What Does The Synchronize File System Right Mean?

https://rohnspowershellblog.wordpress.com/2015/01/16/what-does-the-synchronize-file-system-right-mean/

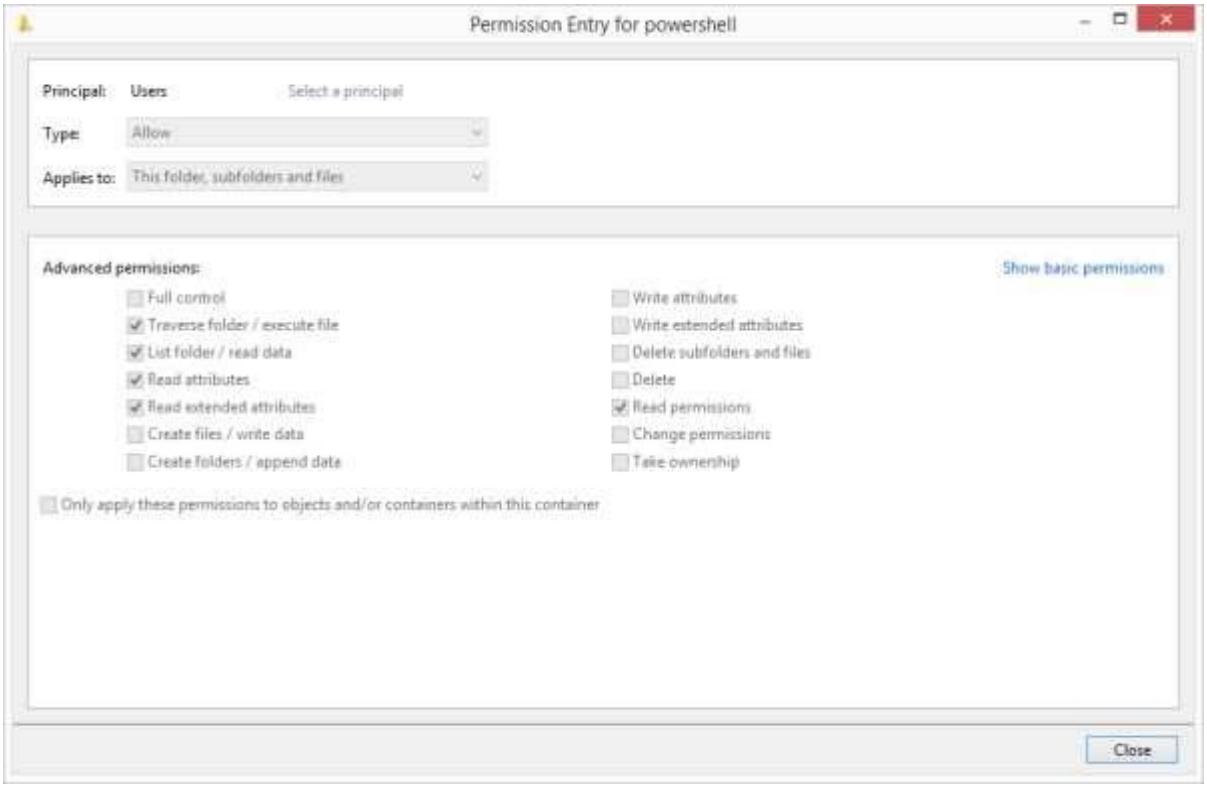
Have you ever tried to use PowerShell (or .NET) to mess with file or folder permissions and wondered what the 'Synchronize' right means? It pops up all over the place, like on existing ACEs:

```
PS C:\> Get-Acl c:\powershell | select -ExpandProperty Access | ft AccessControlT

AccessControlType IdentityReference                  FileSystemRights
----------------- -----------------                  ----------------
            Allow NT AUTHORITY\Authenticated Users         Modify, Synchronize
            Allow NT AUTHORITY\Authenticated Users                 -536805376
            Allow NT AUTHORITY\SYSTEM                          FullControl
            Allow BUILTIN\Administrators                       FullControl
            Allow BUILTIN\Users                 ReadAndExecute, Synchronize


PS C:\> Get-AccessControlEntry C:\powershell | ft -AutoSize

    Path              : C:\powershell
    Owner             : NT AUTHORITY\SYSTEM
    DACL Inheritance : Enabled

AceType Principal           AccessMask                  InheritedFrom AppliesTo
------- ---------           ----------                  ------------- ---------
Allow   Authenticated Users Modify, Synchronize         C:\           0 CC CO
Allow   SYSTEM              FullControl                 C:\           0 CC CO
Allow   Administrators      FullControl                 C:\           0 CC CO
Allow   Users               ReadAndExecute, Synchronize C:\           0 CC CO
```

