(6 p) Given a file system that contains 100 files of 1Kb, 900 files of 4Kb and 500 files of 6Kb; running on a disk whose time to read/write 1 block is 1 ms, compute the disk utilization and the time needed to read all the files for the following block sizes: 0.1 Kb, 0.5 Kb, 1 Kb, 4 Kb, 8 Kb, 16 Kb.
[**HINT:** Compute the number of blocks needed for each file based on its size and for each block size (floor (size/block size)). Once you have the number of blocks, compute reading time (basically divide by 1000 to get seconds) and disk utilization (mutiply blocks by block size)]

(6 p) Given a file system that contains 200 files of 1Kb, 100 files of 4Kb and 250 files of 6Kb; running on a disk whose time to read/write 1 block is 1 ms, compute the disk utilization and the time needed to read all the files for the following block sizes: 0.1 Kb, 0.5 Kb, 1 Kb, 4 Kb, 8 Kb, 12 Kb.
[**HINT:** Compute the number of blocks needed for each file based on its size and for each block size (floor (size/block size)). Once you have the number of blocks, compute reading time (basically divide by 1000 to get seconds) and disk utilization (mutiply blocks by block size)]

(6 p) Given a file system that contains 200 files of 1Kb, 300 files of 2Kb and 550 files of 4Kb; running on a disk whose time to read/write 1 block is 1 ms, compute the disk utilization and the time needed to read all the files for the following block sizes: 0.1 Kb, 0.5 Kb, 1 Kb, 2 Kb, 8 Kb, 16 Kb.
[**HINT:** Compute the number of blocks needed for each file based on its size and for each block size (floor (size/block size)). Once you have the number of blocks, compute reading time (basically divide by 1000 to get seconds) and disk utilization (mutiply blocks by block size)]

(6 p) In the following, you are presented a series of statements about CPU-bound and I/O-bound processes. For each statement, please state if you agree or disagree with the statement, and motivate why.

- Differently from I/O-bound processes, CPU-bound processes require a larger amount of time to complete their execution.
- I/O-bound processes are usually run to exchange messages with the user / other devices, while CPU-bound processes are not.
- preemptive scheduling does not make sense in a system in which most of the processes are I/O-bound.

[**HINT:** Disagree with the 3. Amount of time to complete does not depend on type of processes. CPU-bound does not mean it does not do IO, it could be doing that but it takes time to elaborate the message. It does, imagine priority with a bunch of CPU-bound processes, it could take "forever" to schedule I/O.]

(6 p) During the course, we discussed (several times) that different aspects of an Operating Systems are interdependent. Explain why a dependency exists among the following 2 aspects and provide an additional example of 2 other interconnected aspects of an OS:

1. the procedure to select a process that could be swapped out, and
2. the handling of pending I/O requests.

[**HINT:** if you pick a process that is waiting, where do you store the I/O results? Other examples? see other variations of this question]

(6 p) During the course, we discussed (several times) that different aspects of an Operating Systems are interdependent. Explain why a dependency exists among the following 2 aspects and provide an additional example of 2 other interconnected aspects of an OS:

1. the time it takes to fork a process, and
2. the memory management scheme used by the OS.

[**HINT:** decoupling physical from logical representation (virtual memory) means fork can be take very little. Other examples? see other variations of this question]

(6 p) During the course, we discussed (several times) that different aspects of an Operating Systems are interdependent. Explain why a dependency exists among the following 2 aspects and provide an additional example of 2 other interconnected aspects of an OS:

1. the page fault rate of a process, and
2. the working set of a process.

[**HINT:** if working set is of the right size, peaks on page fault rate only when changing locality. Other examples? see other variations of this question]

(6 p) Would it be possible to rely on the lottery scheduling to approximate the behavior or shortest job first scheduling in a starvation-free manner? Justify your answer and provide examples supporting your claims.
[**HINT:** Yes, at least 1 ticket to each process (starvation free) and then number of tickets inversely proportional to length of process.]

(6 p) Imagine you are developing the scheduler for an Operating System, and suppose such Operating System defines a function that can be used to check if a pipe exists between two processes. If the pipe is unidirectional, the function also allows to understand which process is acting as producer and which process is acting as consumer. How can you use such a function to support your scheduling decisions?
[**HINT:** Can help deciding which processes to co-schedule since they are prod/cons pairs.]

