Week 2

Q1: Resource Allocation The issue of resource allocation shows up in different forms in different types of OSs. List the most important resources that must be managed by an OS in the following settings:

(a) supercomputer

> Processors, memory, GPUs — any resource affecting computational performance of supercomputer

(b) workstations connected to servers via a network

> The network is a resource that is shared by all connected computers. Access to it may be managed by the OS through the network protocol stack that avoids congestion using protocols such as TCP.

(c) smartphone

> A phone has limited battery life, so battery power is a resource that must be conserved by the OS. For example, the OS may decide to put the CPU to sleep or switch off certain unused hardware features.

Q2: OS Kernel What is the kernel of an OS?

> The part of the OS that is always in memory and implements the most commonly executed functions of the OS. The OS kernel executes in kernel or privileged mode and therefore has complete access to all hardware (in contrast to user-mode processes).

Q3: User/kernel Mode Separation Why is the separation into a user mode and a kernel mode considered good OS design?

> A process executing in user-mode can cause less “damage” because it is restricted from executing privileged operations. By splitting an OS design into user-mode and kernel-mode components, the OS designer makes it explicit which parts of the OS require raised privilege and full access to the hardware. This is an example of the “principle of least privilege” that should guide the design of any secure systems.

Give an example in which the execution of a user processes switches from user mode to kernel mode, and then back to user mode again.

```
A system call (using a trap instruction) is an example of this.
```

Q4: Instructions in Kernel Mode Which of the following instructions should only be allowed in kernel mode, and why?
(a) Disable all interrupts
(b) Read the time of day clock
(c) Change the memory map

```
(a), (c), (d)
```

Q5: OS Portability A portable OS is one that can be ported from one system architecture to another with little modification. Explain why it is infeasible to build an OS that is portable without any modification.

```
By definition, an OS must interact with the hardware directly. It
therefore contains code that depends on the specifics of the
processor architecture such as its instruction set and the memory
management system.
```

Describe two general parts that you can find in an OS that has been designed to be highly portable.

```
The platform-specific part contains any OS code that is dependent on
the given processing architecture and platform.

All other OS functionality is implemented as part of the platform
independent part that uses the API exposed by the platform-specific
part to invoke any low-level hardware functionality. When porting the
OS, only the platform-specific part has to be reimplemented for a new
platform. The rest can just be recompiled for the new architecture.

This type of division can be found in the Linux kernel, which makes
it (relatively) easy to port it to new platforms.
```

Q6: Resource Virtualisation What is resource virtualisation?

```
The OS takes a physical resource (such as the processor, or memory,
or a disk) and transforms it into a more general, powerful and
easy-to-use virtual form of itself. Thus, we sometimes refer to the
OS as a virtual machine.
```

Q7: Virtualised Resources Name some resources that OSs virtualise.

- CPU
- Memory
- Disk
- Threads
- Network

Q8: Why Virtualized Resources? Why do OS virtualise resources?

1. Isolate users and applications from one another
2. As a way of providing security
3. Make it simpler to reset the underlying OS
4. Make it easier to use the OS
5. To stop concurrently running processes from interfering with one another
6. Stop a single process from taking over an OS

Q9: Resource Virtualization How does a user application access OS services?

System Calls

What happens when a system call is made?

Typical user applications run in user mode and use a system call to trap into the kernel to request OS services. The trap instruction saves the register state carefully, Changes the hardware status to kernel mode, and jumps into the OS to a pre-specified destination: the trap table.

When the OS finishes servicing a system call, it returns to the user program via another special return-from-trap instruction, which reduces privilege and returns control to the instruction after the trap that jumped into the OS.

Week 3

(1) Why are user threads lighter than kernel threads?

User-level threads do not involve the kernel for context switches

(2) Why are processes created by fork()? If the new process is completely different from the parent, can't we call execve() after assigning resources to the child? Copying everything from the parent seems like a waste of resources.

The child process gets a copy of the parent process using copy-on-write semantics.

