

# CS 2200 Spring 2012 Final Exam 8 AM to 10 AM

Name: \_\_\_\_\_ Kishore \_\_\_\_\_ GT Number: \_\_\_\_\_

Problem	Points	Lost	Gained	Running Total	TA
1	1				
2	10				
3	14				
4	15				
5	15				
6	10				
7	15				
8	10				
9	10				
Total	100				

You may ask for clarification but you are ultimately responsible for the answer you write on the paper. If you make any assumptions state them.

Please look through the entire test before starting. WE MEAN IT!!!

Illegible answers are wrong answers.

Show your work in the space provided to get any credit for problem-oriented questions.

Good luck!

1. (1 point, 1 min)

The newly named chair of the School of Computer Science:

- (a) Annie Anton
- (b) Lance Fortnow
- (c) Zvi Galil
- (d) Ellen Zegura
- (e) Charles Isbell
- (e) George Burdell

# CS 2200 Spring 2012 Final Exam 8 AM to 10 AM

Name: \_\_\_\_\_ Kishore \_\_\_\_\_ GT Number: \_\_\_\_\_

## Memory management

2. (10 points, 10 min)

a) (4 points)

The number of virtual pages is defined by the ratio:

Size of virtual address space / Page size.

The number of physical frames is defined by the ratio:

Size of physical memory / Page size.

The page table is set up by the Operating system.

The page table is looked up on every memory access by the hardware.

b) (2 points) (circle the right choice) Given 32-bit virtual address; 28-bit physical address; 4 KB page size; 8 processes currently executing, the number of page tables in memory is:

1. 4
2.  $2^{20}$
3. 8
4. 28
5. 12
6.  $2^{16}$
7. 32

c) (2 points) One scheme to implement a TRUE LRU policy for page replacement by the OS is the following:

- Have a hardware stack wherein each entry of the stack holds the page frame number accessed by the CPU; CPU updates the stack on each memory access
- Have an instruction in the ISA for the OS to read the LRU page frame from the stack

Why is this idea infeasible to implement?

*All or nothing*

Size of the hardware stack is as big as the number of physical frames...too big to have in hardware.

d) (2 points) How is the reference bit per page table entry used in implementing an approximate LRU using the second chance page replacement policy (limit to 3 or 4 concise bullets please)?

- <sup>+1</sup> Hardware sets the reference bit in the associated PTE during every memory access as part of address translation
- Page replacement policy of the OS is basically FIFO with the following change:
  - if the candidate victim's reference bit is set then reset the bit and move to the next candidate;
  - continue until you find a page whose reference bit is not set;
  - choose that page as the victim for page replacement

# CS 2200 Spring 2012 Final Exam 8 AM to 10 AM

Name: \_\_\_\_\_ Kishore \_\_\_\_\_ GT Number: \_\_\_\_\_

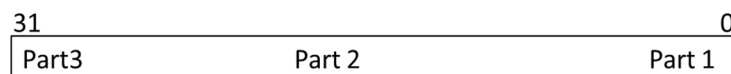
## Cache

3. (14 points, 15 min)

a) (6 points)

A computer has the following characteristics:

- It is **byte-addressed** and the address is 32 bits long (little endian)
- It has a **4-way set-associative cache** with each data block holding 64 bytes of data
- The cache has a total of **16 K data blocks**



*All or nothing*

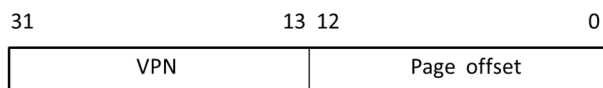
The above figure shows how the address is divided into three parts to perform a cache access. Name the parts (below) and the number of bits associated with each part.

Part 1:	<u>byte offset</u>	<u>6</u>
Part 2:	<u>index</u>	<u>12</u>
Part 3:	<u>tag</u>	<u>14</u>

b) (3 points)(fill in using the words, "capacity", "compulsory", "conflict") Consider a 2-way set associative cache.

