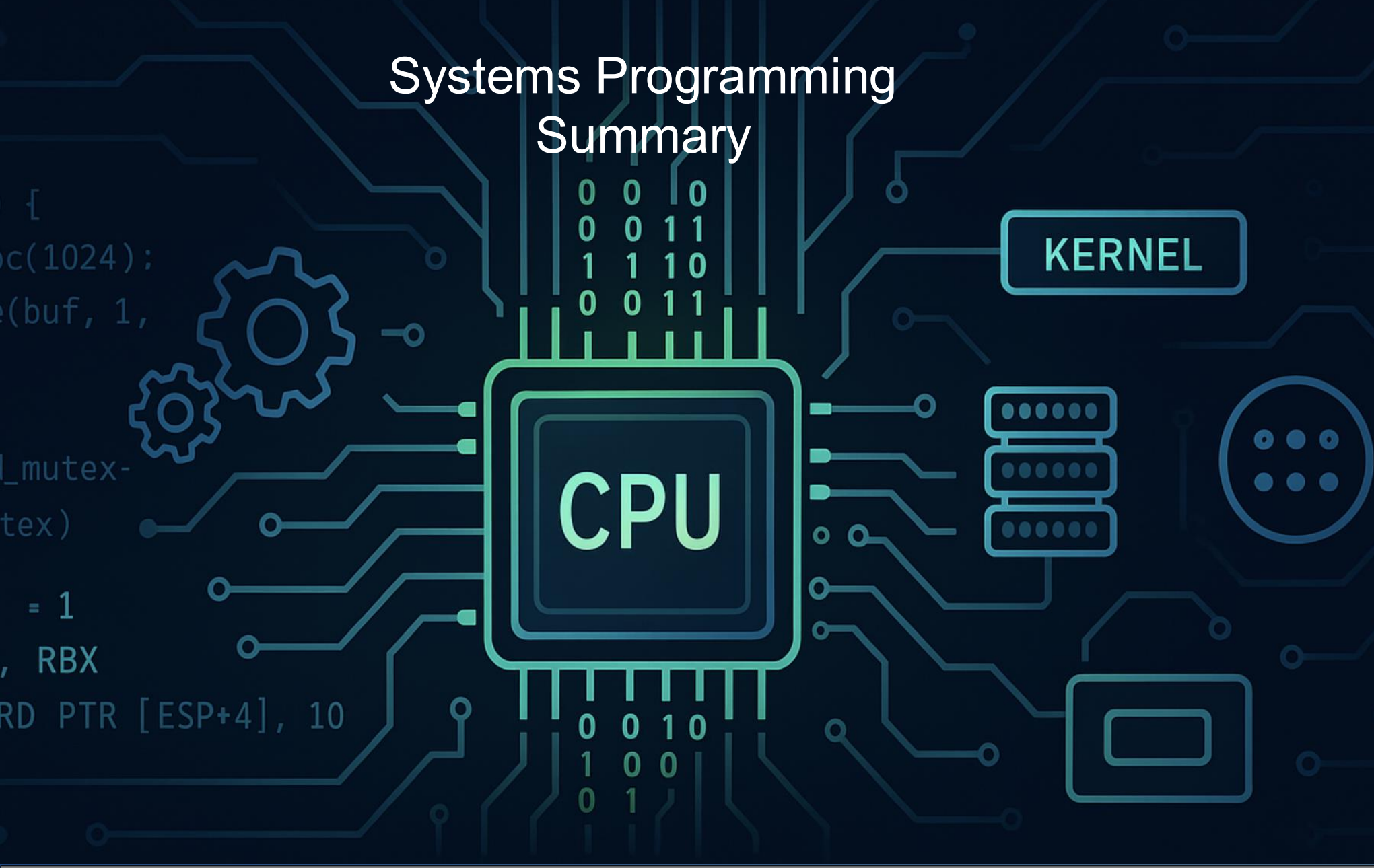
Systems Programming Summary



What we have learned through lectures?

- 01: Tour of Computer System
- 02: Processes: fork, exec, wait, errno
- 03: Signals
- 04: Scheduling
- 05: Memory Management: malloc, free, first/worst/best fit
- 06: Virtual Memory: paging, segmentation

Mid-term

- 07: Threads: thread, data race
- 08: Synchronization: deadlock, mutex, etc
- 09: Files
- 10: Networking: sockets
- 11: Inter-process Communication: pipes, shared memory
- 12: Cryptography: zero-key, one-key, two-key

What we have learned through assignments?

