# Processes

We now look at the programs we want to run

# Processes

We now look at the programs we want to run

The word *process* is used to describe

- the executable code *and*
- its data *and*
- the associated information the OS needs to run it

# Processes

We now look at the programs we want to run

The word *process* is used to describe

- the executable code *and*
- its data *and*
- the associated information the OS needs to run it

Note this is different from *processor* and *program*

# Processes

We now look at the programs we want to run

The word *process* is used to describe

- the executable code *and*
- its data *and*
- the associated information the OS needs to run it

Note this is different from *processor* and *program*

Other words: task, job

# Processes

A single program might possibly use more than one process

# Processes

A single program might possibly use more than one process

For example, one process to compute a picture and another to display it: this is called *structure by process*

# Processes

A single program might possibly use more than one process

For example, one process to compute a picture and another to display it: this is called *structure by process*

Though this is perhaps the exception, these days. Quite often a program uses just one process

# Processes

A single program might possibly use more than one process

For example, one process to compute a picture and another to display it: this is called *structure by process*

Though this is perhaps the exception, these days. Quite often a program uses just one process

And structuring can be done by using multiple *threads of execution*, often running in parallel

# Processes

A single program might possibly use more than one process

For example, one process to compute a picture and another to display it: this is called *structure by process*

Though this is perhaps the exception, these days. Quite often a program uses just one process

And structuring can be done by using multiple *threads of execution*, often running in parallel

But it is coming back in Web browsers using one process per tab to provide security isolation between tabs

# Processes

A single program might possibly use more than one process

For example, one process to compute a picture and another to display it: this is called *structure by process*

Though this is perhaps the exception, these days. Quite often a program uses just one process

And structuring can be done by using multiple *threads of execution*, often running in parallel

But it is coming back in Web browsers using one process per tab to provide security isolation between tabs

Note to think about later: Web browsers use OS process protection and isolation mechanisms to provide tab protection and isolation

# Processes

An OS needs to keep a lots of information about a process, including

# Processes

An OS needs to keep a lots of information about a process, including

- where in memory its code is

# Processes

An OS needs to keep a lots of information about a process, including

- where in memory its code is
- where in memory its data is

# Processes

An OS needs to keep a lots of information about a process, including

- where in memory its code is
- where in memory its data is
- what permissions it has on those parts of memory (MMU flags)

# Processes

An OS needs to keep a lots of information about a process, including

- where in memory its code is
- where in memory its data is
- what permissions it has on those parts of memory (MMU flags)
- how much time it is allocated

# Processes

An OS needs to keep a lots of information about a process, including

- where in memory its code is
- where in memory its data is
- what permissions it has on those parts of memory (MMU flags)
- how much time it is allocated
- how much time it has used

# Processes

An OS needs to keep a lots of information about a process, including

- where in memory its code is
- where in memory its data is
- what permissions it has on those parts of memory (MMU flags)
- how much time it is allocated
- how much time it has used
- similarly for other shared resources, e.g., the amount of I/O or networking done

# Processes

An OS needs to keep a lots of information about a process, including

- where in memory its code is
- where in memory its data is
- what permissions it has on those parts of memory (MMU flags)
- how much time it is allocated
- how much time it has used
- similarly for other shared resources, e.g., the amount of I/O or networking done
- the cpu's PC and registers

# Processes

An OS needs to keep a lots of information about a process, including

- where in memory its code is
- where in memory its data is
- what permissions it has on those parts of memory (MMU flags)
- how much time it is allocated
- how much time it has used
- similarly for other shared resources, e.g., the amount of I/O or networking done
- the cpu's PC and registers
- flags from the MMU

# Processes

An OS needs to keep a lots of information about a process, including

- where in memory its code is
- where in memory its data is
- what permissions it has on those parts of memory (MMU flags)
- how much time it is allocated
- how much time it has used
- similarly for other shared resources, e.g., the amount of I/O or networking done
- the cpu's PC and registers
- flags from the MMU
- and lots more as we shall see later

