COM S 352 Final Exam Study Guide Fall 2022

List of terms from second half reading/lecture

- Deadlock
    - Dining Philosophers problem
    - Conditions for deadlock
        - Mutual exclusion
        - Hold-and-wait
        - No preemption
        - Circular wait
    - Total ordering
    - Deadlock avoidance
    - Detect and recover
- File Systems
    - I/O Computer Architecture
        - CPU
        - Memory
        - Bus(es)
        - Peripheral/Device
        - Device Controller
        - Device Driver
        - Interrupt
        - DMA (Direct Memory Access)
    - Hard Disk Drive (HDD)
        - Sectors
        - Atomic Write
        - Torn Write
        - Platter
        - Surface
        - Track
        - Spindle
        - Disk Arm
        - Disk Head
        - Rotational delay
        - Seek time
        - Random access
        - Sequential access
        - Shortest seek time first
    - File
    - Directory
    - Root directory
    - Absolute path

  - File descriptor
  - System calls
    - open()
    - read()
    - write()
    - close()
    - lseek()
  - hard link
  - soft link
  - metatdata region (simple file system)
  - data region (simple file system)
  - contiguous allocation
  - block allocation
  - inode
  - data bitmap
  - inode bitmap
  - direct indexing
  - indirect indexing (single, double and triple)
  - multiple-level indexing
  - caching (file system blocks)
  - write buffering
  - Berkeley Fast File System (FFS)
  - Block groups
  - Path locality
  - Fsck
  - Journaling (write-ahead logging)
    - Metadata
    - Transaction
    - Commit
    - Checkpoint
  - Flash-based SSDs
    - Cell, page, and block
    - Read page
    - Erase block
    - Program page
    - Flash translation layer (FTL)
    - Log-structured FTL
    - Garbage collection
    - Ware leveling
- Virtual Machines
  - Virtual Machine Monitor (VMM) or hypervisor
  - Goal of transparency
  - Supervisor mode
  - VMM page table - “physical memory” vs machine memory

    - Information gap between host OS and VMM
    - Para-virtualization
- Security
    - Confidentiality
    - Integrity
    - Availability
    - Authenticity
    - Intruder
    - Vulnerability
    - Threat
    - Attack
    - Malware
    - Trojan horse
    - Spyware
    - Ransomware
    - Sniffing
    - Masquerading
    - Man-in-the-middle
    - Denial of service (DoS)
    - Access Control
        - Object (access control)
        - Subject (access control)
        - Access mode
        - Access Control Matrix
        - Access Control List
        - Capability
    - Encryption
    - Decryption
    - Plaintext
    - Ciphertext
    - Symmetric (key system)
    - Asymmetric (key system)
    - Public key
    - Private key
    - Authentication
    - Nonce
    - Defense in Depth (design principle)
    - Layered Architecture
    - Trust
    - Trusted Computing Base (TCB)
    - Trusted Platform Module (TPM)
    - Virtual Machine
    - Sandbox
    - Honeypot

- Networking
    - Protocol
    - Link
    - Packet
    - Switch
    - Host
    - Router
    - Client/server
    - OSI Model
    - Layers of OSI Model
        - Application
        - Transport
        - Network
        - Data Link
        - Physical
    - Socket
    - IP Address
    - Port
    - TCP
    - UDP
    - Bind, listen and accept (what a server does)
    - connect (what a client does)
    - bandwidth
    - latency
    - propagation
- Distributed Systems
    - Transparency (design principle of distributed systems)
    - Stateful (protocol)
    - Stateless (protocol)
    - NFS
    - File handle (NFS)
    - NFS commands
        - LOOKUP
        - READ
        - WRITE
        - GETATTR
    - Generation Number
    - Retry on failure
    - Idempotency
    - Cache consistency problem
    - Update visibility
    - Stale cache
    - Flush-on-close (cache)
    - Attribute cache

    - AFS
    - Scalability
    - AFS commands (only need to know these three)
        - TestAuth
        - Fetch
        - Store
    - Whole-file caching
    - Volumes
    - Callback
    - Heartbeat protocol

List of terms from first half reading/lecture

- Processes
    - Program
    - Process
    - Address space
    - Program counter
    - CPU registers
    - Stack pointer
    - PCB (Process control block)
    - Scheduler
    - Running (state)
    - Ready (state)
    - Blocked (state)
    - Context Switch
    - Parent (process)
    - Child (process)
    - Zombie (process)
    - Orphan (process)
    - Process System calls
        - fork()
        - exec()
        - wait()
        - pipe()
        - dup()
    - Pipes
    - Time-sharing
    - Multiprogramming
    - Multitasking
    - User mode (CPU)
    - Kernel mode (CPU)
    - Privileged instruction

    - System call
    - Trap
- Scheduling
    - Job (or CPU burst)
    - I/O burst
    - CPU Bound
    - I/O Bound
    - Preemption
    - Turnaround time
    - Response time
    - Scheduling policies
        - FIFO
        - SJF
        - STCF
        - RR
        - Lottery
        - Stride
        - MLFQ
        - CFS
    - Time-slice (quanta in RR)
    - Oracle (requirement to see into the future)
    - Tickets (Lottery)
    - Starvation
    - Priority boost (MLFQ)
    - Virtual runtime (CFS)
    - Nice value (CFS)
- Memory
    - Address space (virtual)
    - Physical memory
    - Program code
    - Heap
    - Stack
    - Memory virtualization
    - Virtual address
    - Physical address
    - Address translation
    - Base and Bound
    - MMU (Memory Management Unit)
    - Segmentation
    - Segment (address space)
    - Segmentation fault
    - Sparse address space
    - Extern fragmentation
    - Internal fragmentation

  - Compaction
  - Best fit (free-space management policy)
  - Paging (memory)
  - Page table
  - Page table entry
  - Page
  - Frame
  - VPN (virtual page number)
  - PFN (physical frame number)
  - Offset (from start of page)
  - Valid bit (page table)
  - TLB (Translation-Lookaside Buffer)
  - TLB hit
  - TLB miss
  - TLB cache entry
  - Valid bit (TLB cache entry)
  - Spatial locality
  - Temporal locality
  - Memory hierarchy
  - Swap space
  - Block (swap on disk)
  - Present bit (swap)
  - Cache hit
  - Cache miss
  - Page fault
  - Page out
  - Page in
  - High watermark
  - Low watermark
  - Reference string
  - Cold-start miss
  - Replacement Policies
    - OPT (optimal)
    - FIFO (first in first out)
    - Random
    - LRU (least-recently used)
    - Clock Algorithm
  - Belady's Anomaly
  - Dirty bit (swap)
  - Thrashing
- Concurrency
  - Threads
  - TCB (Thread Control Block)
  - Concurrency vs Parallelism

- Race condition
- POSIX
- pthreads
  - pthread_create()
  - pthread_exit()
  - pthread_join()
- Mutex
  - pthread_mutex_lock()
  - pthread_mutex_unlock()
- Mutual exclusion
- test-and-set
- Spin lock
- Spinning loops
- Condition variables
  - pthread_cond_wait()
  - pthread_cond_signal()
- Producer/consumer (bounded buffer) problem
- Semaphore
  - sem_init()
  - sem_wait()
  - sem_post()
- Binary semaphore
- Reader-Writers problem

Module 1

1. For each state transition below, state whether there is a context switch involved. If there is a context switch, describe the processes involved. How are they chosen?

a. ready -> running
b. running -> blocked
c. blocked -> ready

a. There is a context switch. The scheduler replaces the currently running process and replaces it with the next (based on some algorithm) process from the ready queue.
b. There is a context switch. The currently running process is placed into a blocked state, the scheduler replaces it with the next (based on some algorithm) process from the ready queue.
c. There is no context switch. When a process because ready to run it is placed in the ready queue, assuming there is no preemption, the currently running process remains running.

