1.

a.

i. The first block of the file is read, and the byte at index 2126 (the 70th byte of the immediate data) is the 70th byte of the file. Only one block needs to be read

ii. The first block of the file is read. Bytes 2049-2052 of this block (the indirect block pointer) are read, and the block at the address given by that pointer is read. The first 4 bytes of this new block are read, and the block at the address given by these bytes is read. The 4th byte of this 3rd block is the 2^{20} + 2044th byte of the file. 3 blocks are read in total

b. $2^{20}+2^{22}+2048$ bytes

c.

i. The time of creation is stored in the directory that references the file. This is because the notion of the “creation time” or “ctime” is ambiguous in certain cases (e.g., when the file is restored from a backup, or if certain editors write to a temporary file when editing and then replace the original with the temporary file on saving), and so the notion has much more to do with the specific filesystem being used than the intrinsic metadata of the file

ii. The filename is stored in the directory that references the file. This is so that hard links can exist (the same inode existing with different filenames in different directories)

iii. The file access rights are stored in the inode because of security. If they were instead stored in the directory, hard links would enable a user who does not have these permissions, to create a reference in a different directory to the same inode, but giving themselves permission to read/write/execute the file.

d. This would greatly speed up reading files. If you know the address of the first block of the file, and you know the index of the file you want to read, you can calculate the address of the block you need. This allows you to read any block of the file with only a single block read, where before it could have taken up to 4. Writing to files would be similarly sped up as long as the file size does not increase so as to require an additional block, because again if you know the index of the byte to which you’re writing, you can calculate its address and write to it with only a single block access. However, if another block is required, but the next consecutive block on the disk is not available, this would be a very slow operation as the entire file would need to be copied into a larger free region of contiguous blocks. If no such region exists, the entire disk would need to have its blocks rearranged in order to make space. A similar speed decrease would exist for creating a file. For large, files a large contiguous region of free memory would need to exist in which to put the data, and if no such region exists, the disk’s blocks would need to be rearranged to create one.

2.
   a.
      i. The text segment holds the machine instructions for the program. This stays constant as the process executes

      ii. The data segment contains variables and their values. This grows upwards as more memory is allocated to the process

      iii. The stack segment is used for activation records, such as storing local variables and function parameters. This grows downwards as more functions are called.

   b. The operating system has hardware support to protect this memory region such that it is not accessible unless the processor is running in kernel mode. A process gains access to this region by using syscalls, which switches the processor from user mode to kernel mode.

   c. Blocking I/O makes the process wait until the I/O event has finished before execution can continue. For example, if user keyboard input is required at a command terminal. Non-blocking I/O initiates the I/O event but continues to execute without waiting for the event to finish. For example, if a sound needs to be played to indicate that some event has occurred. Asynchronous I/O is when an I/O event is initiated, and the process can continue to execute while the I/O event has not finished, but when the I/O event does finish, a callback is triggered in the process. For example, to listen for button presses, or incoming network packets.

   d.
      i. Whenever there is a syscall for a file to be read from, written to, or created.

      ii. When an interrupt occurs, the data should be read from the device (or some buffer), write this data into memory, and then resume normal execution of whatever process was running.

      iii. I could cache blocks from the hard disk which are read frequently. I could try to pre-empt which blocks from the hard disk will be read from next and read them in advance.