201 Course Content Summary (Spring 2025)

This summary is meant to highlight what type of material is and is not testable. Many questions may rely on you understanding and applying this knowledge; it is not sufficient to memorize this list, you must be able to use the material.

1. General thoughts

- You will have to write and read code on the test, as well as know the theoretical content.
- You will not be asked to know:
  - Which header files different functions are in.

2. Lecture content

After Midterm

6. Virtual Memory

- Understand memory organizations from before modern approaches:
  - OS & Process with no isolation; one process.
  - OS & Fixed memory regions (fixed size).
- Know Memory Details
  - Variables and code exist in memory.
  - Instructions operate on memory.
  - Know temporal and spacial locality; know how they are useful.
- Know what is virtual memory vs physical memory
  - What benefits does virtual memory provide?
  - What determines size of virtual memory address space?
- Paging
  - Able to explain what is a page, page frame, and page fault.
  - Able to explain how an OS uses physical memory to give each process its own virtual address space.
  - Explain what is a swap space, how it would work, and why it is useful.
- Paging and Address translation
  - Given the number of bits in a pointer and the page/frame size, able to compute how many pages are in a virtual address space (or extract the bits from an address for the page's number).
  - Given a pointer and above info, able to compute the virtual memory page number, and use address translation table to get physical page number.
- Segmentation
  - Understand the difference between paging and segmentation.
  - Know what are internal and external fragmentation. Know how each affect segmentation and/or paging.
- Page Replacement
  - Know what is demand paging, page fault, page replacement, swapping, and swap space.
  - Know page replacement algorithms:
    - Optimal (and its limitations), FIFO, Least Recently Used (LRU), Second-

Chance
- Know how temporal and spacial locality relate to demand paging
- Know what is thrashing

7. Threads

- Know the difference between a thread and a process in terms of memory space, overhead in creating, isolation between them.
  - Know what section of memory is created for each thread.
- Know POSIX functions to work with threads:
  - pthread_create(): What each arg is used for. Able to pass arguments to a thread. Able to return a value from a thread.
  - pthread_self(), pthread_join(), pthread_detach()
- Data races
  - Know what is a data race, race condition, deterministic output.
  - Know what happens with: `x++`

8. Synchronization

- Know what is synchronization with threads of processes.
- Know mutex locks:
  - Creating, initializing, locking and unlocking a mutex.
  - Able to explain the effect of adding a lock to a given piece of code.
  - Know the blocking behaviour of lock.
- Know atomicity, serialization, and interleaving.
- Able to explain what a critical section is
  - know 3 conditions to satisfying a critical section: mutual exclusion, progress, bounded waiting.
  - Know the difference between a thread-safe function and a reentrant function.
  - Understand techniques to make a function reentrant.
- Know deadlocks and livelocks. Able to explain them, identify them, and sometimes correct them.
- Know four necessary conditions for deadlock, and possible solutions to it.
- Know condition variables: the problem they solve, and how to code with them.
  - Understand the consumer/producer pattern
  - Able to write code that uses a condition variable with a mutex to eliminate race conditions and efficiently notify another thread.
- Know semaphores: the problem they solve, and how to code with them.
- Understand (but do not need to code) read/write locks.
- Understand classic problems such as the dining philosopher's and bounded queue. Able to explain what problems can occur in each.

9. File I/O

- Know how to work with unbuffered file I/O:
  - open(), read(), write(), lseek(), close()
  - Nothing on fcntl()
  - Know file descriptors, file offsets
- Know buffered I/O
  - Know file stream vs file descriptor

- Know benefit of using buffered I/O
- Know I/O functions listed on reference page.

- Know blocking vs non-blocking I/O
- Understand Linux's philosophy of everything is a file.
- Understand disk partitions, I-Nodes,
- Know hard vs soft links; know which creates an actual file.
  - Know how to create each type of link with ln terminal command.
- *Nothing on setuid, setguid, sticky.*
- Know about virtual file systems, mount and unmount.

