



# CS 2200 Spring 2006 Test 2 Section B

Name: Kishore GT Number: gt

## Process scheduling

2. (15 points, 10 mins)

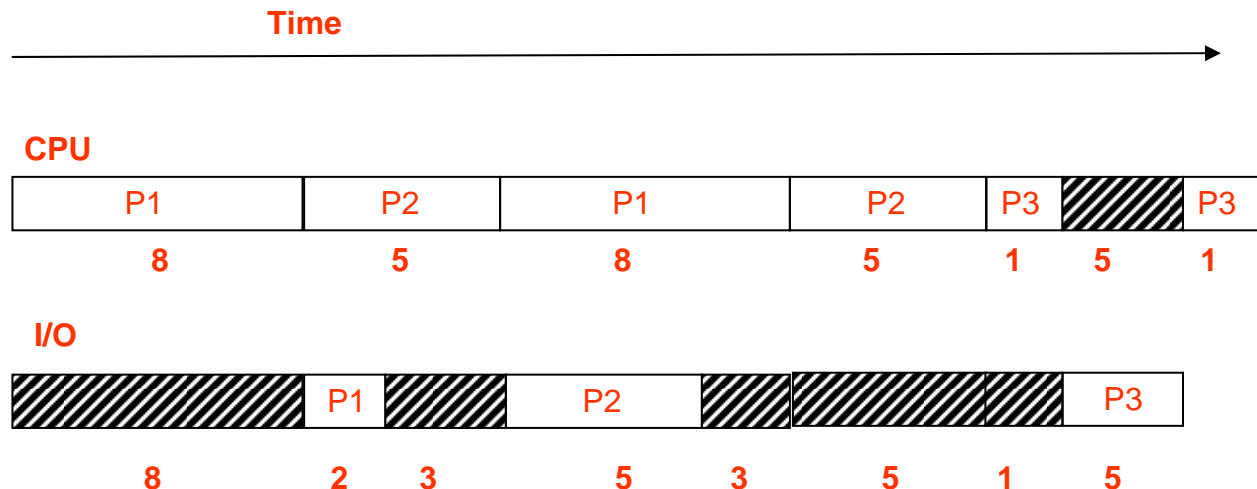
Consider a non-preemptive First Come First Served (FCFS) process scheduler. There are three processes in the scheduling queue and the arrival order is P1, P2, and P3. The arrival order is always respected when picking the next process to run on the processor. Scheduling starts at time t=0, with the following CPU and I/O burst times:

	CPU burst time	I/O burst time
P1	8	2
P2	5	5
P3	1	5

Each process terminates after completing the following sequence of three actions:

1. CPU burst
2. I/O burst
3. CPU burst

a) Show the CPU and I/O timelines that result with FCFS scheduling from t=0 until all three processes complete.



b) What is the waiting time for each process?

Waittime (P1) = 3      Waittime (P2) = 11      Waittime (P3) = 26

c) What is the turnaround time for each process?

Turnaroundtime (P1) = 21      Turnaroundtime (P2) = 26      Turnaroundtime (P3) = 33



# CS 2200 Spring 2006 Test 2 Section B

Name: Kishore GT Number: gt

## Page faults

4. (10 points, 10 mins)

Five of the following operations take place upon a page fault when there is no free frame in memory, while two of them do not. Put the five correct operations in the right temporal order and identify the two incorrect operations.

- a) use the frame table to find the process that owns the faulting page
- b) using the disk map of faulting process, load the faulting page from the disk into the victim frame
- c) select a victim page for replacement (and the associated victim frame)
- d) update the page table of faulting process and frame table to reflect the changed mapping for the victim frame
- e) using the disk map of the victim process, copy the victim page to the disk (if dirty)
- f) look up the frame table to identify the victim process and invalidate the page table entry of the victim page in the victim page table
- g) look up the TLB if the faulting page is currently in physical memory

Your answer:

Step 1: c

Step 2: f

Step 3: e

Step 4: b

Step 5: d \*\*\*

(\*\*\* note: it is OK if they show this as step 3 or 4 so long as the other steps have the same relative order)

a and g do not belong to page fault handling.

# CS 2200 Spring 2006 Test 2 Section B

Name: Kishore GT Number: gt

## Page replacement algorithms

5. (15 points, 10 mins)

Consider the following page replacement algorithm. A "reference bit" is assigned to each page. The hardware sets this bit whenever the CPU accesses any address in that page. Whenever there is need to replace a page, the memory manager does the following:

- 1) Choose the replacement candidate in FIFO order.
- 2) If the reference bit of the replacement candidate is set, the manager clears the reference bit, gives it a new arrival time based on the current time, and it repeats step (1).
- 3) The victim is the first candidate in FIFO order whose reference bit is not set.

a) Why is this algorithm often referred to as "second chance page replacement"?

