FINAL with SOLUTION
Dec 18, 2006

Operating systems, V22.0202, Yap, Fall'06

1. PLEASE READ INSTRUCTIONS CAREFULLY. If you are not sure, please ask! *2. Answer all questions in both parts.* *3. This is a closed book exam, but you may refer to two $8'' \times 11''$ sheets of prepared notes.*

GENERAL COMMENTS ABOUT YOUR ANSWERS:

- Question 1 asked for "brief justifications" but many

PART 1. SHORT QUESTIONS. EACH QUESTION IS TO BE ANSWERED USING AT MOST THAN 3 SENTENCES. YOU RISK LOSING POINTS FOR LONGER ANSWERS. (10 Points Each)

1. Name three forms of interrupts, and describe each in a sentence.

   ANSWER:
   (a) Trap or system calls: software generated interrupt called by user software.
   (b) Exceptions: Generated by error.
   (c) I/O Interrupts: Generated by I/O devices.

2. True or False: Our STM (Simple Toy Machine) has relocatable addressing. Justify your answer. HINT: What is the size of a page table entry? Be sure to use all the data given.

   ANSWER:
   True as the STM uses a base register to compute the physical address. Each logical address is first checked to make sure that it does not exceed the value in the limit register. Then the value of the base register is added to the logical address to yield the physical address.

3. Give 3 advantages of dynamic libraries over static libraries.

   ANSWER:
   (1) Dynamic libraries load the necessary routines at run-time, and so unused routine are not even loaded.
   (2) Library routines can be shared with other processes at runtime.
   (3) The library routines are not directly into the compiled executable code, leading to smaller executable code.
   (4) Even when the dynamic library is recompiled, the user programs does not have to.

4. Consider a paging system with a 32-bit logical address space. Each address refers to a byte in memory. Let the page size be 16 KB, and main memory size be 256 MB. What is the minimal size (in bytes) of the page table? Please show working.

   ANSWER: 0.5MB.

   Each page has 2^{14} Bytes, and main memory has 2^{28} Bytes or $2^{28}/2^{14} = 2^{14}$ frames. Hence the frames have 14-bit addresses. Thus, each entry of the page table need at least 14 bits, or at least 2 bytes. Since logical address space has $2^{32}/2^{14} = 2^{18}$ pages, the page table has $2^{18} \times 2 = 2^{19}$ bytes, or $0.5MB$. REMARK: In practice, we probably want more than 2 bytes per entry: we may need space for the mbit, rbit, protection bits, present/absent bit, etc.

5. The Translation Look-Aside Buffer (TLB) is a hardware cache for storing entries from the page table, typically holding less than 128 entries. Which of the following is true, and justify your choice:
   (I) Using the TLB, memory access is sped up by an order of magnitude.
   (II) Using the TLB, memory access is sped up by at most a factor of two.

   ANSWER:
   (II) is true. Since the page table is normally stored in main memory, each page access requires two memory accesses: once to read an entry in the page table, and once to get the actual frame/page. Using a TLB, we only need one memory access, thus achieving a speed up of almost 2. If the page is not in memory, we still need two memory accesses. REMARK: The average the speed up depends on the cache hits ratio.

6. What is Belady's Anomaly? Name an algorithm that exhibits Belady's Anomaly. *ANSWER: It is the phenomenon where increasing the number of size of the page cache can lead to more page faults. The FIFO algorithm for page replacement exhibits Belady's Anomaly.*

7. Consider the following reference string of length 12:

   $$7\,0\,1\,2,\ 0\,3\,0\,4,\ 2\,3\,0\,3.$$

   For ease of reading, we have listed the page references in groups of 4. Suppose we have three frames (initially empty). How many page faults does this reference string incur using the algorithms indicated below? Be sure to show the contents of the frames during the execution.
   (a) Assume the optimal page-replacement algorithm (OPT).
   (b) Assume the LRU algorithm.

   ANSWER:
   (a) 7 faults. Here are the pages stored in frame cache:

   $$7, 70, 701, 201, -, 203, -, 243, -, -, 203, -$$

   NOTE: the last frame cache of 203 *may be replaced by* 043.
   (b) 9 faults.

   $$7, 70, 701, 201, -, 203, -, 403, 402, 432, 032, -$$

8. What is segmentation and name two advantages of using segmentation.

   ANSWER:
   Segmentation is a memory-management scheme to divide a user's logical space into distinct parts (called "segments"). Two advantages are (1) we can set protection bits (execute/write/read) for each segment independently, and (2) we can share segments across different processes.

