Homework 7 and Final Review
SOLUTION:
Solutions
Computer Systems Organization II, V22.0202.003 Spring 2005, Professor Yap

Out: Apr 26.
Due: Nothing.

SOLUTION:
SOLUTION PREPARED BY Instructor and T.A.s

INSTRUCTIONS:

- This last homework also serves as review for the final exam. We strongly recommend trying to work through these questions.
- Try to solve the problems yourself, before you read the solution which we will publish next week.
- We continue in the style of the midterm review: for each chapter, we tell you what to emphasize, what to deemphaize (or ignore).
- The information up to midterm is applicable for Chapters 1 to 4. No more than 20 % of the question will involve the midterm material. We shall skip Chapter 5 on Input/Output.

---

1 Chapter 6, File Systems

1. Read the chapter up to page 424 (end of Section 6.3.6). The organization of FAT files and I-Nodes files must be mastered. In Section 6.3.6, concentrate only on the File Consistency problem. Skip the rest of Chapter 6 except for Sections 6.4.5 (Unix file systems).

2. Describe the two main methods of implementing file systems: FAT and I-Node. What the relative advantages of each? Illustrate your discussion by considering how the two methods would require for a 60 GB disk.

   SOLUTION:
   Preliminary Remarks: file systems are all based on a subdivision of the disk into **disk blocks**. The obvious solution for files, then, is to use a linked list of such blocks. This solution does not permit fast random access. Hence the two solutions described in this question.

   The FAT solution is based on creating a "FAT" whose entries represent the individual disk blocks. Suppose there are N disk blocks. Then FAT would have N entries. We assume that the i-th entry corresponds to the i-th disk block, and there is a function $diskBlockAddr(i)$ that returns the address on the physical disk.

   If F is the FAT, then $F[i]$ is the i-th entry of the table. The entry is just points to another entry of F. For example, $F[3] = 9$ means that the 3rd disk block is followed by the 9th disk block. If $F[i] = 0$ it means that the i-th disk block is the end of file. In this way, F is just a representation of the linked list corresponding to the files.

   Each file has an entry in some file directory (or file folder). The entry contains the name of the file (ascii string) and other information about the file. One piece of information is the number of first disk block used to store the file data. For instance, if the first block of the file called "classlist.txt" is 3, then we can search the entire file by starting at disk block 3, then going to disk block $F[3]$, then to disk block $F[F[3]]$ until the disk block number is 0.

   Finally, it is important to realize that the FAT is resident in main memory.

   The I-Node Solution is in some sense a compromise between the naive linked list solution and the FAT solution. Here, each I-node stores a fixed size list of all the disk blocks used in the file. The disk blocks addresses can be directly represented, or it can be represented by the block indices (as in FAT files, in which case you need the $diskBlockAddr(i)$ function to translate the indices into address). If more disk blocks are needed, the last entry of the I-node points to another I-node. If each I-node stores k disk block addresses, this approach will give us a factor of k speedup over the linked list solution. (k is on the order of less than a thousand) This is good but is not enough for large files.

For large files, we need an important variation: some I-nodes can be used to store, not disk block addresses or indices, but a list of I-nodes. Such I-nodes acts like a directory, and may be called **indirect I-nodes**. The serve to reduce the number of I-nodes that must be read to get to actual disk blocks. As described in the text book, Unix V7 File System uses up to 3 levels of such indirect I-nodes.

In contrast to FAT files, we only need to keep the I-nodes of opened files in the main memory.

Relative Advantages of FAT vs I-Nodes: FAT can be large, and its size depends on the entire size of the disk, and does not dependent of how many files or disk blocks are actually used. I-Nodes are somewhat slower than FAT (see next 2 exercises).

Here are more exercises:

EXERCISE: In many text editors (e.g. Vi), you can jump to a certain **relative location** in the file. E.g., jump to 50% of the file means to jump to the middle of the file. Jump to 100% of the file means to go to the end of the file. Write the C-code for retrieving the disk block corresponding to any given relative location (given as a percentage) of a file. Assume the availability of the function $diskBlockAddr(i)$ and any similar functions you need.