2. Three programs are serviced in a multiprogramming system. Program A contains 50ms of computation followed by 100ms of I/O on hardware device 1. Program B contains 20ms of computation followed by 50ms of I/O on hardware device 2. Program C contains 50ms of computation followed by 100ms I/O on hardware device 2. Each device can service only one I/O request at a time. What is the minimum time it will take to complete all three programs? Create a table like Figure 4.4 in the reading to show the operation of the three programs.

Time	**A**	**B**	**C**	**Notes**
0	**Ready**	**Ready**	**Running**	
50	**Running**	**Ready**	**Blocked**	**C waiting 100ms for Device 2 I/O**
100	**Blocked**	**Running**	**Blocked**	**A waiting 100 ms for Device 1 I/O**
120	**Blocked**	**Blocked**	**Blocked**	**B waiting 50 ms for Device 2 I/O**
150	**Blocked**	**Blocked**	**-**	**C is finished**
170	**Blocked**	**-**	**-**	**B is finished**
200	**-**	**-**	**-**	**A is finished**

All programs finished at 200.

3. Describe the purpose of each of the following terms individually:

- system call
- interrupt
- trap

Now, describe how the three are related to each other.

System Call – call a system library and transfer control to the OS in kernel mode
Interrupt – signal from a hardware source that gives CPU control to the interrupt handler (owned by the OS)
Trap – signal from a software source that gives CPU control to the interrupt handler (owned by the OS)

System calls need to execute in kernel mode, kernel mode can be entered by an interrupt or trap. Software initiates a system call by causing a trap. Hardware (e.g., a device driver or timer) changes control to the OS by an interrupt.

4. The RISC-V architecture that we will use in examples throughout this class allow processors to be implemented with three privilege levels defined as follows (http://docs.keystone-enclave.org/en/latest/Getting-Started/How-Keystone-Works/RISC-V-Background.html):

> *Privilege level defines what the running software can do during its execution. Common usage of each privilege level is as follows:*
>
> - *U-mode: user processes*
> - *S-mode: kernel (including kernel modules and device drivers), hypervisor*
> - *M-mode: bootloader, firmware*

However, the specification allows simple embedded processors to implement just M-mode (i.e., all code on these simple processor implementations must run at the same privilege level).

What is the purpose of multiple privilege levels? Compare and contrast multiple privilege levels vs a single privilege level, name at least one advantage for each.

We don't want the user processes to be able to access each other's memory or the memory of the OS or other resources that they should not have privilege to access. When a process executes in user mode it is restricted it what instructions it can execute. Only OS code executes in kernel mode where there are generally no restrictions. The RISC-V architecture has an additional mode that can execute instructions that are only required during bootloading.

Advantage of single level: With a single-level, OS libraries can be called with simple function calls. System calls are more expensive because they require a trap and context switch.

Advantage of multiple levels: We can restrict what user processes are able to do to protect memory and resources.

5. The following application code demonstrates what is called a “fork bomb”. What negative consequence might this application have on the system? What might an OS do to prevent it?

```
/* WARNING: DO NOT EXECUTE THIS CODE!       */
/* SSG MAY GIVE YOU A STERN WARNING OR WORSE */
int main(int args, char *argv[]) {
   int rc = 0;
   while(rc == 0) {
      rc = fork();
   }
   wait(NULL);
   return 0;
}
```

The result of the loop is that the child forks a new process and this happens recursively. So an infinite number of processes are created. The parents call wait so they never terminate. Eventually the system will run out of memory or process ids.

To prevent it set a maximum number of processes.

6. A shell is an application that enables users to run and control other applications through a command line interface. Using the POSIX API for processes (e.g., fork, exec, wait, dup and pipe)
list the steps a shell might take to execute the following user commands:

a) Start another application. For example, the user invokes the grep application with
two command line arguments like this:
$ grep COMS352 *

1. **Shell calls fork**
2. **Child calls exec with “grep COMS352 *”**

b) Pipe the standard out of one application into the standard in of another application.
For example, the user pipes the output of the grep application into the input of the wc
application like this:
$ grep COMS352 * | wc –l

Module 2

1. Of these two types of programs:

a. I/O bound (interactive)
b. CPU bound

which is more likely to have voluntary context switches, and which is more likely to have nonvoluntary context switches? Explain your answer.

I/O-bound is voluntary. The program is more likely to wait for I/O to become available before it uses up its time slice. So, it is more likely to voluntarily relinquish the CPU. CPU-bound is non-voluntary. The program is likely to use the CPU for its entire time quantum.

2. OSTEP describes the problem of starvation in section 8.2 and shows how priority boost can be used to avoid it in MLFQ. Consider other scheduler policies. Explain why each of the following policies can or cannot result in starvation?

a. FIFO
b. STCF
c. RR
d. Lottery

(a) starvation – job can have infinite loop and starve other jobs
(b) starvation – many short jobs can prevent longer job from getting run time
(c) no starvation – guaranteed execution of every process within (N * time slice)
(d) no starvation – every lottery selection process has chance of executing based on how many tickets it has

3. CPU efficiency can be defined as:

```
process_running_time / (process_running_time + os_overhead)
```

Between STCF and RR which do think will typically have better CPU efficiency? What are the overheads involved? Explain your answer.

RR has timer preemptions that cause extra context switches that STCF will not have. Context switches contribute to significant overhead of the scheduler. Therefore, RR is less CPU efficient.

4. Consider the following set of jobs, with times given in milliseconds:

Job	Arrival Time	Runtime
A	0	20
B	2	20
C	5	5
D	10	10

a. Draw four Gantt charts that illustrate the execution of these jobs using the following scheduling algorithms: FIFO, SJF, STCF and RR (with quantum = 5). The charts must clearly label the job and the start and end time of each run.

FIFO
```
|   A   |   B   | C |  D  |
0       20      40  45    55
```

SFJ
```
|   A   | C |  D  |   B   |
0       20  25    35      55
```

STCF
```
|A|C|  D  |   A   |   B   |
0 5 10    20      35      55
```

RR
```
| A | B | C | D | A | B | D | A | B | A | B |
0   5   10  15  20  25  30  35  40  45  50  55
```

b. What is the average turnaround time for each of the scheduling policies? Show calculations.

FIFO
((20 – 0) + (40 – 2) + (45 -5) + (55 – 10)) / 4

SFJ
((20 – 0) + (55 – 2) + (25 -5) + (35 – 10)) / 4

STCF
((35 – 0) + (55 – 2) + (10 -5) + (20 – 10)) / 4

RR
((50 – 0) + (55 – 2) + (15 -5) + (35 – 10)) / 4

c. What is the average response time for each of these scheduling policies? Show calculations.

FIFO
((0 – 0) + (20 – 2) + (40 - 5) + (45 – 10)) / 4

SFJ
((0 – 0) + (35 – 2) + (20 - 5) + (25 – 10)) / 4

STCF

((0 – 0) + (35 – 2) + (5 - 5) + (10 – 10)) / 4

RR
((0 – 0) + (5 – 2) + (10 -5) + (15 – 10)) / 4

5. Consider the following set of jobs, with times given in milliseconds.

Job	Arrival Time	Runtime
A	0	200
B	5	20
C	10	10
D	50	10

Assume we are using a MLFQ scheduler with 3 priority levels Q2: quanta=10, Q1: quanta=20, Q0: quanta=50.

The MLFQ follows rules 1-4 from OSTEP, it does not have priority boost rule 5.

Draw a Gantt chart that illustrates the execution of the jobs.