9. True or False: The I-node contains the name of the file. Please give the rationale for your answer, i.e., why the name ought to be (or ought not to be) in the I-node.

   ANSWER:

   False. The reason the I-node does not store the file name is that several hard links might refer to this I-node and each of them has its own name. Instead file name is stored in the directories.

10. We have 4 hard drives, 2 CD-ROM drives and 1 printer. We can represent this information about resources by the vector $E = (4, 2, 1)$. We have processes P, Q, R whose maximum need for these resources are given by the table

    $$M = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 0 \\ 3 & 0 & 1 \end{bmatrix}$$

where row 1 corresponds to process P, row 2 corresponds to Q, and row 3 to R. E.g., Row 1 of M tells us that P never need more than 2 hard drives, 1 CD-ROM drive and 1 printer. Let the current allocation be

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

E.g., Row 2 of A says that Q is currently allocated one CD-ROM drive but nothing else.
(a) Show that allocation A is a safe state.
(b) From state A, if process R requests the printer, should we grant the request? Explain your answer.

ANSWER: Note that you MUST use the Marking Algorithm to justify safety of A (there is no other way we know).

(a) Use the Marking Algorithm to check if A is safe. The "need matrix" is thus

$$N = M - A = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 2 & 0 & 1 \end{bmatrix}.$$

Initially, we have the free vector $F = (2,0,1)$ and we could mark either rows 1 or 3. We mark row 1 and update F to $F + (1,1,0) = (3,1,1)$. Next we can mark either rows 2 or 3, but we choose to mark row 2 and F becomes $F = (3,1,1) + (0,1,0) = (3,2,1)$. Finally we can mark row 3 and F becomes $(4,2,1)$. Since all rows are marked, it is safe.

(b) Yes. Granting this still be safe. If we grant it, then the free vector becomes $F = (2,0,1) - (0,0,1) = (2,0,0)$. We can still mark rows 1 or 3 as before, and everything works as before.

PART 2. LONGER QUESTIONS.

1. Deadlock (40 Points)

   (a) Describe the abstract "code" for two simple processes (call them P, Q), each requiring exclusive access to two resources (call them R, S), such that deadlock can occur if we run P, Q simultaneously.

   (b) Describe an execution sequence of your code in part (a) such that a deadlock actually occurs.

   (c) Draw the sequence of resource graphs (Holt graphs) representing the state after each step of the execution sequence in part (b).

   (d) Modify your code in part (a) so that no deadlock occurs.

   (e) Generalize the technique in part (d) to any number of resources.

   (f) Prove that your answer in part (e) yields deadlock free solutions.

ANSWER:

(a) $P \equiv REPEAT(Request(R); Request(S); Action; Release(R); Release(S))$
$Q \equiv REPEAT(Request(S); Request(R); Action; Release(R); Release(S))$

NOTE: Students misunderstood this question, and gave real world examples of R = printers and S = tapedrive, etc.

(b) $P.Req(R), Q.Req(S), P.Req(S), Q.Res(R)$.

(c) This is shown in Figure 1. The final Holt graph has a cycle.

(d) We just change Q to
$Q \equiv Loop(Req(R); Req(S); Action; Rel(R); Rel(S))$

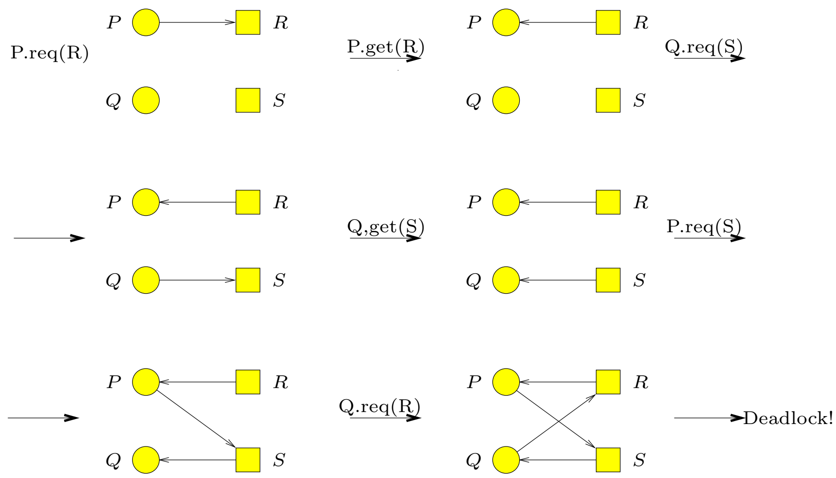

Figure 1: Holt Graphs: execution sequence leading to deadlock.