- 1) CPU accesses memory location X for the **first time**. This is a compulsory miss.
- 2) The cache is **not full**. The CPU has made a number of accesses including to locations X, Y, Z, P, Q, and R in that order. CPU now makes a reference to location X again. If X is **not** in the cache and has to be brought from memory, then this is a conflict miss.
- 3) At some later point in time the **cache is full**. The CPU makes a reference to location X again. If X is **not** in the cache and has to be brought from memory, then this is a capacity miss.

c) (5 points) Consider a **virtually indexed physically tagged** cache. The virtual address is 32 bits. The page size is 8K bytes.



The cache is **direct mapped** with 512 KB and a **block size** of 64 bytes. How many low order bits of the VPN should remain unchanged in the translation process for the above cache to work correctly?

Number of data blocks in the cache = 512 KB/64 bytes = 8 K

Number of index bits needed to address the direct mapped cache

$$= \log_2(\text{cache-size-in-data blocks}) = \log_2(8K) = 13 \text{ bits} \quad (1)$$

Number of bits in virtual address unchanged during translation

$$= \log_2(\text{page size}) = \log_2(8K) = 13 \text{ bits} \quad (2)$$

$$\text{Byte offset in each data block} = \log_2(\text{block-size}) = \log_2(64) = 6 \text{ bits} \quad (3)$$

Number of unchanged bits available to serve as cache index from the virtual address = bits in page offset - bits in byte offset

$$= (2) - (3) = 7 \text{ bits} \quad (4)$$

So additional number of bits needed from the low order bits of virtual address to serve as index to the cache = (1) - (4) = 13 - 7 = 6 bits

*Byte offset missed -2*

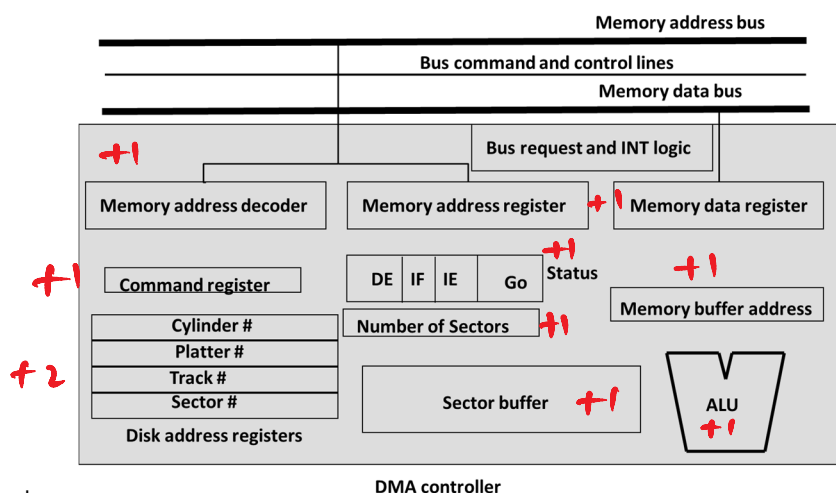
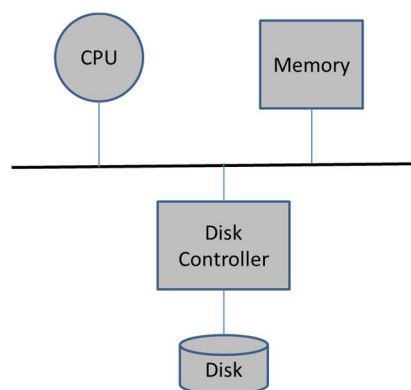
# CS 2200 Spring 2012 Final Exam 8 AM to 10 AM

Name: \_\_\_\_\_ Kishore \_\_\_\_\_ GT Number: \_\_\_\_\_

## Input/Output and Disk

4. (15 points, 20 min)

- a) (10 points) Design a DMA controller for a disk drive. You have to design only the way the controller interfaces with the CPU and the memory bus. You don't have to worry about the device electronics side of the controller. Recall that a disk address is a tuple: {cylinder#, platter#, track#, sector#}. Clearly show the registers in the controller for interfacing to the CPU and the memory, describe succinctly their purpose, and how the CPU communicates with the controller and vice versa, and how the data transfer is effected by the controller.



