CPS 110 Final Study Questions

Fall 1997

1. The program *sparsefile* creates a file, seeks to location 1G-1, and writes a single byte. On a Unix system, the size of the resulting file is one gigabyte as reported by *ls -l*. How much disk space does the file consume? If a program opens the file and reads the locations that were never written, what happens? What if the file is copied with *cp*? How much disk space does the copy consume?

2. Suppose a CPU scheduling algorithm favors those processes that have used the least CPU time in the recent past. Is this a reasonable policy? Why will it favor I/O-bound programs and yet not permanently starve CPU-bound programs? (from Silberschatz)

3. The Seagate Cheetah disc drives have a spindle speed of 10,000 rotations per minute, while most disks on the market today spin at 3600 or 7200 RPM. The Cheetah 9LP has an average of 208 512-byte sectors per track on each of six platters, with a separate head for both surfaces on each platter. There are about 7000 tracks per surface, for an aggregate capacity of 9 GB, and the average seek time is about 6 milliseconds. What is the average access time for a randomly chosen 8K block? What would the access time be for a disk with the same geometry rotating at 7200 RPM? How much does the faster rotation speed improve access time?

4. Suppose the Cheetah 9LP described above is connected to your system through a controller with no track cache, and that a process is reading a file sequentially. Suppose further that the user process and kernel take 300 microseconds to issue each subsequent read request, that the seek time between adjacent tracks is 700 microseconds, that the kernel uses a block size of 8K and does no read-ahead, and that the time to transfer each 8K block from the disk controller buffer across the I/O bus into system memory is 400 microseconds. What is the highest bandwidth the file system can deliver, assuming optimal block placement? What is the block placement scheme that will deliver this bandwidth?

5. Now suppose the controller for the Cheetah 9LP has a track cache, i.e., the access time for a block that passed under the head recently during the current rotation or immediately preceding rotation is equal to the bus transfer time. What is the highest bandwidth the file system can deliver, assuming optimal block placement? What is the block placement scheme that will deliver this bandwidth?

6. Most advanced file systems read ahead for sequential access, i.e., they always issue the next disk access request before the current one has completed. Explain how this works. Suppose that instead of a track cache, the system running the Cheetah uses read-ahead. What is the highest bandwidth the file system can deliver, assuming optimal block placement? What is the block placement scheme that will deliver this bandwidth? How much does read-ahead improve file read bandwidth if the Cheetah has a track cache?

7. Suppose a process is sequentially reading a file that is an array of 128-byte records, issuing a separate 128-byte read system call for each record. Most systems will deliver disk bandwidth for this application, assuming the CPU is able to process each record in less than the time it takes to access each record on average. Explain the key technique used to handle this application, and regurgitate the buzzword that best summarizes why the technique is effective.

8. Advanced file systems can generally deliver full disk bandwidth for streams of sequential writes, as well as reads. The track cache has no value for writes, since it is volatile and is used as a write-through cache; a disk write does not complete until the new data has been written to the nonvolatile disk surface. Explain three techniques that systems use to deliver full disk bandwidth for writes. What is the role of the block buffer cache in implementing these optimizations?

9. Under what circumstances can the file system avoid reading a file block from disk in order to satisfy a write request? (from McKusick)

10. You are writing a program to multiply two matrices A and B. The matrices are 1024*1024 arrays of 4-byte integers, stored in row-major order. A and B are stored in mapped files; the result matrix C is in your "uninitialized static data" segment (e.g., the heap).

    Your initial approach is to use the obvious algorithm, filling in the result matrix left-to-right, top-to-bottom. You run your program over a virtual memory OS that allocates exactly 1026 4K page frames to storing the matrices (unrealistically small but useful for the problem), and uses perfect LRU page replacement with no clustering or prefetching.

    How many page faults will your program incur on the matrices? Assuming the program consumes 10 seconds of user CPU time, the average time to fault a page from disk is 5 ms, and the average time to zero a page is 150 microseconds, how long will the program take to run? (ignoring OS overheads other than page faults on the matrices)

11. Suppose instead that you write your program to fill in the result matrix top-to-bottom, left-to-right (i.e., fill out each column before moving on to the next). How many faults will it incur? How long will it take to run?

    Extra credit: can you devise an even faster algorithm for multiplying matrices that don't fit in memory? This is called an "external memory algorithm", and some researchers in our department make a very good living devising new ones.

12. Suppose that you run your matrix multiply program on 8192*8192 matrices over an NT system that manages address translation tables as described in class and in the Custer book. How many page faults will be incurred to access the first column of any of the matrices? What is the translation table overhead relative to the volume of the data accessed? What if a hashed translation table is used instead?

13. Suppose the process *vmhog* generates a constate rate of 180 4K page faults per second, with each page fault consuming 50 microseconds of system overhead and 5 milliseconds of disk access time. What is the user CPU utilization? If I double the speed of the CPU, how much does this improve application performance? What is the new paging rate?

14. Now suppose all faults are satisfied using remote paging, with the same overhead and 100 microseconds of network latency. What bandwidth will *vmhog* drive on the network? What is the user CPU utilization? Why is user CPU utilization a useful measure of performance in this case? If I double the speed of the CPU, how much does this improve application performance? What is the new bandwidth demand on the network?

15. Suppose the CPU-bound job *cpuhog* is scheduled with the I/O-bound job vmhog on the disk paging system of Problem 10, and that vmhog consumes a total of 10 seconds of user CPU time, while *cpuhog* consumes a total of 100 seconds of user CPU time. How long will it take for both jobs to complete on a system that uses pure round-robin (preemptive FIFO) scheduling with a quantum of 20 milliseconds? How long will it take if the system instead uses a preemptive multi-level scheme that gives priority to I/O-bound jobs?

16. Tanenbaum ch 2 #22.

17. Tanenbaum ch 4 #8 and #10

18. Tanenbaum ch 4 #22 and ch 5 #5

19. Tanenbaum ch5 #8