- A0: Command-line Interfaces (CLI)
- A1: Neovim
- A2: Bash Script
- A3: Compilation
- A4: Build Systems – make/cmake
- A5: Debugging – Assertions, Sanitizers, Fuzzers
- A6: Memory Layout
- A7: Memory Layout, Data Types, Padding and Alignment
- A8: Implement a Simple Shell
- A9: Implement a Memory Allocator
- A10: Implement a Simple Map-Reduce Systems (Multi-thread Programming)
- A11: Implement a Group Chat Server & Client (Network Programming)
- A12: Implement Blockchain Mining (Cryptography)

01: Tour of Computer System

- OS Stack is the layers of service
  - Hardware, Kernel, Application.
- Memory hierarchy
  - allows programs to access large memories quickly
- Pointers hold addresses,
  - 32 vs 64 bits limit how much memory we can access
- Kernel mode gives OS kernel access to all resources
  - User mode limits what an application can do.
- Applications use the OS's ABI to use services
- Virtualization allows parts of the OS stack to be swapped out under software control.

02: Processes (1/2)

- Processes are programs executing from memory (RAM)
  - Each process has its own Memory Space
- C Programming
  - Use man pages to lookup functions
  - Pointers and pointers-to-pointers used as output parameters
- sleep() puts function to sleep
- Waiting on your children: wait(), waitpid()
  - Pass &wstatus to find out why child terminated.
  - Terminated process becomes a zombie until waited on.
  - Terminating the parent creates orphans processes.
- Use errno to find out info
  - Print error message to screen with perror().

02: Processes (2/2)

- Create a new process using fork()
  - Clones current process.
  - fork() returns twice:
    - Parent knows it's the parent because return PID is non-zero (= the child's PID)
    - Child knows it's the child because return PID is zero
- Replace a running program with exec()
  - Pass in what program you want loaded into the current process.
  - Completely replaces the process's memory space

03: Signals

- Signals are notifications with specific meanings.
  - Allow asynchronous communication.
- Configure to receive using sigaction()
  - Configuration done with a struct
  - Set signal handler with a function pointer
- Send any signal with kill()

(content from 08-Synchronization)

- Reentrant: Correctly runs when called again while running (same thread or different threads)

04: Scheduling

- Scheduler picks what job to run next.
- Algorithms:
  - First come, first serve
  - Shortest job first
  - Shortest remaining time first
  - Round Robin
  - Multilevel queue
  - Multilevel feedback queue
  - Completely Fair Schedule
- Drawing process scheduling diagrams
  - Compute Wait time, average wait time

05: Memory Management

- Memory Segments
  - text, data, BSS, heap, memory mapped, stack, kernel.
  - Program break and effect of brk() and sbrk()
- Memory Allocator
  - Linked list of free memory
  - New free blocks go first in the list
- Fragmentation
  - External Fragmentation
  - Coalescing algorithm
- Block selection algorithms
  - (first, smallest, biggest) fit

Final Exam covers from here

06: Virtual Memory (1/2)

- Virtual Memory
  - Process works only in the virtual memory space.
  - OS can flexibly share memory between processes.
  - Gives process memory isolation.
- Address Translation
  - Converting (virtual) address to physical address.

06: Virtual Memory (2/2)

- Paging
  - Virtual memory broken up into identical size pages (usually 4KB).
  - Physical memory broken up into page frames ("frames").
  - Use page table for address translation.
  - Enable memory sharing (more in 11– Inter-process communication).
  - Internal fragmentation.
- Segmentation
  - Like paging, but different size regions (segments).
  - External fragmentation.
- Page replacement algorithms:
  - Optimal, FIFO, LRU, Second Chance

07: Threads

- Threads
  - Lightweight processes that share a memory space.
  - Always have main thread
- PThread
  - POSIX library / API for threads
  - pthread_create(), pthread_join(), ...
- Data Race
  - When two threads may access the same data at the same time.
  - Race condition: more general; correctness depends on timing or order.

