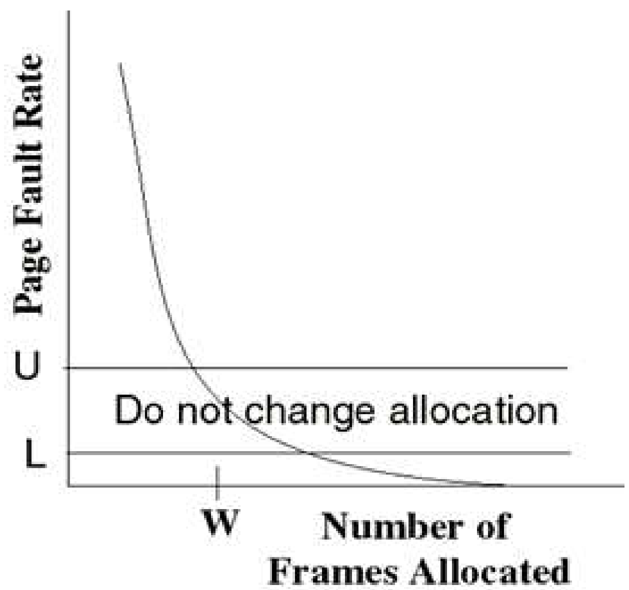
Allocation of Frames

- Page-Fault Frequency (PFF) Scheme
  - Establish “acceptable” page-fault rate:
    - If actual rate too low, process loses frame.
    - If actual rate too high, process gains frame.



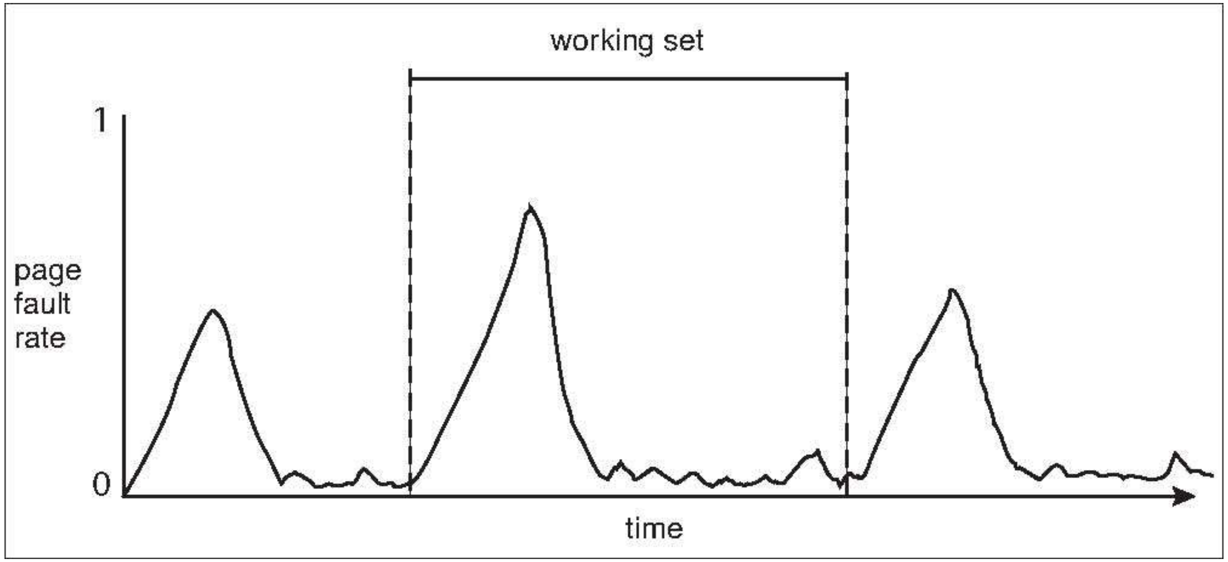
Allocation of Frames

- Page-Fault Frequency (PFF) Strategy
  - Define an upper bound U and lower bound L for page fault rates.
  - Allocate more frames to a process if fault rate is higher than U.
  - Allocate less frames if fault rate is less than L.
  - The resident set size should be close to the working set size W.
  - We suspend the process if the PFF > U and no more free frames are available.



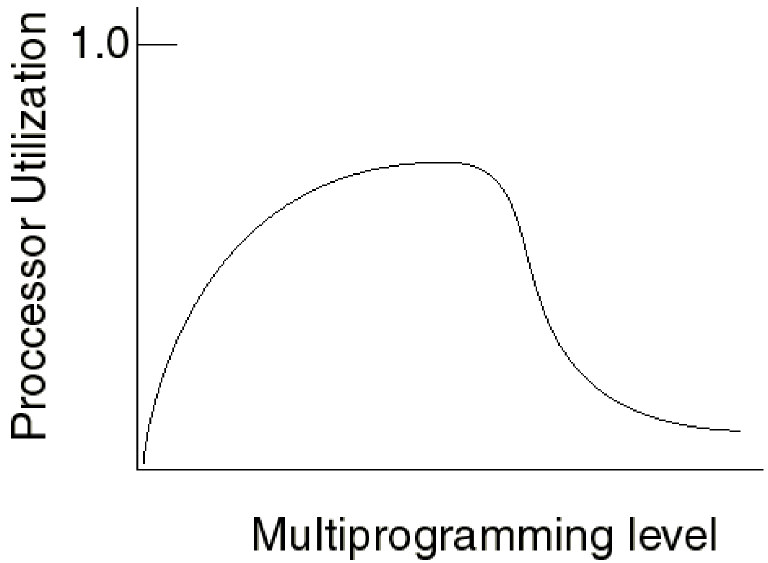
Allocation of Frames

- Working Sets and Page-Fault Rates
  - Direct relationship between working set of a process and its page-fault rate
  - Working set changes over time
  - Peaks and valleys over time



Load Control

- Determines the number of processes that will be resident in main memory (i.e., the multiprogramming level):
  - Too few processes: often all processes will be blocked and the processor will be idle.
  - Too many processes: the resident size of each process will be too small and flurries of page faults will result: thrashing.



Load Control

- A working set or page fault frequency algorithm implicitly incorporates load control:
  - only those processes whose resident set is sufficiently large are allowed to execute.
- Another approach is to adjust explicitly the multiprogramming level so that the mean time between page faults equals the time to process a page fault:
  - performance studies indicate that this is the point where processor usage is at maximum.

Load Control

- Process Suspension
  - Explicit load control requires that we sometimes swap out (suspend) processes.
  - Possible victim selection criteria:
    - Faulting process
      - this process may not have its working set in main memory so it will be blocked anyway.
    - Last process activated
      - this process is least likely to have its working set resident.
    - Process with smallest resident set
      - this process requires the least future effort to reload.
    - Largest process
      - will yield the most free frames.