Processes

Process creation is quite involved

Processes

Process creation is quite involved

- Allocate and create PCB structure

Processes

Process creation is quite involved

- Allocate and create PCB structure
- Find a free PID

Processes

Process creation is quite involved

- Allocate and create PCB structure
- Find a free PID
- Determine and allocate the necessary resources (in particular memory)

Processes

Process creation is quite involved

- Allocate and create PCB structure
- Find a free PID
- Determine and allocate the necessary resources (in particular memory)
- Determine the initial priority of the process

Processes

Process creation is quite involved

- Allocate and create PCB structure
- Find a free PID
- Determine and allocate the necessary resources (in particular memory)
- Determine the initial priority of the process
- Insert PCB into the relevant list

Processes

Process creation is quite involved

- Allocate and create PCB structure
- Find a free PID
- Determine and allocate the necessary resources (in particular memory)
- Determine the initial priority of the process
- Insert PCB into the relevant list

This is what happens in the New state; it can now move to Ready

Processes

Process creation is quite involved

- Allocate and create PCB structure
- Find a free PID
- Determine and allocate the necessary resources (in particular memory)
- Determine the initial priority of the process
- Insert PCB into the relevant list

This is what happens in the New state; it can now move to Ready

Again, the process might not start running immediately, as there could be some higher priority process that must run first

Processes

Most processes are created (*forked/spawned*) by other processes: of course, only the OS can actually create processes

Processes

Most processes are created (*forked/spawned*) by other processes: of course, only the OS can actually create processes

A user process that wants a new process will ask the OS to create one (using a syscall)

Processes

Most processes are created (*forked/spawned*) by other processes: of course, only the OS can actually create processes

A user process that wants a new process will ask the OS to create one (using a syscall)

Processes use resources like memory and CPU, so the OS must be involved

Processes

Most processes are created (*forked/spawned*) by other processes: of course, only the OS can actually create processes

A user process that wants a new process will ask the OS to create one (using a syscall)

Processes use resources like memory and CPU, so the OS must be involved

- A process decides it wants to start another process. E.g., a GUI process as a response to a user clicking on an icon

Processes

Most processes are created (*forked/spawned*) by other processes: of course, only the OS can actually create processes

A user process that wants a new process will ask the OS to create one (using a syscall)

Processes use resources like memory and CPU, so the OS must be involved

- A process decides it wants to start another process. E.g., a GUI process as a response to a user clicking on an icon
- It calls the OS kernel (syscall), telling it what process it want to start (e.g., “start the browser program”)

Processes

Most processes are created (*forked/spawned*) by other processes: of course, only the OS can actually create processes

A user process that wants a new process will ask the OS to create one (using a syscall)

Processes use resources like memory and CPU, so the OS must be involved

- A process decides it wants to start another process. E.g., a GUI process as a response to a user clicking on an icon
- It calls the OS kernel (syscall), telling it what process it want to start (e.g., “start the browser program”)
- The OS can now create a new process according to the specifications given

Processes

Most processes are created (*forked/spawned*) by other processes: of course, only the OS can actually create processes

A user process that wants a new process will ask the OS to create one (using a syscall)

Processes use resources like memory and CPU, so the OS must be involved

- A process decides it wants to start another process. E.g., a GUI process as a response to a user clicking on an icon
- It calls the OS kernel (syscall), telling it what process it want to start (e.g., “start the browser program”)
- The OS can now create a new process according to the specifications given
- The new process can now be scheduled

Processes

The original calling process will generally be the parent of the new process

Processes

The original calling process will generally be the parent of the new process

Of course, the OS can choose not to create the new process if some policy says not to, or there is not enough memory, or some other reason

Processes

The original calling process will generally be the parent of the new process

Of course, the OS can choose not to create the new process if some policy says not to, or there is not enough memory, or some other reason

In that case, the original process usually gets a message back from the OS (via the value returned from the syscall) explaining the problem

Processes

The question arises: if processes are created by other processes, how do we get started?

Processes

The question arises: if processes are created by other processes, how do we get started?

This is the *bootstrapping problem*

Processes

The question arises: if processes are created by other processes, how do we get started?

This is the *bootstrapping problem*

In Unix, there is an ancestor process, sometimes called *init* or *systemd*, PID 1, that gets created at switch-on time and it serves to create all other processes

Processes

The question arises: if processes are created by other processes, how do we get started?

This is the *bootstrapping problem*

In Unix, there is an ancestor process, sometimes called *init* or *systemd*, PID 1, that gets created at switch-on time and it serves to create all other processes

Bootstrapping is complicated as it has to determine the hardware and how it is connected and initialise it, set up all the appropriate datastructures the OS needs, start lots of service processes running, all before it can begin to look at what the user wants

Processes

The question arises: if processes are created by other processes, how do we get started?

This is the *bootstrapping problem*

In Unix, there is an ancestor process, sometimes called *init* or *systemd*, PID 1, that gets created at switch-on time and it serves to create all other processes

Bootstrapping is complicated as it has to determine the hardware and how it is connected and initialise it, set up all the appropriate datastructures the OS needs, start lots of service processes running, all before it can begin to look at what the user wants

This is quite often simply called *booting*

Processes

How does process 1 get started?

Processes

How does process 1 get started?

To get going, the system needs a small chunk of *non-volatile* memory, i.e., memory that doesn't lose its content when power is removed

Processes

How does process 1 get started?

To get going, the system needs a small chunk of *non-volatile* memory, i.e., memory that doesn't lose its content when power is removed

The details of booting are very messy, but in essence:

Processes

How does process 1 get started?

To get going, the system needs a small chunk of *non-volatile* memory, i.e., memory that doesn't lose its content when power is removed

The details of booting are very messy, but in essence:

- When the machine is switched on the processor jumps to a specific location in the non-volatile memory (0xFFFF0000 on PCs)

Processes

How does process 1 get started?

To get going, the system needs a small chunk of *non-volatile* memory, i.e., memory that doesn't lose its content when power is removed

The details of booting are very messy, but in essence:

- When the machine is switched on the processor jumps to a specific location in the non-volatile memory (0xFFFF0000 on PCs)
- At this location is a small program (the *boot loader*, or *bootstrap loader*) that is just big enough to load and run a larger program (from bigger non-volatile memory, or perhaps disk)

Processes

- This might be repeated until we have a big enough program running that is capable of reading from, say, the start of the hard drive

Processes

- This might be repeated until we have a big enough program running that is capable of reading from, say, the start of the hard drive
- This is often itself another bootstrap program (NTLDR, and GRUB are common) that might give the user a choice of operating systems to load, but usually just goes ahead and loads one from disk (or network, or whatever)

Processes

- This might be repeated until we have a big enough program running that is capable of reading from, say, the start of the hard drive
- This is often itself another bootstrap program (NTLDR, and GRUB are common) that might give the user a choice of operating systems to load, but usually just goes ahead and loads one from disk (or network, or whatever)
- This may require the bootloader to have some understanding of how data is laid out on the disk, which itself is non-trivial (see later)

Processes

- This might be repeated until we have a big enough program running that is capable of reading from, say, the start of the hard drive
- This is often itself another bootstrap program (NTLDR, and GRUB are common) that might give the user a choice of operating systems to load, but usually just goes ahead and loads one from disk (or network, or whatever)
- This may require the bootloader to have some understanding of how data is laid out on the disk, which itself is non-trivial (see later)
- Eventually, enough of the operating system kernel is loaded that it get itself going properly, e.g., start init

Processes

Exercise. Some machines do not have disks (or other persistent storage), for example *thin clients*. Read about how these can boot

□