CS 167 Final Exam

2pm May 14, 2016
Closed Book
Please Use a Pen, Not a Pencil!

Do all questions.

1. The implementation given of *thread_switch* in class is as follows:

```
void thread_switch() {
  thread_t NextThread, OldCurrent;

  NextThread = dequeue(RunQueue);
  OldCurrent = CurrentThread;
  CurrentThread = NextThread;
  swapcontext(&OldCurrent->context, &NextThread->context);

  // We're now in the new thread's context

}
```

   a. Suppose we don't have a *swapcontext* routine to call. Instead, we have to use *getcontext* and *setcontext*. Recall that *getcontext* saves all of the calling thread's registers (including the stack pointer, but not the instruction pointer) into its argument, a pointer to a data structure of type *ucontext_t,* and returns nothing. The routine *setcontext* restores the registers from its first (and only) argument (of type *ucontext_t*). Show how to implement *thread_switch* using *getcontext* and *setcontext*. If changes to the thread control block (of type *thread_t*) are required, indicate what they are.

b. Suppose *thread_switch* takes an argument and this argument is referred to after *setcontext* returns. Explain what, if anything, must be done to ensure that what is referred to is what was passed in the most recent call to *thread_switch*.

2. Recall that in Unix, when a file whose set-user-id bit is one is *exec*'d, the process's effective *user ID* and the saved *user ID* become the owner of the file, while the *real user ID* is not modified. A program may set its effective user ID to be anything if it's currently *superuser*, otherwise it may set its effective user ID only to its real, effective, or saved user IDs. Also, file access checks are done using a process's effective UID. A set-UID program uses these features to check if its caller has permission to access a given file, perhaps one passed as an argument. (It first sets its effective UID to that of the caller, opens the file, then sets its effective UID back to the saved UID.) A primary use of these features is to allow the owner of a file to keep others from directly accessing the file, but provide a program (whose set-UID bit is set) that performs operations on the file on the behalf of others, but under the control of the file's owner.

    a. This approach works well if the set-UID program is owned by superuser — such programs are effectively extensions to the kernel. However, it's not that useful if the file containing the program is owned by a normal (untrusted) user. Explain why not.

b. We'd like to fix this: we want to make it possible for some user, say Mary, to create a program that others may run as set-UID-to-Mary, with none of the problems mentioned in the answer to part a. Explain how this might be done. Hint: certain functionality may have to be turned off, and we may need to adopt a convention for passing files as parameters to Mary's program.

3. An operating system is said to be kernel-preemptible if threads may be preempted, even while running in kernel mode, and their processor switched to another thread. There may, of course, be certain code sequences during which such preemption is disabled. An operating system is said to be SMP-safe (symmetric multiprocessor safe) if it can run on a multiprocessor system in which any thread can run on any processor, whether in user mode or privileged mode, and any interrupt handler can run on any processor.

    a. If an operating system is kernel-preemptible, is it necessarily SMP-safe? Explain why not.

    b. If an operating system is SMP-safe, is it necessarily kernel-preemptible? Explain why not.

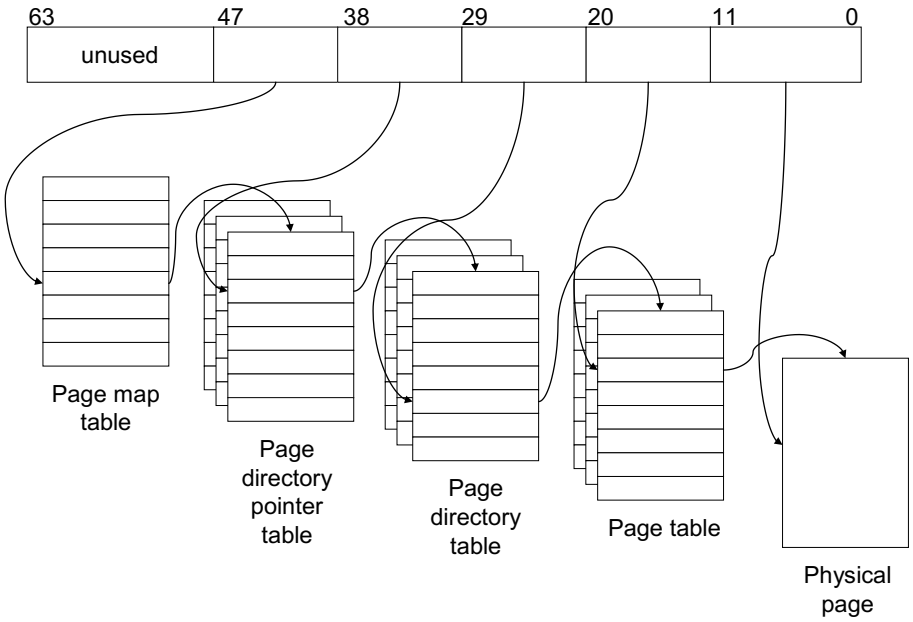
4. Striping is a technique in which files are spread across multiple disks. The striping unit is the block size on each disk; the first striping-unit-sized portion of a file is written to the first disk, the second to the second disk, and so forth. An obvious concern is the size of the striping unit. Assume that data is read from and written to a striped file system in some standard length L. Thus the smallest possible striping unit is the size of a disk sector; the largest useful striping unit is L. You may assume that L is sufficiently large enough that it is worth discussing what the size of the striping unit is.