**The FIFO candidate gets a "second chance" to stay in memory \*if\* it has been referenced by the process since the last time the memory manager cleared all the second reference bits.**

b) Suppose that we only have three physical frames and that we use the previous page replacement algorithm. Show the virtual page numbers that occupy these frames starting from the 4<sup>th</sup> reference. Assume that the three frames are initially empty. ALWAYS SHOW THE NORMAL FIFO REPLACEMENT CANDIDATE AT THE TOP.

Reference number:	1	2	3	4	5	6	7	8	9	10
Virtual page number:	0	1	2	3	1	4	5	3	6	4

To get you started here is the state of the page frames (the top entry with a \*\* is the FIFO replacement candidate) after the first three reference. Use the same notation to show the state of the frames after each reference starting from reference number 4. Note that the reference bit is set when the page is brought into the page frame.

State after reference number 3

**	0	ref bit = 1
	1	ref bit = 1
	2	ref bit = 1

# CS 2200 Spring 2006 Test 2 Section B

Name:  Kishore  GT Number:  gt

Your answer for Q5(b):

The following sequence of figures show the state of the page frames and the associated reference bits **\*\*AFTER\*\*** each reference is satisfied. The top entry is always the normal FIFO candidate.

Page in the frame		ref	Notation						
<p><b>Ref 1</b>                  <b>Ref 2</b>                  <b>Ref 3</b>                  <b>Ref 4</b>                  <b>Ref 5</b>                  <b>Ref 6</b></p>									
0	1		1	0	1	1	3	1	
		1	1	2	0	2	0	1	0
		2	1	3	1	3	1	4	1
<p><b>Ref 7</b>                  <b>Ref 8</b>                  <b>Ref 9</b>                  <b>Ref 10</b></p>									
4	1	4	1	3	0	5	0		
3	0	3	1	5	0	6	1		
5	1	5	1	6	1	4	1		

# CS 2200 Spring 2006 Test 2 Section B

Name: Kishore GT Number: gt

## Effective Memory Access Time

6. (15 points, 10 mins)

Consider a pipelined processor that has an average CPI of 1.5 without accounting for memory stalls. I-Cache has a hit rate of 95% and the D-Cache has a hit rate of 98%. Assume that memory reference instructions account for 25% of all the instructions executed. Out of these 70% are loads and 30% are stores. On average, the read-miss penalty is 20 cycles and the write-miss penalty is 5 cycles. Compute the effective CPI of the processor accounting for the memory stalls.

Your answer:

### In general miss penalty

$$= \text{fraction of instructions affected} * \text{miss rate} * \text{type of miss penalty}$$

$$\text{Instruction miss penalty} = 1 * (1 - 0.95) * 20 = 1$$

$$\text{Data read miss penalty} = (0.25 * 0.7) * (1 - 0.98) * 20 = 0.07$$

$$\text{Data write miss penalty} = (0.25 * 0.3) * (1 - 0.98) * 5 = 0.0075$$

### Effective CPI

$$= \text{Base CPI} + \text{instruction miss penalty} + \text{data read miss penalty} + \text{data write miss penalty}$$

$$= 1.5 + 1 + 0.07 + 0.0075$$

$$= 2.5775$$

# CS 2200 Spring 2006 Test 2 Section B

Name: Kishore GT Number: gt

## Cache design

7. (10 points, 10 mins)

Consider a direct-mapped cache with a data size of 256KB. The CPU generates 32-bit byte-addressable memory addresses. Each memory word consists of 4 bytes. The cache block size is 64 bytes. The cache has a valid bit per cache line. The cache uses the write-through policy. Show how the CPU interprets the memory address (i.e., which bits are used as the cache index, which bits are used as the tag, and which bits are used as the offset?).

Cache block size = 64 bytes

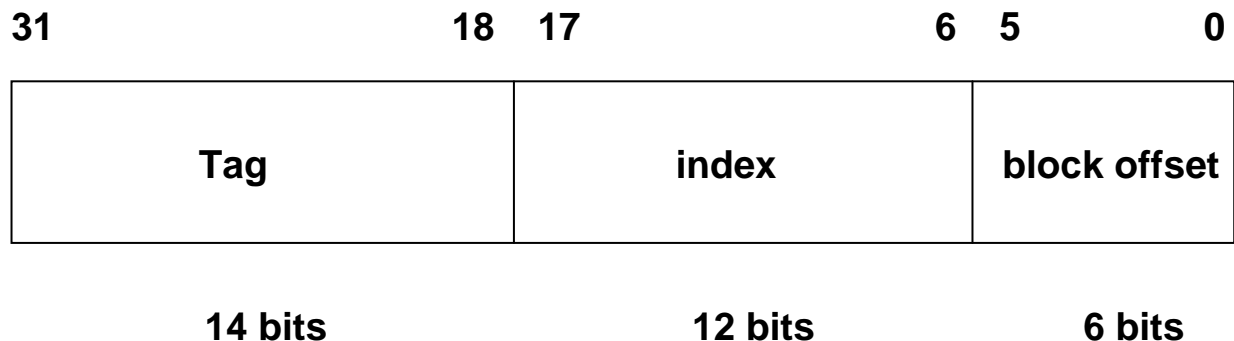
Number of bits to address a byte within a cache block =  $\log_2 64 = 6$  bits  
(these many bits are needed as the offset in a cache block and come from the low order bits of the memory address)