Q2
```
| A | B | C |     | D |
0   10  20  30    50  60
```
Q1
```
            |  A  |   | B |
            30    50  60  70
```
Q0
```
                        |            A            |
                        70                        240
```

6. Assume that two processes, A and B, are running on a Linux system. The nice values of A and B are −5 and +5, respectively. If both A and B have the same positive `vruntime`, and neither has had its `vruntime` reset due to sleeping, which has run the longest actual time on the CPU? Explain your answer.

A is less nice so its vruntime grows more slowly with actual time. Therefore, it will run for more actual time than B which has the same vruntime.

Module 3

1. Suppose an MMU manages memory using base and bounds registers. Assume the base and bounds for a particular process are:

base = 22,000
bounds = 800

A program performs the following loads (reads) and stores (writes) from/to memory. What is the physical memory location of each load or store. If the operation will not be allowed explain why.

a. Load from address 0
b. Load from address 500
c. Store to address 900

virtual_address + base = physical_address

a. 0 + 22,000 = 22,000
b. 500 + 22,000 = 22,500
c. 900 is greater than the bounds which means the process is not allowed to access that memory. The MMU will case a trap so the OS can decide what to do with the process that attempted the illegal memory access.

2. What is the size of addressable memory (maximum number of bytes that can be stored) for each of the following memory address sizes?

a. 16-bit address
b. 24-bit address
c. 32-bit address

a. 2^16 = 65,536 bytes, note that addresses will range from 0 to 65,535.
b. 2^24 = 16,777,216 bytes
c. 2^32 = 4,294,967,296 bytes

3. The following program is run to find the location of code, heap, and stack in virtual memory.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    printf("code  %lu\n", main);
    printf("heap  %lu\n", malloc(1));
    printf("stack %lu\n", &argc);
    return 0;
}
```

The output is:

```
code  5287601
heap  17573467
stack 8731878194284
```

State each of these addresses using binary SI units (e.g., 1KB=2^10) rounded to the nearest whole number. For example, 45,876 is 45,876 / 2^10 = 45KB.

5,287,601 / 2^20	**= 5.0MB**	**= 5MB**
17,573,467 / 2^20	**= 16.8MB**	**= 17MB**
8,731,878,194,284 / 2^40	**= 7.9TB**	**= 8TB**

4. Describe how a process' physical memory can be relocated when using base and bounds. How does it compare to static relocation in memory?

To relocate:

1) **If process is currently running cause a trap to switch out of the process.**
2) **Copy the contiguous block of physical memory to another location.**
3) **Update the base register to be the start address of the new location in physical memory.**

Static relocation of physical process memory requires identifying all pointers in the executing program and updating them to point to the new location in physical memory. Without making assumptions about the program identifying pointers in a running program is extremely difficult if not impossible to do.

5. Suppose an MMU manages memory with per segment base and size registers. Addresses are 16 bits in size. The 2 most significant bits encode the segment. The 14 least significant bits encode the offset. Segments are defined as follows:

Segment Number	**Segment Name**	**Base**	**Size**
0	code	0x0100	0x0100
1	heap	0x0A00	0x0200
2	stack	0x1000	0x0100
3	undefined	-	-

For each of the following virtual addresses state where the data will be stored in physical memory.

a. 0x401A
b. 0x0000
c. 0x8001

a. 0x401A = b0100 0000 0001 1010
segment 1: heap
0x001A + 0x0A00 = 0x0A1A

b. 0x0000 = b0000 0000 0000 0000
segment 0: code
0x0000 + 0x0100 = 0x0100

c. 0x8001 = b1000 0000 0000 0001
segment 2: stack
0x0001 + 0x1000 = 0x1001

Module 4

1. For each of the following memory allocation schemes explain why it does or does not experience external fragmentation or internal fragmentation.

a. Base and bounds

External fragmentation: yes
Internal fragmentation: no
Variable sized allocations can lead to free spaces that are not large enough to fit an entire process.

b. Segmentation

External fragmentation: yes
Internal fragmentation: no
Variable sized allocations can lead to free spaces that are not large enough to fit an entire segment.

c. Paging

External fragmentation: no
Internal fragmentation: yes
Pages and frames are the same size so there will never be wasted external free memory that cannot be allocated to a page. It is possible for the allocated memory to not completely fill a page, therefore, there can be internal fragmentation.

2. Most systems allow a program to allocate more memory to its address space during execution. Allocation of data in the heap segments of programs is an example of such allocated memory. What is required to support dynamic memory allocation in the following schemes?

a. Base and bounds

Copy process to new location in physical memory and then update the base and bounds registers.

b. Segmentation

If there is free space after the segment then the segment can be expanded by updating the size register for that segment.
If there is not enough free space after the segment then it must be copied to a new physical location and the base and bounds registers updated.

c. Paging

Allocated free physical frames to unused pages by updating the page table.

a. 22,000

4KB is 4,096
22,000 / 4,096 = 5
22,000 % 4,096 = 1,520

b. 77,056

77,056 / 4,096 = 18

77,056 % 4,096 = 3,328

c. 197,012

197,012 / 4096 = 48

197,012 % 4096 = 404

4. Suppose a process' logical address consists of 4 pages and the page size is 2KB. The page table of the process is given in below. Compute the physical address for each of the following logical addresses (provided as decimal numbers).

a. 1,018

Page number is 1,018 / 2,048 = 0

Offset is 1,018

Page 0 maps to frame 1

Frame 1 starts at address 2,048

Physical address is 1,018 + 2,048 = 3,066

b. 6,976

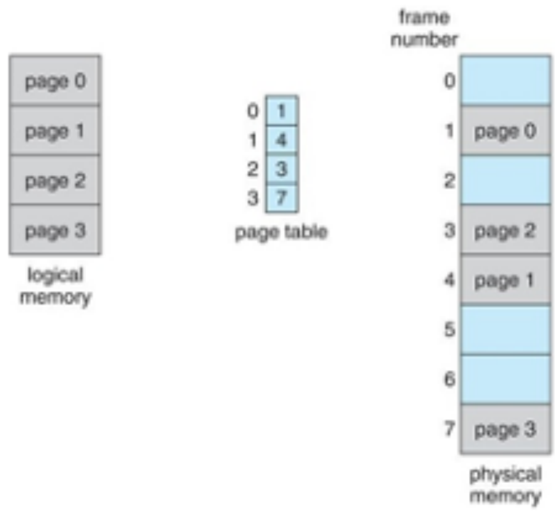
Page number is 6,976 / 2,048 = 3

Offset is 6,976 % 2,048 = 832

Page 3 maps to frame 7

Frame 7 starts at address 7 * 2,048 = 14,336

Physical address is 14,336 + 832 = 15,168



5. On some systems, virtual and physical addresses are different sizes. Consider a small embedded system with a virtual address space of 4,096 pages and a 2KB page size. The physical memory has a maximum capacity of 1,024 frames.

a. How many bits are required for addresses in the virtual address space?
4,096 * 2,048 = 8,388,608 bytes = 2^23 bytes
Therefore 23 bits are required.

b. How many bits are required for addresses in physical memory
1,024 * 2,048 = 2,097,152 = 2^21 bytes
Therefore 21 bits are required.

Module 5

1. Suppose that a machine has 48-bit virtual addresses and 32-bit physical addresses.
   a. If pages are 4KB, how many entries are in the page table if it has only a single level? Explain.

The number of entries in the page table is the same as the number of pages in the virtual address space.

The total size of the virtual address space is 2^48 bytes.

The size of a page is 4KB

Therefore, there are 2^48 / (4*2^10) = ~68.7 billion entries.

   b. Suppose this same system has a TLB with 32 entries. Furthermore, suppose that a program contains instructions that fit into one page and it sequentially reads long integer elements from an array that spans thousands of pages. How effective will the TLB be for this case?

Suppose an integer is 8 bytes long. Then 4*2^10/8 = 512 integers fit on a page. The array spans many pages, on every new page that is accessed there is one cache miss. The array is accessed sequentially, resulting in high spatial locality, which means the page entry is put into cache only once and after it is evicted will never be needed again. With the example numbers there will be a miss rate of about 1/512 = 0.195%.

2. Suppose that a machine has 38-bit virtual addresses and 32-bit physical addresses.

a. What is the main advantage of a multilevel page table over a single-level one?

A multilevel page table does not require page table entries for every unallocated page. Consider a newly created process that only requires 3 pages (one each for code, heap and stack). The multilevel page table would only need to be a few pages, while a single level page table would require millions of pages.

b. With a two-level page table, 16-KB pages, and 4-byte entries, from the VPN how many bits should be allocated for the page directory index and how many for the page table index? Explain.

Number of offset bits: log2(16KB) = log2(2^14) = 14 bits

Number of entries per page: 16 * 2^10 / 4 = 4,096 entries

Number of bits required to address 1 page's entries: log2(4,096) = log2(2^12) = 12 bits

Number of bits required for page directory index: 38 - 14 - 12 = 12 bits

3. Suppose a computer keeps its page tables in main memory. The average overhead required for reading an entry from the page table is 5ns. To reduce this overhead, the computer has a TLB, which holds 32 page table entries, and can do a look up in 1ns. Cache hit rate is the number of hits divided by total accesses. What hit rate is needed to reduce the mean overhead to 2ns?

When the page entry is not in the TLB it takes 1ns for the lookup and 5ns to read from the page table, 6ns total.

When the page entry is in the TLB it takes 1ns. If *hit* is hit rate, then the miss rate is *(1 –hit)*.

2 = 6 * (1 -hit) + 1 * hit
2 = 6 - 6 * hit + hit
4 = 5 * hit
hit = 4/5 = 0.8 hit rate

Modules 6 and 7

1. Race conditions are possible in many computer systems. Consider an online auction system where the current highest bid for each item must be maintained. A person who wishes to bid on an item calls the `bid(amount)` function, which compares the amount being bid to the current highest bid. If the amount exceeds the current highest bid, the highest bid is set to the new amount. This is illustrated below:

```
  void bid(double amount) {
    if (amount > highestBid) {
      highestBid = amount;
    }
}
```

Describe how a race condition is possible in this situation and what might be done to prevent the race condition from occurring.

There is a race condition on the variable highestBid. Suppose that two people wish to bid on a computer, and the current highest bid is $1,200. Two people then bid concurrently: the first person bids $1,300, and the second person bids $1,400. The first person invokes the bid() function, which determines that the bid exceeds the current highest bid. But before highestBid can be set, the second person invokes bid(), and highestBid is set to $1,400. Control then returns to the first person, which completes the bid() function and sets hightestBid to $1,300. The easiest way of fixing this race condition is to use a mutex lock:

```
void bid(double amount) {
    acquire(mutex);
    if (amount > highestBid)
        highestBid = amount;
    release(mutex);
}
```

2. Consider the code example for allocating and releasing processes.

```
#define MAX_PROCESSES 255
int number_of_processes = 0;
/* the implementation of fork() calls this function */
int allocate_process() {
  int new_pid;
  if (number_of_processes == MAX_PROCESSES) {
    return -1;
  } else {
    /* allocate necessary process resources */
    number_of_processes++;
    return new_pid;
  }
}

/* the implementation of exit() calls this function */
void release_process() {
  /* release process resources */
  number_of_processes--;
}
```

a. Identify the race condition(s).
b. Assume you have a mutex lock named mutex with the operations lock() and unlock(). Indicate where the locking needs to be placed to prevent the race condition(s).

a. There is a race condition on the variable number_of_processes.
b. A call to acquire() must be placed upon entering each function and a call to release() immediately before exiting each function.

3. Assume a network is shared among different services, each of which wants to have multiple concurrently open TCP connections. The network sets a limit on the total number of open TCP connections. The purpose of the methods below is to keep track of the number of open connections so that the limit is never exceeded. For example, if a service want to open 5 new connections it must first call `request(5)`.
Implement the methods (pseudocode is fine) using only **locks** and **condition variables** to manage the concurrency.

```
/* initialize the network capacity to n connections */
void
init(int n);


/* The purpose of this function is to block (not return) until
* n network connections are granted to the caller.
*
* On return, the caller assumes they are allowed to open n
* network connections.
*/
void
request(int n);

/* release n connections */
release(int n);

cond_t cond;
mutex_m mutex;
int available;
```

```
void
init(int n) {
  // assuming this is called before the other threads are
  // created, it that is not then case then mutex lock needs to
  // be acquired before updating available
  available = n;
}
void
request(int n) {
  pthread_mutex_lock(&mutex);
  while (n > available) {
    pthread_cond_wait(&cond, &mutex);
  }
  available -= n;
  pthread_mutex_unlock(&mutex);
}

void
release(int n) {
  pthread_mutex_lock(&mutex);
  available += n;
  pthread_cond_signal(&cond);
  pthread_mutex_unlock(&mutex);
}
```

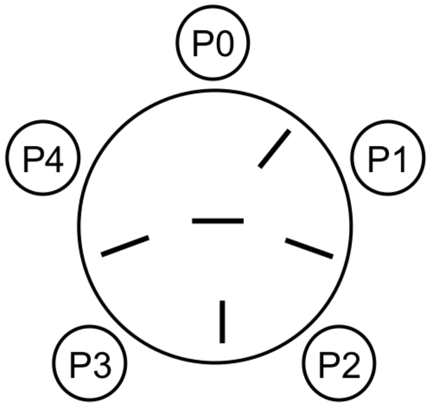
4. Consider the following sushi bar problem. There is a sushi bar with one chef and N seats. When a customer arrives, they take a seat at the bar if there is an empty one or they leave if no seats are empty. After taking a seat, they wait for the sushi chef to be available and then give their order. After the sushi chef serves them, they eat and leave. Write a solution to the sushi bar problem that uses only **semaphores** to manage the concurrency. You should write three procedures: `init()`, `chef()` and `customer()`.
This problem shares some similarities to a classic problem we didn't cover in class, the sleeping barber problem.
https://en.wikipedia.org/wiki/Sleeping_barber_problem

```
sem_t customersReadyToOrder = 0; /* can be 0 up to N */
sem_t chefTakingOrder; /* will always be 0 or 1 */
sem_t customerGivingOrder; /* will always be 0 or 1 */
sem_t chefServingFood; /* will always be 0 or 1 */
sem_t seatsMutex = 0; /* only one thread at a time allowed
                         to read/write seatsAvailable */
int seatsAvailable = N; /* can be 0 up to N */

void
init() {
    sem_init(customersReadyToOrder, shared, 0);
    sem_init(chefTakingOrder, shared, 0);
    sem_init(customerGivingOrder, shared, 0);
    sem_init(chefServingFood, shared, 0);
    sem_init(seatsMutex, shared, 0);
    int seatsAvailable = N;
}

void chef() {
    while (true) {
        sem_wait(customersReadyToOrder); /* wait for at least one
                                            customer to be ready */
        printf("Welcome to the sushi bar.\n");
        /* take order */
        sem_post(chefTakingOrder);
        printf("What will you have?\n");
        sem_wait(customerGivingOrder);
        /* serve customer */
        printf("Here is your food.\n");
        sem_post(chefServingFood);
    }
}
```