Datapath description:

- Memory mapped registers in the controller for interfacing with the CPU: disk address registers (cylinder#, platter#, track#, sector#), command register, number of sectors, memory buffer address and status
- CPU can read/write the controller registers using simple L/S instr.
- Memory address decoder monitors the address bus to see if the L/S from the CPU is addressed to one of the controller registers
- Bus request and INT logic allows controller to compete for memory bus cycles for completing the DMA request from the CPU
- Memory address register and memory data register allow the controller to interface with memory
- ALU is for doing address arithmetic and for decrementing the count for data transfer

Commands (operations supported by controller via commands issued by writing to the Command Register):

- Seek (cylinder#), Read (disk address, number of sectors, memory buffer address), Write (disk address, number of sectors, memory buffer address)

Status register: IE - INT enable set/reset by CPU; IF - INT pending from controller; Go - set by CPU to get the controller started on the command issued by the CPU by writing to the command register

# CS 2200 Spring 2012 Final Exam 8 AM to 10 AM

Name: \_\_\_\_\_ Kishore \_\_\_\_\_ GT Number: \_\_\_\_\_

b) (5 points) Given the following specifications for a disk drive:

512 bytes per sector  
400 sectors per track  
6000 tracks per surface  
3 platters  
Rotational speed 15000 RPM

What is the transfer rate of the disk?

Time for one revolution =  $60/15000$  secs **+1** (1)

The disk can read and transfer one track worth of data in every revolution

Amount of data transferred in one revolution

= number sectors in one track \* number of bytes per sector

=  $400 * 512$  bytes **+2** (2)

Transfer rate of the disk

= Amount of data transferred/time for one revolution

=  $(2)/(1) = (400 * 512)/(60/15000)$  bytes/sec

=  $400 * 512 * 15000 / 60 = 400 * 512 * 250$  bytes/sec **+2**

## File Systems

5. (15 points, 15 min)

a) (6 points)

Notes:

- Unix "**touch** <file>" command creates a zero byte new file or updates the timestamp of the named file
- Unix "**ln** <file1> <file2>" command creates a hard link
- Unix "**ln -s** <file1> <file2>" command creates a sym link
- Unix "**rm** <file>" removes the named file

In the following table, assume **none of the files exist to start with** in the current directory. Fill in the table. The reference count in the table pertains to the i-node that is affected by the command in that row. If a new i-node is created, show the old reference count for that i-node as 0.

Command	New i-node created (yes/no)	Reference count	
		old	new
touch f1	Yes	0	1
ln f1 f2	No	1	2
ln f2 f3	No	2	3
rm f1	No	3	2
ln -s f2 f4	Yes	0	1
ln f4 f5	No	1	2

Use this area for rough work for this question.

**+ for each line correct**  
**- 0.5 for each entry wrong**  
**no double jeopardy**

# CS 2200 Spring 2012 Final Exam 8 AM to 10 AM

Name: \_\_\_\_\_ Kishore \_\_\_\_\_ GT Number: \_\_\_\_\_

b) (9 points)

