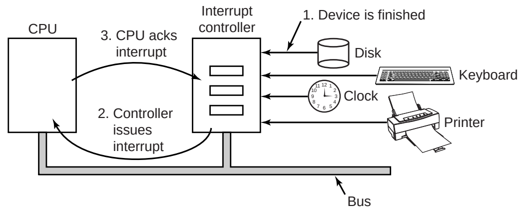
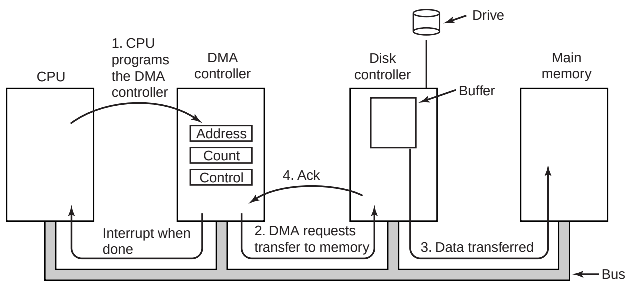
Real–time Systems
Input/Output

[2] **Interrupts Revisited (hardware level)**



- when an I/O device has finished the work given to it, it causes an interrupt by asserting a signal on a bus line that it has been assigned,
- signal detected by the interrupt controller chip. If no other interrupts pending, the interrupt controller processes the interrupt immediately. If another one in progress or there is a simultaneous request on a higher-priority interrupt request line, continues to assert until serviced by the CPU.
- the controller puts a number on the address lines and asserts a signal that interrupts the CPU,
- that number used as an index into a table called the **interrupt vector** to start a corresponding interrupt service procedure,
- the service procedure in certain moment acknowledges the interrupt by sending some value to some controller's port. That enables the controller to issue other interrupts.

[3] **Direct Memory Access (DMA)**



- DMA modules control data exchange between main mamory and external devices,
- usage of free bus time or (usually) delaying of the CPU for a one cycle from time to time to send a word by bus (**cycle stealing** and **burst mode**),

- only one interrupt after the whole transfer, avoidance of context switches,

[4] **Input/Output Handling**

Division of I/O devices:

- **block devices**, read/write of each block possible independently,
- **character devices**, deliver stream of characters without division into blocks, not addressable, without seek operation,
- **communication/network devices** sometimes distinguished as a separate group because of their specificity,
- some devices, like **timers**, do not fit in to this classification scheme,

[5] **Differences in Input/Output Handling**

Differencies in I/O handling:

- complexity of service,
- additional hardware support requirement,
- priorization of services,
- throughput unit,
- data representation,
- device response type,
- error handling,
- programming method.

[6] **Input/Output Programming Goals**

- device independence,
- uniform naming,
- error handling – the closer the hardware the better,
- transfer type – synchronous/ asynchronous,

- buffering.

[7] **Communication with External Devices (I)**

How processor communicates with control registers and how accesses external devices buffers. Two communication techniques:

1. **I/O ports**, with each control register some port with established number associated. Communication with special instructions:

   ```
   IN   REG, PORT
   OUT  PORT, REG
   ```

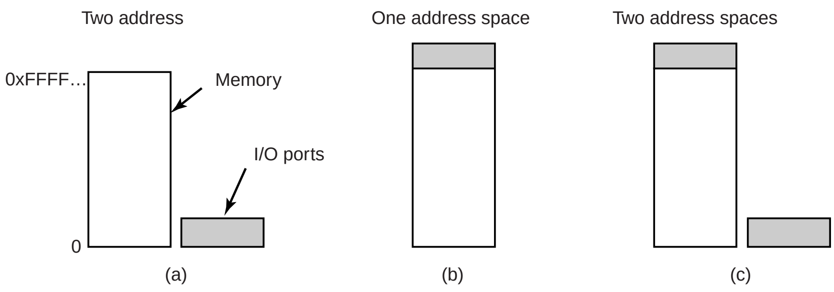
2. **memory-mapped I/O**

   - driver may be completely written in C, without any assembly code pieces, because access only via standard read/write calls,
   - no need for separate special protection mechanism,
   - faster testing of the contents of control registers,

   but

   - cache must be disabled for mapped region,
   - complicates the architecture with different type buses.

[8] **Communication with External Devices (II)**



a. separate I/O and memory space,

b. memory-mapped I/O,

c. hybrid solution, i.e. Pentium architecture: 640kB - 1MB addresses reserved for external devices still having I/O ports space 0 – 64K.

