History

The next issue is memory protection: this must stop a program from writing and/or reading the memory used by another program or by the OS

History

The next issue is memory protection: this must stop a program from writing and/or reading the memory used by another program or by the OS

The OS must be allowed to read and write any part of memory

History

The next issue is memory protection: this must stop a program from writing and/or reading the memory used by another program or by the OS

The OS must be allowed to read and write any part of memory

Again, there must be hardware support to do this to make it fast

History

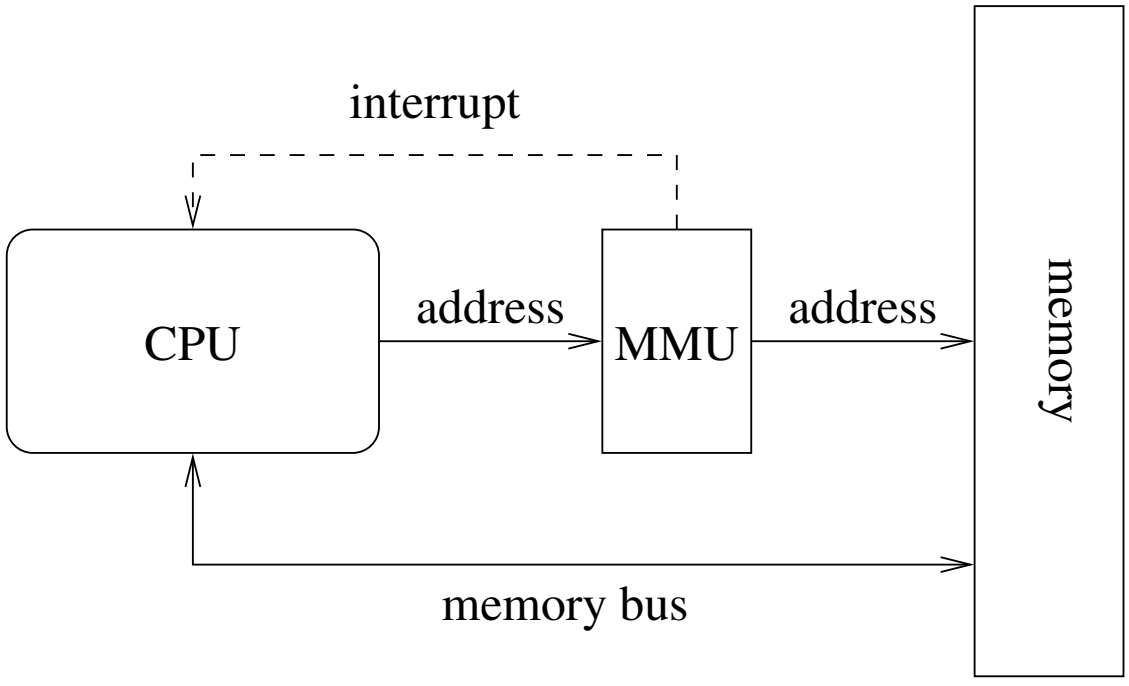
The next issue is memory protection: this must stop a program from writing and/or reading the memory used by another program or by the OS

The OS must be allowed to read and write any part of memory

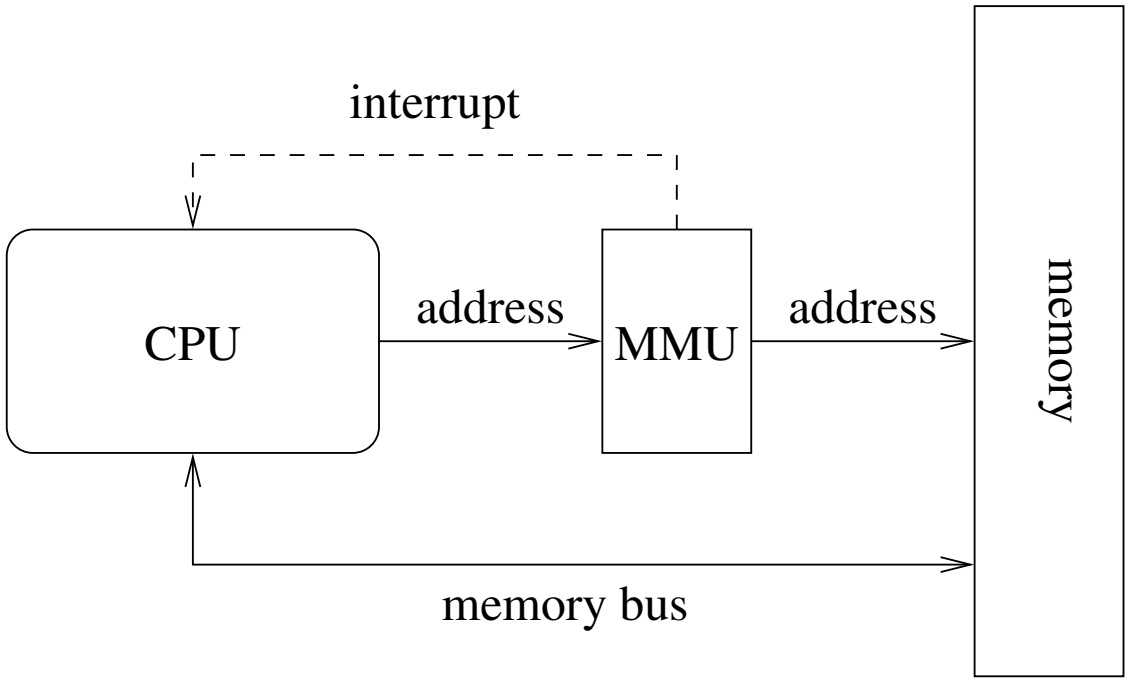
Again, there must be hardware support to do this to make it fast

