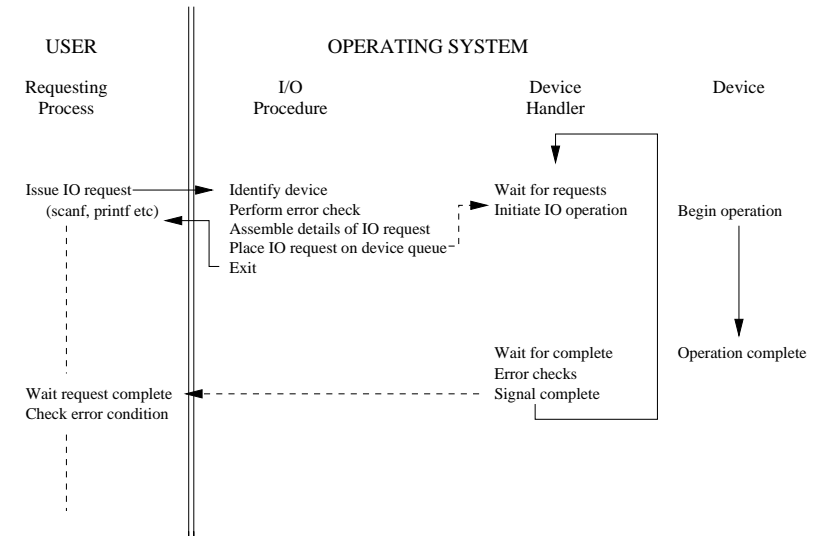


DMA (Direct Memory Access) I/O

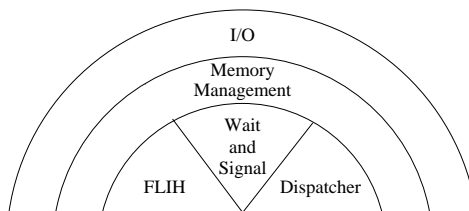
- usually multiple characters (often KBs or MBs) are sent in sequence
- with DMA, a special purpose external processor (DMA controller) does the programmed I/O for us, and then tells us when it is finished
- number of interrupts is reduced dramatically over that of interrupt-driven I/O
- each transfer uses the bus ([O'H&Bryant, fig 1-4]); what if CPU needs to use it?
- example: output 4096 characters from address 1024 onwards to a disk:
 - send start address 1024 and the length 4096 to the DMA controller
 - in turn, sends a code (number) for the disk device (e.g. 4), and a write code
 - the DMA controller transfers the data from memory to the disk using programmed I/O
 - when finished, the DMA controller sends an interrupt signifying completion
- when directly communicating with a device (e.g. DMA controller), the CPU reads/writes to registers on that device:
 - this could be done via special instructions (drawbacks?)
 - or memory-mapped I/O: map these registers to special locations in main memory (thus memory can provide an *abstraction* to devices)

Basic Input/Output Operation



Responsibility: Virtual I/O

- if direct access to I/O devices was allowed at user-level:
 - inconvenient: requires users to have substantial knowledge of the devices
 - also when reading and writing, many different (correctable) errors may occur
 - arbitration problems: what if two users tried to access the disk simultaneously?
 - this would leave the machine without any data security!
- instead, O/S level I/O instructions (traps) provide an abstraction to I/O:
 - users usually don't want to know (just want a successful transfer)
 - hide much of this complexity from the user and provide a level of security
- the parts of the OS which access devices are called device drivers
- OS structure for virtual I/O:



File Organisation and Types

- files provide a means of organising and accessing data in a convenient manner
- a file is a linear sequence of 8 bit bytes indexed from 0 to some maximum (e.g. $2^{32}-1$ or $2^{64}-1$)
- associated with every open file is a pointer to the next byte to be read or written
- in UNIX, even devices such as printers and terminals are treated as (special) files, e.g. /dev/lp /dev/tty; also selected kernel data (/proc)
 - this enables these to be read/written to as files - a powerful abstraction
- file types:
 - sequential files:
 - five basic operations: open, read, write, rewind, close
 - read/write from/to next item in file
 - rewind operation permits return to beginning of file
 - access analogous to that of a magnetic tape
 - random access files:
 - file position (pointer) can be explicitly set