(3) Is it possible to never use processes and always use threads instead?

Processes provide an address space and other kernel resources.

(4) How would one recycle threads?

Use a thread pool.

(5) Signal handlers: which thread handles a signal, and does it need to be a kernel thread?

Signals must be delivered to a kernel visible thread, as this is done by the kernel. The thread picked depends on the signal type.

Q1: Processes Which of the following are per process items:

Signals, **Open Files**, **Address Space**, **Child processes**, **Pipes** are per process.
Stack, **Registers** are per thread.
Kernel is per machine.

Q2: Threads Why do OSs provide their users with a thread abstraction?

1. They want to separate the number of processes that can run from the numbers of CPU cores that exist
2. Allow time sharing on a common resource (CPU)
3. Provide a common API on top of CPU threads so programs do not need to know the specifics of the CPU they are executing on
4. They do not provide a security boundary (That is for processes)

Q3: Processes vs. Threads Why use processes instead of threads?

1. Having a separate address space between processes reduces bugs
2. Blocking IO is easier to use and has less effect on processes that have a single responsibility

Even though:

1. Inter-process communication is slower and less efficient than inter-thread communications
2. Managing a group of processes is harder than managing a group of threads.

Q4: Process Communication How do processes communicate with one another?

- Files
- Signals
- Events, Exceptions
- Pipes
- Message Queues
- Mailslots
- Sockets
- Shared memory
- Semaphores

Q4: UNIX Signals List some UNIX Signals – and explain what they do.

- SIGINT - Interrupt from keyboard
- SIGABRT - Abort signal from abort
- SIGFPE - Floating point exception
- SIGKILL - Kill signal
- SIGSEGV - Invalid memory reference
- SIGPIPE - Broken pipe: write to pipe with no readers
- SIGALRM - Timer signal from alarm
- SIGTERM - Termination signal

Q5: Issues with using Threads Using threads as compared to using processes introduces several types of bugs:

- Concurrency
- Synchronisation
- Race conditions
- Dead locks

- Live locks

Q6: Thread Yielding Why would a thread voluntarily give up the CPU by calling thread_yield()?

1. Threads know better than the Kernel when they are willing to give up CPU cycles.
2. Time slices are given to a process, not a thread, so if a thread surrenders the CPU another thread in the process will be given the CPU.

Q7: Registers The register set is per-thread rather than per-process. Can you explain why?

A thread virtualises a CPU core. CPU cores are a physical construct that exists within a CPU. Each CPU core has a set of registers.

Q8: Process Creation What API calls will create a new process?

Fork for Linux. **CreateProcess** for Windows.

Q9: Inter-Process Communication When two processes communicate through a pipe, the kernel allocates a buffer (e.g. 64KB). What happens when the process at the write-end of the pipe attempts to send additional bytes on a full pipe?

The thread that is attempting to write to the pipe blocks until enough data is read from the pipe.

Week 4

Q1: Scheduling Algorithms What are the goals of a scheduling algorithm?

- **Ensure fairness** - Comparable processes should get comparable services
- **Avoid indefinite postponement** - No process should starve
- **Enforce policy** - E.g. priorities
- **Maximize resource utilisation** - CPU, I/O devices
- **Minimise overhead** - From context switches, scheduling divisions

Q2: Scheduling Algorithms What are the advantages and disadvantages of “FirstCome First-Served” scheduling?

- **+** No indefinite postponement
- **+** Easy to implement
- **-** Throughput and turnaround time can be unpredictable

Q3: Scheduling Algorithms Imagine you are writing a “Preemptive Fair-Share Round Robin Scheduler” where the quantum is 1 second.
In the following example calculate in what order the following processes will run in:

User 1 has 3 processes: A,B,C
User 2 has 1 process: D

And each process runs for 2 seconds

A, D, B, D, C, A, B, C

Q4: Scheduling Order 3 jobs are waiting to run. Their expected runtimes are 1, 3, 5. In what order should they run to minimise their average turnaround time?

