Process Protection

We now need to re-visit the idea of process protection

Process Protection

We now need to re-visit the idea of process protection

Recall by forcing access to resources via the kernel we can ensure that one user cannot interfere with the processes of another user

Process Protection

We now need to re-visit the idea of process protection

Recall by forcing access to resources via the kernel we can ensure that one user cannot interfere with the processes of another user

Thus a process must include the notion of *user*

Process Protection

We now need to re-visit the idea of process protection

Recall by forcing access to resources via the kernel we can ensure that one user cannot interfere with the processes of another user

Thus a process must include the notion of *user*

This is usually encoded as a simple integer, the *userid*

Process Protection

We now need to re-visit the idea of process protection

Recall by forcing access to resources via the kernel we can ensure that one user cannot interfere with the processes of another user

Thus a process must include the notion of *user*

This is usually encoded as a simple integer, the *userid*

Each user has their own unique userid and the OS uses this to determine whether one process can access files, other processes and so on

Process Protection

We now need to re-visit the idea of process protection

Recall by forcing access to resources via the kernel we can ensure that one user cannot interfere with the processes of another user

Thus a process must include the notion of *user*

This is usually encoded as a simple integer, the *userid*

Each user has their own unique userid and the OS uses this to determine whether one process can access files, other processes and so on

The userid also plays a role in Fair Share scheduling, of course

Process Protection

Userids are used everywhere

Process Protection

Userids are used everywhere

- Memory: a chunk of memory has a userid associated. This tells the kernel which processes are allowed to access it

Process Protection

Userids are used everywhere

- Memory: a chunk of memory has a userid associated. This tells the kernel which processes are allowed to access it
- Files: each file has a userid associated. This tells the kernel which processes are allowed to access it

Process Protection

Userids are used everywhere

- Memory: a chunk of memory has a userid associated. This tells the kernel which processes are allowed to access it
- Files: each file has a userid associated. This tells the kernel which processes are allowed to access it
- Similarly for other resources

Process Protection

Userids are used everywhere

- Memory: a chunk of memory has a userid associated. This tells the kernel which processes are allowed to access it
- Files: each file has a userid associated. This tells the kernel which processes are allowed to access it
- Similarly for other resources

We shall see more of this when we get to memory and files

Process Protection

Userids are used everywhere

- Memory: a chunk of memory has a userid associated. This tells the kernel which processes are allowed to access it
- Files: each file has a userid associated. This tells the kernel which processes are allowed to access it
- Similarly for other resources

We shall see more of this when we get to memory and files

Exercise. Find out the userid allocated to you on the Uni's `linux.bath.ac.uk` machine

Process Protection

A new process (usually) inherits the userid of its parent process

Process Protection

A new process (usually) inherits the userid of its parent process

Of course, this lead to another bootstrapping problem: how can a user get a process going in the first place?

Process Protection

A new process (usually) inherits the userid of its parent process

Of course, this lead to another bootstrapping problem: how can a user get a process going in the first place?

If there are no processes running with my userid, how can I ever get a process to be created?

Process Protection

A new process (usually) inherits the userid of its parent process

Of course, this lead to another bootstrapping problem: how can a user get a process going in the first place?

If there are no processes running with my userid, how can I ever get a process to be created?

So there is a distinguished user, variously called the *superuser* or *root* or *administrator*

Process Protection

A new process (usually) inherits the userid of its parent process

Of course, this lead to another bootstrapping problem: how can a user get a process going in the first place?

If there are no processes running with my userid, how can I ever get a process to be created?

So there is a distinguished user, variously called the *superuser* or *root* or *administrator*

This is a **normal user**, but the OS allows it full access to other users' files, processes, etc.

Process Protection

A new process (usually) inherits the userid of its parent process

Of course, this lead to another bootstrapping problem: how can a user get a process going in the first place?

If there are no processes running with my userid, how can I ever get a process to be created?

So there is a distinguished user, variously called the *superuser* or *root* or *administrator*

This is a **normal user**, but the OS allows it full access to other users' files, processes, etc.

