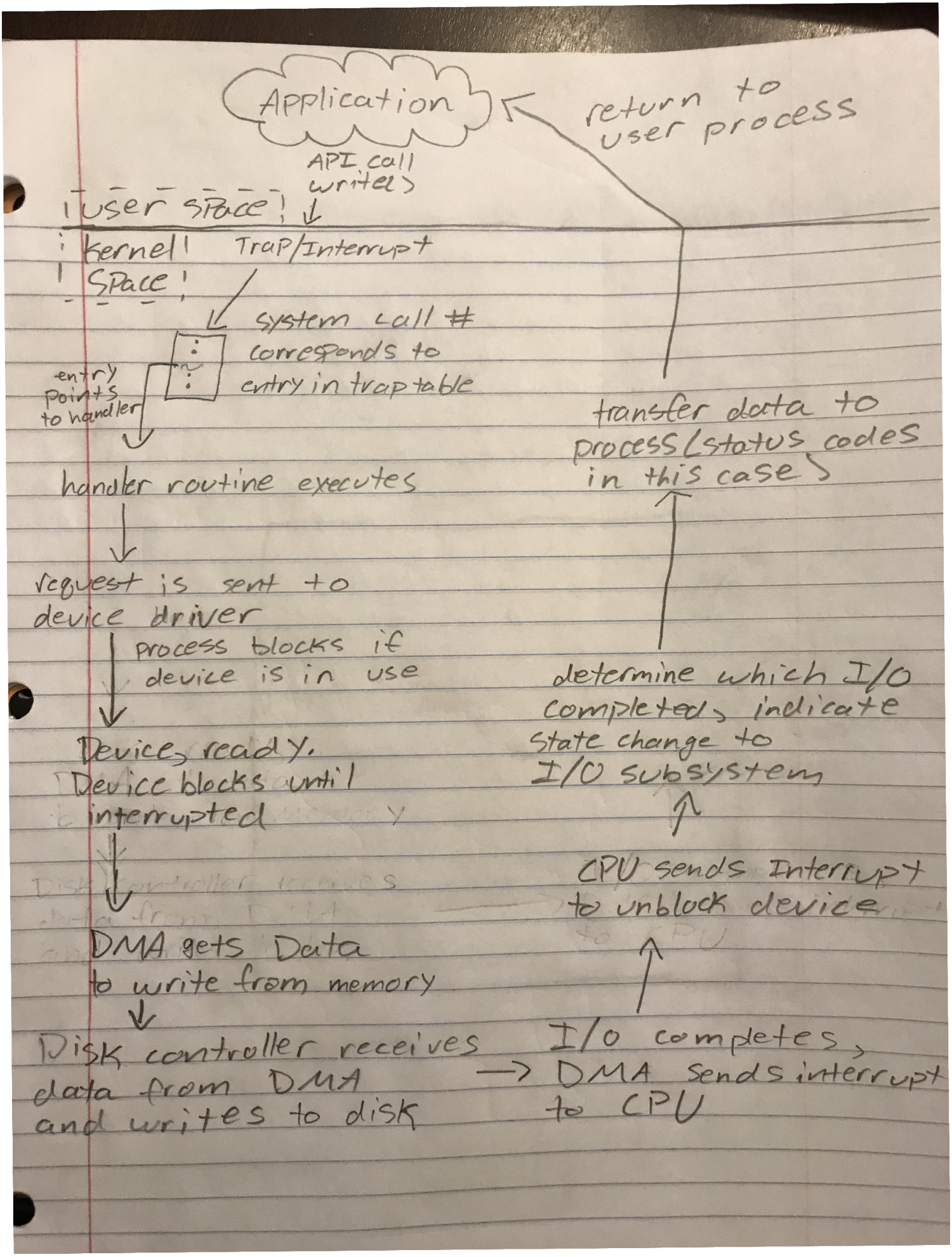
1. A trap is a software generated interrupt. A trap can be caused by an error or by a system call from a user program. When a trap is called, the system changes from user mode to kernel mode and the operating system takes control.

2. When a system call is executed, the system treats it as a software interrupt. The trap table (or interrupt vector) tells the operating system which interrupt handler routine to execute.

