Software Systems Answers 1/98

Process scheduling, race, block consistency

1.a. There are lots of answers here. A few:

Minimizing response time and turnaround time are contradictory. The first favors small interactive processes while the second favors the batch processes.

Maximizing fairness and throughput are contradictory. Fairness means giving everyone some "fair" chunk of the cpu in a timely fashion, while maximizing throughput would favor short batch jobs over the others.

Maximizing efficiency and response time can be contradictory. Minimizing response time can mean context switching more often, and the CPU isn't during "useful" work while context switching.

1.b. Round robin, First Come First Server, and FIFO are all reasonable, since the processes are all of equal priority. ~~As a process returns from waiting on I/O, it gets put at the end~~ of the run queue. Time slices can be small, although the interactive processes are likely to block before the end of their time slice in any case.

1.c. The video feed needs continual, frequent attention from the CPU, or the video will be jerky or you'll have to drop frames to keep up, etc. Giving the video process higher priority may help, but that can hurt the performance of the small interactive processes. Some guarantee of some amount of CPU per unit of time would be good for the video.

1.d. You may want to consider process size. If memory is in high demand, then swapping out a large process may make room for several smaller processes to run. If you bring back in the large process, this takes time to bring its pages in, so you'll want to run it for longer to amortize this cost. (Fewer but longer time slices.)

1.e. Cheaper memory has meant more memory on computers, which makes it easier to avoid swapping and paging problems. There are still a few really huge processes, depending on your workload, that may need to be swapped or paged, but most processes are more likely to avoid this problem.

2. If more than one process or thread accesses and modifies a shared resource (variable, data structure, file, etc.) in a non-atomic way, then the interleaving of these processes' actions can leave data in an inconsistent state.

For example, if two processes are allowed to add blocks to a file at the same time, we could end up with a corrupted file.

As another example, consider the following code:

```
P1
---
R := x;
R ++;
x := R;
```

```
P2
---
R := x;
R ++;
x := R;
```

50

If the three lines of code are protected by a lock, then if both processes run, in either order, x will be incremented twice (the desired outcome).

But if the code sections are not protected, but interleaved, then the processes can interfere with each other, and the effect of one's increment can be lost. For example, assuming x = 1, P1 executes line 1 and line 2 and is then descheduled. P2 executes all three statements, incrementing x to 2. Then P1 continues running and also sets x to 2, since it is still using the old initial value of x.

3.a. The block can be neither in a file nor on the free list (both counters = 0)

The block may be in more than one file (file counter >= 1)

The block can be both in a file or files and on the free list (both counters >= 1)

The block may be on the free list more than one time (free list counter >= 1) Note that this can't happen if the free list is implemented as a bitmap.

3.b. To fix the first problem, assign the block to the free list since it's not in use.

To fix the second problem, assuming the block appears in N files, make N-1 copies of it in previously free blocks and assign these blocks to those N-1 files in place of the multiply referenced block.

To fix the third problem, remove the block from the free list. If the block also appears in multiple files, handle it as per the second problem.

To fix the fourth problem, remove the extra block references from the free list leaving just one reference to the block.