(6 p) As we discussed in the course, the Hypervisor in charge of coordinating several VMs maintains a shadow page table, which needs to be synchronized accordingly to the changes made by each VM to their own page tables. To perform such synchronisation, the Hypervisor does not mark the pages used by the VMs to maintain their page tables as read-only, but instead relies on the page faults issued by each VM to update its own shadow page table. What are the pros and cons of such an approach?
[**HINT:** Pros: Hypervisor does not gain control for changes that do not really require its intervention. Cons: freed-up pages will be found later, only when a page fault is induced by the Guest.]

(6 p) As we discussed in the course, the Hypervisor in charge of coordinating several VMs maintains a shadow page table, which needs to be synchronized accordingly to the changes made by each VM to their own page tables. To perform such synchronisation, the Hypervisor marks the pages used by the VMs to maintain their page tables as read-only, instead of relying on the page faults issued by each VM to update its own shadow page table. What are the pros and cons of such an approach?
[**HINT:** Cons: Hypervisor might gain control for changes that do not really require its intervention. Pros: freed-up pages will be found immediately, not only when a page fault is induced by the Guest.]

(6 p) Do all guest-induced page faults need to be re-injected to the guest OS? Motivate your answer.
[**HINT:** Yes, they belong to the guest OS and need to be handled by it, no matter the extra steps the Hypervisor applies for virtualization. Check lecture about virtualization.]

(6 p) Suppose you decide to take advantage of a multicore CPU by running one dedicated OS in each core, while partitioning the main memory so that each OS has its own portion of it. Can such an approach guarantee consistent views on the data accessed by the OS? Motivate your answer.
[**HINT:** No (think about writes/reads to/from disk files. Check lecture about scheduling.]

(6 p) The following is a proposal for a solution to the critical section problem for n threads. Argue about its safety, progress and fairness properties.

```
shared var
    num: array[0..n-1] of int, init 0
    K: int, init 1
    T: int, init 1


thread-i {
    repeat
        [Other code]

        num[i] := FAI(&K)
        while num[i] != T do [nothing] end-while

        [Critical Section]

        FAI(&T)

    forever
}
```

where FAI is defined as follows

```
int FAI(int *t){
  int r := *t
  *t++
  return(r)} // Executed atomically
```

[**HINT:** Similar argumentation as for Lamport's algorithm discussed in class, although simpler here, as this builds on stronger HW primitives, i.e., the atomic RMW instruction FAI (fetch and increment). The variable K distributes keys ("nummerlappar") and the variable T tells whose key's turn is to enter the CS.]

(2 + 4 = 6 p)

- Consider two threads, ThreadP and ThreadC. Thread ThreadC must execute operation opC only after thread ThreadP has completed operation opP. How can you guarantee this synchronization using semaphores?

  [**HINT:**
  ```
  binary semaphore S, init 0
  ThreadP:
   opP
   signal(S)
  ThreadC:
   wait(S)
   opC
  ```
  argue why the required order is guaranteed, as we did in the lecture studying this synchronization mechanism.]

- Consider two threads, ThreadP and ThreadC, that must take turns executing operation opP and operation opC respectively for 100 times. ThreadP must be the one that executes opP first. How can you guarantee that using semaphores? Formulate the safety and progress requirements and argue about the correctness of your solution.

  [**HINT:** Arguing as for the producer consumer problem discussed in the lectures. Now the threads work in a loop, using semaphores SP and SC: in each of their respective loop iterations, ThreadP must wait for opC, ie wait(SC) (except the first time, hence SC is initialized to 1) and C must wait for opP, ie wait(SP), with SP initialized to 0. After opP (resp opC), ThreadP executes signal(SP) (resp ThreadC executes signal(SC), to generate the required signal to enable the other one to proceed. i.e.

  ```
  binary semaphore SP, SC, init 0 and 1 respectively
  ThreadP:
   for i = 1 to 100 do
    wait(SC)
    opP
    signal(SP)
   end-for
  ThreadC:
   for j = 1 to 100 do
    wait(SP)
    opC
    signal(SC)
   end-for
  ```
  The required properties are that no thread will block forever [progress] and that (i) it cannot be the case that we have more than one opP (respectively opC) between successive opC (respectively successive opP) and (ii) opP is executed first [safety]. Using the methodology discussed in the lectures (i.e. assume that the wrong thing can happen and show that it cannot due to the ordering of events) and the above arguments, show that they hold.

  Alternatively argue that this is a bounded buffer problem with buffer size 1 and adapt the bounded buffer solution discussed in the lectures.]