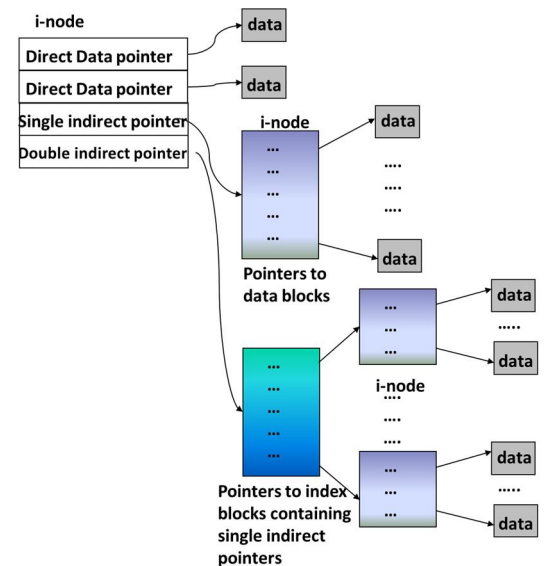
Using C like syntax, write the i-node data structure. Note that an i-node may represent a directory, a data file, or a symbolic link. If it represents a data file it has the structure shown in the figure: it could be a top level i-node, or an intermediate index block as shown in the figure.

```
#define NUM_PTRS_IN_INODE 128
#define NUM_DIR_ENTRIES 99
typedef enum {dir, sym-link, data, first-level-ind,
second-level-ind} i_node_type;
```

```
typedef struct disk_address_t {
    int cylinder_num;
    int platter_num;
    int track_num;
    int sector_num;
} disk_address;
```

```
typedef struct disk_address_t *disk_address_ptr;
typedef struct i_node_t *i_node_ptr;
```

```
typedef struct i_node_t {
    i_node_type type;
    union {
        struct dir_t { /* directory i-node */
            char *name;
            int refcnt;
            char *entries[NUM_DIR_ENTRIES];
        } dir;
        struct sym_link_t { /* sym_link entry */
            char *name;
            int refcnt;
            char *path;
        } sym_link;
        struct data_t { /* data entry */
            int refcnt;
            disk_address_ptr direct_block_ptr1;
            disk_address_ptr direct_block_ptr2;
            i_node_ptr first_level_inode_ptr;
            i_node_ptr second_level_inode_ptr;
        } data;
        struct first_level_ind_t { /* first-level-ind */
            disk_address_ptr data_block_ptrs[NUM_DATA_PTRS_IN_INODE];
        } first_level_ind;
        struct second_level_ind_t { /* second-level-ind */
            i_node_ptr first_level_inode_ptrs[NUM_DATA_PTRS_IN_INODE];
        } second_level_ind;
    } i_node_details;
} i_node;
```



+2 for completeness

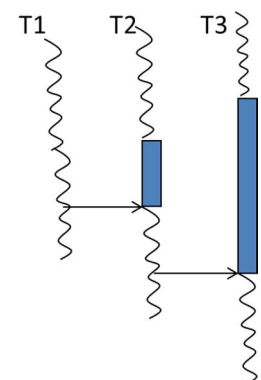
# CS 2200 Spring 2012 Final Exam 8 AM to 10 AM

Name: \_\_\_\_\_ Kishore \_\_\_\_\_ GT Number: \_\_\_\_\_

## Parallel Systems

6. (10 points, 10 min)

`X = 0;`



The Shared variable X is initialized to 0;

- T3 is waiting for X to become 2
- T2 is waiting for X to become 1
- T1 changes X to 1 and signals T2
- T2 changes X to 2 and signals T3

Write the code snippets that implement the above synchronization among the threads T1, T2, and T3.

Note:

- Busy waiting solution gets NO CREDIT.
- Loss of concurrency **except when needed** to implement the required synchronization will not get full credit.
- Of course you have to ensure there are no deadlocks.

```
Mutex-lock    m;
Cond-var      cx1, cx2

T1:
    Lock(m);
    X = 1;
    Cond-signal(cx1);
    Unlock(m);
    ...

T2:
    Lock(m);
    While (X != 1)
        Cond-wait(cx1, m);
    Unlock(m);
    ...
    Lock(m);
    X = 2;
    Cond-signal(cx2);
    Unlock(m);
    ...

T3:
    ...
    ...
    Lock(m);
    While (X != 2)
        Cond-wait(cx2, m);
    Unlock(m);
    ...
    ...
```

# CS 2200 Spring 2012 Final Exam 8 AM to 10 AM

Name: \_\_\_\_\_ Kishore \_\_\_\_\_ GT Number: \_\_\_\_\_

7. (15 points, 20 min)

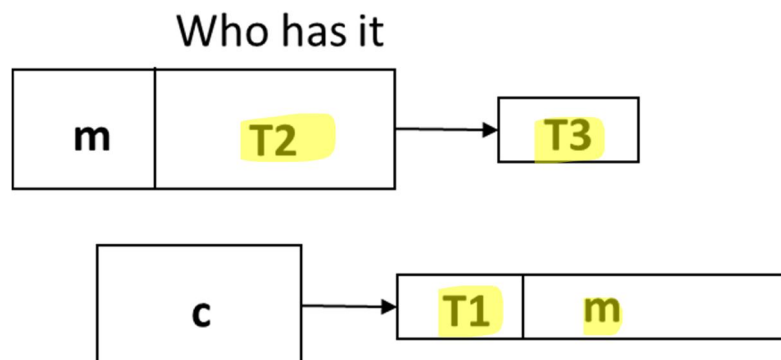
a) (7 points) Given mutex lock  $m$ , and condition variable  $c$ , the following events happen in the order of occurrence shown below:

- T1 executes `mutex-lock(m)`; assume no one has the lock so T will win
- T1 executes `cond-wait(c, m)`
- T2 executes `mutex-lock(m)`
- T3 executes `mutex-lock(m)`
- T2 executes `cond-signal(c)`

Show the waiting queues associated with  $m$  and  $c$ .

