Advanced Systems Topics 2004
Paper 8 Question 5 (TLH)

This question is testing the lectures on DSVM and on scalable mutex design.

A *distributed shared virtual memory* (DSVM) programming model is often used on cluster computers because it can allow multi-threaded applications to be distributed across a set of machines without needing to be re-written.

(*a*) Describe the implementation of DSVM using a centralized page manager. Your answer should identify:

(*i*) What data structures are maintained by the page manager.

The centralized page manager runs on a single node. For each page it records (i) the page's owner (who last created or wrote to the page) and (ii) the set of nodes which hold copies of the page. The owner is the only node allowed to make updates to the page. For strong consistency these updates must be propagated to nodes which hold copies of the page.

(*ii*) What happens when a machine performs a read operation to a page.

There are two cases to consider. If the node already holds a copy of the page then it can be read directly. Otherwise, the node contacts the page manager to request an up-to-date copy of the page. The manager adds the node to the page's copy set and forwards the request to the page's owner. The owner sends a copy of the page to the requesting node. The requester acknowledges this to the page manager.

(*iii*) What happens when a machine perfoms a write operation to a page.

The node contacts the page manager to invalidate the copy set for the page (the values held in the copies may become out-of-date after the write). The node must then become the page's owner. If it is not already the owner then it contacts the manager, who in turn contacts the previous owner, who then sends the current contents of the page to the new owner. The new owner acknowledges this to the page manager.

[8 marks]

(*b*) Someone observes that the centralized page manager may form a *bottleneck* and a *single point of failure*. Do you agree with these observations? [2 marks]

The centralized page manager could certainly become a bottleneck: observe how in this basic scheme it is being contacted on every write operation. It is unfair to call it a single point of failure: the system is likely to become unusable if *any* node fails while owning pages.

(*c*) Sketch the implementation of a scalable spin-lock for use on shared-memory multiprocessor machines. You may assume the existence of an atomic *compare-and-swap* operation. [5 marks]

The best approach is an MCS-style queue-based spin-lock in which processors wait by performing reads on per-processor locations. When one processor releases the lock it updates the location that the next in line is spinning on. This avoids write contention by spinning processes and avoids stampedes when the lock is released.

A less good approach is a simple test-and-test-and-set lock.

A test-and-set spin lock is completely inappropriate as this question requests a 'scalable spin-lock'.

(*d*) Do you think that your spin-lock design would be appropriate for use on a DSVM system? Either explain why it will perform well, or suggest an alternative implementation which would be appropriate. [5 marks]

A spin lock in which processors spin locally performing read-only operations has the potential to work acceptably – the key thing to look at is whether there will be messages going between the nodes while they are waiting.

In order to be certain of good performance the programmer will want to avoid *false sharing* on the pages that hold the locations that processors spin on, perhaps by placing each queue node on a per-processor page along with other data that is generally private to that processor.