# Processes

An OS needs to keep a lots of information about a process, including

- where in memory its code is
- where in memory its data is
- what permissions it has on those parts of memory (MMU flags)
- how much time it is allocated
- how much time it has used
- similarly for other shared resources, e.g., the amount of I/O or networking done
- the cpu's PC and registers
- flags from the MMU
- and lots more as we shall see later

It uses this information to schedule and protect the process

# Processes

A process can be in one of several *states*. In a simplified model, the five main states are

# Processes

A process can be in one of several *states*. In a simplified model, the five main states are

1. New. A process that has just been created

# Processes

A process can be in one of several *states*. In a simplified model, the five main states are

1. New. A process that has just been created
2. Running. It is currently executing on the CPU

# Processes

A process can be in one of several *states*. In a simplified model, the five main states are

1. New. A process that has just been created
2. Running. It is currently executing on the CPU
3. Ready. It is ready to run, but some other process (or the OS) is currently using the CPU

# Processes

A process can be in one of several *states*. In a simplified model, the five main states are

1. New. A process that has just been created
2. Running. It is currently executing on the CPU
3. Ready. It is ready to run, but some other process (or the OS) is currently using the CPU
4. Blocked. Waiting for some event or resource to become available. E.g., waiting for a block of data to arrive from the disk

# Processes

A process can be in one of several *states*. In a simplified model, the five main states are

1. New. A process that has just been created
2. Running. It is currently executing on the CPU
3. Ready. It is ready to run, but some other process (or the OS) is currently using the CPU
4. Blocked. Waiting for some event or resource to become available. E.g., waiting for a block of data to arrive from the disk
5. Exit. A process that has finished

# Processes

A process can be in one of several *states*. In a simplified model, the five main states are

1. New. A process that has just been created
2. Running. It is currently executing on the CPU
3. Ready. It is ready to run, but some other process (or the OS) is currently using the CPU
4. Blocked. Waiting for some event or resource to become available. E.g., waiting for a block of data to arrive from the disk
5. Exit. A process that has finished

Real OSs will have more states than this, but these are the important ones

# Processes

We shall assume, for simplicity, that we have just one processor

# Processes

We shall assume, for simplicity, that we have just one processor

The OS will have lists of processes in each state, so the scheduling decision is making the choice of which process to move between which states

# Processes

We shall assume, for simplicity, that we have just one processor

The OS will have lists of processes in each state, so the scheduling decision is making the choice of which process to move between which states

Again, in real OSs, these will not be simple lists. They might be arranged in priority order, or might be some more sophisticated datastructure: e.g., a pair of lists, one for real-time processes and the other for non-real-time; or a tree

# Processes

Example: in Unixes, processes are arranged in *trees*

```
systemd-+-ModemManager---2*[{ModemManager}]
        |-NetworkManager---2*[{NetworkManager}]
        |-Thunar---3*[{Thunar}]
        |-accounts-daemon---2*[{accounts-daemon}]
        |-agetty
        |-atd
        |-auditd---{auditd}
        |-avahi-daemon
        |-chrome-+-2*[cat]
        |        |-chrome-+-chrome-+-chrome---12*[{chrome}]
        |        |        |        |-chrome---19*[{chrome}]
        |        |        |        |-3*[chrome---11*[{chrome}]]
        |        |        |        |-chrome---15*[{chrome}]
        |        |        |        |-chrome---17*[{chrome}]
        |        |        |        |-chrome---16*[{chrome}]
        |        |        |        |-chrome---10*[{chrome}]
        |        |        |        `-chrome---23*[{chrome}]
        |        |        `-nacl_helper
        |        |-chrome-+-chrome
        |        |        `-7*[{chrome}]
```

# Processes

This allows control of a whole bunch of processes as a group

# Processes

This allows control of a whole bunch of processes as a group

A group within the tree has a *session leader*

# Processes

This allows control of a whole bunch of processes as a group

A group within the tree has a *session leader*

For example, killing the session leader would typically kill all the processes in the group

# Processes

This allows control of a whole bunch of processes as a group

A group within the tree has a *session leader*

For example, killing the session leader would typically kill all the processes in the group

In the example above, exiting the chrome session leader would kill it and all its subprocesses