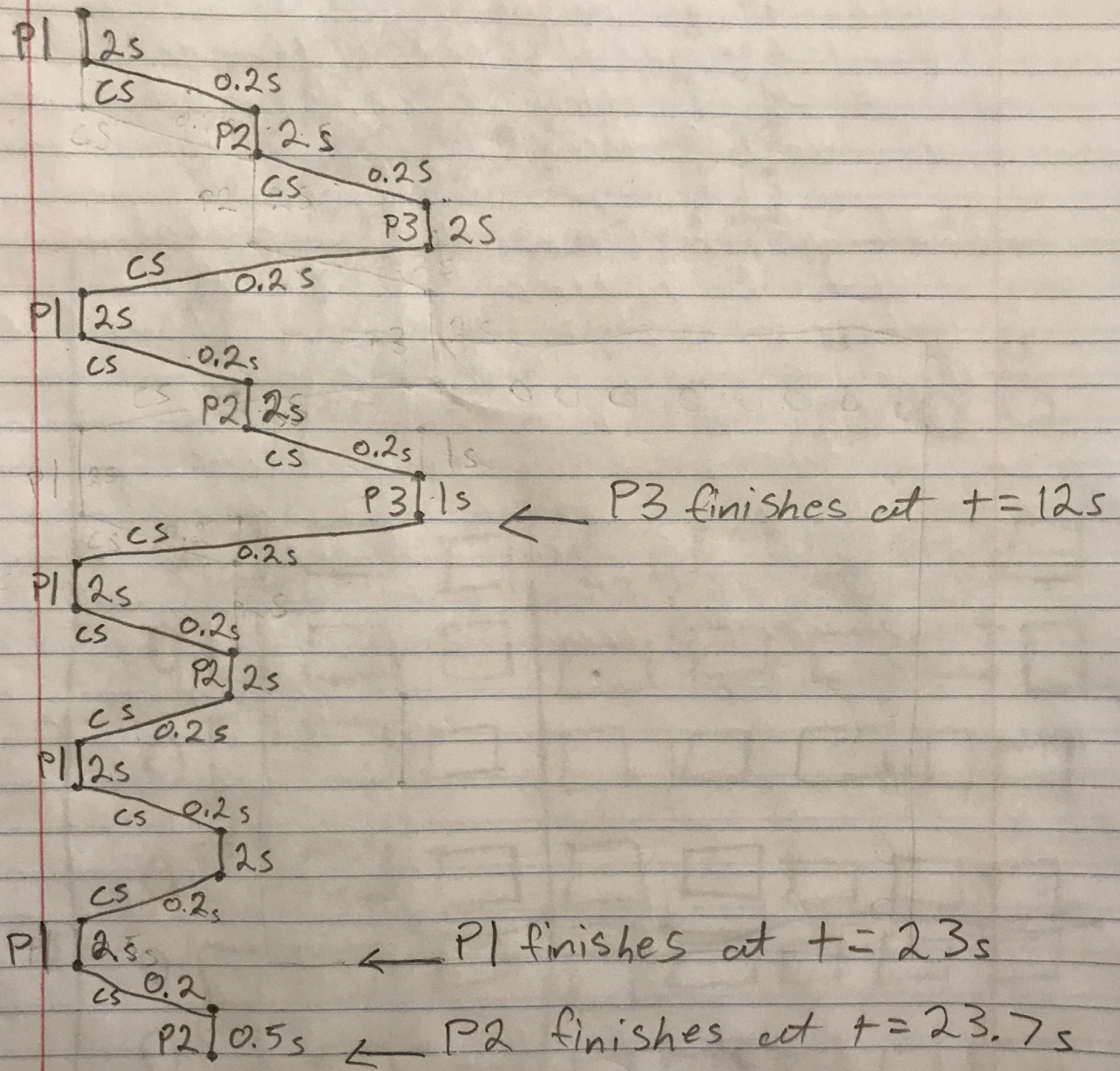
3. In order for the kernel to communicate with a device, it needs a device driver. One way to implement a device driver is with a Loadable Kernel Module. An LKM exists in the form of an executable object file (or .o file).
   - To insert an LKM use 'insmod <module-name>.o'
   - To check which modules have been loaded, use 'lsmod'
   - To remove a module, use 'rmmod <module-name>'
   - The 'modprobe' command can be used to intelligently insert or remove LKMs. This command is easier to use than insmod and rmmod.

   - In Linux, devices are special files in the /dev directory. You can create a new device by using the makedev program. makedev uses the mknod command to set the device name, type, and major and minor device numbers. makedev also sets the file permissions and ownership properties. Because the mechanisms that pass requests between devices and applications are general, devices and drivers can be introduced into the computer without recompiling.

4. One way that I/O can be overlapped with CPU execution is with interrupt driven I/O. In this case, a device will notify the CPU when it is ready for use with an interrupt. This method is better than polling because the CPU only has to deal with the device when it is ready for input, when output is complete, or when an error has occurred. Another way of overlapping I/O and device execution is with DMA (or Direct Memory Access). This method uses a special purpose processor used for I/O called a DMA controller. The DMA controller operates the memory bus directly, without the help of the CPU. This method is advantageous because the CPU only receives an interrupt when the I/O process is finished.


Application
return to user process
API call writes
user space
kernel space
Trap/Interrupt
system call # corresponds to entry in trap table
entry points to handler
handler routine executes
request is sent to device driver
process blocks if device is in use
Device ready. Device blocks until interrupted
DMA gets Data to write from memory
Disk controller receives data from DMA and writes to disk
I/O completes, DMA sends interrupt to CPU
CPU sends Interrupt to unblock device
determine which I/O completed, indicate state change to I/O subsystem
transfer data to process (status codes in this case)

P1: 10s P2: 8.5s P3: 3s
Time slice: 2s CS overhead: 0.2s



Total time: 23.7s
Time executing Processes: 21.5s
Total time context switching: 2.2s
% overhead = 2.2s/23.7s = 0.092 ≈ 9%