And on new ACEs that you create (even if you don't include it):

```
PS> New-Object System.Security.AccessControl.FileSystemAccessRule Users, Read, Allow


FileSystemRights  : Read, Synchronize
AccessControlType : Allow
IdentityReference : Users
IsInherited       : False
InheritanceFlags  : None
PropagationFlags  : None



PS> New-AccessControlEntry -Principal Users -FileRights Read

FileSystemRights  : Read, Synchronize
AccessControlType : Allow
IdentityReference : S-1-5-32-545
IsInherited       : False
InheritanceFlags  : None
PropagationFlags  : None
```

If you try to check permissions using the ACL Editor, you won't see it anywhere. Here's the ACE for 'Users' from the 'C:\powershell' folder shown in the first screenshot above:



So, what is this mysterious right, and why does PowerShell/.NET insist on showing it everywhere? Let's start with the definition from MSDN:

> The right to use the object for synchronization. This enables a thread to wait until the object is in the signaled state. Some object types do not support this access right.

The first time I read that, I didn't think it sounded all that important. It turns out, though, that it's critical for working with files and folders.

Before I explain a little bit more about why that right shows up, let's briefly cover what makes up an access control entry's access mask. It's a 32-bit integer, which means that, theoretically, there are 32 different rights that can be controlled (32 bits means 32 different on/off switches). In practice, you don't get that many rights, though. No matter what type of object you're working with (file, folder, registry key, printer, service, AD object, etc), those 32-bits are broken down like this:

- Bits 0-15 are used for object specific rights. These rights differ between object types, e.g., bit 1 for a file means 'CreateFiles', for a registry key means 'SetValue', and for an AD object means 'DeleteChild'.
- Bits 16-23 are used for "Standard access rights". These rights are shared among the different types of securable objects, e.g., bit 16 corresponds to the right to delete the object, and it means the same thing for files, folders, registry keys, etc. As far as I know, only bits 16-20 in this range do anything.
- Bit 24 controls access to the SACL.
- Bits 25-27 are reserved and not currently used.
- Bits 28-31 are "Generic access rights". They are a shorthand way of specifying four common access masks: read, write, execute, and all (full control). These bits are translated into a combination of object specific and standard access rights, and the translation differs depending on the type of object the ACE belongs to.

The 'Synchronize' right is controlled by bit 20, so it's one of the standard access rights:

```
PS> [math]::Log([System.Security.AccessControl.FileSystemRights]::Synchronize, 2),20
```

If you manage to remove the right (or if you explicitly deny it), bad things will happen. For folders, you won't be able to see child items. For files, you won't be able to view the contents. It turns out some very important Win32 APIs require that right to be granted, at least for file and folder objects. You get a hint of it from this MSDN page:

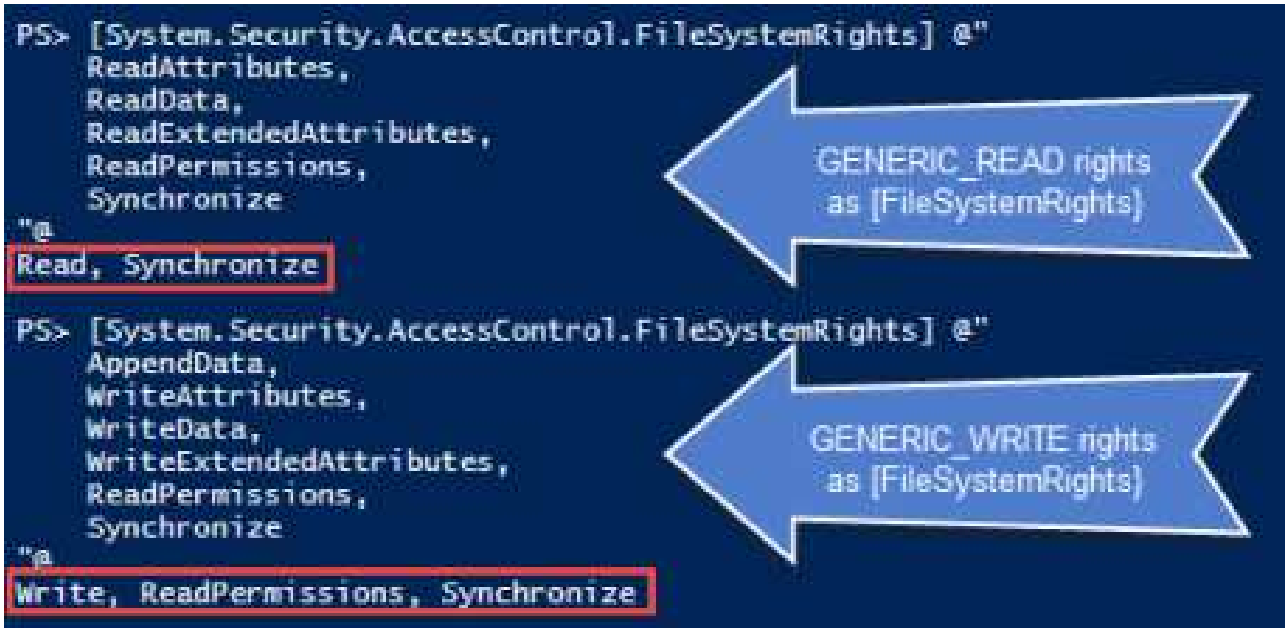
''Note that you cannot use an access-denied ACE to deny only **GENERIC_READ** or only **GENERIC_WRITE** access to a file. This is because for file objects, the generic mappings for both **GENERIC_READ** or **GENERIC_WRITE** include the **SYNCHRONIZE** access right. If an ACE denies **GENERIC_WRITE** access to a trustee, and the trustee requests **GENERIC_READ** access, the request will fail because the request implicitly includes **SYNCHRONIZE** access which is implicitly denied by the ACE, and vice versa. Instead of using access-denied ACEs, use access-allowed ACEs to explicitly allow the permitted access rights.

