Figure 1: Main Components of an Operating System

**CIS2380 (second term) - Week 7**

**Input/Output**
The role of the OS in computer I/O is to manage and control I/O operations and I/O devices. Figure 1 shows a modular operating system which includes I/O manager - as well as other components we have already seen.

Standardisation means we can incorporate improved versions of devices into old computers. However there are also many kinds of new device - and these present a challenge. To encapsulate the details of these new devices the kernel of an operating system is structured to use device-driver modules. These device drivers present a uniform device access interface to the I/O subsystem. Devices can be (roughly) categorised:

- Storage (discs, tapes):
  - transmission (network cards, modems)
  - human-interface (screen, keyboard, mouse).

Other devices might include the steering of a space shuttle.

Devices can also be categorised by whether they are Block Oriented, Stream oriented or Network. These diverse forms of INPUT/OUTPUT have much in common - for example every device communicates with a computer system by sending signals over a cable or through the air.

**Definitions**
The device communicates with the machine via a **port**.

If one or more devices use a common set of wires the connection is a **bus**:

- a set of wires and a rigidly defined protocol that specifies a set of messages that can be sent on the wires.

A **controller** is a collection of electronics that can operate a port, a bus or a device. A controller may be in charge of more than one device - for example a **Small Computer Systems Interface (SMIS)** which may be implemented as a separate circuit board. Some devices have their own controller - for example a disc drive.
A controller has registers which are commonly separate from main memory. (See Figure 2 (b) for an example.) These registers are dual ported - there are two paths into and out (unlike 'usual' memory). One side is connected to the data bus of the computer and the other to the device (See Figure 2 (a).) The data register holds data and the status register provides information about the data (e.g. bit 7 set might mean there is a byte in the data register bit 7 clear would mean a byte has just been read.)

There are several methods of data transfer: **Polling, Interrupt Driven** and **Direct Memory Access** and there are also several ways of dealing with the transferred data (see later).

**Polling:** The CPU checks the status register for each device to see if it needed service. This is simple but wasteful.

**Interrupt Driven** An interrupt line connects each device with the CPU. (This is the dotted line in Figure 2(b).) When a device has data - it put an electric pulse on the interrupt line. The CPU *interrupts* its current process - and commences polling which is now in order to discover which device has data. After the data has been transferred and the bit cleared, the CPU resumes the process.

**Interrupts (Revisited):**
1. The CPU completes the current instruction
2. Saves part of the program State - eg PC
3. Jumps to address of interrupt handler (or one of a set of interrupt handlers determining priority)
4. Handles reason for interrupt (transfers data)
5. Returns CPU to execution state prior to interrupt.

See Figure 3.

**Direct Memory Access:** The two previous methods of data transfer are wasteful of CPU time which only transfers one byte at a time. A less wasteful method is *Direct Memory Access*. (See Figures 4 and 5). If a file is
Figure 3: How an interrupt happens. The connections between the devices and the interrupt controller actually use interrupt lines on the bus rather than dedicated wires.

Figure 4: Direct Memory Access from a Disc
to be transferred it would involve interrupts for every byte - an intolerable situation. The DMA in Figure 4 is different from the device controller in Figure 2(b) for there is no data register. Instead the status register is extended. There are two extra registers - address and length and some extra processing ability (not shown). Consider the example of an 8KB file where the block of data to be transferred is in main memory not a register. Transfer is as follows (Figure 5):

1. The CPU sets up the bits in the status register - that data is ready to be transferred and also
2. Puts its memory address in the address register and its length in the length register.
3. The DMA processor moves the data from memory, sharing the address and data buses with the CPU and interleaving with the CPU in its other tasks.
4. When the CPU is not using the bus or memory, the DMA uses it to transfer data.
5. When the required number of bytes have been transferred, the controller interrupts the CPU to let it know.

**Handshaking:** Interleaving is done via ‘handshaking’ between the DMA controller and the device controller - there are a pair of wires called DMA-request and DMA-acknowledge. The device controller places a signal on the DMA-request wire when a word of data is available for transfer. The DMA controller seize the memory bus, places the desired address on the memory address wires and places a signal on the DMA-acknowledge wire. When the device controller receives the DMA-acknowledge signal, it transfers the word of data from memory and removes the DMA-request signal. (Obviously transfer can be TO memory also.)

Figure 5 provides a diagrammatic representation of the operation - showing transfer TO memory. The ‘control’ register(s) specify the direction of the transfer (read or write) etc.

**Memory Mapped I/O** If a page fault occurs then disc access is required. When data is read in from a disc one way is to input the file (through DMA or other means) and transfer the content to a buffer. However if the data is needed because of a page fault, another transfer is required to transfer the contents of this buffer to a page frame in physical memory. To overcome this, the file can be directly transferred to particular page(s) in physical memory. This association of a file with page(s) in physical memory is known as memory mapping. This also works with segmented memory management. However in every case care has to be taken if a file is accessed by two processes, that they are accessing the correct version - if one process modifies a page, that modification will not be reflected on the disc version until the page is evicted.

**Global File Table** Information about open and closed files (and associated devices) in a process is kept in tables. An important table is the *global*
file table where each open file and device has one entry. (This is to maintain consistency where devices, files etc are used by more than one process.)

The Device Switch Each device has an associated driver (program) and to coordinate calls to these drivers, a mechanism known as a device switch is used. This is an array which is filled in when the system is booted. Each entry in the array is a collection of pointers to functions appropriate to that device. (read, write, etc.) When a device is opened there is a pointer in the global file table to the array entry in the device switch.

Input/Output Request Block Another important data structure is an input/output request block or IORB which contains:
• flags to indicate read or write, synchronous or asynchronous.
• methods of indicating completion eg semaphore name.
• location of data in memory
• number of bytes in memory
• device number (etc)
In some operating systems (eg Unix) processes requiring I/O initiate device driver code passing the IORB as a parameter. While waiting for the device the CPU is given to another process. The device interrupts when ready. Other operating systems assign each device driver a process which sleeps when not required. It is woken up via interprocess communication - such as a message queue with the IORB in the queue.

Operations on Devices The functions particular to each driver require implementing and the following is a general description:
Read it checks the device is powered up and online then resets device to known state. Eg printing starts top left hand corner.
Close Resets device and puts it offline.
Read implements synchronous read requests. All information is built into an IORB (one of the read parameters).
Write: Similar to read using the IORB.

Next Week: Viruses and Security