The number of lines (or sets) in the cache =  $256 \text{ KB} / 64 \text{ bytes} = 4 \text{ K lines}$

Number of bits needed to index into the cache =  $\log_2 4096 = 12$  bits  
(these many bits are needed as the index into the cache starting right after the block offset)

The remaining high order bits are the tag.

Thus the CPU interprets the memory address as follows:



What is the total size of the cache considering both data and meta-data?

Number of bits in one cache line  
= number of data bits + meta data bits  
= number of data bits + valid bit + tag bits  
=  $64 * 8 + 1 + 14 = 527$  bits  
Total size of cache =  $4\text{K} * 527 = 2,158,592$  bits

# CS 2200 Spring 2006 Test 2 Section B

Name: Kishore GT Number: gt

## Thread synchronization

8. (10 points, 10 mins)

The following two functions are used by a thread when it wants to print something (`access_printer`) and after it is done using the printer (`release_printer`). Suppose that we have three threads (T1, T2 and T3) that want to access the printer at about the same time. Describe a scenario in which two threads manage to get access to the printer at the same time. How can you modify the code to avoid this error?

```
boolean printer_state = NOT_BUSY;
mutex_lock_type printer_mutex;
cond_var_type printer_not_busy;

access_printer() {
    thread_mutex_lock(printer_mutex);
    if (printer_state == BUSY)
        thread_cond_wait (printer_not_busy, printer_mutex);
    printer_state = BUSY;
    thread_mutex_unlock(printer_mutex);

    print_function();
}

release_printer() {
    thread_mutex_lock(printer_mutex);
    printer_state = NOT_BUSY;
    thread_cond_signal(printer_not_busy);
    thread_mutex_unlock(printer_mutex);
}
```

T3 is here  
\*\*\*\*

T2 is here

T1 is here

Answer:

The following events happen in time order

- T1 finishes printing and releases mutex lock
- T3 acquires the lock checks the printer\_state is NOT busy, sets the state to busy, and enters the print\_function
- T2 is woken up by T1's cond\_signal, sets the printer\_state to busy (again), and enters the print\_function

Now we have both T1 and T3 in the print\_function concurrently.

Solution:

Change the \*\*\*\* statement above to:

```
while (printer_state == BUSY)
    thread_cond_wait (printer_not_busy, printer_mutex);
```

# CS 2200 Spring 2006 Test 2 Section B

Name: Kishore GT Number: gt

## Threaded OS

9. (15 points, 10 mins)

a) (6 points) The contents of the thread control block (TCB) of a multi-threaded process typically contains the following items (select the ones that apply; correct choice gets +2 points; incorrect choice -2 points):

- 1) program counter value
- 2) page table pointer
- 3) user account information
- 4) contents of the register file
- 5) contents of the TLB
- 6) contents of the stack pointer

Choices 1, 4 and 6 are right

b) (3 points) (Select one correct choice) Deadlock

- 1) is a condition where threads are not using mutex locks
- 2) is a condition where all the lock variables are in use
- 3) is a condition that will happen only in a multiprocessor
- 4) is a condition where one or more threads are waiting for an event that will never happen
- 5) None of the above
- 6) All of the above

Choice 4 is the right choice

c) (3 points) (Select one correct choice) A thread starts its execution

- 1) in main
- 2) at some procedure entry point decided at compile time
- 3) at some procedure entry point decided at run time
- 4) All of the above
- 5) None of the above

Choice 3 is the right choice

d) (3 points) Consider LC-2200 that has atomic read and write instructions. Is this sufficient to implement a mutual exclusion lock (without turning off interrupts)? Explain why or why not.

Answer:

- This is insufficient.
- To implement a lock:
  - o read a memory location  $m$  into a register (use ld instruction of LC 2200)
  - o if  $m$  is 0 then lock call is successful, else lock call failed (use beq instruction of LC 2200)
  - o write a 1 into  $m$  (use st instruction of LC 2200)
- The above three operations have to be done "atomically".
- There is no way to guarantee atomicity (without turning off interrupts) since only individual instructions in LC 2200 are atomic.