I couldn't do a good job of translating the actual definition of 'Synchronize' earlier, but I think I can translate this paragraph. It's saying that you can't create an access denied ACE for just GENERIC_READ or just GENERIC_WRITE as they are defined, because each of those sets of rights include 'Synchronize', and you'd effectively be denying both sets of rights. GENERIC_READ (bit 31) and GENERIC_WRITE (bit 30) are two of the four "Generic access rights" mentioned above. When they're translated/mapped to their object-specific rights, they make up a combination of bits 0-20 of the access mask (object specific and standard rights).

Once translated, GENERIC_READ is very similar to [FileSystemRights]::Read, and GENERIC_WRITE is very similar to [FileSystemRights]::Write. From the same MSDN page, here's a list of the object specific and standard rights that make up the generic rights (the [FileSystemRights] equivalents are listed in parenthesis):

- **GENERIC_READ**
  - **FILE_READ_ATTRIBUTES** (ReadAttributes)
  - **FILE_READ_DATA (ReadData)**
  - **FILE_READ_EA (ReadExtendedAttributes)**
  - **STANDARD_RIGHTS_READ (ReadPermissions)**
  - **SYNCHRONIZE (Synchronize)**
- **GENERIC_WRITE**
  - **FILE_APPEND_DATA (AppendData)**
  - **FILE_WRITE_ATTRIBUTES (WriteAttributes)**
  - **FILE_WRITE_DATA (WriteData)**
  - **FILE_WRITE_EA (WriteExtendedAttributes)**
  - **STANDARD_RIGHTS_WRITE (ReadPermissions)**
  - **SYNCHRONIZE (Synchronize)**

The [FileSystemRights] enumeration has values for Read and Write that almost match what is defined above. Since PowerShell coerces strings into enumerations, and enumerations will attempt to show you combined flags where possible, let's take a look at how those rights are seen when they're cast as a FileSystemRights enumeration:



Hopefully that makes sense. It's showing that GENERIC_READ in [FileSystemRights] translates to 'Read, Synchronize', which means that GENERIC_READ is not the same as [FileSystemRights]::Read since 'Read' doesn't include 'Synchronize'. GENERIC_WRITE and [FileSystemRights]::Write are almost the same, except [FileSystemRights]::Write is also missing 'ReadPermissions' in addition to 'Synchronize'.

So, why don't the generic rights translate to the same numeric values for [FileSystemRights]? It goes back to the warning from the MSDN page above: if you want to deny 'Read' or 'Write' only, you have to remove the 'Synchronize' right first. The ACL editor does this, and it doesn't give you any control over the 'Synchronize' right: if you create a new ACE it will determine whether or not the right is added, and it never shows it to you. The creators of the file/folder access control .NET classes didn't get that luxury. Each ACE has a numeric access mask, and that access mask needs to be translated with a flags enumeration. If the 'Synchronize' bit is set, then the flags enumeration string is going to show it, and vice versa. So, they did the next best thing: they pulled 'Synchronize' from the combined 'Read' and 'Write' rights in the [FileSystemRights] enumeration, and made sure that creating a new allow ACE or audit rule automatically adds the 'Synchronize' right, and creating a new deny ACE removes it. If an application wants to hide the 'Synchronize' right from the end user, that's fine, but the underlying .NET object will show it if it's present.

I hope that makes sense and clears that up. If not, please leave a comment where something needs to be explained a little better, and I'll try to expand on it some more.