FINAL with SOLUTION
May 10, 2005

Comp.Sys.Org.II, V22.0202.003, Yap, Spring'05

1. PLEASE READ INSTRUCTIONS CAREFULLY. *2. Answer all questions in the space provided.* *3. Use the REVERSE SIDE of exam paper for scratch work.* *4. 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*

QUESTION 1. SHORT QUESTIONS. (10 Points each)

1. Explain external and internal fragmentation in memory management: how they arise and how they are solved in practice.

   ANSWER:

   External fragmentation happens when the spaces between allocated chunks of memory are too small to be used by any process. Internal fragmentation happens when we allocate space in discrete units – the unused space in the allocated memory are said to be internal fragments. To solve external fragmentation, we can run "defragmentation" algorithms which makes all allocated blocks contiguous. A partial solution for internal fragmentation is to choose page sizes that are relatively small (like 1 or 2K). Then internal fragmentation is not serious, but we gain many other advantages.

2. Describe 3 uses of segmentation of processes.

   ANSWER:

   (1) Sharing of code segments,
   (2) Management of multiple address spaces for independent datastructures that can grow and shrink unpredictably,
   (3) Improved protection control for data and code.
   (4) Paging can be improved too.

3. List all the similarities and differences between the following two operating system concepts: paging and FAT.

   ANSWER:

   (1) Both map pages in disk to pages in main memory.
   (2) Both uses a table (page table and FAT)
   (3) Paging relates pages to processes, but FAT relates pages to files.

4. Recall the R-bit and M-bit for paging in virtual memory. How are these two bits used in the Not Recently Used (NRU) Policy?

   ANSWER:

   Read (R, M) as a binary number from 0 to 3, called its "category". We choose the page of the lowest category for eviction. E.g., $R = 1$ and $M = 0$ gives us category 2, and such a page would not be evicted if there is a page of category 0.

5. Describe the two sources of coding error when we convert a continuous signal (like sound or images) into a discrete representation (using zeros and ones)?

   ANSWER:

   Quantization error and sampling error.

6. Suppose we want to transmit human voice over the telephone using 8-bits per sample. The frequency range for human voice is 300 to 3400 Hz. What bandwidth is sufficient to transmit such data?

   ANSWER: Nyquist sampling rate requires only 6800 samples per second. Thus a bandwidth of 54400 bits per second is enough. (NOTE: 56 Kbits per second is standard for telephone modems.)

7. The basic paradigms for interprocessor communication are represented by the following two sets of primitives: LOAD/STORE and SEND/RECEIVE. Discuss their differences and describe situations where one paradigm is more appropriate than the other.

   ANSWER:

   (1) LOAD/STORE (L/S) is used in shared memory systems. SEND/RECEIVE (S/R) is used in distributed memory systems.
   (2) L/S involves variables (memory) while S/R involves messages.
   (3) The former is suited for tightly coupled systems, the latter for loosely coupled systems.
   (4) The main synchronization for L/S is mutex. The synchronization for S/R are various handshaking protocols.

8. Suppose that an I-Node can store the addresses of 254 disk blocks. We can distinguish two kinds of I-Nodes: (A) The first 253 addresses refer to disk blocks which storing file data, and the last address points to the next I-Node. (B) All 254 addresses to point to I-Nodes. Some systems only use Type (A) I-Nodes, but it may be advantages to also use Type (B) I-Nodes. Illustrate their differences using a 1 million block file as example.

   ANSWER:

   Type (A) nodes allows sequential linear-time access of any disk block. E.g. with one million blocks, we need $2^{20-8} = 4K$ I-Nodes to represent the file. We have to read $4K$ such I-Nodes to get to the end of file. Type (B) Nodes allows us to rapidly logarithmic-time access to any disk block. With 1 million blocks, you need to read at most $\lceil \lg(10^{20})/\lg(254) \rceil = \lceil 20/8 \rceil = 3$ disk blocks to get to any disk block.

9. In a 32 processor hypercube, consider the processor whose address is (10011). List the addresses of its neighbors.

   ANSWER:

   $(00011), (11011), (10111), (10001)(10010)$.

10. Consider the following three periodic tasks, with the period P_i and computation times C_i shown (assume the time unit is seconds):

	Period P_i	Computing Time C_i	Phasing
Task A	100	20	50
Task B	150	30	0
Task C	200	100	30

   The third column (Phasing) tells us when each of these tasks are to begin running (relative to some absolute time 0). Show how the above tasks are scheduled using Rate Monotonic Scheduling.

   ANSWER:

   Draw a time line....

   B[0..30], C[30..50], A[50..70], C[70..150], A[150..170], etc...

11. Refering to the same tasks A, B, C in the previous question. Suppose the phasing information (column 3) is unknown. Can we still schedule these 3 tasks? Explain. HINT: This question is subtle! Note that $3(2^{1/3} - 1) = 0.779$.

    ANSWER:

Let $U = (20/100) + (30/150) + (100/200) = 9/10$. Since $U > 3(3^{1/3} - 1)$, the theorem of Liu/Layland DOES NOT guarantee that these 3 tasks can be scheduled. But it does not mean that the tasks CANNOT be scheduled.

Indeed, by a direct argument, we can prove that they can be scheduled: first show that TASK C can will always meet its deadline – that is because in a period of C, there are most 40 seconds allocated to TASK A and 60 seconds for TASK B. So TASK C can be scheduled. Likewise, it any period of TASK B, there are at most 40 seconds for TASK A, so TASK B can be scheduled.

NOTE: this example shows that the Liu/Layland result is not tight.