```
void customer() {
    sem_wait(seatsMutex); /* get exclusive access to seatsAvailable
                             to prevent race conditions */
    if (seatsAvailable == 0) {
        sem_post(seatsMutex); /* release the lock */
        printf("I'm not waiting around, bye.\n");
        return;
    }
    printf("This seat looks good.\n");
    seatsAvailable--; /* customer takes an available seat */
    sem_post(seatsMutex); /* release the lock */

    sem_post(customersReadyToOrder); /* customer signals they are
                                        ready to order */
    sem_wait(chefTakingOrder); /* customer waits, the chef might be
                                  serving other customers */

    printf("Hello.\n");

    printf("I will have one roll.\n");
    sem_post(customerGivingOrder);
    sem_wait(chefServingFood);
    printf("That was good.\n");

    set_wait(seatsMutex); /* get exclusive access to seatsAvailable
                             to prevent race conditions */
    seatsAvailable++;
    sem_post(seatsMutex); /* release the lock */
}
```

Module 8

1. Consider a modified version of the dining philosophers problem, where one of the chopsticks is placed at the center of the table, available for any of the philosophers to pick up with either hand.

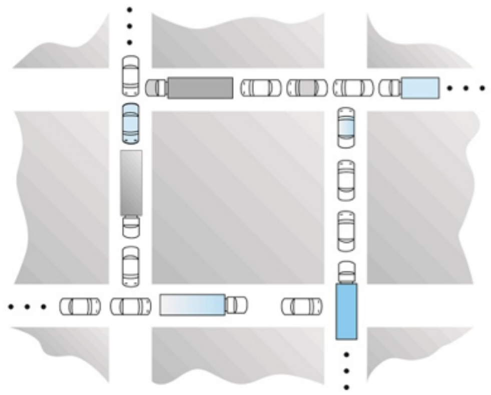


Can deadlock be prevented if no philosopher is allowed to pick up the center chopstick as their first chopstick? Explain you answer.

Yes, deadlock can be prevented.

Consider the four outer chopsticks, which are the only ones that are allowed to be picked up first. Suppose that four of the philosophers each pick up one of them and a fifth philosopher has none. Is it possible for there to be a circular wait on the chopsticks? The fifth philosopher that has none is not allowed to pick up the center chopstick first. Therefore, one of the four philosophers will pick up the center chopstick and be able to eat. In other words, because there must always be one philosopher that has no chopsticks, it is not possible for there to be a circular wait, and likewise not possible for there to be a deadlock.

2. Consider the traffic deadlock depicted in the figure below.



a. Show that the four necessary conditions for deadlock hold in this example.
b. State a simple rule for avoiding deadlocks in this system.

a.

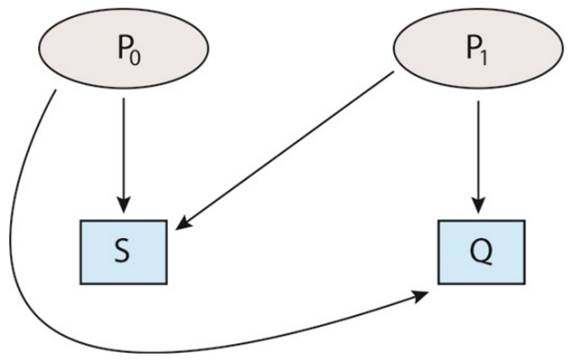
Mutual exclusion – two vehicles cannot physically be in the same place at the same time
Hold and wait – There are vehicles that have blocked an intersection, thus preventing other traffic from entering the intersection (e.g., holding a resource). At the same time these vehicles are waiting for the intersection ahead of them to open up before they can clear their current intersection (e.g., waiting for a resource).
No preemption – It appears that none of the vehicles are willing or able to divert from their current path in order to open up one of the intersections.
Circular wait – there are four flows of traffic and four intersections, each flow of traffic is waiting on an intersection to clear that another flow of traffic is currently occupying forming a cycle of dependencies.

b.

A vehicle may enter an intersection only if both that intersection and the intersection ahead are clear. This rule means that if three intersections are occupied by three flows of traffic, a fourth flow of traffic cannot enter the fourth intersection which prevents circular wait.

3. The resource-allocation graph blow shows that both processes P0 and P1 need both resources S and Q to complete.

a. Explain how a deadlock is possible in this system.
b. Although a deadlock is possible, the system will not always enter a deadlock, the scheduler plays a role. Assume P_0 and P_1 have CPU bursts of 20 ms each, draw example Gantt charts that demonstrate FCFS and RR (with quanta=5). Use the charts to explain which algorithm is more likely to result in a deadlock?



FCFS:
Because FCFS doesn't use process preemption (different than resource preemption), it will not cause a hold and wait, therefore it will not cause a deadlock. For example, both processes in the chart below are to complete.

```
| P0 lock(s); lock(q); unlock(s); unlock(q); | P1 lock(q); lock(s); unlock(s); unlock(q); |
```

RR:
An RR scheduler uses process preemption, which can cause a deadlock, assuming there is no resource preemption. In the example below both P0 and P1 become deadlocked (enter a permanent wait state) at the statements highlighted.

```
| P0 lock(s); | P1 lock(q); | P0 lock(q); | P1 lock(s); |
```

Module 9

1. Suppose that a computer can read or write a memory word in 10 nsec. Also suppose that when an interrupt occurs, all 32 CPU registers, plus the program counter are pushed onto the stack. What is the maximum number of interrupts pre second this machine can process.

To service an interrupt the CPU registers need to be written to memory and before the interrupt returns the registers need to be read back from memory. Assuming there is no caching 10 nsec is taken for each read or write of each word.

33 words * 10 nsec * 2 = 660 nsec

The maximum number of interrupts per second is: 1 / 660 nsec = 1.5 million

2. Consider a 320 GB SATA drive described in the figure below. Suppose the workload is 10,000 reads to sectors randomly scattered across the disk. How long with these 10,000 requests take (total) assuming the disk services requests in FIFO order?

Size	
Form factor	**2.5 inch**
Capacity	**320 GB**
Spindel speed	**5,400 RPM**
Average seek time	**12.0 ms**
Maximum seek time	**21 ms**
Track-to-track seek time	**2 ms**
Transfer rate (surface to buffer)	**850 Mbit/s (maximum)**
Transfer rate (buffer to host)	**3 Gbit/s**
Buffer memory	**8 MB**

disk access time = seek time + rotation time + transfer time
Seek time:

- **Each request requires a seek from a random starting track to a random ending track.**
- **For seek time use the average of 12ms.**

Rotation time:

- **Once the disk head settles on the right track, it must wait for the desired sector to rotate under it.**
- **The average time it will take is one half of a rotation**
- **0.5 * 60s/5400RPM = 5.56ms**

Transfer time:

- **The disk's surface bandwidth is at least 100 MB/s, so transferring one sector (512 bytes) takes at most 5120ns (000512 ms).**

Total time:

- **Num requests * (seek + rotation + transfer)**
  = 10000 * (12ms + 5.56ms + 0.00512ms)
  = 10000 * 17.56512ms

= 175.6s

3. Consider a 320 GB SATA drive described in the previous figure. Suppose the workload is 10,000 reads to 10,000 sequential sectors on the outer-most tracks of the disk. How long will these 10,000 requests take (total) assuming the disk services requests in FIFO order?

Seek time and rotation time per request are the same as above.

Transfer time: 10,000 sectors * 512 bytes/sector * 1/(100 MB/s) = 51.2ms

So, 10,000 requests will take about 12ms + 5.56ms + 51.2ms = 68.76 ms (much faster that random access)

4. Linux provides a command-line utility rm that has the purpose of removing a file
from a directory. When executing rm inside of the strace utility we can see that rm makes a
system call to unlink(). Here is an example.

