SOLUTION NOTES

Operating System Functions 2001 Paper 4 Question 7 (SMH)

What is demand paging?

In a demand-paged virtual memory system, pages of text and data are dynamically transferred to and from physical memory. Pages are brought into memory as and when they are referenced — i.e. "on-demand". It is implemented by marking the PTEs of the pages in question as "invalid, but available" in some manner. Then when a page fault occurs, the handler can arrange for the correct data to be brought in from disk/demand-zeroed.

What is temporal locality of reference?

Temporal locality of reference means that if a particular piece of memory m is accessed at time t, then it will with high probability be accessed at time $t + \Delta t$. In English: after we access a particular procedure or data item for the first time, it is likely we will access it again in the near future.

How does temporal locality of reference affect page replacement?

If we assume that temporal locality of reference is being exhibited by our process(es), then we can use recent behaviour as a predictor for the near future. In particular, we can assume that if a page has been referenced recently (i.e. it is "hot") that it will be referenced again shortly, and hence should be retained in memory. A page which has not been referenced recently ("cold") is a good candidate for eviction should it be necessary. This is the policy followed by the least recently used (LRU) page replacement algorithm, which evicts the page which has not been used for the longest time. Other algorithms using the same principle are: NRU, clock / second-chance.

What is spatial locality of reference?

Spatial locality of reference means that if a particular piece of memory m is accessed at time t, then $m + \Delta m$ will with high probability also be accessed. In English: after we access a particular memory location, we'll likely access neighbouring locations.

How does spatial locality of reference influence VM design?

The assumption of spatial locality of reference is fundamental to the effective use of aggregate memory regions such as pages or cache-lines. In the VM system, we assume that if any part of a page is needed, then the entire page is needed. This assumption means that we need only keep statistics and perform allocation on a per-page basis rather than on a per memory-location basis (which would be exorbitant).

Loading/unloading "objects" and "procedures"

This question is essentially asking for a view on "demand segmentation". I don't use the term however since I want them to think it through for themselves.

In support of this suggestion, we can say we would expect objects or procedures to better fit the properties of spatial and locality of reference than pages. This is because most of our observed locality of reference derives from language-level constructs such as loops and extended data-structures. Hence if we were to load / unload objects and procedures rather than pages we could expect to be able to capture more meaningful statistics for replacement strategies, to be able to save space (by packing objects adjacently within e.g. pages), and to save time (by e.g. faulting in an entire multi-page object at once rather than waiting for each part to be individually referenced).

On the down side, we now need (a) some way for the operating system to recognise higher-level constructs such as objects and procedures, and (b) per-object / per-procedure data structures in memory. We could in principle finesse (a) by doing everything at user-level after upcalls from the OS (i.e. by making it all application-specific). Not clear how to avoid (b) save by trying to encourage "objects" or "procedures" of a reasonable size.

Also have the old problem of external fragmentation re-rearing its ugly head – if we choose to evict a given variable sized object, then we have a variable sized gap into which we must load other things. Unless we're very lucky we'll have to either live with the wastage (obviating one of the arguments for doing this), or introduce a compaction scheme (CDC Cyber anyone?).

Finally we have the problem of dynamic linkage — once we begin to decouple translation from loading then we lose the ability to pre-link everything. So either we bite the bullet and go back to h/w supported demand segmentation, or we patch up references to procedures when those procedures are loaded/unloaded. Since this in general cannot cope with e.g. computed jumps or function pointers, we then need to add in some stub routines, and hence add an extra couple of instructions to every invocation. Nonetheless this approach has been tried with procedures at least (DPP) and shown to be fairly effective. Notably DPP did not replace but rather augment the VM system.