10. Networking

- Understand the networking stack: Physical, Link(MAC), IP, Transport, Application.
  - Know what each layer does.
  - Know MAC address, IP address, IP Port
  - Know UDP and TCP, what the abbreviations mean, and what they do.
- Know sequence of function calls for TCP & UDP, client & server.
- Know how to implement a TCP server (multi-threaded), TCP client, UDP server, UDP client
- Network Socket
  - Know what each of AF_UNIX and AF_INET do, and able to write socket programs using them.
  - Understand how to convert between IP address and name using inet_pton(), inet_ntop()
  - Know network byte order, htonl(), htons(), ntohl(), ntohs()
  - Understand getaddrinfo(), getnameinfo().
- Know techniques to handle multiple clients; know benefits and drawbacks of each.
  - Single threaded, one at a time (able to code it).
  - Multi-threaded, ore per connection (able to code it).
  - Single threaded, non-blocking reads/writes (able to code it).
  - Single threaded, using epoll() or select() (no need to code it).

11. Inter-Process Communication

- Pipes
  - Know the purpose of a pipe, and when they are usable.
  - Know how to create, open, use, and close pipes.
  - Know how to use a pipe between a parent and child process.
  - Know which file descriptor is for read and which is for write.
  - Know what happens when the write end of the pipe is closed.
  - Know duplicating file descriptors with dup2(): what is the purpose? How is it used?
- FIFO
  - Know the purpose of a FIFO.
  - Understand how to use a FIFO.
- Message Queue
  - Understand the purpose of a message queue and how it differs from a pipe and FIFO.
  - *Do not have to write code for a message queue.*

- Memory Mapping
  - Know that not setting the MAP_ANONYMOUS flag loads a file into memory.
  - Know that setting the MAP_ANONYMOUS flagt allocates empty memory.
  - Know that setting the MAP_SHARED flag shares the memory with other (child?) processes, and if it's a file, writes changes to the file.
  - Know that setting the MAP_PRIVATE flag makes the memory local to this process, and if it's a file, does not write changes to disk.
  - Know the four uses of mmap()
  - Know to unmap file with munmap()
- Shared Memory with shm_open()
  - Know purpose of using shm_open() over just MAP_ANONYMOUS memory.
  - Know how to use shm_open() with mmap().

12. Cryptography

- Know CIA: Confidentiality, Integrity, Availability
- Know what is meant by plain text, cipher text, encryption, decryption.
- Know the problem with traditional cryptography (with a secret algorithm) vs modern cryptography with a public algorithm but a secret key.
- Know the idea of a window of validity.
- Hash functions
  - Know their purpose.
  - Know the four desired properties: easy to compute, one way, weak collision resistance, strong collision resistance.
- Know Private key cryptography: how it works (in general).
  - Know encryption and decryption both need the same private key.
- Know Public key cryptography: how it works (in general).
  - Know that encrypting with one key requires decrypting with the other key.
  - Know how it can be used to keep a secret, verify a sender, or both.
- Storing passwords
  - Know systems only store password hashes.
  - Know how salts are used, and how they guard against rainbow tables.
- Verifying Documents
  - Know what a secure digest is, how it is computed (in general), and when it is useful.
  - Know what a digital signature is, what it is used for, and how it is used by the signer and verifier.
- Digital Certificates
  - Know what a digital certificate is, and why it is useful.
  - Know the process used for creating and verifying a digital certificate.
  - Know root of trust, and chain of trust.
- Birthday Attack
  - Know the birthday attack.
  - Know why a hash collision is a problem for cryptographic hash functions.
  - Know the process an attacker may use to exploit a weak hash function.
  - Understand how strong collision resistance is needed to combat this type of attack.
- Know how a block chain functions, as based on A12.