```
$ strace rm test.txt
...
unlinkat(AT_FDCWD, "test.txt", 0)       = 0
...
```

Explain why rm unlinks the file. How does unlinking result in the removal of the file? Why does Linux not have a system call such as delete() that rm could use to directly remove the file?

There may be more than one file name mapped to the inode.
If the file is deleted (inode is deleted), those other directory mappings would become invalid.
Unlinking removes the directory mapping.
Only when all files (references) to the inode are unlinked will the inode be deleted.

5. Linux provides the command mv that has the purpose of "moving" a file from one
file path name to another file path name. When executing mv inside of the strace utility we
can see that mv makes a system call to rename().

```
$ strace mv oldname.txt newname.txt
...
renameat(AT_FDCWD, "oldname.txt", AT_FDCWD, "newname.txt") = 0
...
```

Is there any difference at all between renaming a file and just copying the file to a new file with the new name, followed by deleting the old one? Explain.

Renaming a file means modifying the directory mapping of file name to inode. The file contents do not need to be copied.

The alternative approach would not be the same, besides having an extra copy, there would be complications such as other hard links to the file becoming irrelevant.

Module 10

1. How many disk operations are needed to fetch the inode for the file `/var/log/boot.log`? Assume that the inode for the root directory is in memory, but nothing else along the path is in memory. Also assume that all directories fit in one disk block.

Assuming the inode for the root directory is in memory, the reads will be:

1. **Root data block**
2. **Var inode block**
3. **Var data block**
4. **Log inode block**
5. **Log data block**
6. **boot.log inode block**

Therefore, it will be 6 read operations.

2. In the Very Simple File System (VSFS) the inodes are kept together near the start of the disk. An alternative design is to allocate an inode when a file is created and put the inode at the start of the first data blocks of the file.

a. Assume a hard disk drive (HDD) is being used. What is an advantage to putting inodes at the start of the disk? What is an advantage to putting inodes at the start of the first data blocks of the file?

   When inodes are together at the start of the disk there may be less seek time to find multiple files in a directory. When inodes are close to the contents of the file there is less seek time to go from the inode to the data of the fill.

b. In the Fast File System (FFS) the inodes for a file are placed near the start of the same block group as the file's data blocks. Again assuming a HDD, explain an advantage to this approach rather than placing all the inodes near the start of the disk.

   This has the same advantages of putting inodes at the start of the disk for files that are close together in the directory tree (thus get put in the same group). But there is an additional advantage that the inodes are closer to the file data then they would be if placed at the start of the disk.

c. Now assume a Solid State Drive (SSD) is being used. Does the FFS approach still have any advantages or would the alternative design of putting the inode at the start of the first data blocks of the file be better?

SSD still benefits from faster sequential access, however, there is not seek latency so whether blocks are close or far in the logical address space doesn't make a difference. Therefore, there would be no difference in either of the two approaches

3. What would happen if the bitmap containing the information about the free disk blocks was completely lost due to a crash? Is there any way to recover from this disaster, or is it bye-bye disk?

It is possible to completely recover the bitmap. Starting at the root directory every directory and file can be recursively visited. When the inode of a file or directory is visited it is possible to discover what other blocks that inode is pointing to. If the bits in the bitmap are marked as used for each block encountered, at the end the bitmap will be recovered.

4. Some file systems allow disk storage to be allocated at different levels of granularity. For instance, a file system could allocate 4 KB of disk space as a single 4-KB block or as eight 512-byte blocks.

a. How could we take advantage of this flexibility to improve performance?

   The advantage is less internal fragmentation will lead to less wasted space. Suppose a file only uses half of a block, then the other half would have been wasted space. With 512-byte subblocks the file will occupy exactly two sub-blocks and waste no space.

b. What modifications could be made to the bitmap free-space management scheme in order to support this feature?

   There any many possible ideas. The most naive approach would be to make a bit map that is 4 times larger so that every bit represents one sub-block.

   A more efficient approach might be to combine the ideas of a bitmap and free list. The bitmap would indicate if the entire block is free. When no free blocks remain a linked list indicates which sub-blocks are free.

5. Consider a file system that uses Unix-style inodes (index nodes) to represent files. The system uses 16-KB blocks and 8-byte pointers. Multi-level indexing is used with an inode containing 12 direct pointers, 1 single indirect pointer, 1 double indirect pointer and 1 triple indirect pointer.

a. What is the maximum size of a file that can be stored in the file system? Show calculations.

   A block can hold 16KB / 8 bytes = 2,048 pointers
   (12 + 2048 + 2048^2 + 2048^3) * 16KB

(12 + 2^11 + 2^22 + 2^33) * 2^14
8,594,130,956 * 16,384

(8,594,130,956 * 16,384) / 2^40 TB = 128 TB

b. For each of the following addresses describe how many blocks (including the inode and data block) need to be read to perform a read of the data at the address. Explain your answers.

4,096
32,768
1,048,576
268,435,456

4,096 / 16,384 = block # 0
The block is less than 12, therefore it is directly pointed to by the inode, so 2 reads.

32,768 / 16,384 = block # 2
The block is less than 12, therefore it is directly pointed to by the inode, so 2 reads.

1,048,576 / 16,384 = block # 64
The block is >= 12 but less than 2060, therefore it is pointed to by a single indirect pointer, so 3 reads.

268,435,456 / 16,384 = block # 16,384
The block is >= 2,060 but less than 4,196,364, therefore it is pointed to by a double indirect pointer, so 4 reads.

6. Suppose an SSD consists of raw flash with the following performance characteristics: reading one page takes 25µs, programming one page takes 200µs and erasing one block takes 1500µs (note these refer to physical SSD pages and blocks). The following figure represents the current state of the SSD.

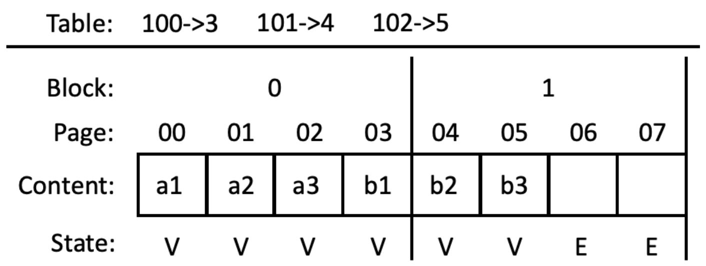


Table: 100->3 101->4 102->5

Block:	0				1			
Page:	00	01	02	03	04	05	06	07
Content:	a1	a2	a3	b1	b2	b3		
State:	V	V	V	V	V	V	E	E

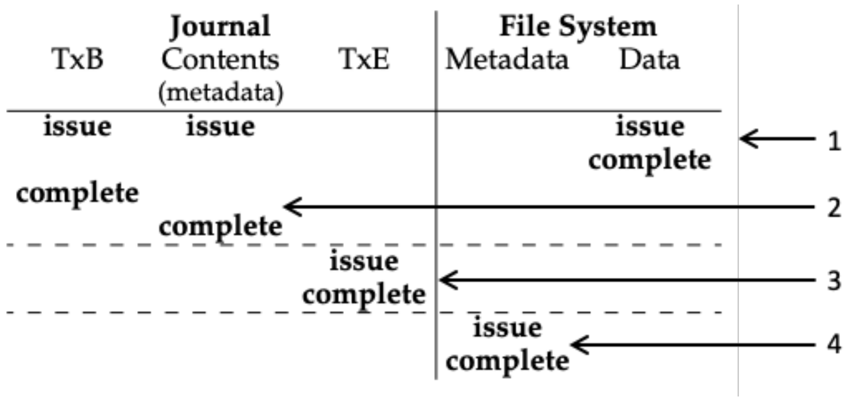
How much time will it take for garbage collection to reclaim physical block zero? Assume all operations must be performed sequentially (no parallel hardware optimization) and none of the pages are cached. Explain your result.

Read b1 from page 3
Program b1 to page 6
Erase block 0.

Therefore the time is: 25 + 200 + 1500 = 1725µs

Module 11

1. The figure below shows a timeline (top to bottom) of when different actions required to write to a block to a file occur. Explain how the file system recovers from a failure at each of the 4 points indicated and say if any data is lost after the recovery.