1, 3, 5 (Lowest first)

Q5: Multiprocessor Scheduling Imagine you are writing a scheduler that needs to schedule threads on a computer that has 2 CPUs. A problem many multi-CPU schedulers have is that threads lose CPU cache affinity.

Can you explain the issue with losing CPU cache affinity?

CPU caches significantly decrease the access time for reading or writing to memory.

CPU caches are also hierarchical, and each level of the hierarchy is more expensive to access than the next (by an order of magnitude), but can be accessed by more cores/CPUs.

Scheduling threads to run on CPUs results on threads missing caches more often.

Week 5

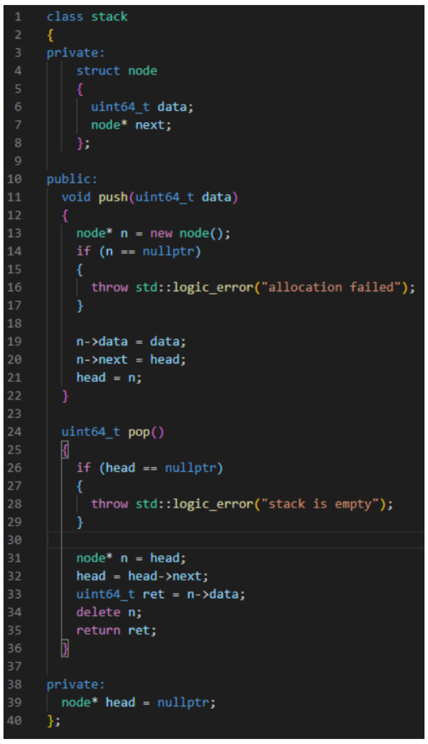
Q1: Thread Interleavings Consider the following two threads:
T1: a = 1 b = 1
T2: a = 2 b = 2
How many possible thread interleavings exist?

```
1. a = 1; a = 2; b = 1; b = 2
2. a = 1; b = 1; a = 2; b = 2
3. a = 1; b = 1; b = 2; a = 2
4. b = 1; a = 1; a = 2; b = 2
5. b = 1; a = 1; b = 2; a = 2
6. b = 1; b = 2; a = 1; a = 2
```

```
6 Interleavings
```

Q2: Thread Synchronization Consider the simple stack implementation on the right. There are race conditions in this code.

In the function push between what lines would you need to insert enter_critical() and where would you insert leave_critical()?



```
1   class stack
2   {
3   private:
4       struct node
5       {
6           uint64_t data;
7           node* next;
8       };
9
10  public:
11      void push(uint64_t data)
12      {
13          node* n = new node();
14          if (n == nullptr)
15          {
16              throw std::logic_error("allocation failed");
17          }
18
19          n->data = data;
20          n->next = head;
21          head = n;
22      }
23
24      uint64_t pop()
25      {
26          if (head == nullptr)
27          {
28              throw std::logic_error("stack is empty");
29          }
30
31          node* n = head;
32          head = head->next;
33          uint64_t ret = n->data;
34          delete n;
35          return ret;
36      }
37
38  private:
39      node* head = nullptr;
40  };
```

```
Enter => lines 19-20
Leave => lines 21-22

Enter => lines 25-26
```

```
Leave => lines 27-28, 32-33
```

Q4: Choosing Locks Now that we have identified which areas of the code require mutual exclusion, please consider what type of lock you will use.

```
Spin lock
```

Q5: Disabling interrupts One way of implementing mutual exclusion is by turning interrupts on and off. Please describe the issue with this technique.

```
This technique does not work when there is more than 1 CPU core.
Therefore, it is not useful for general purpose OSes such as Linux or
Windows.

Custom operating systems such as ones that run on a microcontroller
still use this technique.
```

Q6: Disabling interrupts Why does Windows and Linux not allow application to turn interrupts off?

```
If a user application can turn off interrupts the kernel's scheduler
would not have any techniques available to stop a running thread.

This would mean a user thread could run forever.
```

Q7: Strategies For Dealing With Deadlock There are 4 strategies for dealing with deadlocks. Please select (1), name it, and write a 1 to 2 sentence description of it.

