Comprehensive Spring 97: Operating Systems

1 Short Questions

Short answers expected (not more than half a page each).

1. Define the term "utilization".
2. What is the key difference between capabilities and access lists?
3. What is a covert channel?
4. What is the key difference between a thread and a Unix process?

2 Disks and File Systems

Reading a block from disk involves three separate activities: (1) the seek, moving the disk to the cylinder on which the block resides, (2) the rotational latency, waiting for the beginning of the block to appear under the disk head, and (3) the data transfer, moving the data to the controller while the block passes under the disk head.

1. Give some typical values for the cost of these three operations on current disks.
2. Given a disk with n cylinders, and assuming access to disk blocks is uniform (meaning, each disk block is equally likely to be accessed) and independent (meaning, there is no relationship between the current block being accessed and the next), what is the average seek length, expressed in cylinders?
3. What do file systems do to ameliorate this situation?
4. Do you think such uniform and independent accesses are likely? If not, why not? How could a file system take advantage of this?

3 Synchronization

Budget cuts in the country of Galvania have forced plans for the construction of a bridge over the Galva river to be scaled back to where the bridge is only wide enough for one single person. Multiple people can be on the bridge at the same time, but not one next to the other. In particular, there is no way for two people going in opposite directions to cross each other on the bridge. This, combined with the fact that Galvanians are incredibly stubborn and never ever back up, has led to a rather pressing problem.

Provide a solution to this problem using semaphores. Your solution should provide efficient use of the bridge, but avoid starvation.

4 Memory Management

In an operating system, it is often necessary to copy a large object, for instance, a large file, from one address space to another. When done by plain copying, this is an expensive operation because of the time taken by the copy and because we need to allocate twice the number of frames necessary to hold the object, once for the source address space and once for the destination. Some operating systems use a clever memory management trick, called *copy-on-write*, to lower the cost of such operations, both in terms of execution time and main memory usage.

For simplicity, assume that the object consists of a number of pages in the source address space, and starts and ends on a page boundary. When the copy is invoked, no new memory frames are allocated and no physical copy operation is performed. Instead, the appropriate page table entries in the destination address space are set to point to the memory frames to which the object is mapped in the source address space. Also, in both address spaces, the protection for those pages is set to *read-only* so that writes will cause a protection violation.

As long as all processes in the source and destination address spaces only read (or do not access) the page, they continue to actually share the same physical memory frames. When a process in one of the address spaces however attempts to write to the object, a protection violation occurs, This protection violation is called a *copy-on-write fault*. In response to such a fault, the operating system allocates a new frame, copies the page into the newly allocated frame, and changes the page table entry for that page in the current address space to point to the new frame rather than the original. Also, the protection for the pages in both address spaces is set to its original value.

Your task is to design an implementation of copy-on-write, with the following important simplification. You may assume that it will never happen that an object is first copied from address space A to address space B, and then from address space B to address space C, etc. You must, however, be prepared to handle the case in which an object is first copied from A to B, then from A to C, etc. You may, in addition, assume that processes do not share pages in any other way.

Your solution should consist of three parts:

1. A description of the data structures used by your implementation.
2. A pseudo-code description of the steps taken at the time of a copy.
3. A pseudo-code description of the steps necessary to handle a copy-on-write fault. When your trap handler is called, you are supplied with the virtual memory address of the memory access that caused the fault.

You may assume the existence of the following primitives:

1. `get_active_pid()` returns the process identifier of the process currently running.
2. `lookup_frame(pid,va)` returns the frame number corresponding to virtual address `va` of process `pid`.
3. `lookup_prot(pid,va)` returns the protection status of the page containing virtual address `va` of process `pid`.
4. `update_page_table(pid, va, fn, prot)` updates the page table entry for virtual address `va` of process `pid` to contain frame number `fn` and protection `prot`.
5. `get_new_frame()` returns the frame number of a newly allocated frame. You need not worry about how physical memory allocation is done, and you may assume that this routine will always succeed.
6. `copy_frame(frame1,frame2)` copies the contents of `frame1` to `frame2`.
7. `resume_write_instruction()` resumes a faulted write instruction.

Finally, what do you think about the efficiency of copy-on-write? On uniprocessors? On multiprocessors?