3. Cheatsheet for Final

```
- Thread functions: pthread.h
    int pthread_create(pthread_t *thread, pthread_attr_t *attr,
         void *(*funct)(void *), void *arg);
    void pthread_exit(void *retval);
    int pthread_join(pthread_t thread, void **retval);
    int pthread_detach(pthread_t thread);
    pthread_t pthread_self(void);

- Mutex: pthread.h
    pthread_mutex_t myLock = PTHREAD_MUTEX_INITIALIZER;
    int pthread_mutex_lock(pthread_mutex_t *mutex);
    int pthread_mutex_trylock(pthread_mutex_t *mutex);
    int pthread_mutex_unlock(pthread_mutex_t *mutex);

- Condition Variables: pthread.h
    pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
    pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
    pthread_cond_signal(pthread_cond_t *cond);
    pthread_cond_broadcast(pthread_cond_t *cond);

- Semaphores: semaphore.h
    sem_t sem;
    sem_init(sem_t *sem, int pshared, unsigned int value);
        // pshared = 0 for threads; 1 for processes.
    sem_wait(sem_t *sem);
    sem_post(sem_t *sem);

- Read-Write Lock: pthread.h
    pthread_rwlock_rdlock(pthread_rwlock_t *rwlock);
    pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);

- Unbuffered I/O: fcntl.h, unistd.h
    //Defined fd's: STDIN_FILENO, STDOUT_FILENO, STDERR_FILENO
    int open(const char *pathname, int flags);
        // flags: O_RDONLY, O_WRONLY, O_RDWR, O_APPEND, O_CREAT,
O_NONBLOCK
    int open(const char *pathname, int flags, mode_t mode);
        // mode, when creating: S_IRWXU, S_IRUSR, S_IWUSR, S_IRWXG
    ssize_t write(int fd, const void *buf, size_t count);
    ssize_t read(int fd, void *buf, size_t count);
    int close(int fd);
    off_t lseek(int fd, off_t offset, int whence);
        // whence = SEEK_SET, SEEK_CUR, SEEK_END
    int fcntl(int fd, int op, ... /* arg */ );

- Buffered I/O: stdio.h
    // Defined FILE: stdin, stdout, stderr
    FILE *fopen(const char *pathname, const char *mode);
    int fclose(FILE *stream);

    int fprintf(FILE *stream,
                  const char *format, ...);
    int fscanf(FILE *stream,
```



```
                const char *format, ...);
    int fputc(int c, FILE *stream);
    int fputs(const char *s, FILE *stream);
    int fgetc(FILE *stream);
    char *fgets(char s[.size], int size, FILE *stream);
    FILE *fdopen(int fd, const char *mode);
    int fileno(FILE *stream);
    int fflush(FILE *stream);


Networking: sys/socket.h
    int socket(int domain, int type, int protocol);
        // domain: AF_UNIX, AF_INET, AF_INET6
        // type: SOCK_STREAM, SOCK_DGRAM
        // protocol: 0
    int bind(int socket, const struct sockaddr *address, socklen_t
address_len);
    int listen(int socket, int backlog);
    int accept(int socket, struct sockaddr *address, socklen_t
*address_len);
    int connect(int socket, const struct sockaddr *address, socklen_t
address_len);

    ssize_t recvfrom(int socket, void *buffer, size_t length,
           int flags, struct sockaddr *address, socklen_t *address_len);
    ssize_t sendto(int socket, const void *message, size_t length,
           int flags, const struct sockaddr *dest_addr, socklen_t
dest_len);
    ssize_t recv(int sockfd, void *buf, size_t len, int flags);
        // flags: MSG_DONTWAIT, MSG_PEEK
    ssize_t send(int sockfd, const void *buf, size_t len, int flags);
        // flags: MSG_DONTWAIT

    struct sockaddr_un {
        sa_family_t sun_family;        /* AF_UNIX, AF_INET, AF_INET6 */
        char        sun_path[108];     /* Pathname */
    };

    struct in_addr {
        in_addr_t s_addr;
    };
    // in_addr_t constants: INADDR_LOOPBACK, INADDR_ANY

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

IPC: unistd.h
    int pipe(int fildes[2]);
    int dup2(int oldfd, int newfd)
    int mkfifo(const char *pathname, mode_t mode)
    int unlink(const char *path);
```



```
void *mmap(void *addr, size_t length, int prot, int flags,
    int fd, off_t offset);
    // addr usually NULL
    // prot:
    // flags: MAP_SHARED, MAP_PRIVATE,
    //        MAP_ANONYMOUS (offset=0, fd=-1 if not shm_open())
int munmap(void *addr, size_t length);
int shm_open(const char *name, int oflag, mode_t mode);
    // oflag: O_RDONLY, O_RDWR, O_CREAT
    // mode, when creating: S_IRWXU, S_IRUSR, S_IWUSR, S_IRWXG
int ftruncate(int fd, off_t length)
int shm_unlink(const char *name)
```