1. **Ignore it** - "The ostrich algorithm", contention for resources is low and deadlocks infrequent
2. **Detection and recovery** - After system is deadlocked:
    a. detect the deadlock
    b. recover from it
3. **Dynamic avoidance -** Dynamically consider every request and divide whether it is safe to grant it; needs some information regarding potential resource use
4. **Prevention -** Prevent deadlocks by ensuring at least one of the four deadlock conditions can never hold

Week 6

Q1: Hierarchical Page Tables Why do operating systems make their page tables hierarchical, and what are the drawback(s)?

Hierarchical page tables reduce the amount of memory required to represent the virtual memory range for every application.

This was not a major issue when using 32bit processors (~4MB). On a 64bit processor (~30PB), using a flat page table is not possible.

The key drawback is performance: for every memory access, 2 lookups are required.

Q2: Dynamic Storage Allocation When allocating memory fragmentation is an issue. There are several strategies that exist to counter the effects of fragmentation.

Describe the strategies and write a one sentence description of its properties.

- **First-fit -** Allocates first hole that is big enough
  - Shortest possible traversal of the free list
  - No control over the size of leftover holes
- **Best-fit -** Allocate smallest hole that is big enough
  - Must search entire list, unless ordered by size
  - Produces smallest leftover hole

Q3: fork Explain how fork() is optimized to use virtual memory?

Gives a child its own page table pointing to the parent's page that is marked as read only. When any process writes to a page:

1. Protection fault causes trap to the kernel
2. Kernel allocates new copy of page so that both processes have their own private copies
3. Both copies are marked read-write

Q4: Page Replacement (LRU) Least recently Used (LRU) is a page replacement algorithm. Describe how you could implement this algorithm in 2 – 3 sentences.

Each page entry has counter

- When page referenced, copy clock into counter
- When page needs to be replaced, choose lowest counter

Exam Question: Address Translation I An embedded system uses a 16-bit big-endian architecture. It supports virtual memory management with a one-level page table. It has a page size of 1 KByte. The least significant bit of each page table entry represents a valid bit; the second least significant bit is the modified (dirty) bit.

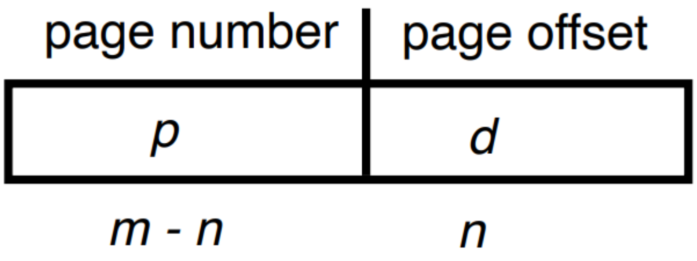
The following are the current entries in the page table (in hexadecimal notation):

0x2C00
0x0403
0xCC01
0x0000
0x7C01

Translate the following virtual memory addresses to physical memory addresses using the page table given above (if possible):

(i) 0xB85 (ii) 0x1420 (iii) 0x1000 (iv) 0xC9A

For given logical address space 2^m and page size 2^n:

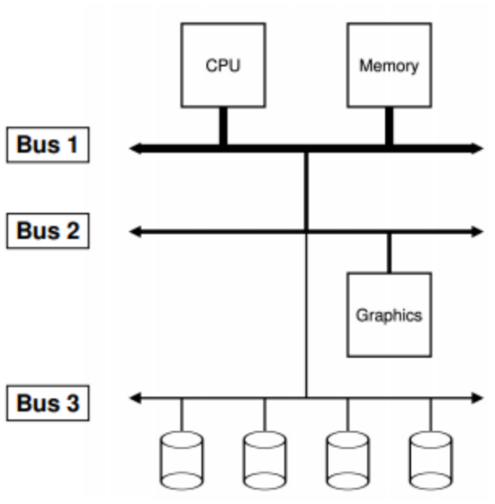


```
0. 0x2C00 = 001011 0000000000 (invalid)
1. 0x0403 = 000001 0000000011 (dirty)
2. 0xCC01 = 110011 0000000001
3. 0x0000 = 000000 0000000000 (invalid)
4. 0x7C01 = 011111 0000000001

0x0B85 = 000010(Entry 2) 1110000101 => 110011 1110000101 = 0xCF85
0x1420 = 000101(Entry 5)  0000100000 => page fault (beyond end)
0x1000 = 000100(Entry 4) 0000000000 => 011111 0000000000 = 0x7c00
0x0C9A = 000011(Entry 3) 0010011010 => page fault (invalid)
```

Week 7

Q1: Buses The figure below shows a typical system architecture for how the CPU interacts with different I/O devices.Please explain the purpose of the buses (bus 1, 2, 3)?



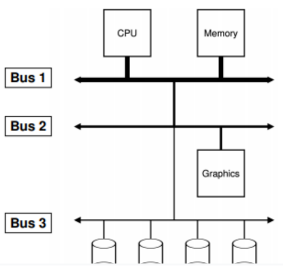
Bus 1 - High bandwidth & low latency devices. Devices that can function at the same speed as RAM
Bus 2 - High bandwidth & medium latency devices. Devices that have high throughput and require low latency but at slower than memory speeds. E.g. Graphics cards, network
Bus 3 - Other. Slower devices or devices that have lower bandwidth. E.g. block devices (disk)

Q2: I/O devices List four objectives that an operating system has when it manages I/O devices.

1. Fair access to shared devices
2. Allocation of dedicated devices
3. Exploit parallelism of I/O devices for multiprogramming
4. Provide uniform simple view of I/O
    a. Hide complexity of device handling
    b. Give uniform naming and error handling

Q3: I/O devices A new type of I/O device that operating systems must manage is non-volatile RAM (NVRAM). Conceptually an NVRAM I/O device offers both the access bandwidth of memory and the persistent storage capability of disks.

To which system bus, Bus 1, 2 or 3, would you connect an NVRAM I/O device?



```
Bus 1 / Bus 2
```

Q4: Disk seeks Suppose that the current position of the disk arm is over cylinder 200. The disk request queue contains requests for sectors on the following cylinders:

424, 24, 134, 443, 900

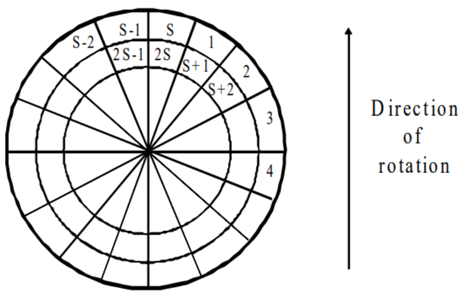
In which order will the requests be handled under (i) CSCAN & (ii) SSTF (Shortest seek time first) and briefly explain the policies.

```
(CSCAN) 424, 443, 900, 24, 134
(SSTF) 134, 24, 424, 443, 900
C-SCAN - Services requests in one direction only
```

Q5: Disk seeks A disk drive has S sectors per track and C cylinders. For simplicity, we will assume that the disk has only one, single-sided platter, i.e., the number of tracks per cylinder is one. The platter spins at w rotations per millisecond. The following function gives the relationship between seek distance d, in cylinders, and seek time, t_{seek}, in milliseconds:

$t_{seek} = 0$	$d = 0$
$t_{seek} = 6 + 0.06d$	$0 < d <= C$

The sectors are laid out and numbered sequentially, starting with the outer cylinder, as shown in the diagram below. Suppose the disk read/write head is located over cylinder 7. The disk receives a request to read sector S-1. What is the expected service time for this request?



```
seek time = 6 + 0.06 * 7 = 6.42 ms
rotational latency = (1 / 2) (1 / ω)
transfer time = (1 / S) (1 / ω)
service time = seek time + rotational latency + transfer time
```

Q6: SSD vs HDD SSDs have:

– More bandwidth (1GB/s read/write vs 100MB/s)
– Smaller latencies (microseconds vs milliseconds)

Provide an example of why HDDs are used.