EXERCISE: Compute the space needed for the FAT for a file system for a disk with 60 GB of memory, and disk blocks are 2K bytes. Assuming that computation is instantanteous for data in main memory, determine how much faster is the FAT solution than the I-node solution to access an arbitrary relative location in a 10 Megabyte file.

3. Assume that files are broken up into blocks. Let each block be 1 KB (1024 bytes), and each disk block number be 32 bits. Thus there can be up to $2^{32} = 4$ billion blocks. We maintain a linked list L of all the free blocks, using free blocks themselves as nodes of L. Assume that we hold only head node of L in main memory. Getting a new block B from this list, or releasing a block B to this free list, are done by the instructions: $B = GetBlock()$ and $FreeBlock(B)$. Write the algorithms for FreeBlock and GetBlock. Be sure that your algorithms avoid I/O thrashing (which could happen when there is alternating FreeBlock/GetBlock Requests).

SOLUTION:

Each block can hold $2^{10-2} = 256$ disk addresses. Assume that the first 4 bytes is a count of the number of block addresses that are stored in the current block, and the last 4 bytes points to the next block. Hence we only have 254 disk addresses.

Among the free blocks, some are "used" and some are "unused". Used free blocks are those that are nodes in a linked list which we call "FreeList". The block addresses of unused free blocks are stored in FreeList. We must explicitly say "B=Read(blockaddr, B)" where B is buffer space in main memory where the block is read into. Similary, we must say "Write(B, blockaddr)" to write it back to disk.

Let FreeList.head stores the first block of the FreeList. This block is always in main memory. We have another buffer space in main memory to store a block, called FreeList.tmpBlock.

Organization: If H is a node (i.e., block) of FreeList, then let $H[255]$ point to the address of next node in FreeList. If $H[255] = 0$, it means H is the last node of the FreeList. Also, let $H[0]$ be the count of the number of unused free block addresses stored in H. Thus $H[0]$ ranges from 0 to 254. Hence for $i = 1, 2, \ldots, H[0]$, $H[i]$ contain the block address of some unused free block.

```
blockaddr GetBlock()
H := FreeList.head.
If $H[0] > 0$, let $A = H[H[0]]$, decrement $H[0]$ and Return($A$).
Now, $H[0] = 0$.
```

```
Let $A = H[255]$ and Read($H[255]$, FreeList.head).
Return($A$)
```

```
FreeBlock(blockaddr $A$)
H := FreeList.head.
If $H[0] < 254$, increment $H[0]$ and let $H[H[0]] = A$. Return.
Now, $H[0] = 254$. – We will insert A as a new node in FreeList.
Let $B$=FreeList.tmpBlock and Read($A, B$).
Let $B[0] = 127$ and $H[0] = 127$.
Let $B[255] = H[255]$ and $H[255] = A$.
For $i = 1$ to 127, let $B[i] = H[i + 127]$.
Write($B, A$) and Return.
```

Note: a new node of FreeList is always half-full. Hence there are at least 127 GetBlocks or FreeBlocks between two disk access.

4. What are the File Consistency and Block Consistency problems? Describe the algorithms to check file and/block consistency, repairing if necessary.

   SOLUTION:

   See Lecture 13.

2 Chapter 7, Multimedia

1. Read Chap 7 up to page 475. Topics to focus on: basics of audio encoding, video encoding (including JPEG, MPEG), simple calculations of frequencies and bandwidths, rate monotonic scheduling and EDF scheduling.

2. What is the Nyquist Sampling Rate? Suppose we want to capture audio frequencies up to 20 KHz, and we sample at 16-bits, what is the bandwidth necessary to transmit such signals?

   SOLUTION:

   This is the sampling rate which is twice the frequency of the highest frequency component in a signal.

   For instance, in audio signals, the highest frequency that humans can hear 20 KHz. Hence we can assume this is the highest frequency in audio signals. The Nyquist sampling rate says we should sample such signals 40,000 times per second.

   The bandwidth at a 16-bit sampling resolution at 40,000 times per second is 640,000 bits/sec or 640 Kbaud.

