# Operating Systems Problem session 6

Dibran Dokter 1047390 & Marnix Lukasse 1047400

October 22, 2019

**10.6** The OS can read the rest of the file before the program needs it because it knows it is reading it sequentially. This reduces lookup time and increases the performance.

**11.8** We assumed that 8kb = 8000 bytes, 2kb = 2000 etc. If we understand correctly: we have 12 direct disk blocks (12 * 8Kb = 96Kb).
single indirect disk block: 8000/4 = 2000 pointers to 8Kb blocks ( 2000 * 8 = 16000Kb).
double indirect disk blocks: 8000/4 = 2000 pointers to 2000 pointers to 8Kb blocks (2000 * 2000 * 8 = 32000000Kb).
triple indirect disk blocks: 8000/4 = 2000 pointers to 2000 pointers to 2000 pointers to 8Kb blocks (2000 * 2000 * 2000 * 8) = 64000000000Kb

Because of the increase in size, only the triple indirect disk block really counts and the other numbers are negligible. This turns out to be 64 petabytes.

**13.5** To handle an interrupt we need to save the state of the current process and then handle the interrupt. After the interrupt has been handled we need to restore the running process. To handle the interrupt we also need to find the correct interrupt handler. This adds overhead to handling the interrupts.

**13.6** Blocking I/O should be used in the following circumstances:
- When we need to wait for the user input before the process can continue.
- When we want to read the entire file in one go and we need it before we can continue.
- A server waits for input and when there is input it handles it all at one time.

Non-Blocking I/O should be used in the following circumstances:
- A user interface that receives user input while handling the display of the data.
- A video player that reads data from the disk while it decompresses and displays it.
- A networked program that sends data over the network all the time and changes when the user inputs data.

We should not implement non-blocking I/O with busy-wait because then the program that calls the I/O needs to be waiting where with normal non-blocking I/O it would be able to do different things.

**14.2** Yes, this is the same.

**14.13** Decreases the harm a potential system break-in can cause because the user has limited permissions.

**15.12** In general, symmetric is less safe but faster to encrypt/decrypt and asymmetric is safer but it also takes more computational power to encrypt/decrypt and its therefore slower.

In a distributed network, one would usually use asymmetric for really private data (like login credentials), and symmetric when you need to send and less private data (video stream/raw data). An approach seen often, is to use asymmetric encryption to share a key, which then can be used for symmetric encrypt/decrypt.

**15.13** Because in this asymmetric encryption protocol, one party sends over the public key. Anyone in the network might be able to intercept this public key, and use this key to encrypt some data. So when the named original party receives a message, it can't be sure who the sender is.
It could be used for a login procedure for example, the server then authenticates the right user by having the user encrypt the correct password and send it to the server.

This could be used for sending an encrypted message to the holder of the private key. So we can ensure encryption between the holders of the public key and the holder of the private key.

**15.14** a)
This can be ensured by having the user encrypt a message using his private key. Because only that user has the private key we can be sure that he is the sender.
b)
This can be ensured by encrypting the message with the users public key. Since only the private key can decrypt the message we can be sure that nobody else can read it. c)
For this we need both parties to have a set of public/private key. When we use both (double encryption) we can fulfill the conditions.