1999 Operating Systems

p1 q11
SMH

Question 2 (20 Marks)

Most operating systems provide each process with its own *address space* by providing a level of indirection between virtual and physical addresses.

Give three benefits of this approach. [6 marks]

Are there any drawbacks? Justify your answer. [2 marks]

A processor may support a *paged* or a *segmented* virtual address space.

a) Sketch the format of a virtual address in each of these cases, and explain using a diagram how this address is translated to a physical one. [8 marks]

b) In which case is physical memory allocation easier? Justify your answer. [2 marks]

c) Give two benefits of the segmented approach. [2 marks]

Answers to Question 2

Benefits of per-process address space

Two marks for each benefit.

There are a number of benefits including:

- Independence: each process can use (nearly) all of the address space without worrying about whatever other processes might currently be running.
- Portability: the size of physical memory is hidden behind the level of indirection ⇒ programmers do not need to know about the precise amount of RAM installed on the machine.
- Protection: a process cannot by default read or modify information within another process's address space.
- Static relocation: having the extra layer of indirection means that processes can be linked at a fixed *virtual* address, and so do not need to be relocated at load-time.
- (related to above): means that demand paging/segmentation is possible.
- Sharing: the same piece of physical memory can be mapped into multiple address spaces (e.g. shared libraries, kernel).

There are certainly more benefits and/or different ways to phrase things. Any valid benefit should receive the marks.

plqn
SMH

Drawbacks of per-process address space

Two marks for one (decent) drawback.

The main drawback is the fact that *sharing* between processes needs to be mediated by the operating system, whether explicit (e.g. IPC via mailboxes, pipes, ports, ...) or implicit (shared libraries as mentioned above).

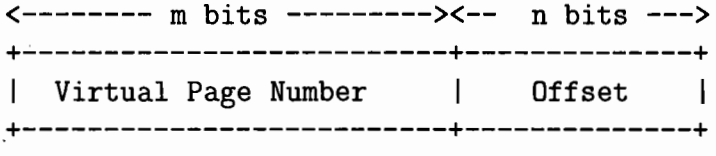
A second drawback is that additional overhead is incurred upon context switches due to the necessity to swizzle page or segment tables, and the subsequent cost of re-filling address translation caches (TLBs, segment registers or CAMs, virtual tagged caches, etc).

A third drawback is that, even in the best case, additional time (e.g. cost of TLB hit) and complexity (e.g. transistor cost of TLB + extra control logic for handling misses) is required. In practice a certain %age of accesses will be even more expensive due to page-table or segment table lookups.

Paged/Segmented Virtual Addresses

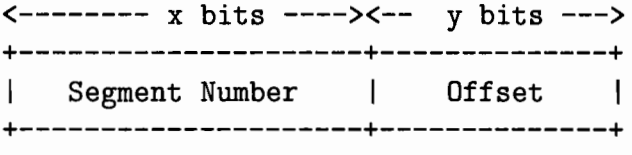
One mark for each virtual address format, and a further three marks for each explanation of how translation occurs. The level of detail given below should be enough for full marks.

A virtual address on a paged machine looks something like:



The first m bits (the *virtual page number* (VPN)) are translated into a k-bit *physical frame number* (PFN), where $k \leq m$. This translation occurs using a *page table*: a h/w or software supported data structure which maps VPNs to *page table entries* (PTEs). Each PTE contains the PFN to which the VPN should be mapped, along with various other bits. The diagram explaining the translation process should show the translation of the VPN to the PFN via the page table. It is not necessary to mention TLBs, etc.

A virtual address on a segmented machine looks something like:



The first x bits (the segment number) are an index into a *segment table*, each entry of which contains a *segment descriptor* (SD). The SD contains a *base* and a *limit*, along with various other bits. The translation proceeds by a) checking that the offset is $\leq$ the limit

and, if so, b) adding the offset to the base to produce the physical address. The diagram explaining the translation should show the segment table (and the base and limit fields within it). It is not necessary to mention CAMs, or two level segment tables.

Physical Memory Allocation

Physical memory allocation is simpler when using a *paged* scheme. This is since allocation is performed at a (relatively) coarse granularity (the physical frame). This reduces the amount of book-keeping the operating system must do. It also removes the problem of external fragmentation.

Benefits of Segmentation

One mark for each benefit.

Although memory allocation is more difficult, segmentation provides some nice features:

- The virtual address space is divided into logically distinct units. This provides a nice abstraction from both the OS and programmer points of view.
- Protection occurs at the segment level $\Rightarrow$ can occur at an arbitrary (e.g. byte) granularity.
- Since segments represent logical units, the segment level is ideal for supporting sharing.
- By using an explicit "attach", the OS (and the segment table) must only be concerned with segments which are actually in use.
- The above, coupled with the fact that segments tend to be much larger than a page, mean that less space is required in the CAM for equivalent performance.

The above is not an exhaustive list: any other valid benefits should also award a mark.