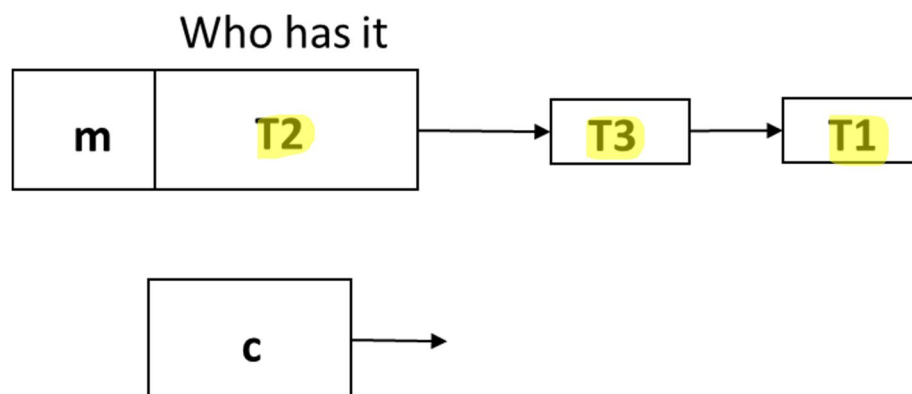
- Clearly, show which thread is currently holding the mutex lock, and which threads are in the waiting queue for the lock.
- Note that if a thread is waiting on a condition variable, you should also show the mutex lock it needs for resuming execution.

(i) State of waiting queues before T2 executes `cond-signal`



+1 for each highlighted answer

(ii) State of waiting queues after T2 executes `cond-signal`





# CS 2200 Spring 2012 Final Exam 8 AM to 10 AM

Name: \_\_\_\_\_ Kishore \_\_\_\_\_ GT Number: \_\_\_\_\_

b) (3 points)

Fill in the blanks using a subset of the following phrases exactly once.

**Hardware**                      **operating system**                      **page table**                      **cache**                      **registers**

In an SMP that supports hardware cache coherence, the hardware is responsible for ensuring that the copies of the same memory location in the different caches are kept consistent; the operating system is responsible for ensuring that the TLBs in the different processors are kept consistent; the page table is shared among all the threads of the same process.

c) (5 points)

Given a **dual core cache-coherent** processor with **per core cache** and shared memory:

Cache coherence protocol: **write-invalidate**

(i.e., write by a core to its cache invalidates the peer core cached copy if present)

Cache to memory policy: **write-back**

(i.e., cached copy in a core may be more up to date than memory; upon a cache miss for a memory location, a peer core will supply the data if its copy is more up to date compared to memory)

Initially:

The **caches** are **empty**.

**Memory** locations: **A** contains **33**; **B** contains **15**

(use **NP** for a memory location not present in the cache; **I** if the cached content is invalid)