08: Synchronization (1/2)

- Mutex
  - Used for Mutual Exclusion from a critical section.
  - Guarantees only one thread can hold the lock
- Critical Section
  - Area of the code which accesses a shared variable that must not be concurrently accessed from another thread.
- Thread safe: Correctly runs with multiple threads.
- Reentrant: Correctly runs when called again while running (same thread?)
- Deadlock: Two threads blocking each other. Necessary conditions:
  - Hold and wait
  - Circular wait
  - Mutual exclusion
  - No preemption

08: Synchronization (2/2)

- Condition Variable
  - One thread signals another for an event.
  - Paired with a mutex for mutual exclusion.
- Produce-Consumer Pattern: Shared data structure storing waiting items.
- Semaphore
  - Synchronization with a count
- Read-Write Lock
  - Multiple readers allowed; only one writer.
- Classical problems
  - Dining Philosophers: worry about deadlock / livelock
  - Bounded Buffer: elegant semaphore solution

09: Files (1/2)

- 6 Syscalls for File Access: open, write, read, close, lseek, fcntl
- Syscalls vs Library functions
  - write() vs fprintf()
  - Non-buffered vs buffered IO
  - File descriptor (int) vs stream (FILE*)
  - fprintf buffer control: setbuf(), fflush()

09: Files (2/2)

- Everything is a file: Use file operations to access almost anything. (/proc for process info, /dev for devices, /sys for system info)
- Partitions split up disks
- I-Nodes used for metadata about each file/directory.
- Hard/soft links allow two entries for one file.
- Mounting places one file tree inside another.

10: Networking (1/2)

- Network Stack has layers (bottom-up)
  - phy, link, IP, transport, application
- Socket: Connect to communicate across network.
- TCP:
  - Connection-oriented; in-order delivery.
  - Server:
    socket(), bind(), listen(), accept(), read(), write()... close()
  - Client: socket(), connect(), write(), read(), ... close()
- UDP:
  - Connectionless
  - Server: socket(), bind(), recvfrom(), sendto(), ... close()
  - Client: socket(), sendto(), recvfrom(), close()

10: Networking (2/2)

- Use AF_INET for IPv4

```
struct in_addr {
   in_addr_t s_addr;
};

struct sockaddr_in {
   sa_family_t    sin_family;
   in_port_t      sin_port;
   struct in_addr  sin_addr;
   unsigned char   __pad[X];
}
```

- Network Byte Order

–Big-Endian: Biggest byte is first.

```
#include <arpa/inet.h>
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

- Server must likely handle many sockets at once:
  - Can create a new thread per socket.
  - Can use non-blocking IO to busy-wait checking for ready sockets
  - Can use epoll() or select() to have kernel monitor sockets

11: Inter-Process Communication

- Pipes: Send data between two related processes
- FIFO: Send data between unrelated processes
- Message Queue: Send full messages, rather than a byte stream
- Memory Sharing
- mmap(): Creates a memory mapping of a file or some memory.
  - Usually copied by fork()
  - Useful for parent-child shared memory.
  - Types: private/shared, anonymous/non-anonymous
- shm_open()
  - Creates a named shared memory object.
  - Useful for unrelated processes to share memory.

12: Cryptography (1/2)

- Cryptography
  - From plain text, create cipher text that others cannot read or change.
- Types of algorithms
  - 0 Keys: Hash function
  - 1 Key: Symmetric encryption (private-key)
    - Both sides know the same secret key.
  - 2 Keys: Asymmetric encryption (public-key)
    - You share a public key with the world.
    - Anyone can encrypt messages for you using this key.
    - Only you can decrypt messages using your secret private key which matches the public key.

12: Cryptography (2/2)

- Passwords
  - Store salted and hashed passwords to avoid rainbow tables.
- Digest
  - A hash of a document.
- Digital Signatures
  - Sign a hash with a private key.
- Digital Certificates
  - Sign document to show who really owns a public/private key.
  - Chain of trust for distributing certificates.
  - Root of trust built into OS.
- Hash Collisions
  - Duplicate hash (digital signature) is a security issue.
  - Birthday attack to find duplicates.

Last Slide

- Thanks for taking this course!
- Hope you enjoy system programming & sense the beauty of computer system design ☺
- Following schedule:

  - Apr 10 11:50 PM: A12 due (maximum extension to Apr 13)
  - Apr 15 3:30-6:30 PM: final exam

  We will arrange an online score viewing session before reporting grades – time TBA through massmail