

CS 574
Midterm Exam
October 6, 2003

The following exam is open book and open notes. You may feel free to use whatever additional reference material you wish, but **no electronic aids** are allowed. Please note the following instructions. There will be a ten point deduction for failure to comply with them:

- start each problem on a new sheet of paper
- write your social security number, but not your name, on each sheet of paper you turn in
- show your work whenever appropriate. There can be no partial credit unless I see how answers were arrived
- be succinct. You may lose points for facts that, while true, are not relevant to the question at hand

You have until 12:20 to finish the exam. The questions are equally weighted.

1. File Systems

One of the advantages claimed for ReiserFS is that it can be more efficient than a conventional Unix-like file system such as Ext2, since files do not need to begin on block boundaries. Would it be possible to create a Unix-like file system that was very similar to Ext2, but allowed files to start at arbitrary locations (in particular, not necessarily block boundaries)? How would the i-node structure have to be changed to accommodate this modification? How would free space management have to be changed?

Yes. The main change that would be required would be the addition of a "byte offset" field in the i-node, telling where in the block a short file starts (there's already a field telling exactly how long the file is, so the amount of space it occupies in its last block can be derived). Free space management would need to be completely re-written using an algorithm and data structure appropriate for variable-length objects instead of the system used now.

2. Scheduling

One requirement for a "hard" real time system is bounded response: it must be possible to guarantee that, within some fixed time after a process becomes runnable, it does in fact run. Can a system using start-time fair queuing as its top-level scheduler meet this requirement? You can assume that the weights of all the process classes, and the length of a quantum, are known in advance. *Note: there are a variety of other factors, such as performance of device drivers and many others, which also affect whether a system can meet this requirement. I don't want you to consider the effects of any of these; only think about the scheduler algorithm.*

Yes, since SFQ breaks time into quanta and guarantees each class gets some time within each quantum.

The two most popular blind alleys were:

- (a) *Instead of thinking about the response criterion, thinking about whether the tasks meet their deadlines. As a matter of fact, this can actually be done too, but that isn't the question.*
- (b) *Instead of asking whether a bound exists based on weights and quanta, asking whether any arbitrary bound we care to set can be met. Of course the latter can't be satisfied; it can't be satisfied by any scheduler!*

3. Virtual Memory

Consider a computer system with a 64 bit virtual address, a 40 bit physical address, and a 64K page size.

- (a) Assuming an inverted page table (IPT), how many bits of each IPT entry are required for representing a virtual page number? How many for the "next entry" field?

Since the page size is 64K, it takes 16 bits of page offset. This leaves a 48 bit virtual page number, which is the necessary size of that field in the IPT entry. Since the physical address is 40 bits, there are potentially $2^{40-16} = 2^{24}$ physical pages, so the "Next" field has to be 24 bits.

- (b) Suppose you've got a very, very small hardware configuration with only 1 MB of physical memory. Use the picture on the page appended to this exam to show a possible HAT and IPT if the following virtual addresses can be accessed without page faults:

1234123412341234

49ac3472c324aad3

3419c0a8811100ac

798a6531bac497b2

0013aaf45231ac37

For your hash function, just extract the most appropriate address bits (same hash function as used by caches). Denote the end of a hash collision chain by leaving the `Next` field blank.

Solution is on the figure. Note that this solution is not unique: the important parts are that the three pages with a 4 in the fifth digit from the right all map to hash bin 4, and get chained together. Likewise, the two addresses with a 1 in that position both map to bin 1, and have a collision chain.

4. IPC

Suppose we had a CORBA implementation on some computer which did not support sockets (not either Windows or Unix, obviously!). Would it be possible to create an implementation of stream sockets on top of CORBA?

Not just possible, but easy. All we need is a "stream socket" server, which keeps track of all the stream sockets on the machine and who they're bound to, and implements a buffer for each one. If you wanted, you could even put Berkeley's socket API "on top of" this implementation.

