

INDIANA UNIVERSITY
Computer Science Department

Qualifying Exam
Sample Questions

7/8/2002

This is a sample systems qualifying exam for the Computer Science Department at Indiana University.

This is only a sample.

Had this been a real exam, you would be given the following instructions.

This exam contains 9 short-answer questions, of which you may choose any 8 to answer. Indicate on the first page of your exam booklet which question you are not answering. If you do not indicate a question, one will be chosen for you at random. You will not receive extra credit for answering all 9 questions.

1. Consider a microprocessor MMU with the following design parameters.

Page size: 4096 bytes

Virtual address: 40 bits

Physical address: 40 bits

TLB: Two-way set associative with 256 sets

A fully associative TLB stores and retrieves page table entries using the page number as a key. Recall, however, that a TLB is simply a form of cache. A less expensive kind of cache than fully-associative is set-associative. With a set-associative TLB, each page-table entry is mapped to a certain set of TLB locations.

A two-way set associative TLB with 256 sets has two cache locations in each set. Page table entry b is mapped to set " $b \bmod 256$ " and may be stored in either of the two locations in that set with the page number (without the lower eight bits) as a tag. To determine whether block b is in the TLB, set " $b \bmod 256$ " is searched associatively for the tag. (In a fully associative TLB, all entries must be searched.)

- (a) How many entries are needed for the page table?

(b) Suppose a program has three arrays beginning at the following addresses:

```
x: 0x0012300000  
y: 0x0012400000  
z: 0x0012500000
```

Explain what will happen in the MMU during execution of an arbitrary iteration of the following loop:

```
for (i = 0; i < 0x100000; ++i)  
    x[i] = y[i] * z[i];
```

(c) Which of the following sets of array addresses would *not* exhibit the problem of part 1b? Circle all answers that apply.

i. x: 0x0012300001
y: 0x0012400000
z: 0x0012500000

ii. x: 0x0012400000
y: 0x0012400000
z: 0x0012500000

iii. x: 0x0012200000
y: 0x0012300000
z: 0x0012500000

iv. x: 0x0012300000
y: 0x0012400800
z: 0x0012500000

v. x: 0x0012300000
y: 0x0012400000
z: 0x0012501000

vi. x: 0x0112300000
y: 0x0012400000
z: 0x0012500000

2. Consider the following program:

```
int a[128][128];

int main() {

    for(int i = 0; i != 128; ++i)
        for(int j = 0; j != 128; ++j)
            a[i][j] = 0;

    return 0;
}
```

Assume the following about the virtual memory system of the computer executing this program:

- the page size is 256 words,
- an int occupies one word,
- the virtual memory system uses demand loading and LRU page replacement,
- the code and stack are in page 0,
- the first word of array a is stored in the first word of page 1, and
- three page frames are allocated to this process.

The definition of C++ states that arrays are stored in row-major order; this means, for example, that the first three words on page 1 contain $a[0][0]$, $a[0][1]$, and $a[0][2]$.

(a) How many page faults occur while running the above program?

(b) How many page faults occur while running a variant of the above program in which $a[i][j]$ is changed to $a[j][i]$?

4. Design a system in which threads $T_0; T_1; T_2; \dots; T_N$ take turns performing actions. Specifically, the sequence of actions that should occur is $a_0; a_1; \dots; a_N; a_0; a_1; \dots; a_N; \dots$. Thread T_i is responsible for performing action a_i . The code for T_i is:

```
while (TRUE) {  
    WaitForTurn(i);  
    ai;  
    FinishedTurn(i);  
}
```

Give code for WaitForTurn(i) and Finished(i) and declarations (and initializations, if necessary) for your global variables. Your code should use semaphores. Do not use other synchronization constructs (locks and condition variables, TSL, etc.). Do not use busy-waiting.

5. A group of operating system designers for the Itty Bitty Machine company are thinking about ways to reduce the amount of backing store needed in their new operating system. The head guru has suggested not bothering to save the program text in the swap area at all, but just page it in directly from the binary file whenever it is needed.

(a) Under what conditions, if any, does this idea work for the program text?

(b) Under what conditions, if any, would this idea work for the program data?

6. Consider a "standard" UNIX filesystem (with i-nodes, indirect blocks, etc.). Assume that the file block cache is initially empty, and that the i-node cache initially contains the current working directory. Assume "bar" exists in the current working directory.

Suppose a program executes the following sequence of instructions:

```
fd = Open("bar");           // open the file
Seek(fd,0);                 // seek to beginning of file
Write(fd,"A",1);           // write one byte
Close(fd);                  // close the file
```

Reading or writing one block or i-node requires 1 disk access.

- (a) What is the *minimum* number of disk accesses that this sequence of instructions could possibly require? If it is not possible to give an exact value, explain why and indicate what additional information is needed.

- (b) What is the *maximum* number of disk accesses that this sequence of instructions could possibly require? If it is not possible to give an exact value, explain why and indicate what additional information is needed.

8. Assume an operating system with preemptive scheduling, and assume that a timer interrupt may occur between any two machine instructions. Initially, address x contains 2, and address y contains 10. Assembly language pseudo-code is given below for two threads, T_1 and T_2 , which execute concurrently.

```
T1: Load x, r1 // load contents of address x into register r1
     Load y, r2 // load contents of address y into register r2
     Add r2, r1 // r1 := r1+r2
     Store r1, x // store contents of register r1 into address x
```

```
T2: Load x, r1 // load contents of address x into register r1
     Load y, r2 // load contents of address y into register r2
     Sub r2, r1 // r1 := r1-r2
     Store r1, y // store contents of register r1 into address y
```

What are (all) the possible pairs of final values of x and y , i.e., the values after both threads have finished? Just to clarify the English, the possible pairs of values might be something like: ($x=1$ and $y=2$) or ($x=3$ and $y=4$).

9. Write pseudo-code for translating virtual addresses to physical addresses using 2-level page tables. Assume the following constants, types, and global variables are defined.

```
#define PgSz    ...           // page size in bytes
#define VPN1    ...           // number of entries in L1 page table
#define VPN2    ...           // number of entries in L2 page table.

class L2Entry {
public:
    int PFN;                // page frame number
    bool valid;             // valid bit ...
                            // the other members are irrelevant
};

// a level-2 page table is an array of L2Entry's.
typedef L2Entry L2PgTbl[VPN2];

// pgtbl1 is a global variable pointing to the level-1 page table.
// If pgtbl1[i]!=NULL, then the i'th level2 page table isn't present in memory
L2PgTbl* pgtbl1[VPN1];

enum ExceptionType { NoException, PageFaultException, PageFaultException2,
                    AddressErrorException };
```

Give pseudo-code for the following procedure

```
ExceptionType Translate(VirtualAddress v, PhysicalAddress* p)
```

This procedure normally stores the translated physical address in *p and returns NoException. Translate() should return:

- PageFaultException if the necessary page isn't in memory
- PageFaultException2 if the necessary level-2 page table isn't in memory
- AddressErrorException if the virtual address is invalid.

You can introduce other exception types, if you think they are needed.