3. Problem 5 on page 500 (Chapter 7). A CD holds 74 minutes of music or 650 MB of data. Estimate the compression factor for music. NOTE: audio CD Sampling rate is 44.1KHz at 16-bits.

   SOLUTION:
   44.1 KHz at 16-bits requires 88200 bytes per second. Multiply by 60×74 to get bytes per CD. But this amounts to 392 MB per CD. Since a CD can store 650 MB, we do not need any compression!

   I am not sure what has has gone wrong in this question from the book.

4. Problem 12 on Page 500 (Chapter 7). Two realtime processes are running. The first runs every 25 msec for 10 msec. The other runs every 40 msec for 15 msec. Will RMS always work?

   SOLUTION:

RMS guarantees its performance for m processes if $\sum_{i=1}^{m} C_i/P_i \leq m(2^{1/m} - 1)$. In our problem, we need to check the inequality: $(10/25) + (15/40) \leq 2(\sqrt{2} - 1) \sim 0.828$. This is true, so RMS always works.

Remark: to "always" work means that we do not care when the m processes choose to begin their periodic scheduling.

5. Problem 14 on Page 500. Figure out when the CPU will first go idle in an EDF Scheduling (see Figure 7-13).

   SOLUTION:

6. Outline the major steps of JPEG encoding.

   SOLUTION:

   Read p.464 (Section 7.3.1 of Text).

7. (a) Give scenarios where the uses of P- and B- frames in MPEG encoding would be result in substantial compression. (b) The MPEG Standard does not tell us how to encode: suggest algorithms to search for P- and B-frames during encoding.

   SOLUTION:

   (a) If we begin with a landscape frame and then a small figure (person, animal, car) starts to move across this landscape, then it is clear that P-frames would be very efficient capture most of the transitions!

   Conversely, suppose we begin with a close up shot of a person's face that covers the whole frame. Then, the face starts to shrink as we zoom out, introducing more and more of the landscape. The final frame is the landscape with just a tiny face. Here, the B-frames would be most effective.

   (b) There are many strategies. Here is a simple one. Let us suppose we arbitrarily choose every 10 frames to serve as an I-frame. For each of the remaining frames in between the I-frames, we try to determine if it should be a P- or B-frame. We started out with IX^9I where X is unknown (either P- or B-frame) and I is an I-frame. Thus X^9 is just a sequence of 9 unknown frame types. A brute force way to to try all 2^9 ways of assigning P or B to the 9 intermediate frames.

   For each choice, it is easy to determine what is the best way to encode each frame. For instance, suppose $X^9 = BPBPPBBBP$. Then, for each P frame, we can compare it to the previous P- or I-frame to search for the best macro block, etc. Similarly, we can compute the best encoding for each B-frames (since we know the previous or future P- or I-frame).

   Finally, we choose the best compression achieved by the 2^9 possibilities. If we replace 10 by 20, we would presumably achieve greater compression ratio, but it would be costlier to compute. In any case, it makes no difference to the decoding algorithm!

3 Chapter 8, Parallel Computation

1. Read Chapter 8 up to page 559 (end of Section 8.3.3). Some key concepts to learn are routing algorithms, network topology (see my lecture notes), and distributed shared memory (Section 8.2.5). I use the general term "parallel computer" to describe any system that has more than one processing unit (PE).

   SOLUTION:

2. Describe the book's classification of parallel computers. Describe Flynn's classification of parallel computers (this is not in the Textbook).
   SOLUTION:

3. Interprocessor communation can be done in one of two ways: using STORE/LOAD primitives or using SEND/RECEIVE primitives. Discuss the differences in these two paradigms, and when one is be more appropriate than the other.

   SOLUTION:

4. Draw the Omega-network for 8 processors. Describe the routing scheme is such networks, and show that it is correct.
   SOLUTION:

5. Describe the routing scheme for hypercube topology. Show that it is correct.

   SOLUTION:

6. In a general computer network, what are the two principle ways for communication?

   SOLUTION:

7. Describe the various network topology and discuss their pros and cons. Use concepts such as diameter, cut width, degree.

   SOLUTION: