Solution notes

Operating Systems (Part IA) 2005 – Paper 1 Question 12 (20 Marks) (SMH)

This question is partly based on the Windows XP case study, and partly on the general material on I/O.

Structure of XP

Diagram here similar to that in p154 of notes.

Windows XP is a layered system of modules. At the bottom we have the hardware abstraction layer (HAL) which provides a common interface to various platforms (regular x86, NUMA, etc). Next is the kernel which is non-preemptive and implements a set of kernel objects. At the same layer approximately are the device drivers which interface with specific pieces of hardware.

The next layer is the executive; this contains a bunch of modules including the I/O Manager, the VM manager, the Cache Manager, the Object Manager and File System Drivers. The executive deals with executive objects, and all runs in kernel mode.

Above this is the NT native API, and then the (user-space) subsystems. These include the win32 subsystem as well as the posix and OS/2 subsystems, security subsystem, etc.

Object Namespace vs. Directory Namespace

The XP object namespace is extremely similar to the Unix directory namespace. Both have a single distinguished root, allow recursive construction via directories (or directory object), and limplement a DAG (via symbolic links). The key differences are that the Unix namespace is limited to files (and named pipes), at least originally (today have /proc and /sys and other interesting things) while the XP namespace is used for *all* objects, whether they be files, processes, or internal kernel objects. A second difference is that the XP namespace is not directly visible at the UI level (save via e.g. an object manager browser) while the Unix namespace is a clear part of the API and UI.

Blocking, non-blocking and async I/O

2 marks for each variant

In blocking I/O, the invocation does not return until 'complete' — i.e. until the data has been written or read. Hence, using a keyboard for example, the blocking call would return promptly if data were available in the keyboard buffer, but otherwise would block

the process until a key was pressed.

In non-blocking I/O, the invocation always returns immediately, but may or may not be complete. In our keyboard example, the non-blocking read would return immediately with e.g. a return code stating many characters (≥ 0) have been read from the keyboard buffer.

In asynchronous I/O, the invocation always comprises two halves: a request followed sometime later by a response. In our keyboard case, the initial invocation would enqueue a read request along with an I/O completion handler. Then whenever keyboard data became available (perhaps with no delay), the completion handler would be invoked and could then process the data.

Improving I/O

1 mark each for any sensible suggestions.

Options include: reducing number of copies, using DMA wherever possible, using buffering to match device and processor speeds, using caching to avoid I/O wherever possible, and scheduling device requests.