QUESTION 2. NETWORK ROUTING (45 Points)

1. Draw the Omega network for 8 processsors. Label the processors from $0 = (000)_2$ to $7 = (111)_2$. Similarly, label the 2×2 switches in each stage from $0 = (00)_2$ to $3 = (11)_2$. Here, we write $(b_1 b_2 \cdots b_n)_2$ ($b_i \in \{0, 1\}$) for the binary representation of an n-bit number.

   ANSWER:

   FIGURE HERE.

   There are 8 input processors in the left column, and 8 output processors in the right column. In between we have 3 columns of switches, each column with 4 switches. The connection between successive layers is the shuffle exchange network (next question).

2. Let $n \geq 1$ be an integer. Describe (in words) the interconnection between two successive layers of an Omega network with 2^n processors. HINT: what are the "addresses" of the two input (and two output) lines of switch number $(b_1 b_2 \cdots b_{n-1})_2$?

   ANSWER:

   Basically, if a wire has address $(b_1 b_2 \cdots b_n)$ then it is connected to switch number $(b_2 b_3 \cdots b_n)$, using the lower input line if $b_1 = 1$ and upper input line if $b_1 = 0$. Thus address $(b_1 b_2 \cdots b_n)$ is connected to address $(b_2 b_3 \cdots b_n b_1)$ (circular left-shift). You can draw a picture to help.

3. Describe the routing algorithm for Omega network. Say the source address is $(s_1 s_2 \cdots s_n)_2$ and the destination address is $(d_1 d_2 \cdots d_n)_2$.

   ANSWER:

   Starting in position $(s_1 \cdots s_n)$, at each step, we follow the wire to the next switch. This moves us (by previous question) to position $(s_2 \cdots s_n s_1)$. This position is an input to switch number $(s_2 \cdots s_n)$. Then we use the first bit (d_1) of the destination to operate the switch – this will move us to position $(s_2 \cdots s_n d_1)$.

   In general, at the the beginning of the ith stage, we will be at some input line of switch $(s_{i+1} \cdots s_n d_1 \cdots d_{i-1})$. Then, using the bit d_i to operate the switch, we will find ourselves at position $(s_{i+1} \cdots s_n d_1 \cdots d_{i-1} d_i)$.

4. Illustrate your routing algorithm in the previous question by showing how to route from processor No. 23 to processor No. 10 in a 32 processor Ω-network. Note: do not to draw the network. Just describe the 2×2 switches that the route passes through in each stage (these switches are numbered from $0 = (0000)_2$ to $15 = (1111)_2$).

   ANSWER:

   Processor 23 has address $(s_1 s_2 \cdots s_5) = (10111)$ and processor 10 has address $(d_1 d_2 \cdots d_5) = (01010)$.

   Stage 1: (10111) is connected to switch (0111) and since the first bit d_1 of the destination is 0, we are switched to (01110).

   Stage 2: (01110) is connected to switch (1110) and since $d_2 = 1$, we are switched to (11101).

   etc.

QUESTION 3. TEST AND SET LOCK (20 Points)

1. The textbook gives the following code for entering and leaving a critical region:

```
enter_region:
      TSL REGISTER,LOCK          # TSL operation
      COM REGISTER, #0           # was LOCK zero?
      JNE enter_region           # if nonzero, loop
      RET                        # enter region
leave_region:
      LOAD LOCK, #0              # TSL operation
      RET                        # leave region
```

Prove that this code is correct.

ANSWER:

A process enters the critical region only if REGISTER is 0, i.e., the LOCK value was 0 when we executed TSL. Since TSL set the LOCK to 1, we can be sure that our process is the UNIQUE process to change the LOCK variable to 1. So when we enter the critical enter under this condition, we can be sure that no other process are in the critical region.

2. Suppose, instead of TSL, we have a new instruction `SWAP RX, LOCK` which exchange the values of the register `RX` and the variable `LOCK`. This exchange is an indivisible step. Give the code using `SWAP` corresponding to the previous question.

   ANSWER:

   To simulate TSL, we do the following:

   enter_region:
   LOAD REGISTER, #1 # load a non-zero into RX
   SWAP REGISTER, LOCK
   COM REGISTER, #0 # was LOCK zero?
   JNE enter_region # if nonzero, loop
   RET # enter region

3. Prove that your solution is correct.

   ANSWER: There is exactly ONE process whose SWAP action causes the transition of LOCK value from 0 to 1. Because of the indivisibility of SWAP, the process whose action causes this WILL SEE THIS. NO OTHER PROCESSES WILL SEE THIS. Hence, if we see this transition, we can proceed to enter the critical region, with guaranteed that no one else will.

4. What can go wrong in our enter_region code when used in a multi-processor computer which shares global memory over a bus? Give a simple fix to this problem, just by modifying the semantics of TSL.

   ANSWER:

   Two processors may think that they have access to the critical region if they both read the LOCK variables before either has a chance to change it. To fix this problem, we can make TSL instruction lock the bus for exclusive use before reading/writing into LOCK.

5. Suppose our multi-processor computer of the previous question uses local cache. What new issues arise because of our modified semantics of TSL? Outline a solution.

   ANSWER:

   Since TSL causes a write, this will invalidate the cache block that holds the LOCK variable for ALL the other processors! In fact, the loop which repeated test the LOCK variable to see if it is equal to 0 should not be done using TSL. We should simply use an ordinary READ instruction. Only when the variable is found to be 0 should we attempt a TSL. Of course, we could fail at the TSL, but at least we know that some process had managed to enter the critical region.

   REMARK: this is a reasonably good solution because the number of times each processor has to fetch the cache block is AT MOST equal to the number of times ANY processor enters the critical region.