CIS3110 Summary Notes - March 4, 2025

1. Assignment Clarifications

Assignment 3 (Threads vs. Processes)

- A3 involves mapping A2 code (processes-based) to a threads-based implementation
- Grading will focus on the mapping implementation, not functionality issues from A2
- Only core functionality (like makefile working, having functional code) is required from A2

Assignment 2 Debugging Tips

- For randomness issues: Remove randomness by using a fixed list of integers
- Use random seed setting to get consistent behavior:

  ```c
  srand(42); // Replace 42 with any constant value for consistent results
  ```

- When using random numbers with threads, protect the random generator as a critical section
- Default random seed uses time(0) and other system values, causing different sequences each run

Token Ring Implementation Approach

- Focus on solving producer-consumer relationship between nodes
- Implement byte-level communication first, then move to full ring communication
- Prevent deadlocks by making sender node behave differently than other nodes
- Use if-statements to differentiate sender from non-sender nodes
- Standard producer-consumer pattern:

  ```
  wait(empty)
  // send data
  signal(filled)

  wait(filled)
  // receive data
  signal(empty)
  ```

Process vs. Thread Structure

- In A2: One parent process + children processes (one per node), communicating through shared memory
- In A3: One process with multiple threads (one per node)
- Threads automatically share the data and text segments
- Each thread has its own private stack
- With threads, synchronization is needed for any shared data access
- Conversion: fork → pthread_create, wait → pthread_join, exit → pthread_exit

2. Memory Management (Chapters 9 & 10)

Page Replacement Policies (10.4, 10.5)

- **Clock Algorithm** recap: Uses use bits and dirty bits with a virtual "hand" going around to select pages
- **Working Set vs. Resident Set**:
  - Working Set: Pages needed for efficient execution
  - Resident Set: Pages currently in memory
  - Problem occurs when working set > resident set (continuous page swapping)
  - Inefficient when working set < resident set (wasting memory)

Global vs. Local Page Replacement (10.4)

- **Global Policy**: Apply algorithm across all pages regardless of process ownership
  - Process A may lose frames to satisfy Process B
  - All frames in one list
- **Local Policy** (more common): Each program gets a subset of frames
  - Algorithms can be tailored to specific process needs
  - Example: Database joins benefit from Most Recently Used, text editors don't

Variable Partition Policies (10.5, 10.6)

- Dynamically adjust memory allocation based on program behavior
- Programs often need different amounts of memory at different stages
- OS monitors page fault rate to adjust memory allocation:
  - High fault rate → significantly increase allocation (often exponentially)
  - Low fault rate → slowly decrease allocation (linearly, ~5%)

Memory Utilization Curve

- Traditional view: Inverse relation between page frames and page faults
- Modern reality: Most systems operate in the flat part of the curve
- When memory is highly constrained, exponential degradation occurs
- Emergency response: Swapping vs. Paging
  - **Paging**: Moving individual pages to/from disk (normal operation)
  - **Swapping**: Moving entire processes to disk (emergency measure)

Memory Optimization Techniques (9.4, 9.5)

- **Read-Only Pages**: Can be shared between processes with single frame
- **Copy-on-Write**: Delay page copying until a write actually occurs
  - After fork(), parent and child share pages until one attempts to modify
  - Only then is a private copy made for the modifying process
  - Saves resources by avoiding unnecessary copying
  - Especially effective before exec() calls

Address Space Adjustments (9.2, 9.3)

- **Stack Growth**: Downward in memory
  - Function calls may require additional pages
  - Allocated as needed
- **Heap Growth**: Upward in memory
  - Malloc/new operations request more memory
  - Kernel functions:
    - brk(): Sets absolute end of data segment
    - sbrk(): Relative adjustment to data segment
- New pages are zeroed for security
- Page allocation is block-wise (minimum one full page)
- Segmentation faults occur only when accessing beyond allocated pages

Memory Allocation Algorithms (Assignment 1 Recap)

- First Fit: Use first space large enough
- Best Fit: Use smallest space that fits

- Worst Fit: Use largest available space
- Performance depends on allocation patterns:
    - Same-sized structures → Best fit works well
    - Mixed sizes (especially strings) → First fit is generally good enough
    - Linux uses Buddy System algorithm

Next Class

- Finishing memory management
- Starting I/O Systems (Chapter 12)