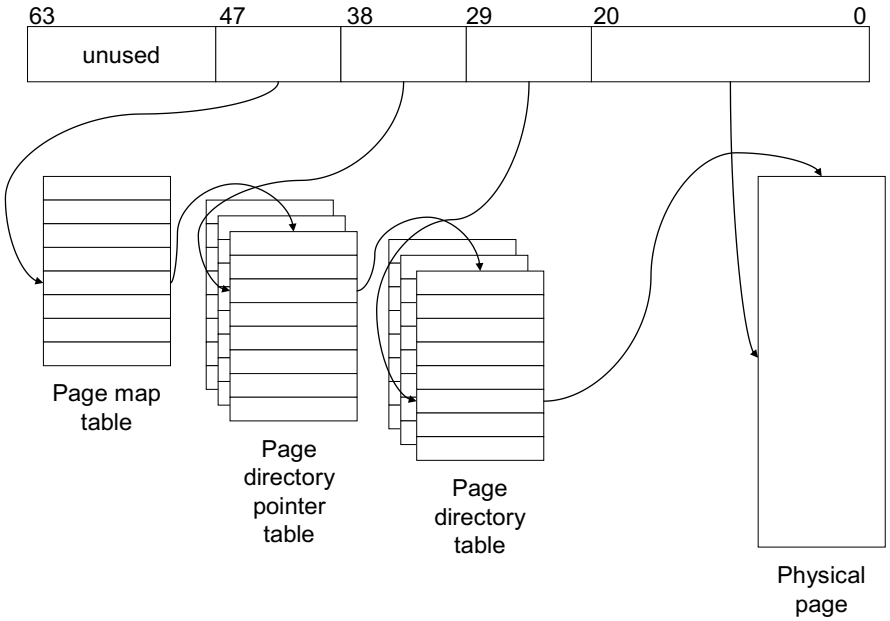
    a. Explain why it makes sense for the striping unit to be small (such as one sector in length) on systems in which just a single thread is generating file-system requests.

    b. Explain why it makes sense for the striping unit to be large on busy servers (i.e., on systems in which many threads are generating file-system requests). (Hint: you might answer this by showing why a small striping unit is not good.)

    c. Suppose we replace the disks with flash memory, i.e., we are striping files over a number of flash devices. Would your answers to parts a and b be different? Explain.

5. The x64 architecture supports both 4KB and 2MB pages as shown below.





a. A little-known Linux feature allows one (if the kernel is properly configured) to map a file into the address space using 2MB pages rather than 4KB pages. What would be the advantage of doing this? Why isn't it always done? (Be sure to consider, among other things, the implementation of fork on Unix and typical use of fork.)

b. Suppose the x64 architecture supported 4KB and 4MB pages. Could the two different page sizes coexist in the same address space as the 4KB and 2MB page sizes do in the actual architecture? Explain.

c. Consider two possible implementations of fork using copy on write. One, which we call shallow copy on write, involves sharing just the actual page frames, but both processes have their own private copies of the all page tables (including page directory tables, page directory pointer tables, etc.). The other, which we call deep copy on write, involves sharing both page frames and all page tables. Thus with deep copy on write, after fork, both parent and child share the page map table and nothing is immediately copied. When page frames are modified, all page tables (and page directory tables, etc.) that must be modified are also copied. But with shallow copy on write, when a process forks, all page tables, including the page map table, are copied at the time of the fork. It should be clear that, with deep copy on write, forks are cheaper than with shallow copy on write. Why might it be advantageous, nevertheless, to use shallow copy on write? While we're looking for a qualitative and not a quantitative explanation, you should mention any important data structures that might be required in one but not the other.

d. How might one optimize shallow copy on write to make it much cheaper for the usual case of fork, in which it's followed quite soon by an exec? (Hint: this is yet another application of "lazy evaluation").

6. NFS v2 and v3 systems allow clients to "hard-mount" remote file systems, so that in the event of a server crash, clients repeatedly retry RPC calls until the server comes back up. If clients held locks on any of the server's files, the NLM protocol recovers these locks for them.

    a. Assume that all RPC calls are to idempotent procedures and that the only possible failure is a server crash (i.e., clients don't crash and the network is well behaved). Other than timing issues, will server crashes have any adverse effects on clients (assuming the server comes back up)? Explain.

    b. Suppose now that the network is not well behaved — it contains routers and it is possible that the network might partition itself for short periods so that some clients can communicate with the server and others cannot. Are clients immune from any ill effects of server crashes combined with network partitions? Explain.

c. In DFS, servers give clients tokens that allow clients to use cached data from files. These tokens have time limits associated with them, so that if the token times out, the client has to assume that any cached data it has from the server is invalid and must obtain a new token (and new data) from the server. Explain how tokens might be used to help with the situation of part b. (Hint: it won't necessarily solve the problem, but it might help client applications become aware of problems.)

d. Explain why DFS does not support hard-mounted file systems.