[9] **Principles of I/O Software**

Three ways of I/O communication/ programming:

1. programmed I/O (with polling, busy waiting behaviour),
2. interrupt-driven I/O,
3. I/O using DMA.

[10] **Programmed I/O**

```
copy_from_user(buffer, p, count);                 /* p is the kernel bufer */
for (i = 0; i < count; i++) {                     /* loop on every character */
      while (*printer_status_reg != READY) ;      /* loop until ready */
      *printer_data_register = p[i];              /* output one character */
}
return_to_user( );
```

An example of programmed I/O: steps in printing a string.

[11] **Interrupt-Driven I/O**

(a)

```
copy_from_user(buffer, p, count);
enable_interrupts( );
while (*printer_status_reg != READY) ;
*printer_data_register = p[0];
scheduler( );
```

(b)

```
if (count == 0) {
     unblock_user( );
} else {
     *printer_data_register = p[i];
     count = count – 1;
     i = i + 1;
}
acknowledge_interrupt( );
return_from_interrupt( );
```

An example of an interrupt-driven I/O: writing a string to the printer.

a. code executed when the print system call is made,

b. interrupt service procedure.

[12] **I/O Using DMA**

```
copy_from_user(buffer, p, count);        acknowledge_interrupt( );
set_up_DMA_controller( );                unblock_user( );
scheduler( );                            return_from_interrupt( );

            (a)                                  (b)
```

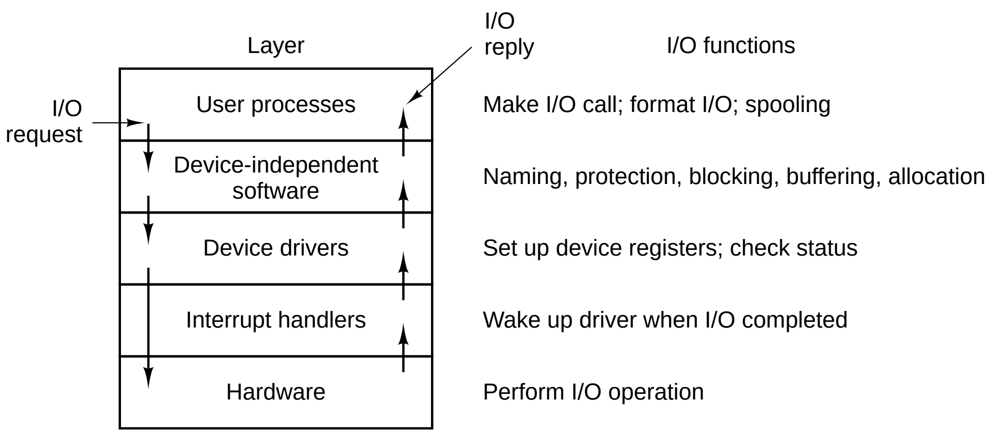
An example: of I/O using DMA: printing a string.

a. code executed when the print system call is made,

b. interrupt service procedure.

- advantage: reduction of number of interrupts from one per character to one per buffer printed,
- not always the best method – aspects of transfer scope size and relative speed of CPU and DMA controller.

[13] **I/O Software Layers**



[14] **Device-Independent I/O Software**

- uniform interfacing for device drivers,

- under Unix: naming devices with usage of **major** and **minor** numbers,
- protection against unauthorized access,
- providing a device-independent block size,
- buffering mechanisms (example: **double buffering**),
- management of accessability of devices,
- handling of allocation and releasing of devices,
- management of resource to user allocation,
- part of errors handling.

[15] **Disk Access Performance**

- **seek time** the time ti move the arm to the proper track,
- **rotational delay/latency** the time for the proper sector to rotate under the head,
- **access time** seek time + rotational latency,
- seek time decides about performance,
- important role of the cache memory (replacement algorithms LRU, LFU).

[16] **Disk Arm Scheduling Algorithms**

Because of requester:

RSS random scheduling,

FIFO the most fair one,

PRI with priorities,

LIFO *Last In First Out*, locality and resource usage maximization,

Because of service:

SSTF **shortest service time first** with the smallest move of arm,

SCAN **elevator algorithm**, the arm moves alternately in two directions (up and down) servicing all requests,

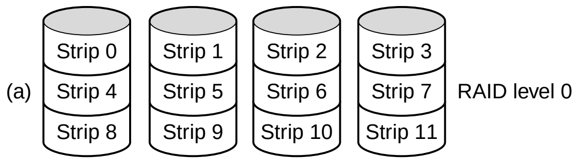
C-SCAN cyclic SCAN, servicing during the move only in one direction with a fast return to the start position.