There is a table of flags in a special piece of hardware: the *memory management unit* (MMU). These flags say whether the *currently running* (user mode) program can read or write a given area of memory

History

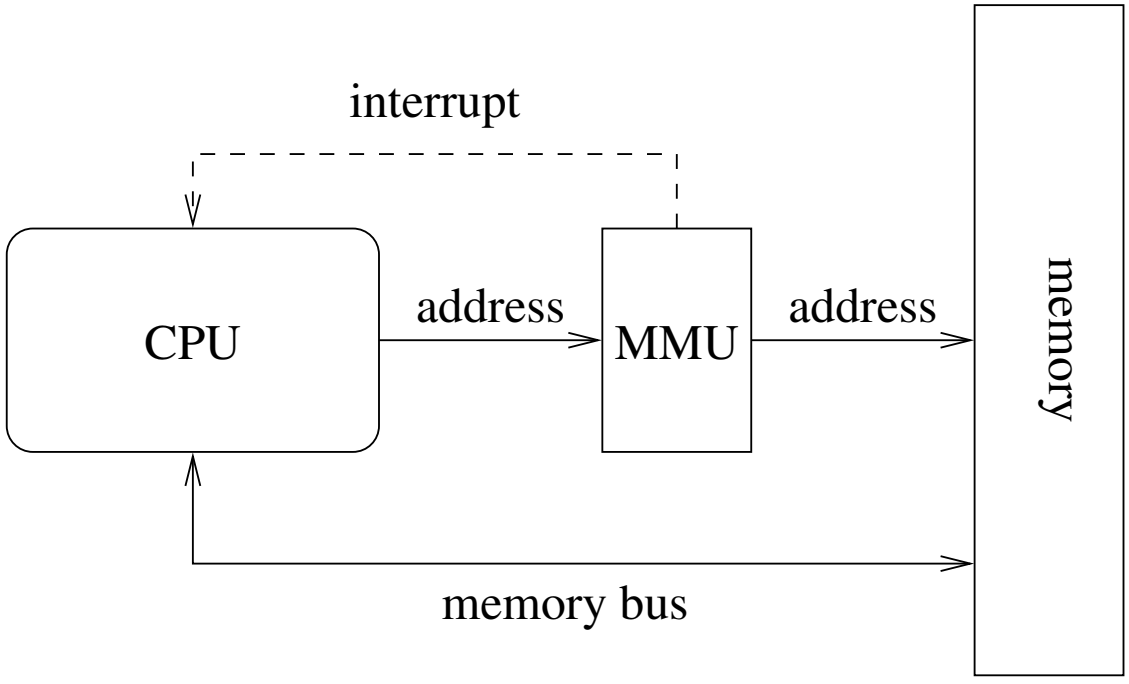


History



One bit to say if an area is readable; another to say if it is writable

History



One bit to say if an area is readable; another to say if it is writable

It is often useful to separate ability to read from ability to write

History

Setting these flags in the MMU is a privileged operation, of course

History

Setting these flags in the MMU is a privileged operation, of course

And if an unprivileged program tries to read or write to an area of memory for which it does not have the required permission (say some other program's or the OS's memory) the MMU raises an interrupt and the OS takes control again

History

Setting these flags in the MMU is a privileged operation, of course

And if an unprivileged program tries to read or write to an area of memory for which it does not have the required permission (say some other program's or the OS's memory) the MMU raises an interrupt and the OS takes control again

It would not be feasible to have control like this on a byte-by-byte level, so memory is divided into blocks called *pages*

History

Setting these flags in the MMU is a privileged operation, of course

And if an unprivileged program tries to read or write to an area of memory for which it does not have the required permission (say some other program's or the OS's memory) the MMU raises an interrupt and the OS takes control again

It would not be feasible to have control like this on a byte-by-byte level, so memory is divided into blocks called *pages*

A page is just a contiguous area of memory: 4096 bytes is popular on modern machines, though current hardware can support 4MB pages

History

A page is marked as read/writable as a whole: this makes this technique practical

History

A page is marked as read/writable as a whole: this makes this technique practical

Exercise. How many flags (bits) are needed to cover 2GB? How many bytes of flags does that correspond to?

