CSCI 5103
Operating Systems

Instructor: Abhishek Chandra

Memory Management
- Frame Allocation
- Thrashing and Working Set Model
- Memory Management Issues
- Kernel Memory Allocation

Frame Allocation
- Fixed number of frames in physical memory
- Varying number of processes:
  - Different memory requirements
  - How to allocate frames to each process?

Bounds on Number of Frames
- What is the minimum number of frames per process?
- Factors to consider:
  - Performance: Want to keep minimum memory required by process
  - Architecture: Some instructions may need to touch multiple pages simultaneously
  - What is the maximum number of frames per process?
Frame Allocation Algorithms

- Equal allocation:
  - Each process gets m/n frames for m total frames, n processes
  - Problem?

- Proportional allocation:
  - Each process gets no. frames in proportion to its total size
  - Problem?

Local vs. Global Allocation

- Local Allocation:
  - A process can replace a page from its allocation only
  - Benefit?
  - Problem?

- Global Allocation:
  - A process can replace a page from another process

Thrashing

- Process/system spending more time paging than executing
- How could thrashing occur?
  - Process does not have enough pages for its need
  - Process may borrow pages from other processes causing them to thrash

Preventing Thrashing

- Option 1: Use local page replacement
  - Can this still impact other processes?
- Solution: Provide each resident process enough pages based on its need
  - Working-Set Model
  - Page-Fault Frequency
Locality Principle

- Locality: Set of pages accessed together
  - E.g.: function code, local vars, referenced data
- Process execution: moves between localities
  - E.g.: moves from one function call to another, moves from one part of data to another
- Goal: Allocate enough frames to fit current locality of process

Working Set Model

- Tries to identify the current locality of a process
- Working set: Set of pages in the “most recent” page references
- How do we define “most recent”?  
  - Parameter $\Delta$: Window size
  - How to choose $\Delta$?
- Computing working set:  
  - Similar to LRU-approximation: Can use reference bit vectors

Working Set Model

- To avoid process thrashing:  
  - Assign number of frames = working set size
- To avoid system thrashing:  
  - Sum of working sets of all processes $\leq$ total frames
- What if this condition is violated?  
  - OS has to suspend and swap out one (or more) process

Page Fault Frequency

- Working set may be difficult to determine. Why?
- What is the symptom of thrashing?  
  - High page fault frequency (PFF)
- How is the page fault rate related to working set?  
  - High PFF $\Rightarrow$ not enough frames
  - Low PFF $\Rightarrow$ too many frames
**Page Fault Frequency**

- Thresholds on PFF:
  - High watermark: Increase number of frames
  - Low watermark: Decrease number of frames
- PFF may temporarily increase when:
  - Process changes localities
  - How to handle such temporary spikes?

**Memory Management Issues**

- Architectural: Page Size
- Application-side: Program Structure

**Page Size**

- What should be the right page size?
- What if page size is large?
  - Smaller page table
  - More efficient disk I/O
  - Fewer page faults
- What if page size is small?
  - Less internal fragmentation
  - Better resolution to match program locality

**Program Structure: Example**

```c
int A[1024][1024];
for (i=0; i<1024; i++)
  for (j=0; j<1024; j++)
    A[i][j]=1;
```

```c
int A[1024][1024];
for (j=0; j<1024; j++)
  for (i=0; i<1024; i++)
    A[i][j]=1;
```

- How many page faults would happen?
  - Assume page size=4K, num frames < 1024
  - Assume array is page-aligned
Program Structure Impact

- Program structure can impact performance:
  - Result in more page faults
- Causes:
  - Memory access patterns. E.g.: poor locality of reference
  - Choice of data structures. E.g.: array vs. linked list
  - Choice of language. E.g.: Too many pointers/references

Program Structure: Compiler Techniques

- Compiler optimizations. E.g.: Loop interchange
- Separate read-only code from data
- Loader: Place functions within page boundaries, group together frequent caller-callee functions

Kernel Memory Allocation

- Memory allocation techniques so far can lead to:
  - Non-contiguous memory allocation
  - Internal fragmentation
  - Swapping
- These techniques fine for user memory allocation
  - How about kernel memory allocation?

Kernel Memory Requirements

- Kernel pages should typically reside in memory
  - Pinned to physical memory
  - Must conserve memory
- Kernel data structures are variable size and/or small
  - Avoid internal fragmentation to save space
- Kernel memory may have to be physically contiguous
  - For correct functionality. E.g.: to receive block of data from an I/O device
  - For efficiency reasons. E.g.: what if PCB straddles non-contiguous pages?
Kernel Memory Allocation

- Kernel free-memory pool:
  - Separate from user memory pages
  - Physically contiguous set of frames
- Allocation Techniques:
  - Buddy System
  - Slab Allocation

Buddy System

- Contiguous block of memory:
  - Can be partitioned recursively into equal-size chunks
  - Buddies: Set of neighboring chunks
- Power-of-2 allocator:
  - Allocate a chunk rounded up to next power-of-2
- Coalescing:
  - Combining two buddies into one contiguous chunk
- Problem?

Slab Allocation

- Slab: Set of contiguous frames
- Cache: Set of kernel objects of same type
  - E.g.: cache of PCBs, cache of locks, etc.
- Cache initialization:
  - Memory pre-allocated from one or more slabs
  - All objects marked as free
- Object creation:
  - Assign a free object from a cache
  - Prefer a partially used slab
  - Allocate more slabs as needed

Slab Allocation: Benefits

- Fast memory allocation
  - Memory is pre-allocated
  - Only need to mark objects as used/free
- No internal fragmentation
  - Each object is pre-assigned to its actual size
  - Some space may be wasted in a slab