Operating System Questions

Matthew Johnson, Trinity Hall

May 27, 2002

1 2001 Paper 1 Question 12

1.1

Polled I/O: The device driver repeatedly checks the device to see if it is ready to send the IO, and communication happens then, with the cooperation of the device driver for the whole transfer

```
requestIO()
while ( NOT ioReady() ) do wait()
transferIO()
return IOlocation
```

Interrupt IO: Here the device driver makes the initial call, but then runs the scheduler so that other tasks can be performed while the IO device is waiting for the IO. When the device is ready it makes an interrupt, and the scheduler starts the device driver again to perform the IO

```
requestIO()
schedule()
.....
//in the device driver interrupt handler
transferIO()
return IOlocation
```

Direct Memory Access: With this model of IO, the device it self can write to a specified area of memory. Hence, when an IO request is made, the device driver runs the scheduler as before, but doesn't get interrupted until the I/O transfer is complete into memory.

```
requestIO()
schedule()
.....
//in the device driver interrupt handler
return IOlocation
```

1.2

With Blocking I/O, the program halts execution, and waits for the IO to be completed before continuing. In the case of a keyboard, the program might request some input from the user, and would then wait until the user had typed in the input before continuing. With non-blocking I/O the programmer would call the routine to get input and it would return immediately with only the data in the keyboard buffer at the moment, rather than waiting for a line-feed or EOF. Asynchronous IO uses interrupts to call the appropriate code whenever the input happens. So, in the case of a keyboard you would register an interrupt handler for the keyboard, and when the user typed a key, your code would be called.

2 1999 Paper 1 Question 12

2.1

The directory service provides a mapping from the unique path/filename name-space to the file-system ID, and also handles access control from permissions.

2.2

A directory hierarchy is a way of increasing the name-space of files, and providing facilities of overlapping names for different files. This is managed by having a tree structure from either a single root, or a restricted number of roots, where each node can either refer to a file or itself be a directory, and contain further nodes. Each node in the tree has a name, which must be unique between siblings, and a particular node is referenced by the concatenation of all the notes between it and the root, separated by a separator character

2.3

File meta-data contains such information as the file size, access, creation & modification dates, and data about the access permissions of various users. File-systems such as Fat {12,16,32} merely store a read only flag, where as traditional Unix file-systems store the user & group of the file, and a read, write & execute flag for each of those, and for any other users. Some file-systems, such as XFS, or NTFS, implement Access Control Lists, which allow an arbitrary number of objects to be specified, and each given a unique set of permissions.

2.4

A hard link is an extra directory entry pointing to the same file-system ID. This is exactly equivalent to the original directory entry, and so 2 names in the root name-space can be used to access the same file. to implement hard links, file meta data needs to be stored in the FS, rather than the directory entry, since otherwise there could be 2 different link to a file, with different access restrictions.

2.5

A symbolic link is a file that merely contains another path & filename. When accessing the symbolic link, the directory service looks up the name in the file, and accesses that file instead. This allows flexibility in that it doesn't create recursive directory trees, like using hard links, and it allows links across file-systems. Because the symbolic link is a file, not just a directory entry, it will have a separate set of file meta data. The file-system drivers must be careful not to use the permissions of the symbolic link, but of the target file.

2.6

A Unix inode is a level of indirection below the directory space. Directory entries point to inodes, which contain the file-system meta data as outlined above, and also a link count, which is the number of hard links to the inode. This count is used to know when an inode can be disposed of. Also stored in the inode is a list of file-system blocks occupied by the file. These can be non-contiguous, which helps to reduce external fragmentation of the file-system. There are 2 types of access of the file-system blocks. Small files just use the direct accessing, which lists the blocks in the inode. This is relatively fast. For larger files some of the inode is reserved for 2 or 3 level indirection. These are links to other inodes which in turn contain the links to files. This is slower, but allows a much bigger file to be accessed while keeping inode size small.

3 1998 Paper 1 Question 11

3.1 name resolution

Files are identified by a name, but to resolve naming conflicts between different parts of the file-system, and to allow overlapping name-spaces, a directory hierarchy is specified. The directory service ensures that the correct file is accessed by a particular path through the directory hierarchy, and that the limitations of unique names between siblings in a given directory is preserved. This hierarchy may descend from a single root (as in unices) or from multiple roots (as in windows).

3.2 access control

Also controlled by the Directory Service, this is responsible for checking file-system meta data, and checking the access permissions of the file against the user trying to access it. In Unix permissions are specified by user, group or all. The directory service looks at the groups that the user trying to access the file is part of, and checks them & the user against the permissions bits. If the type of requested access (read, write, etc) is allowed then it passes the program to the fs-access layer, otherwise it returns an error.

3.3 existence control

As with most of the high-level functions, the directory service is responsible for checking that the directory entry exists, and creating it if necessary

3.4 concurrency control

Concurrency control means checking that a file is not updated by one application while it is open for writing by another process. This is done by having a bit in the file-system meta-data that is set when the file is opened for writing, and the file is then locked for writing by other processes. This would be in the actual file-system access layer.

4

4.1

If you have a file-system simultaneously mounted in several location the concurrency issue are more complex, but that can be solved if the other mount points are mounted read only.

4.2

In some senses the main memory is a cache for the hard disk, in that with DMA/UDMA controllers the data is copied into main memory by the hardware before being read by the CPU. Data can also be cached there before being flushed to disk. However, the majority of the RAM is not written to the disk. It would be more accurate to call the Hard disk an extension of the memory, since pages are swapped from memory into the hard disk, and us a form of non-volatile memory.

5

5.1

Communication via pipes can only be done between processes with a common ancestor because the pipe ID is stored in the ancestor thread, or memory shared by the processes, and that can only be done with child / ancestor processes.

5.2

Caching meta data causes concurrency issues, which is less of a problem with actual data, but with meta-data it can mean that a process sees the wrong access restrictions when accessing a file, and that is a security issue.

5.3

The original design for NT was for a micro-kernel system. Hence, enlarging the kernel with originally user-space code violates it's design. However, this will speed up the code, since optimizations can be made, and less indirection is needed, and it gives you the opportunity to do tricks with the scheduler, depending on which window is currently focussed.