History

A page is marked as read/writable as a whole: this makes this technique practical

Exercise. How many flags (bits) are needed to cover 2GB? How many bytes of flags does that correspond to?

Note that these flags are part of a program's state that must be saved and restored when that program is re-scheduled

History

A page is marked as read/writable as a whole: this makes this technique practical

Exercise. How many flags (bits) are needed to cover 2GB? How many bytes of flags does that correspond to?

Note that these flags are part of a program's state that must be saved and restored when that program is re-scheduled

There is usually also an *executable* flag: can you execute code from this memory address?

History

Every read or write to memory is checked by the MMU before it is allowed: this means the hardware that does this check has to be very fast

History

Every read or write to memory is checked by the MMU before it is allowed: this means the hardware that does this check has to be very fast

We shall not be going into this in depth here, because in modern machines this is enhanced by the notion of *virtual memory*

History

Every read or write to memory is checked by the MMU before it is allowed: this means the hardware that does this check has to be very fast

We shall not be going into this in depth here, because in modern machines this is enhanced by the notion of *virtual memory*

This we shall cover later, but it builds on the ideas above and provides a much more flexible method of protection

History

Thus we can see some of the requirements of an operating system

History

Thus we can see some of the requirements of an operating system

- Resource management

History

Thus we can see some of the requirements of an operating system

- Resource management
  - in particular program scheduling (CPU time)

History

Thus we can see some of the requirements of an operating system

- Resource management
  - in particular program scheduling (CPU time)
  - also disk, network, . . .

History

Thus we can see some of the requirements of an operating system

- Resource management
  - in particular program scheduling (CPU time)
  - also disk, network, . . .
- Protection

History

Thus we can see some of the requirements of an operating system

- Resource management
  - in particular program scheduling (CPU time)
  - also disk, network, . . .
- Protection
  - in particular memory

History

Thus we can see some of the requirements of an operating system

- Resource management
  - in particular program scheduling (CPU time)
  - also disk, network, . . .
- Protection
  - in particular memory
  - also files, network data, . . .

History

Thus we can see some of the requirements of an operating system

- Resource management
  - in particular program scheduling (CPU time)
  - also disk, network, . . .
- Protection
  - in particular memory
  - also files, network data, . . .
- Efficiency

History

Thus we can see some of the requirements of an operating system

- Resource management
  - in particular program scheduling (CPU time)
  - also disk, network, . . .
- Protection
  - in particular memory
  - also files, network data, . . .
- Efficiency
  - in particular with regards to time

History

Thus we can see some of the requirements of an operating system

- Resource management
  - in particular program scheduling (CPU time)
  - also disk, network, . . .
- Protection
  - in particular memory
  - also files, network data, . . .
- Efficiency
  - in particular with regards to time
  - also size, energy, . . .

History

By making privileged operations *only* available to the OS, the OS can enforce policy on access and ensure fair distribution of shared resources

History

In current large OSs we have:

History

In current large OSs we have:

- Windows. Preemptive multitasking from Windows NT (1996) onwards. Previously (Windows 95 etc.) was little more than a monitor with a pretty interface on top

History

In current large OSs we have:

- Windows. Preemptive multitasking from Windows NT (1996) onwards. Previously (Windows 95 etc.) was little more than a monitor with a pretty interface on top
- Linux. A Unix re-implementation. Preemptive multitasking from inception (1991). (Recall that Unix had preemption from early 1970s)

History

In current large OSs we have:

- Windows. Preemptive multitasking from Windows NT (1996) onwards. Previously (Windows 95 etc.) was little more than a monitor with a pretty interface on top
- Linux. A Unix re-implementation. Preemptive multitasking from inception (1991). (Recall that Unix had preemption from early 1970s)
- MacOS. MacOS X is a Unix derivative (BSD), from 1999 onwards. Earlier systems (MacOS 9 and earlier) were completely different, with no preemption, only cooperative

History

- Solaris. A Unix derivative (System V). Preemptive multitasking from inception (1992), an extensive rewrite of the earlier SunOS (1983), another Unix variant (BSD)

History

- Solaris. A Unix derivative (System V). Preemptive multitasking from inception (1992), an extensive rewrite of the earlier SunOS (1983), another Unix variant (BSD)
- OS/2. Initially from Microsoft and IBM (1997), then just IBM as Microsoft went off to do its own thing. Intended to be the followup to DOS. Multitasking when the hardware could support it: OS/2 2.0 (1992) could run multiple copies of DOS/Windows simultaneously. Previously used a lot in bank ATMs (until IBM ended support in 2006). OS/2 3.0 became Windows NT

History

And thousands of others: but the major players in the PC market are either derived from Windows NT, or from Unix

History

And thousands of others: but the major players in the PC market are either derived from Windows NT, or from Unix

In contrast, in the embedded market are things are much more mixed, with both purpose-built OSs and slimmed-down derivatives of the general-purpose OSs all having major representation

History

With Windows, rebooting is the first thing an admin tries to fix a problem; with Unix, it's the last

Anon.

□