1. No recovery as the data has not been logged to the journal or file system.
2. No recovery as the metadata has not been logged to the journal or file system.
3. No recovery as we don't know if the journal is complete (TxE has not been issued ye
t).
4. Yes. We can replay the journal to recover the metadata in the file system. Data is alr
eady in the file system since it is done before the issue of TxE.

2.

a. Suppose the file system issues a write command to update the contents of virtual page 2000 with data "c1". Assume the current state of the SSD drive is shown in the figure below. Draw the most likely state of the SSD after the write has been completed.

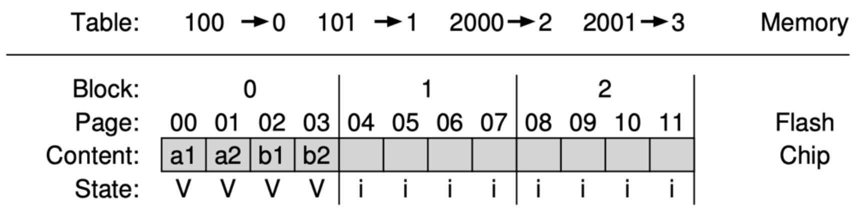


Table: 100 → 0 101 → 1 2000 → 2 2001 → 3
Memory
Block: 0 1 2
Page: 00 01 02 03 04 05 06 07 08 09 10 11
Content: a1 a2 b1 b2
State: V V V V i i i i i i i i
Flash
Chip

b. Now suppose, following the previous write, the file system deletes a file and removes virtual pages 100 and 101. Assume the FTL decides to garbage collect block 0. Draw the most likely state of the SSD after the garbage collection has been completed.

a.
table: 100->0, 101->1, 2000->4, 2001->3
| a1 | a2 | b1 | b2 | c1 |

b.
table: 2000->4, 2001->3
| e | e | e | e | c1 | b2 |

3. Suppose a guest process is running on top of a guest operating system on top of a host operating system. Consider the memory mappings below. Assume page size is 1KB (1,024 bytes).

Guest OS Page Table

VPN (Virtual Page Number)	PFN (Physical Frame Number)
0	10
1	2
2	3
3	5
4	0
5	1
6	6
7	7

Hypervisor (Shadow) Page Table

PFN (Physical Frame Number)	Machine
0	11
1	12
2	5
3	6
4	9
5	10
6	13
7	14

Where on the machine is virtual byte 2,555 stored?

VPN = 2,555 % 1,024 = 2
PFN = 3
Machine frame number = 6
Base address = 6 * 1,024 = 6,144

Offset = 2,555 – (1,024 * 2) = 507

Machine address = 6,144 + 507 = 6,651

Module 12

1. Select the security goal from the CIA Triad (confidentiality, integrity and availability) that has been violated in each scenario.

a. A user replaces wininit.exe on a Windows file system with a modified version that contains a backdoor to allow unauthenticated access to the machine in the future.
b. A botnet (a group of compromised Internet-connected devices) is being used to target a webserver with high amounts of network traffic with the objective of overwhelming the server's resources.
c. A user is logged into their account at Secure Bank & Trust, an attacker has launched a man-in-the-middle attack that allows the attacker to observe the communication.

a. **Integrity: system resource replaced by unauthorized**
b. **Availability: distributed denial of service DDoS**
c. **Confidentiality: observing secret/private data**

2. Discuss how the asymmetric encryption algorithm can be used to achieve the following goals.

a. Authentication: the receiver knows that only the sender could have generated the message.
b. Secrecy: only the receiver can decrypt the message.
c. Authentication and secrecy: only the receiver can decrypt the message, and the receiver knows that only the sender could have generated the message.

Let,

k^{s}_{e} be the public key of the sender,

k^{r}_{e} be the public key of the receiver,

k^{s}_{d} be the private key of the sender, and

k^{r}_{d} be the private key of the receiver.

- **Authentication: Authentication is performed by having the sender send a message that is encoded using Private Key of Sender, k^{s}_{d}.**
- **Secrecy: Secrecy is ensured by having the sender encode the message using Public Key of the Receiver k^{r}_{e}.**
- **Authentication and Secrecy: Both authentication and secrecy are guaranteed by performing double encryption using both Private Key of Sender k^{s}_{d} and Public Key of Receiver k^{r}_{e}.**

3. Many computer architectures offer more than just two security modes, for example, they have user, kernel and machine mode. If the operating system is trusted to execute with kernel mode privileges, why is there need for a machine mode?

Machine mode allows adding an extra level of protection such as putting the OS in a virtual machine, hypervisor, or sandbox or using a Trusted Platform Module.

4. How do operating systems that use paged virtual memory prevent processes from reading or writing the physical memory of other processes? Be specific about the mechanisms in place that prevent this.

All memory accesses by a process are treated as virtual.
The hardware MMU translates virtual addresses to physical addresses by looking in the page table.
If the page table does not have a translation, then the memory access is not allowed.
The page table only maps to addresses the process is allowed to access.

5. An operating system provides access control which prevents unauthorized reading of data, what are the advantages of encrypting data stored in the computer system?

The advantage is security in depth. Even though the OS has access control, it should not be relied on as the only security mechanism for sensitive data such as passwords. Systems are hacked, if the attacker gets access to the password file it is better if the file is encrypted to add an extra layer of defense.

6. Describe each of the following three kinds of access control mechanisms in terms of (a) ease of determining authorized access during execution, (b) ease of adding access for a new subject, (c) ease of deleting access by a subject, and (d) ease of creating a new object to which all subjects by default have access.

a. per-subject access control list (that is, one list for each subject tells all the objects to which that subject has access)
b. per-object access control list (that is, one list for each object tells all the subjects who have access to that object)
c. access control matrix

a.

   i. **Ease of determining authorized access during execution – it is easy to find the object that is being accessed in the list of objects for the subject**
   ii. **Ease of adding access for a new subject – it is easy to add a new subject and add the object into that subjects list.**
   iii. **Ease of deleting access by a subject – find the object in the subjects list and remove it from the list.**

   iv. **Ease of creating a new object to which all subjects by default have access – this is more time consuming, it requires visiting each subject and adding the object to that subject's list**

b.

   i. **Ease of determining authorized access during execution – it is easy to find the find the subject in the list for the object**
   ii. **Ease of adding access for a new subject – it is easy to add a new subject to the list for the object**
   iii. **Ease of deleting access by a subject – find the subject in the object's list and remove it from the list.**
   iv. **Ease of creating a new object to which all subjects by default have access – create a new object which a list of every subject.**

c.

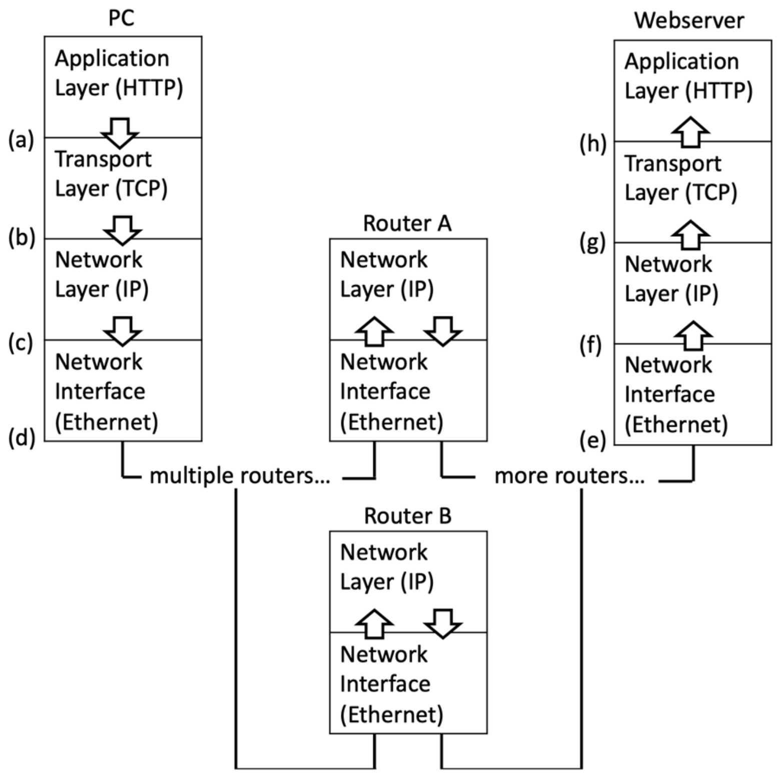
   i. **Ease of deterring authorized access during execution – find the row and column in the matrix**
   ii. **Ease of adding access for a new subject – add a new subject row and set the value in the correct column**
   iii. **Ease of deleting access by a subject – fix the row and column in the matrix and clear the access**
   iv. **Ease of creating a new object to which all subjects by default have access – add an object column and the value in every row.**

7. Suppose a per-subject access control list is used. Deleting an object in such a system is inconvenient because all changes must be made to the control lists of all subjects who did have access to the object. Suggest an alternative, less costly means of handling deletion.