In particular, root can suspend or kill any user's processes

Process Protection

Don’t confuse the root user with kernel mode

Process Protection

Don’t confuse the root user with kernel mode

Root’s processes run in user mode, just like other users’ processes

Process Protection

Don’t confuse the root user with kernel mode

Root’s processes run in user mode, just like other users’ processes

Hardware access is still mediated by the OS, but the inter-user protections are not enforced by the OS

Process Protection

Don’t confuse the root user with kernel mode

Root’s processes run in user mode, just like other users’ processes

Hardware access is still mediated by the OS, but the inter-user protections are not enforced by the OS

In the OS there is the equivalent of

```
if uid_of_process == uid_of_resource or
   uid_of_process == uid_of_root
then
    allow access
else
    disallow access
```

Process Protection

Note that the privilege separation between superuser and normal user is used for protection of OS resources in exactly the same way as kernel mode and user mode is used for protection of hardware resources

Process Protection

Note that the privilege separation between superuser and normal user is used for protection of OS resources in exactly the same way as kernel mode and user mode is used for protection of hardware resources

It is the same idea being used in two different contexts

Process Protection

Critically, root can change the userid of its processes: by doing so it gives away its privileges, but thereby allows a normal user to have a process

Process Protection

Critically, root can change the userid of its processes: by doing so it gives away its privileges, but thereby allows a normal user to have a process

When a user logs in to a system a process, owned by root, starts up, changes its userid to the user, and then starts other processes as that user

Process Protection

Many resources are restricted by the OS so only the superuser can use them: this provides an extra level of protection to resources that are sensitive

Process Protection

Many resources are restricted by the OS so only the superuser can use them: this provides an extra level of protection to resources that are sensitive

For example, shutting down the computer. We can't allow any user process to turn off the computer, so this operation is restricted by the kernel to the root user

Process Protection

Many resources are restricted by the OS so only the superuser can use them: this provides an extra level of protection to resources that are sensitive

For example, shutting down the computer. We can't allow any user process to turn off the computer, so this operation is restricted by the kernel to the root user

Any shutdown program will need to have root ownership and this will be carefully policed by the system

Process Protection

Root is generally trusted by the kernel

Process Protection

Root is generally trusted by the kernel

So root-owned processes can completely trash everyone's programs and data on the machine if they want to

Process Protection

Root is generally trusted by the kernel

So root-owned processes can completely trash everyone's programs and data on the machine if they want to

This is why you should keep the use of the administrator account to a minimum

Process Protection

Root is generally trusted by the kernel

So root-owned processes can completely trash everyone’s programs and data on the machine if they want to

This is why you should keep the use of the administrator account to a minimum

Doing everyday stuff as administrator is just asking for trouble, and is throwing away many of those protection mechanisms that OSs have developed to provide

Process Protection

This user-level protection is what prevents my processes from interfering with your processes: as we have different userids, the kernel knows to keep them separate

Process Protection

This user-level protection is what prevents my processes from interfering with your processes: as we have different userids, the kernel knows to keep them separate

In particular, if I download an application or web page that contains a malicious worm or virus, properly working protection will limit the damage that malware can do to just my files and my processes

Process Protection

This user-level protection is what prevents my processes from interfering with your processes: as we have different userids, the kernel knows to keep them separate

In particular, if I download an application or web page that contains a malicious worm or virus, properly working protection will limit the damage that malware can do to just my files and my processes

Not ideal, but better than letting the malware have full reign over the entire machine

Process Protection

A big part of the spread of malware in Windows OSs is the weakness of this kind of barrier to their spread: too many programs run as administrator and this can ultimately cause the entire system to be affected

□

Process Protection

A big part of the spread of malware in Windows OSs is the weakness of this kind of barrier to their spread: too many programs run as administrator and this can ultimately cause the entire system to be affected

Note that if your OS *requires* the use of a virus checker, this is a strong sign that your OS is not confident in its implementation of process protection

□

Process Protection

A big part of the spread of malware in Windows OSs is the weakness of this kind of barrier to their spread: too many programs run as administrator and this can ultimately cause the entire system to be affected

Note that if your OS *requires* the use of a virus checker, this is a strong sign that your OS is not confident in its implementation of process protection

Virus scanners address the *symptom*, not the *problem*

□