1.1 1. Referee: Allocation of resources e.g. CPU, physical memory, etc. Communicates between users and applications. Prevents bugs to crash applications or the entire computer. 2. Illusionist: Each application seems to run independently. Provides fancy object for users, for example, files. 3. Glue: Provides a set of common, standard services to applications. Allows copying data from one application and pasting it onto another application.

1.2. All of the criteria are important. However, when comparing these 3 criteria, portability is the least important to consider. Although the operating system should be able to run regardless of the hardware, but for game consoles, it takes a while to be built and released. This means that there are not that much hardware to be adapt.

1.3. Valid bootloader code must have magic signature of 0xAA at byte 510 and 0x55 at byte 511.

1.4.

In order to print out a character in a bootloader, it requires mov ah, 0Eh and int 10h to invoke the interrupt to print out the character.

```
bits 16                 ; Tell NASM this is 16 bit code

org 0x7c00              ; Tell NASM to start outputting stuff at offset 0x7c00

start:

boot:

    mov si, message     ; Point SI register to our message buffer

    mov ah, 0Eh         ; Specify 'int 10h' 'teletype output' function

    lodsb               ; Load byte at address SI into AL, and increment SI

    int 10h             ; Invoke the interupt to print out the character

    cli                 ; Clear interrupt flag

    hlt                 ; Halt execution

data:

    message db 'a', 10, 13, 0

; Pad to 510 bytes (boot sector size minus 2) with 0s, and finish with the two-
byte standard boot signature

times 510-($-$$) db 0

dw 0xAA55               ; Magic bytes - marks this 512 byte sector bootable!
```

However, you will need loop or recursion (with the same technique as printing out 1 character) to print out a string.

```
bits 16                 ; Tell NASM this is 16 bit code

org 0x7c00              ; Tell NASM to start outputting stuff at offset 0x7c00

start:

boot:

    mov si, message     ; Point SI register to our message buffer

    mov ah, 0x0e        ; 0x0e means 'Write Character in TTY mode'

.loop:

    lodsb

    or al, al           ; is al == 0?

    jz halt             ; if (al == 0) jump to halt label

    int 0x10            ; runs BIOS interrupt 0x10 - Video Services

    jmp .loop

halt:

    cli                 ; clear interrupt flag

    hlt                 ; halt execution

data:

    message db 'Hello', 10, 13, 0

; Pad to 510 bytes (boot sector size minus 2) with 0s, and finish with the two-byte standard boot signature

times 510-($-$$) db 0

dw 0xAA55       ; Magic bytes - marks this 512 byte sector bootable!
```

2.1. There are 2 mode of operations: Kernel and user modes. CPU always in one of the two modes. Dual-mode operation forms the basis for I/O protection, memory protection and CPU protection. Because sometimes, we need trusted codes while sometimes, we need untrusted code, we have to switch between modes.

2.2. 1. Hardware Interrupt (Asynchronous). 2. Processor exception (Synchronous): Segmentation fault, divide by zero. 3. System calls (Synchronous): User process calls OS services voluntarily.

2.3. 1. fork(). 2. execvp(const char* file, const char* argv[]). 3. wait(NULL).

3.1. 2^number of fork() = 2^3 = 8 processes.

3.2. You would execute the new executable and never call fork. It means if we call exec() before fork(), in the calling process, this system call (exec()) simply replaces the current process with a new program and the control is not passed back to the calling process (current process will terminate).

3.3. 1. Ignore the signal (do nothing). 2. Terminate the process (with optional core dump). 3. Catch the signal by executing a user-level function called signal handler.

3.4. The kill command in UNIX enables the user to send a signal to a process.

3.5. An exit status is the number returned by a computer process to its parent when it terminates. Its purpose is to indicate either that the software operated successfully, or that it failed somehow. The value of an exit status is an integer. To extract the exit condition, use the WIF- macros e.g.

- WIFEXITED(status): child exited normally
    - WEXITSTATUS(status): return code when child exits
- WIFSIGNALED(status): child exited because a signal was not caught
    - WTERMSIG(status): gives the number of the terminating signal
- WIFSTOPPED(status): child is stopped
    - WSTOPSIG(status): gives the number of the stop signal

4.1. a. It uses CPU and disk equally. Use more threads.

4.1. b. It uses too much CPU, but doesn't use much disk. Get faster CPU.

4.1. c. It uses too much disk, but doesn't use much CPU. Get faster disk.

4.2. a. Yes. Since threads do not ensure the ordering. It may happen that one is holding lock a and wants to acquire lock b, while another is holding lock b and wants to acquire lock a. Because neither can complete what they are trying to do both will end up in deadlock. If a process is unable to change its state indefinitely because the resources requested by it are being used by another waiting process, then the system is in a deadlock.

4.2. b. We need an ordering to lock the lock. It must be the same ordering for all threads. We can use min and max of the input so that it is guaranteed that it will be in the same ordering. For example, use min(a, b) and max(a, b) to arrange them.

4.3. a. Kernel threads because I/O operation is needed and it is a long-running task.

4.3. b. User-level threads because no I/O operation needed.

4.4. **4**: Execute thread 1 completely, then execute thread 2. **7**: Interrupt thread 1 before c = b + a, and then execute thread 2, followed by executing thread 1 again. **6**: Execute thread 2 till b = 10 is done, then interrupt it, and execute thread 1 completely. **13**: Execute thread 2 completely, then thread 1. **-3**: In thread 1, check a < b, which is false. This takes you to the else condition. Read the value of b, which is 0. Switch to thread 2, execute it completely. a is now -3. Switch back to thread 1, read a, which is -3. Then, add and assign -3 to c.

5.1. 1. It is non-preemptive algorithm, which means the process priority doesn't matter. If a process with very least priority is being executed, more like daily routine backup process, which takes more time, and all of a sudden, some other high priority process arrives, like interrupt to avoid system crash, the high priority process will have to wait, and hence in this case, the system will crash, just because of improper process scheduling. 2. It is not optimal Average Waiting Time. 3. Resource's utilization in parallel is not possible, which leads to Convoy Effect, and hence poor resource (CPU, I/O etc) utilization.

5.2. It can only estimate the length which should be similar to the previous one, then pivk precoess with shortest predicted next CPU burst.It can be done by using the length of previous CPU bursts, using exponential averaging: tau_n+1 = (alpha)(t_n) + (1 – alpha)(tau_n). t_n = actual length of previous nth CPU burst. tau_n+1 = predicted value for the next CPU burst. alpha, 0 <= alpha <= 1, commonly alpha is set to 1/2.

5.3. a. A small unit of CPU time, usually 10-100 ms, that each process gets. After this time has elapsed, the process is preempted and added to the end of the ready queue.

5.3. b. Using a small quantum will not cause the response time of the processes to be too much which may not be tolerated in interactive environment. This means that it will have time for context switch to do other works as well.

5.3. c. Using a large quantum will not cause unnecessarily frequent context switch leading to more overheads resulting in less throughput. This means that it will have time to do the current work, not just keep doing context switching.

6.1. The principle of least privilege is a concept in computer security that limits users' access rights to only what are strictly required to do their jobs. Users are granted permission to read, write or execute only the files or resources necessary to do their jobs. In the amusement park, there might be several kinds of ticket. People with riding roller coaster only ticket cannot play carousel. People with taking a picture only cannot play anything.

6.2. a. Access list: Expensive and needs to be performed every time the object is accessed. For example, it can be seen that the guard has to give a sticker to everyone when entering the building, which is really expensive.

6.2. b. Capability list: It is too flexible and lead to lack of control. For example, a stranger can enter the Science Division by stealing someone’s ID card.