Keep both a per-subject and per-object access control list. The per-object control list can be used to efficiently find all subjects that have access to the object. Then removing the object for all subject lists is more efficient.

Module 13

1. The diagram shows the scenario of a PC making a request of a webserver. The Internet is represented between the PC and the webserver. There are two possible paths for packets to take, one passes through router A and the other through router B. Assume that it is equally likely for packets to take either path. Use the example to answer the following questions.



a) Explain the difference between the transport layer and the network layer.

Both are part of the network stack, and the transport layer relies on the network layer to complete its tasks. The transport layer provides communication between processes (usually on different machines), the processes are identified by port numbers, and it is expected to be resilient to packet loss or out of order packets. The network layer provides communication between hosts (e.g., machines), the hosts are identified by IP addresses, and no attempt is made to resent, or reorder lost or out of order packets.

b) Why do the routers not have a transport layer?

Routers participate at the network layer and below, they look at each packet's IP address and forward the packet to the next appropriate node, they do not try to check for lost or out of order packets. Adding transport layer mechanisms at the router level would make routing slow, expensive, and inflexible.

c) For efficient sharing of the network infrastructure, TCP breaks long application layer messages into small packets (typically less than 1500 bytes). Explain why the packets arriving at the webserver can be out of order.

   The hosts (PC and webserver) have no control over the route packets will take over the network, some packets might go to router A and others to router B. Communication over the network is asynchronous, some packets may be lost, some packets may take a longer/slower route, so packets from router A and router B could arrive at the webserver in any order.

d) The arrows in the diagram show an example TCP communication from the PC to the Webserver, the layer boundaries have been labeled (a) to (h). List all boundaries (a to h) where packets might **arrive** out of order.

   On the PC, the application must provide the correctly ordered message to the transport layer, where there would be no reason for the transport layer, network layer or network interface to change the order of packets. Once the packets reach the network there is not guarantee over which route packets will take or how long a particular route will take. The transport layer (TCP) on the webserver maintains a receive window, it collects packets in the window and uses acknowledgements to have lost packets resent. Only a complete (contiguous) window of packets is forwarded to the application layer.

2. What are the three parts of a socket? Which layer of the 7-layer OSI model does each part deal with?

Protocol – transport layer
Port – transport layer
IP Address – network layer

3. What are the advantages and disadvantages of TCP and UDP? Give examples of applications where each might be preferred.

TCP is reliable as it guarantees the delivery of data to the destination router. A disadvantage of TCP is that it is slower and more expensive because it relies on establishing and maintaining a connection which requires extra packets being sent.

UPD is a connectionless protocol, packets are sent without any guarantee of delivery. The advantage is no overhead – no handshake packets sent to establish a connection, no acknowledgement packets sent for every data packet, etc. The disadvantage is that packets can be lost or delivered out of order, it is up the application to deal with that.

TCP is common for file transfer (e.g., FTP), email (e.g., SMTP), and web (e.g., HTTP). UDP is common for multimedia applications (e.g., VoIP) and client-server communication that doesn't need to send large messages (e.g., DNS).

4. Calculate the total time required to transfer a 1000-KB file in the following cases, assuming an RTT of 50 ms, a packet size of 1 KB data:

(a) The bandwidth is 1.5 Mbps, and data packets can be sent continuously.

Bandwidth = 1.5 Mbps = 0.1875 MB/s

Packet size = 1 KB = 1,024 B

Transmit time per packet = size/bandwidth = 1,024B / 187500B/s = 5.461ms

Transmit time for 1,000 packets = 5,461ms

Propagation + queuing time = RTT / 2 = 25ms

Transfer time (latency) = 5,461ms + 25ms = 5,486ms

(b) The bandwidth is 1.5 Mbps, but after we finish sending each data packet we must wait one RTT (for data receive and acknowledgement) before sending the next.

(5.461ms + 50ms) * 1,000 = 55,461ms

Module 14

1. Suppose a distributed system is using acknowledgement-based communication:

- **a.** How does the sender distinguish between the message being lost, the receiver being not available, or the acknowledgement being lost?

  The sender has no way to know.

- b. Explain, how the sender deals with a lost acknowledgement.

  After a timeout, it resends the original message. Messages have a sequence number attached so the receiver will know if it receives the same message twice.

- c. Explain, how the sender deals with an acknowledgement that is significantly delayed.

  From the perspective of the sender this is the same as a lost acknowledgement. If the acknowledgement does not arrive before the timeout, it resends the original message.

2. In most operating systems file access permissions are checked only when the file is opened and not with each read or write. Thus if you open a file successfully for both reading and writing, obtaining file descriptor *f*, and if you subsequently change the file's access permissions to read-only, you can still write to the file using *f*.

- a. Explain why supporting this feature is difficult in distributed file systems whose servers do not hold open-file state, such as NFS.

  A stateless server such as NFS does not remember what it told a client in a previous response. Because of this, for security, the NFS server much check file permission on every request.

- b. How might you "approximate" this feature so that it's supported on NFS for what's probably the most common case: the file's owner is accessing the file?

  If the files owner is always given both read and write permission at the server (which seems reasonable given that it is the owner), then additional permission restrictions can be supported at the client.

3. Suppose we wish to add to basic NFS a remote procedure for appending data to the end of a file.

- a. Why does NFS not already have an *APPEND* procedure?

Append is not a idempotent operation, so it was omitted from the client-server protocol.

b. Explain why such a remote procedure is needed: why can't its effect be obtained through the use of the *GETATTR* procedure (which, among other things, returns the size of a file) and the *WRITE* procedure?

   The approach of using GETATTR followed by WRITE is not an atomic operation. This could result in a race condition between client where they overwrite each other appends.

c. Assuming a well behaved network and a server that never crashes, will there be any problems if an append request or response is lost? Explain.

   Yes, the client will assume the request failed and resend to the append. This can potentially result in multiple appends to the file.