211 — Operating Systems – Tutorial
Device Management

Peter Pietzuch <prp@doc.ic.ac.uk>

Note that the solution notes below only briefly list (some of) the key points that should be included in an answer. They are by no means complete. In an exam, you are expected to spell out the solution more fully and include a detailed explanation of your reasoning.

1. In which of the four I/O software layers (*user-level I/O software*, *device-independent OS software*, *device drivers* and *interrupt handlers*) is each of the following done?

   (a) Computing the track, sector and head for a disk read.

   (b) Maintaining a cache of recently used blocks.

   (c) Writing commands to the drive registers.

   (d) Checking to see if the user is permitted to use the device.

   (e) Converting binary integers to ASCII for printing.

   Solution Notes:

   (a) *Device driver (or hardware):* This requires familiarity with the actual disk layout used, so only the hard disk driver will have the necessary information. For modern hard disks, the mapping will often be done entirely in hardware by the disk controller because only the built-in disk hardware may be familiar with the location of bad blocks etc.

   (b) *Device-independent OS layer:* A block cache is useful across a wide-range of block I/O devices. By putting this functionality into the device-independent layer, it can be reused/shared by multiple different devices.

   (c) *Interrupt handler (or device driver):* If this is a quick but time-critical operation, then it can be performed within the interrupt handling routine. If updating the drive registers requires more time (or is device-specific), then it would be done by the device driver.

   (d) *Device-independent OS layer:* Performing access control checks for devices is functionality that is applicable to many devices. Therefore it makes sense to factor it out and provide it in the device-independent layer. The OS can then enforce access control policy uniformly across all devices.

   (e) *User-level I/O layer:* Typically conversion between data representations for I/O will be done by a user-level I/O library that an application can linked against. This gives the user maximum flexibility in choosing the right conversion strategy. In addition, the conversion can be done cheaply in user space because it does not need any privileged kernel access.

2. What is the difference between

   (a) a *device driver* and a *device controller*?

   (b) a *block-oriented* device and a *character-oriented* device?

   Solution Notes: A device driver is software: that part of the OS that is responsible for interacting with a particular device. A device controller is a piece of hardware that controls the device and implements an interface between the device and the rest of the system. Block devices store and retrieve data in units of fixedsize blocks. Character devices supply (or consume) streams of characters, i.e., bytes.

3. What is *memory-mapped IO*? Why is it sometimes used?

   Solution Notes: Memory-mapped I/O is a mechanism by which I/O devices are mapped into the regular memory address range. This has the advantage that memory and I/O accesses can be treated uniformly by the programmer. No special instructions are required in a programming languages to carry out I/O operations. In addition, the regular OS mechanisms for memory protection can be used to protect I/O devices too. To implement memory-mapped I/O, the memory controller (e.g. as part of the Northbridge) must intercept requests to address ranges that represent I/O devices and put them on the PCI bus instead of the memory bus. Care must be taken to bypass any memory caching for I/O devices.

4. An alternative to using interrupts for I/O is *polling*. Are there any circumstances when using polling is a better choice?

   Solution Notes: Polling has the benefit that it is a simple mechanism and thus easier to implement than interrupt-driven I/O. This is big advantage for embedded devices that are resource-constraint in terms of their OS footprint. Also, in an embedded system, the CPU may not have anything else to do while waiting for I/O to finish. It may also be fine to use I/O polling when the I/O operation is supposed to finish quickly, making the overhead of a context switch in interrupt-driven I/O unnecessary.

5. Explain what *direct memory access (DMA)* is and why it is used.

   Although DMA does not use the CPU, the maximum transfer rate is still limited. Consider reading a block from disk. Name three factors that might ultimately limit the rate of transfer.

   Solution Notes: Direct Memory Access (DMA) is a technique to relieve the CPU from the task of handling low-level device I/O. A special hardware device called a DMA controller takes over the responsibility of handling I/O. For example, the CPU can request the reading of certain blocks from a disk by telling the DMA controller which blocks to read and where to put the result in memory. It then carries out other tasks, while the DMA controller communicates with the disk controller to do the I/O. After all blocks have been read, the CPU is notified by the DMA controller using an interrupt that the blocks are available in memory. Simple DMA controllers can only handle a single read at a time, whereas more sophisticated ones support multiple reads and putting the results at different memory locations (scatter-gather DMA).

   The transfer rate of blocks from a disk to memory using DMA will be limited by:

   (a) the transfer rate supported by the hard disk, i.e. how fast the disk can read blocks and put them on the bus.

   (b) The transfer rate supported by the bus and potential contention for bus access.

   (c) The access bandwidth supported by the memory and, again, potentially contention with the CPU for memory access.

6. What is *spooling*?

   Why is a printer spooling system better than direct user access to printers?

