Instructions:

- Print your name in the space provided below.
- This examination is closed book and closed notes, aside from the permitted one-page fact sheet. No calculators or other computing devices may be used.
- Answer each question in the space provided.
- If you want partial credit, justify your answers, even when justification is not explicitly required.
- Most of the questions involve making a conclusion and supporting it. Be sure you clearly state your conclusions. Being vague or hedging your bets will not be rewarded.
- There are 7 questions, priced as marked. The maximum score is 100.
- When you have completed the test, sign the pledge at the bottom of this page and turn in the test.
- Note that either failing to return this test, or discussing its content with a student who has not taken it is a violation of the Honor Code.

Do not start the test until instructed to do so!

Name Solution

printed

Pledge: On my honor, I have neither given nor received unauthorized aid on this examination.

signed

1. Recall the role of the *translation look-aside buffer*.

   a) [8 points] Suppose a process *P* is currently running. Describe the significance of the TLB entries from the perspective of the process *P*.

   The TLB contains page number/page table entry pairs that correspond to pages that were recently accessed by the process P.

   b) [6 points] The inclusion of a TLB is, of course, justified on the grounds that its presence should improve performance. Would the TLB be expected to improve performance if pure paging was used (i.e., paging without virtual memory)? Explain carefully.

   Yes. Even if pure paging is used, logical addresses must still be translated to correct physical addresses at runtime, and that will still require looking up the referenced pages in a page table in order to determine the corresponding frame numbers. (The only real difference is that there is no possibility that a page will not be resident.)

   As with paged virtual memory, it will still be much faster to look up a PTE at the processor level in the TLB than to do so in a page table in main memory (or even in a primary cache).

   The payoff may not be quite as great as when paged virtual memory is used, since there is no possibility that the relevant portion of the page table might itself be in a non-resident page.

2. Consider the following observation regarding the relative merits of global versus local page replacement policies in a demand-paged virtual memory system:

   With global replacement, a process may steal frames from other processes, causing those other processes to have additional page faults. As a result, those processes may also steal frames. Therefore, under global replacement, if one process begins thrashing it may cause other processes to also thrash, resulting in an epidemic of thrashing across the system.

   However, with local replacement, if one process starts thrashing, it cannot steal frames from another process and cause the latter to thrash as well. Thus, adopting a local replacement policy will prevent processes that begin to thrash from having an adverse effect on processes that are exhibiting good locality.

   a) [6 points] Critique the first paragraph above. In particular, address whether the paragraph describes a potentially real phenomenon.

      It is certainly true that if global replacement is used then a faulting process may "steal" a frame from another process. Since that causes the unloading of a formerly resident page of the victimized process, the result may certainly be that the victimized process will subsequently incur a page fault when it attempts to access that page, and that page fault would not have occurred (at least at the same time) if the frame had not been stolen.

      The possibility that the phenomenon could cascade, resulting in a general increase in faulting is low, but it cannot be discounted.

      So, the claim is true, although possibly exaggerated.

   b) [6 points] Critique the second paragraph above. In particular, address whether the claim about the superiority of local replacement policies is correct, or incorrect, or exaggerated.

      Again, it is certainly true that if local replacement is used then a faulting process cannot "steal" frames from other processes, and therefore that a faulting process cannot cause other processes to incur faults they would not have experienced anyway.

      On the other hand, a thrashing process will still cause the device that stores the virtual space to suffer an increase in I/O traffic. That would result in somewhat longer average access times for other processes that performed I/O to the same device, even if their I/O was planned rather than a result of the paging mechanism.

      So, the claim is exaggerated.

3. a) [6 points] Describe the phenomenon of starvation (in the context of an operating systems course).

Starvation is the phenomenon that a process or thread is continually denied some necessary resource due to the particular allocation policy that is used for that resource.

(This differs from deadlock in that there is no implication of a circular wait amongst processes.)

b) [6 points] Explain how aging can be used to prevent starvation.