I. **Core 1** executes: **Load A**

Show the contents of the caches and memory for memory address A

+1

A

Core 1 cache	Core 2 cache	Memory
33	NP	33

II. **Core 2** executes: **Store #99, A**; (write immediate value 99 into A)

Show the contents of the caches and memory for memory address A

+1

A

Core 1 cache	Core 2 cache	Memory
I	99	33

III. **Core 1** executes: **Store #22, B** (write immediate values 22 into B)

Show the contents of the caches and memory for memory address B

+1

B

Core 1 cache	Core 2 cache	Memory
22	NP	15

IV. **Core 2** executes: **Load B**

Show the contents of the caches and memory for memory address B

+1

B

Core 1 cache	Core 2 cache	Memory
22	22	15

V. **Core 2** evicts **A** from its cache

Show the contents of the caches and memory for memory address A

+1

A

Core 1 cache	Core 2 cache	Memory
I	I	99

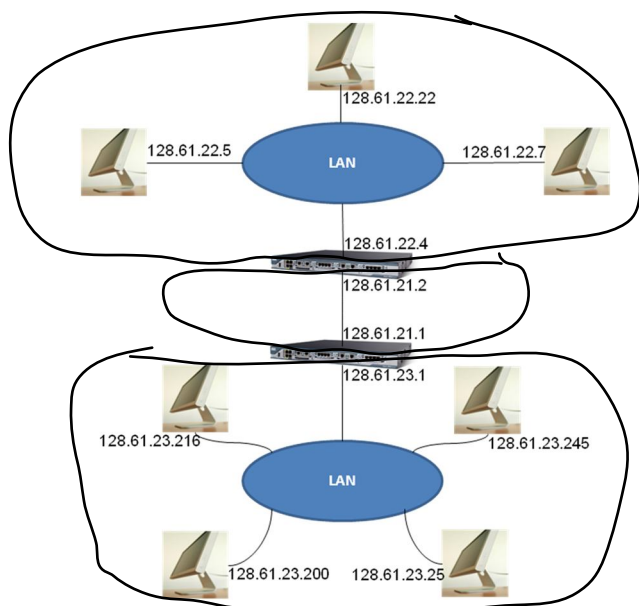
# CS 2200 Spring 2012 Final Exam 8 AM to 10 AM

Name: \_\_\_\_\_ Kishore \_\_\_\_\_ GT Number: \_\_\_\_\_

## Networking

8. (10 points, 10 min)

a) (2 points) How many IP Networks are there in the following Figure? Assume that the top 24 bits of the 32-bit address name an IP network.



Your answer: 3

All or nothing

b) (3 points) Give three short reasons to justify the need for a separate network layer in the protocol stack.

- Multiple network interfaces on a host +1
- Multiple hops between source and destination (packet has to go through intermediate routers) +1
- Routes from source to destination not static (due to network churn, load, etc.) +1

c) (5 points)

A packet header consists of the following fields:

Destination address	8 bytes
Source address	8 bytes
Number of packets in message	4 bytes
Sequence number	4 bytes
Actual packet size	4 bytes
Checksum	4 bytes

Assuming that the maximum packet size is 1574 bytes, what is the maximum payload in each packet?

Size of header =  $(8+8+4+4+4+4) = 32$  bytes

Maximum size of payload = maximum packet size - size of header +3  
=  $1574 - 32$   
= 1542 bytes } +2

# CS 2200 Spring 2012 Final Exam 8 AM to 10 AM

Name: \_\_\_\_\_ Kishore \_\_\_\_\_ GT Number: \_\_\_\_\_

9. (10 points, 10 mins)

Given the following:

Sender overhead	= 1 ms
Message size	= 200,000 bits
Wire bandwidth	= 100,000,000 bits/sec
Time of flight	= 2 ms
Receiver overhead	= 1 ms

Compute the observed bandwidth. Recall that the message transmission time consists of sender overhead, time on the wire, time of flight, and receiver overhead. Ignore ACKs.

Total time for message transmission

$$\begin{aligned} &= \text{sender overhead} + \text{transmission delay} + \text{time of flight} + \text{receiver overhead} \\ &= 1 \text{ ms} + \text{message-size/wire-bandwidth} + 2 \text{ ms} + 1 \text{ ms} \\ &= 1 \text{ ms} + 200,000 / (100,000,000 / 10^{-3}) \text{ ms} + 2 \text{ ms} + 1 \text{ ms} \\ &= 1 \text{ ms} + (200,000 * 1000) / 100,000,000 \text{ ms} + 2 \text{ ms} + 1 \text{ ms} \\ &= 1 \text{ ms} + 2 \text{ ms} + 2 \text{ ms} + 1 \text{ ms} \\ &= 6 \text{ ms} \end{aligned}$$

+3

} +3

$$\begin{aligned} \text{Observed bandwidth} &= \text{message size} / \text{transmission time} \\ &= 200,000 / (6 * 10^{-3}) \text{ bits/sec} \\ &= (200/6) 10^6 \text{ bits/sec} \end{aligned}$$

} +4