CS 33211 **OS**

Homework #3

Due in class at 12:30pm on Monday 26 November 2007
typed answers preferred

1. **(counts double) Deadlock can be prevented by eliminating the "no preemption" condition through the following algorithm: If a process A requests a resource held by another process B, and that process B is waiting for other resources as well, the resource requested by A is preempted from B and given to A so that A can proceed. Then B can only continue when it gets the resources it was waiting on and also recovers the resource preempted by B.**
   Will this algorithm prevent deadlock? Explain.
   Are there any problems with this algorithm?

   Assuming the resources are preemptable and their state can be saved and restored, it does prevent deadlock. However, starvation is possible if a process continually loses some of its resources while sleeping.

2. **Why is linking multiple object files together into a single executable file difficult?**

   Combining all the individual text, data, bss, heap, and stack segments together into larger combined text, data, etc. segments isn't difficult, but the difficulty is that any addresses for instructions or data need to be modified to reflect the new locations of those instructions / data. Further, external references from one object file to addresses in other object files must be resolved using the patch list so that those instructions have the correct address in the executable file.

3. **Does partitioning with fixed-size partitions suffer from external or internal fragmentation? Explain. How about partitioning with dynamic (variable-size) partitions?**

   Partitioning with fixed-size partitions suffers from internal fragmentation since the space in the partition will likely not be fully used.

   Partitioning with dynamic partitions suffers from external fragmentation if space between two partitions is too small to be useful.

4. **Why is the valid-invalid bit needed in a page table? Why not simply make the page table exactly large enough to hold all the valid pages?**

   The valid-invalid bit is used to quickly identify a page that is not in memory, triggering a page fault to page it in.

   It would be horribly wasteful of space to make the page table as large as the largest possible process, so in general it should be the exact size needed to represent the process. However, since it is indexed by an address of size N it must be 2**N elements in length rather than some arbitrary length. As a result a process with 5 pages will require 8 entries in its page table, the last 3 of which will always be invalid. Of the 5 pages, those in memory will be represented by a valid bit and those not in memory will be represented by an invalid bit and while it would use less space to store only the valid ones in the table that would break the indexing into the table.