CIS3110: Operating Systems - Lecture 3 Summary Notes

Assignment 2: Token Ring Network Overview

- The assignment models a token ring network using processes as nodes (Processes 3.4, 3.5)
- Token ring is a networking strategy for moderate to long-haul networks
  - Example: University of Guelph is a node on ORANO network connecting institutions in Southern Ontario
- Each node (computer) is connected to others in a ring with unidirectional flow
- Connections between nodes are modeled as shared memory buffers (6.5, 6.6)
- Communication protocol:
  - Only the process with the "token" (like a baton) can send messages
  - Token passes around the ring continuously
  - Message format: receiver ID, sender ID, message length, and data
  - Messages travel around the ring until they reach intended recipient
  - Recipient copies message while passing it on
  - Sender removes message after a full loop to prevent infinite circulation

Semaphores and IPC (Continued from previous lecture)

- Assignment uses POSIX semaphores for synchronization (6.1, 6.2, 6.4)
- SEM_WAIT and SEM_SIGNAL macros wrap Linux semaphores
- Potential deadlock issues in ring structures (8.2, 8.3)

Signals (6.4, 6.5)

- Signals are interrupt-like structures for inter-process communication
- Unlike interrupts or traps, signals are for process-to-process communication
- Process A sends a signal to Process B to indicate an event occurred
- Process B can register signal handlers (functions) to respond to specific signals
- When a signal is received, the process stops execution, runs the handler, then resumes
- Common signals and their standard numbers:
  - 1 (SIGHUP): Hang up - terminal disconnection or config reload for servers
  - 2 (SIGINT): Interrupt - generated by Ctrl+C, default handler calls exit()
  - 3 (SIGQUIT): Quit - graceful termination

    - 4 (SIGILL): Illegal instruction - CPU cannot execute the instruction
    - 5 (SIGTRAP): Trap - used by debuggers
    - 9 (SIGKILL): Kill - cannot be overridden, forcefully terminates process
    - 11 (SIGSEGV): Segmentation violation - accessing invalid memory
    - 15 (SIGTERM): Termination signal
    - 30/31 (SIGUSR1/SIGUSR2): User-defined signals
- Historical context:
    - Signals predate Unix
    - Different implementations (Bell Labs and Berkeley) led to different numbering systems
    - The `kill` command can send any signal type, not just termination signals
- Signal handler implementation:
    - Include `signal.h`
    - Create a function with signature: `void handler_name(int signal_code)`
    - Register the handler using `signal(SIGNAL_TYPE, handler_function)`
    - The function returns the previous handler (can be saved if needed)

Pipes (12.2, 12.3)

- Pipes are shared buffers managed by semaphores for inter-process communication
- Used when typing vertical bar (|) between commands in shell
- Implemented as producer-consumer relationships
- Fixed-size buffer where one process writes, another reads
- Writer blocks when buffer is full; reader blocks when buffer is empty
- Operates like a file in memory with two file descriptors (read/write)

File Descriptors and I/O

- File descriptors are integer indexes into the process's open file table
- Standard file descriptors:
    - 0: Standard input
    - 1: Standard output
    - 2: Standard error
- Lower-level operations (read/write) use file descriptors

- Higher-level formatted I/O (printf, scanf) use FILE* structure
- The "F" in functions like fprintf stands for "formatted"
- File descriptor interface extends to many byte-stream entities:
  - Regular files
  - Pipes
  - Sockets (network communications)
  - And other I/O streams

Process Scheduling (5.1, 5.2, 5.3)

- Scheduling determines which process runs next on an available CPU
- When a process is running, it may stop due to:
  - Requesting I/O (blocked state)
  - Calling exit() (terminated)
  - Time slice expiration (preemption)
- Scheduling algorithm:
  1. When a process stops running, it's marked as runnable or blocked
  2. Process context (CPU registers) is saved in the process table
  3. Kernel searches for another runnable process
  4. Selected process is marked as running
  5. Process context is restored to CPU
- Scheduling algorithms/queuing disciplines:
  - FIFO (First In, First Out) / Round Robin
    - Simple queue implementation (often a linked list)
    - Processes cycle through the queue
    - Fair but not necessarily efficient
  - Shortest Job First
    - Mathematically optimal for throughput
    - Unfair and impractical (long jobs may never run)
    - Requires knowing job runtime in advance
- Preemptive vs. Non-preemptive Scheduling:
  - Preemptive: Uses alarm clock to limit time slices
    - Better for interactive systems

    - Smoother user experience
  - Non-preemptive: Processes run until completion
    - More efficient (fewer context switches)
    - Poor responsiveness for interactive use
- Priority considerations:
  - Nice utility allows adjusting process priority
  - Dynamic priority updates based on recent behavior
  - Time slice granularity trade-offs:
    - Fine granularity: smoother response but more overhead
    - Coarse granularity: better throughput but choppier response

Memory Management (Preview)

- Brief mention that memory topics will be covered next, including RAM and virtual memory concepts (9.1-9.5, 10.1-10.6, 10.8)