[17] **Redundancy in Disk Service**

RAID Redundant Array of Independent Disks – (formerly: Inexpensive) the name and classification originating from Berkeley University.

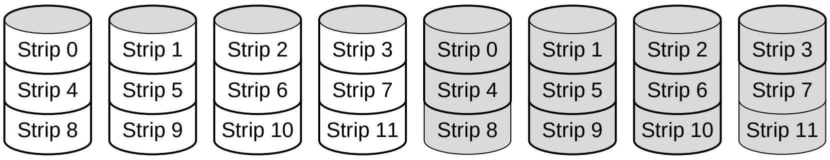
- a technique of creation of virtual disks (with logical volumes), with some features related to reliability, efficiency and serviceability, from a group of disks,
- data distributed over the matrix of disks,
- redundancy used to improve fault tolerance, especially tolerance to physical medium damage.
- RAID as opposed to (before) SLED (Single Large Expensive Disk) or (now) **JBOD** (Just a Bunch of Disks).

[18] **RAID Solutions (RAID 0)**



- no data redundancy,
- division into **concatenation** and **striping**,
- performane and flexibility improvement, low cost solution with lack of fault tolerance.

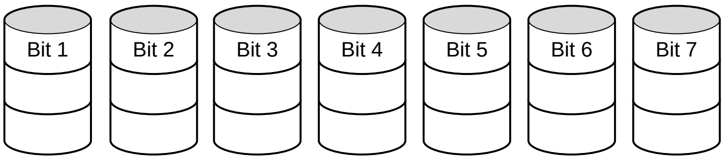
[19] **RAID Solutions (RAID 1)**

(b)


RAID level 1

- **mirroring**, full data redundancy,
- from the point of fault tolerance the best solution,
- expensive solution.

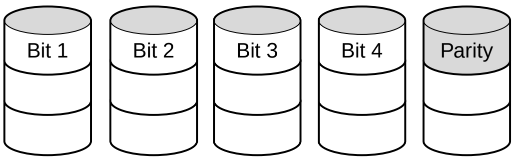
[20] **RAID Solutions (RAID 2)**

(c)


RAID level 2

- correction code computed from data bits,
- usage of correction-detection codes (Hamming's code),
- expensive solution which requires many disks.

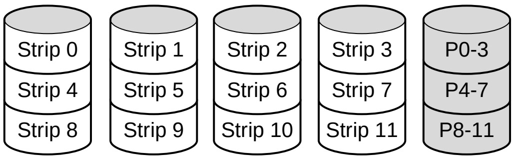
[21] **RAID Solutions (RAID 3)**

(d)


RAID level 3

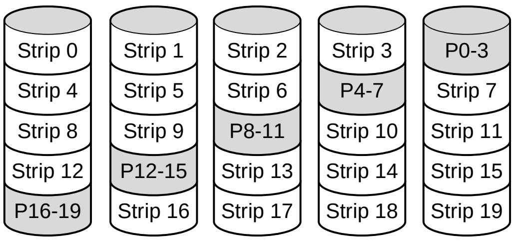
- analogoues to RAID 2, with parity bits instead of correction-detection codes,
- good throuhput in data size per time, poor performance in number of serviced requests in time.

[22] **RAID Solutions (RAID 4)**

(e)  RAID level 4

- RAID 4 – RAID 6, independent access to disks, independent requests may be serviced in parallel, better performance in number of serviced requests in time,
- striping with big stripes,
- parity computer on bit basis still requires read of a block.

[23] **RAID Solutions (RAID 5)**

(f)  RAID level 5

- striping with added parity bits,
- economical solution - redundancy costs exactly one disk,
- good read performance, noticable degradation of write performance,
- quality and efficiency of solution determined by the parameters tuning process.

[24] **RAID - Additional Aspects**

- RAID 6, as RAID 5 with two independent parity bits (stripes),
- RAID 10 = 1 + 0
- hardware RAID and software RAID,

- in common use RAID: 0, 1, 5, 1+0, 0+1,
- typical server configuration:
  - RAID 1 for small critical data (i.e. disks with operating system),
  - RAID 5 for huge databases (i.e. disks in some external matrix).