   Solution Notes: Spooling is the allocation of all interaction with a given device to a single process; other processes that want access to the device must go through the spool process. A non-spooled printer is vulnerable to two or more people printing on it at once: in such cases the result will be gibberish. A spooled printer does not suffer from the problem, since it has better means of communication with client processes (e.g. depositing and noticing a file in the spool directory). Spooling also frees processes from having to supervise the printing process; the lpr command can return without waiting for the file to be printed. This is much better than the usual situation on non-spooled single-tasking systems.

7. An operating system has to support I/O devices with very diverse properties. Complete the following table, as exemplified below, using your best guesses.

Device	**Data rate**	**Type** (**C**haracter/**B**lock)	**Operation** (**R**ead, **W**rite, **S**eek)
Clock			
Keyboard			
Mouse			
56k Modem	*7 KB/sec*	*C*	*R,W*
ISDN line			
Laser Printer			
Scanner			
52x CD-ROM			
FastEthernet			
EIDE (ATA-2)disk			
ISA bus			
Fire Wire (IEEE 1394)			
USB 2.0			
XGA Monitor			
Gigabit Ethernet			
Serial ATA disk			
SCSI Ultrawide4 disk			
PCI bus			

Solution Notes:

Device	**Data rate**	**Type** (**C**haracter/**B**lock)	**Operation** (**R**ead, **W**rite, **S**eek)
Clock	7.5 bytes/sec	???	???
Keyboard	10 bytes/sec	C	R
Mouse	100 bytes/sec	C	R
56k Modem	7 KB/sec	C	R,W
ISDN line	16 KB/sec	C	R,W
Laser Printer	100 KB/sec	C	W
Scanner	400 KB/sec	C	R
52x CD-ROM	8 MB/sec	B	R,S
FastEthernet	12.5 MB/sec	C	R,W
EIDE (ATA-2)disk	16.7 MB/sec	B	R,W,S
ISA bus	16.7 MB/sec	C	R,W
Fire Wire (IEEE 1394)	50 MB/sec	C	R,W
USB 2.0	60 MB/sec	C	R,W
XGA Monitor	60 MB/sec	C	R,W
Gigabit Ethernet	125 MB/sec	C	R,W
Serial ATA disk	300 MB/sec	B	R,W,S
SCSI Ultrawide4 disk	320 MB/sec	B	R,W,S
PCI bus	528 MB/sec	C	R,W

8. Explain how one can provide an *asynchronous* I/O API on top of a *blocking* I/O system call interface.

   Solution Notes: It's possible to implement a blocking I/O system call in a separate thread that executes the I/O operation concurrently. When the I/O system call has finished, the I/O thread notifies the main thread using a thread synchronisation primitive.

   You have to implement a web server that should handle thousands of concurrent incoming connections. What would be the advantages of using a non-blocking I/O interface for this?

   Solution Notes: In this scenario, blocking I/O would involve many threads handling concurrent connections, which could have a considerable context-switching overhead. Non-blocking I/O means that the web server can be implemented using an event-driven design: An event loop polls all active file descriptors (using a "select" system call) to obtain a set of file descriptors with outstanding I/O operations. It can then service them using non-blocking read/write calls.

9. Write a C program that implements the *copy* (`cp`) command. Your program should be invoked as

```
mycp <source file> <destination file>.
```

(a) Write your program on a sheet of paper. Make sure that you use the correct Linux I/O calls.

(b) Now try running your program on a computer. How efficient is your implementation compared to the standard `cp` command? You can use the `time` command to measure execution times for various file sizes. If there is a performance difference, can you explain it?

(c) The `strace` command can be used to trace the system calls that a program makes. Compare the system calls between `cp` and `mycp`. Again, can you explain the differences?

Solution Notes:

(a)
```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>

void pHelp(char *mycp) {
  fprintf(stdout, "usage: %s <srcfile> <destfile>\n", mycp);
  exit(0);
}

int main(int argc, char *argv[]) {
  int inFile, outFile;
  char line[512];
  int bytes;

  if(argc < 3)
    pHelp(argv[0]);

  if((inFile = open(argv[1], O_RDONLY)) == -1) {
    perror("couldnt open input file");
    exit(-1);
  }

  if((outFile = open(argv[2], O_WRONLY | O_CREAT)) == -1) {
    perror("couldnt open output file");
    exit(-1);
  }

  while((bytes = read(inFile, line, sizeof(line))) > 0)
    write(outFile, line, bytes);

  close(inFile);
  close(outFile);
}
```

(b) Any performance difference can probably be explained by the larger block size (32KB) that cp uses by default.

(c) The standard cp command probably does additional calls to stat64 and fstat64 to check for the existence of the source and destination files and ensure their successful access.