NOTE: Many students use synchronization semaphores, or "release of resources". These are OK, but you need to make sure that your solution do not introduce other bad features such as starvation (e.g., the processes may repeatedly get and release resources without making progress), or unfairness (a slower process might be losing out to a faster process).

(e) Suppose there are $R_1, \ldots, R_m$ resources (one copy of each), and any process that requires two resources R_i, R_j must request R_i before R_j for $i < j$. Of course, a process may require any subset of the m resources.

(f) Proof of Deadlock freeness: Among the processes, let P be the one that has been granted R_i where i is the largest index among the allocated resources. Then we claim that P is cannot be waiting on any other process. Thus P can run to completion. Thus, there is no deadlock.

2. Unix Files (20 Points)

Write the high level algorithm called `SeekBlock` for locating the data block which contains a given relative address within the unix file. More precisely, assume function is called `SeekBlock(ino, off)` where `ino` is the pointer to the file's i-node, and `off` is the 32-bit address of a byte within the file. Assume that the block size is 4K, and you must outline all the necessary calculations. Hint: there are three cases, depending on how large the offset is.

Make explicit any assumptions you need about how i-nodes and other structures of the file system are organized. For instance, you may assume the 15 pointers of an i-node `ino` can be accessed via `ino.Pointers[i]` where `i` is between 0 and 14. Also, if `ptr` points to a data block, then `GetBlock(ptr)` will return the data block.

ANSWER:

`SeekBlock (ino, off):`

Step 0. Let `B = (off / 4K)`. This gets the block number. Remark: `K`=1024 ino these calculations.

Step 1. If `(B<12)`, `Return(ino.Pointers[B])`.

Step 2. Else if `(B-12) < 1K`, then let `SingleIndirect = GetBlock(ino.Pointers[12])`. We view `SingleIndirect` as an array with 1K pointers, and `SingleIndirect[i]` will return the i-th pointer in the array (for i=0,1,...,1023).

`Return(SingleIndirect[B-12])`.

Step 3. Else we know `(B-12) >= 1K`.

Let `SingleIndirect = GetBlock(ino.Pointers[13])`

and `DoubleIndirect = GetBlock(SingleIndirect[(B-12) / K])`.

`Return(DoubleIndirect[(B-12) % K])`.

3. (This question was deleted to shorten the exam :-) The following stm program prints the message "Hello" and then terminates. It is based on a slightly nhanced assembler (`assem.pl`).

```
##############################################################
       PRINT:
LOA R15, TWO         # TRP 2 for output
LOA R14, FIVE        # R14=5 indicates string of length 5
LOA R13, MESSAGE     # R13 is address of output string
TRP                # Print it!
       HALT:
LOA R15, ZERO        # TRP 0 to halt
TRP                # Halt!
       DATA:
ZERO: CON 0          # integer constant 0
TWO: CON 2           # integer constant 2
FIVE: CON 5          # integer constant 5
MESSAGE: CON "Hello" # string constant "Hello"
##############################################################
```

Specifically, our assembler allows the following:
(1) We can define a label by writing its identifier with a colon (E.g., `TWO:`).
(2) We can refer to labels in the argument of instructions (E.g., `LOA R15, TWO`, where `TWO` is a label).
(3) Load integer or string constants using the key word `CON` (E.g., `CON 2` to load the constant 2 in the current location, or `CON "Hello"` to load the string constant "Hello" in the next five consecutive locations).

Please write a stm program to prompt the user for an input integer, prints the double of the integer, then halt.

ANSWER:

```
##############################################################
 PROMPT:
LOA R15, TWO         # TRP 2 for output
LOA R14, EIGHTEEN    # R14=18 indicates string of length 18
LOA R13, MESSAGE     # R13 is address of output string
TRP             # Print it!
 READ:
LOA R15, ONE         # TRP 1 for input
LOA R14, ZERO        # R14=0 indicates integer input
TRP             # Read input into R13
 DOUBLE:
CPR R1, R13          # Copy R13 into R1
ADD R1, R13, R2      # Now R2 has double of the input integer
 OUTPUT:
LOA R15, TWO         # TRP 2 for output
LOA R14, ZERO        # R14=0 indicates integer output
LOA R13, R2          # R13 has the doubled input
TRP             # Print it!
 HALT:
```

```
CPR R13, R2          # Copy R2 to R13
TRP             # Halt!
 DATA:
TWO: CON 2           # integer constant 2
MESSAGE:
CON "Input an integer: "
EIGHTEEN: CON 18     # integer constant 18
ZERO: CON 0          # integer constant 0
ONE: CON 1           # integer constant 1
##############################################################
```