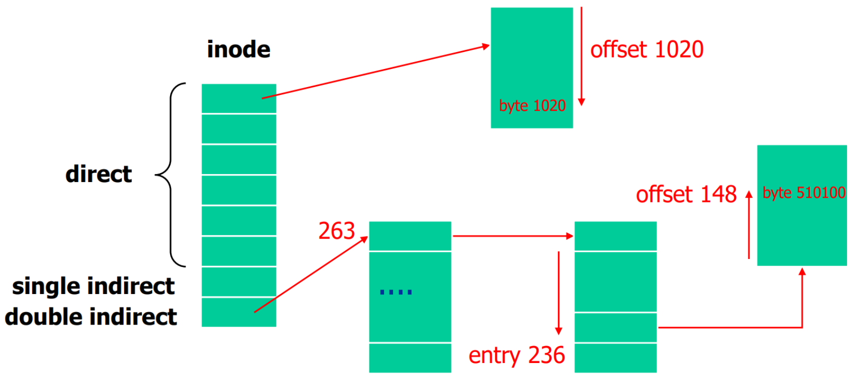
Used as basic storage by AWS, Azure, GCP

- GB/$ are much cheaper
- Lots of software is optimized for HDD

Week 8

Q1: Inodes In a particular OS, an inode contains 6 direct pointers, 1 pointer to a (single) indirect block and 1 pointer to a doubly indirect block.
Each of these pointers is 4 bytes long. Assume a disk block is 1024 bytes and that each indirect block fills a single disk block.
How many disk block reads will be needed to access:
– the 1020th data byte?
– the 510,100th data byte?

	Block chaining	File allocation table	Inodes
Byte 1020	2	2	2 (assuming inode not yet in memory)
Byte 510100	501	best case: 3 worst case: 500	4 (assuming inode not yet in memory)



1020 / 1024 = 1st block
510100 / 1024 = 499th block

Q2: File Access Consider a file system that maintains a unique inode for each file in the system. Each inode includes 8 direct pointers, a single indirect pointer, and a double indirect pointer. The file system block size is 1024 bytes, and a block pointer occupies 4 bytes.

How many disk operations will be required if a process reads data from the 513th block of a file? Assume that the file is already open, the buffer cache is empty, and each disk operation reads a single file block.

```
if 1 ≤ N < 8, then 1 operation is required
if 8 ≤ N < 8 + 256, then 2 operations are required
if 8 + 256 ≤ N < 8 + 256 + 256^2, then 3 operations are required
```

Q3: Fragmentation Explain what it means to "defragment a file system"?
Please describe a scenario where a file system will not benefit from defragmentation?

> Defragmenting a file system means that blocks belonging to the same file are moved around to ensure that they occupy consecutive disk blocks. This improves the read/write performance because it avoids unnecessary disk seeks.
>
> File systems on flashed storage do not usually benefit from defragmentation as accessing cells can occur in parallel.

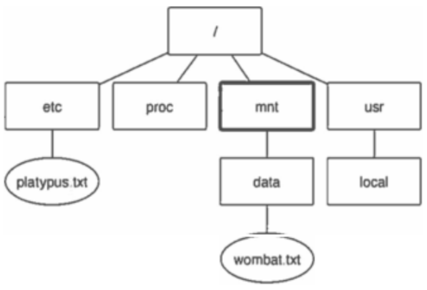
Q4: Exam 2016 Explain what it means for a file system to be mounted under Linux?

> Mounting is the process of taking multiple file systems and combining them under one name space.
>
> This lets us reference multiple file systems from the same root directory.

Q5: Exam 2016 Describe the actions that the Linux kernel would take when executing the following system call?

