Operating Systems 2000 p1q12 SMH

Action on Interrupt

1 mark for I/O handling in each of programmed I/O and DMA cases; 2 marks for interaction with scheduler.

When an interrupt occurs, the CPU transfers control to a/the interrupt handler. This is low-level software which may need to e.g. demultiplex the interrupt, save some state, etc.

Next an interrupt-specific device driver is invoked which will:

1. for programmed I/O device:
   - transfer data
   - clear interrupt (sometimes a side effect of transfer)
2. for DMA device:
   - acknowledge transfer
3. request another transfer if any more I/O requests pending on device
4. unblock any waiting processes.
5. enter scheduler or return

In summary, on an interrupt OS will (a) try to keep I/O device[s] busy and (b) potentially unblock (and dispatch) a process/processes.

Problems with Interrupt Load / DMA

2 marks for each the below, or viable substitutes.

Two main problems can occur. The most notable is that if the interrupt load is high enough, one doesn't get any time to schedule processes at all ("interrupt livelock"). Secondly, if a large amount of DMA traffic is going on, contention on the bus might cause degradation in processor performance.

Paper 1 Question 12 (20 Marks)

In the context of memory management, under which circumstances do *external* and *internal* fragmentation occur? How can each be handled? [4 marks]

What is the purpose of a page table? What sort of information might it contain? How does it interact with a TLB? [3 marks]

Describe with the aid of a diagram a two-level page table. Explain the motivation behind the structure and how it operates. [5 marks]

What pieces of information make up the *meta-data* of a file? [4 marks]

Describe the basic access control scheme used in the UNIX filing system. How does UNIX support more advanced access control policies? [4 marks]

Answers to Paper 1 Question 12

Fragmentation

2 marks for each of external and internal

External fragmentation occurs when we are allocating variable sized partitions/segments; after a number of segments of variable size have been allocated and subsequently freed, small portions of free memory will be left between the allocated regions. These may be too small to be useful. We can "solve" this problem by compacting memory.

Internal fragmentation occurs when we are allocating fixed size pages; requests must be rounded up to a multiple of the fixed page size, and so space is wasted. We cannot really solve this per se, although we can reduce the average waste by using smaller page sizes. Unfortunately this is at odds with other concerns...

Page Table Basics

One mark for each answer.

A page table is responsible for mapping from *pages* [fixed size portions of the virtual address space] to *frames* [fixed size portions of the physical address space].

Each entry in a page table contains the corresponding frame (if any), protection information, and a set of flags such as valid, referenced and modified.

The TLB acts as a *cache* for entries in the page table. The page table is only searched whenever an entry is not found in the TLB. Once the information has been retrieved from the page table it is inserted into the TLB.

Two-level Page Table

Diagram as per notes.

Two level page table is essentially a tree of depth two in which every node has the same [large] branching factor. Typically a node will be the same size as a page/frame, although this is not necessary.

Lookup operates by splitting the virtual address into three portions; the lowest s bits are the page offset (where the page size is 2^s) and are ignored. The remaining bits are divided into two [probably] equal portions. The most significant of these is used to index into the *root page table*. The result of this operation is either (a) a pointer to a second-level page table or (b) a "translation fault" [a discussion of superpages is not required]. In the former case, the second less significant portion of the address is then used to index into the second-level page table. This results in a page table entry which describes the mapping [if any] for the original page.

The motivation is that a linear table would be far too large to manage, and it is expected that virtual addresses will be allocated and accessed with some sort of locality of reference. Hence ideally few second-level page tables will be required, and those that are will be relatively full.

File meta-data

Two marks for getting below

The most important part of file meta-data is the information describing where on disk the contents of the file are to be found. This may be a simple table of blocks, or a hierarchy of tables, etc. etc.

One mark each for any further points. Example below.

Other file meta-data information includes the file size (i.e. how much of last block valid), the file type (dir, link, reg), the owner of the file, permissions on the file, the time of last access/creation/modification, the number of links (references) to the directory, etc.

Unix Access Control

Two marks for each bit.

For the first bit, just want the 3 bits (RWX) for owner, group, world along with some explanation of the user and group concept. Semantics when used with directories would be nice, but not necessary.

Unix allows fairly arbitrarily complex access control policies to be implemented by means of the setuid/setgid bits. A program with one/both of these bits set will execute with the relevant effective uid/gid. In addition, the program can programatically check other

factors such as calling uid/gid, time of day, day of week, etc, etc, and decide to terminate or continue based on this.