Dept. Computer Science	**Operating Systems**
Vrije Universiteit	**30.08.2006**

1a Disabling interrupts used to be a standard technique for protecting critical sections in operating systems. Why does this technique fail for multiprocessor operating systems? *5pt*

The answer is quite simple: disabling interrupts lets a processor continue execution without indeed being interrupted. That means that a single thread of control is executing. However, in a multiprocessor system we need to do something about two or more processors accessing the same memory locations, and disabling interrupts will not help in that case.

1b Explain how the test-set-lock (TSL) instruction works, and how it can be used to implement the operations enter_region and leave_region for protecting a critical section. *10pt*

The TSL instruction atomically copies the value of a variable lock into a register rx and sets lock to 1. It can be used to implement the two operations as follows:

```
enter_region:
    tsl RX, LOCK
    cmp RX, #0
    jne enter_region
    ret

leave_region:
    move lock, #0
    ret
```

1c Some solutions for protecting critical sections require busy waiting. When is busy waiting acceptable? Give an example when it is acceptable. *10pt*

Busy waiting is acceptable if the time that is known that the waiting will last is small. For example, when implementing higher level synchronzation primitives, such as semaphores, the data structures that implement the semaphore need to be protected as well. This can be done by a busy-waiting solution assuming that blocking while manipulating those data structures will not happen.

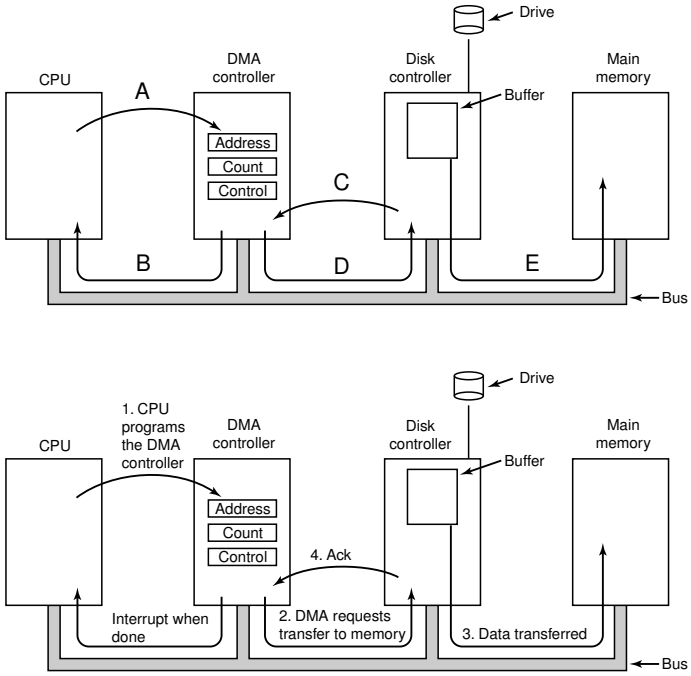
2a Many system calls are wrapped in a library function. Explain the steps that take place between calling and returning from that function. *10pt*

When calling the function, the parameters for the system call are put in a place specific to the operating system, such as the current stack. In addition, a number identifying the required call is placed in a designated register. At that point, a trap to the kernel is made, essentially switching from user mode to kernel mode. Inside the kernel, control continues with a dispatch function that looks up the address of the system-call implementation, taking the system-call ID as its input. The implementation is executed, after which control returns to the library function, then to the caller, possibly after copying results to a specific place (again, e.g., the stack).

2b We have always argued that an operating system can be implemented in a high-level language such as C. What about the library function wrapping system calls? *5pt*

No, we will generally have to assume that assembly code is needed to switch to the kernel using a machine-specific instruction. In practice, this means that a few lines of assembly code need to be included in the wrapper function.

3a Explain how DMA works by describing and ordering the actions associated with each of the arrows *A–E* in the following figure. (Note that the current labels are unrelated to the ordering of actions.) *5pt*



3b The disk controller for DMA has an internal buffer. Why is this buffer needed? *5pt*

First, internal buffering allows the controller to check for errors during transfer. Second, as soon as a disk transfer starts, the bits are streamed to the controller. They cannot be written immediately to main memory as they may be lost when the memory bus turns out to be occupied.

3c Where does the operating system come into play with DMA? Be precise! *5pt*

In fact, it hardly does play a role. The OS only needs to set the proper registers for the DMA controller, and be able to process interrupts generated by that controller.

4a Explain the purpose of a page table for virtual memory systems. *5pt*

A page table is nothing but a collection of mappings from virtual page numbers to page frames. The idea is that when a virtual page is looked up in the table, the OS will find the associated page frame, if any. If the virtual page is not in memory, a page fault will allow the OS to replace a page frame with the referenced virtual page.

4b Explain how a multilevel page table works, and why they are often needed. *5pt*

Page tables can be very big, certainly when virtual memory grows. However, not all programs need that much memory, in which case it is more efficient to divide the page table into multiple parts. The root page table then contains references to next-level page tables. Each virtual address contains two pointers: one to an entry in the root table*, from which the second-level table can be identified. The second pointer is in index into the second table, from which subsequently the associated page frame can be identified.*

4c What is the purpose of the translation-lookaside buffer (TLB)? *5pt*

The TLB is used to cache recent page lookups. Before actually inspecting the page tables, the MMU first sees whether it can find the addressed virtual page in the TLB, often by doing a parallel search. If found, this saves a relatively expensive table lookup.

4d Some page frames may never be evicted to make place for other pages. Give an example of page frames that should always stay in memory. *5pt*

Typically, the pages that contain the data and code of the process that decides on page replacement should be pinned down. Likewise, it would not be a good idea to remove pages containing the code for accessing the disks.

5a What is the difference between a hard link and a symbolic link in a file system? *5pt*

A symbolic link is a special file containing the path name to another file. When resolving a symbolic link, name resolution continues with the path name found in the link. A hard link is a reference to an i-node which is stored as a normal directory entry. An important difference between the two types of links is that hard links can only refer to files in the same file system as where they occur.

5b An i-node in main memory has a flag indicating that it is "mounted on." Describe how a file lookup proceeds when such an i-node is reached. *5pt*

When the i-node is reached, name resolution proceeds by looking in the superblock table to see which file system has been mounted on the i-node. Each entry in the superblock table points to such an i-node. Once the entry has been identified, resolution continues with the root i-node of the associated file system.

5c Assume that `/dev/hda1` has already been mounted. Can it be mounted again, but at a different point in the current file system? *5pt*

As long as the superblock table can support multiple mounts of the same file system, there is no reason why this shouldn't be possible.

***Grading:** The final grade is calculated by accumulating the scores per question (maximum: 90 points), and adding 10 bonus points. The maximum total is therefore 100 points.*