1. Traverse directory hierarchy to find file
2. Check access validity based on inodes
3. Initialise file descriptors to point to file

Q6: Exam 2016 Consider this diagram where: the root directory is indicated by / and the directory mnt is a mount point with a mounted file system.



Please decide if wombat.txt could be a hard link and/or a soft link to platypus.txt?

> Soft link but not hard linked

Week 9

Q1: One-way functions One-way functions are important for computer security and cryptography. Please describe what one-way functions are and give an example of how they are used in computer security today.

> `A one-way function is a function f where it is easy to compute f(x) = y but hard to compute x given y. In most operating systems, passwords are encrypted using a one-way function in order to prevent attackers, including system administrators trying to abuse their privileges, from obtaining the passwords.`

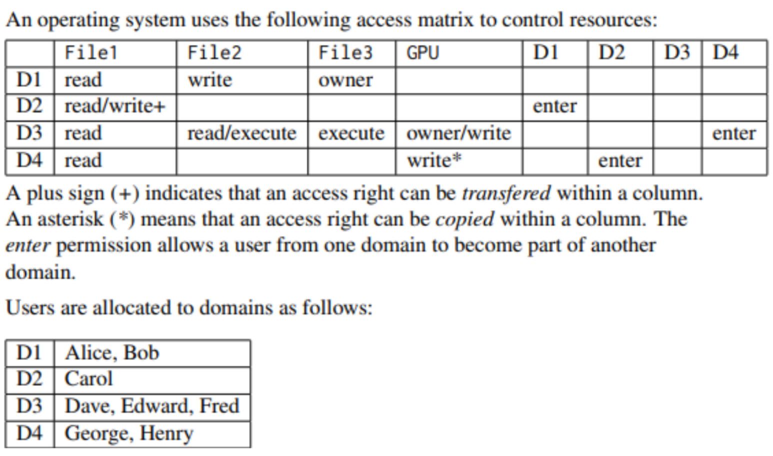
Q2: Usage of salt values Why do UNIX systems store passwords with a salt value ?

> `Salt is an additional parameter to the one-way hash function. It ensures that the same password will result in a different encrypted version.`
>
> `It also means that encrypted passwords are specific to the chosen salt value, so a single rainbow table for an attack cannot be pre-computed and reused to attack multiple accounts.`

Q3: Principle of least privilege Please give a brief definition of the principle of least privilege and explain why it should be employed by systems?

> `The principle of least privilege states that each user, program, process, or entity in general be granted the minimal amount of privilege that it needs to accomplish its designated tasks.`
>
> `It makes good sense as a system structuring principle because without excess privilege, entities cannot accidentally or maliciously perform actions or modify data they are not supposed to.`

Q4: Exam Question



An operating system uses the following access matrix to control resources:

	`File1`	`File2`	`File3`	`GPU`	D1	D2	D3	D4
D1	read	write	owner					
D2	read/write+				enter			
D3	read	read/execute	execute	owner/write				enter
D4	read			write*		enter		

A plus sign (+) indicates that an access right can be *transfered* within a column. An asterisk (*) means that an access right can be *copied* within a column. The *enter* permission allows a user from one domain to become part of another domain.

Users are allocated to domains as follows:

D1	Alice, Bob
D2	Carol
D3	Dave, Edward, Fred
D4	George, Henry

How can Carol write to the GPU device?

    A user in D4 can copy the write access to D2 or D1. Carol can access D2 & D1.

Who can access File2?

    Alice, Bob, Carol, Dave, Edward, Fred, George, Henry

Q6: Exam Question An attacker has physical access to a machine. Describe how you would implement the enforcement of access control decisions to files by the operating system so that the attacker cannot circumvent them.

With physical access to computer/peripherals one can:

- Read contents of memory/disks
- Listen to network traffic including (unencrypted) passwords
- Alter contents of memory/disks
- Forge messages on network
- Steal machine or set it on fire

Data must therefore be encrypted to prevent direct access. The encryption keys must not be under control of the operating system.