Aging policies gradually increase the priority of a waiting process or thread with respect to the allocation of a resource. As a result, low-priority processes will eventually achieve a priority level that guarantees access to the resource. In a sense, this amounts to promoting a waiting process until it achieves a rank that allows it to gain the resources it needs.

In essence, aging policies make priority a monotonically increasing function of wait time.

The classic example is when scheduling priorities are aged so that low-priority processes are not delayed indefinitely even if there is a steady supply of high-priority processes.

Note that it is not sufficient to apply a policy of *demotion*, lowering the priorities of processes as they accumulate runtime (or possession time of some other resource). If demotion is used alone, long-lived processes will not be able to hog a resource, but a steady stream of short-lived high-priority processes could still do so.

4. a) [8 points] Explain what is meant by "the binding of instructions and data to memory".

Instructions contain references to data and to other instructions, such as variable and function names and labels in branch statements. At runtime, each of those variables and functions and instructions will be stored at some specific physical address (possibly a changing address, but that isn't relevant here.)

Binding refers to the act of mapping those names to specific physical addresses.

b) [8 points] Which of the following methods is performed by most general-purpose operating systems? Why?

compile-time binding load-time binding execution-time binding

In principle, any of them could be used, but the first two are highly impractical and thus not widely used in modern systems.

Compile-time binding would require the compiler to determine just where the text, stack and data segments will be loaded into physical memory. That is logically possible. However, it would reduce portability and it would eliminate the possibility of relocating a process image during execution. Hence, it would eliminate virtual memory.

Load-time binding would require a similar effort on the part of the loader. This would cause a substantial increase in the start-up time for a process since the loader would have to write the correct physical addresses into the process image. Technically, this could be extended to support relocation, but that would cause a tremendous slow-down since the addresses would have to be re-written whenever relocation was needed.

Execution-time binding is the norm on contemporary systems. As described in the context of paged virtual memory, this requires an efficient mechanism for address translation, but the problem is solved by combining hardware support such as the TLB and straightforward table structures. Execution-time binding supports relocation, and even fragmented relocation as in paged VM.

5. [16 points] There are four conditions that must hold simultaneously in order for deadlock to occur. List them and give a brief description of each.

Mutual exclusion	at least some resources can be used by only one process at the same time
Hold-and-wait	if a process requests a resource and the request is denied, the process will wait until the request can be granted, and it will continue to hold its resources while it waits
Non-preemption	there is no provision for taking a resource away from one process in order to grant it to another, waiting process
Circular wait	there must be a collection of processes, each of which is waiting to be allocated a resource that is held by another process in the collection

6. a) [6 points] What does the term *quantum* refer to in the context of CPU scheduling?

The *quantum* is the length of time a process is allowed to remain in the running state before the scheduler will choose another process to run next.

b) [8 points] How should the length the quantum compare to the time to perform a context switch? Why?

Let the quantum be Q and the time for a context switch be C. From the perspective of the scheduler, time consists of alternating periods of length Q and C. So, the fraction of time spent performing context switches would be given by:

$$C / (Q + C)$$

Since this represents pure overhead (no user work is being performed during a context switch), it is important that this fraction not be too large. Therefore, the quantum should be substantially larger than the time for a context switch.

7. [16 points] Carefully describe the procedure by which a logical (virtual) address is translated into a physical address.

Assume a page size of 2^K.

Given a logical address L, the corresponding virtual page number is obtained by taking the high-order bits 31:K of L.

The virtual page number is looked up in the TLB (or, if necessary the page table). This may trigger a page fault if the referenced page is not resident, but that is tangential to the process of translating the address.

Looking up the virtual page number yields the number of the frame of physical memory in which that page is stored.

The offset of the referenced data or instruction within the page is obtained by taking the low-order bits K-1:0 of the logical address.

The physical address is then obtained by concatenating the frame number and the offset.