University of California, Berkeley
College of Engineering
Computer Science Division – EECS

Spring 2014 Anthony D. Joseph

Midterm Exam #1

March 12, 2014
CS162 Operating Systems

Your Name:	
SID AND 162 Login:	
TA Name:	
Discussion Section Time:	

General Information:
This is a **closed book and one 2-sided handwritten note** examination. You have 80 minutes to answer as many questions as possible. The number in parentheses at the beginning of each question indicates the number of points for that question. You should read **all** of the questions before starting the exam, as some of the questions are substantially more time consuming.

Write all of your answers directly on this paper. ***Make your answers as concise as possible.*** If there is something in a question that you believe is open to interpretation, then please ask us about it!

Good Luck!!

QUESTION	POINTS ASSIGNED	POINTS OBTAINED
1	34	
2	9	
3	15	
4	18	
5	24	
TOTAL	100	

NAME: _____

1. (34 points total) Short answer
   a. True/False and Why? **CIRCLE YOUR ANSWER.**
      i) A fully associative cache will *always* have a higher hit rate than a direct mapped cache (on the same reference pattern).

TRUE FALSE

Why?

ii) When the total execution times of each of the long-running (no I/O) processes waiting to be scheduled are equal, FIFO is the better choice (in terms of response time) of scheduling algorithm to use instead of round robin.

TRUE FALSE

Why?

iii) In a multi-level address translation scheme, the amount of memory used by the translation tables for each process' virtual address space is always proportional to the size of the virtual address space.

TRUE FALSE

Why?

iv) Monitors are more powerful than Semaphores because Monitors can implement solutions to synchronization problems that Semaphores cannot solve.

TRUE FALSE

Why?

NAME: _____

b. (8 points) Resource Allocation Tables.

Consider the following snapshot of a system with 5 processes (P1, P2, P3, P4, P5) and 4 resources (R1, R2, R3, R4). There are no outstanding queued unsatisfied requests.

	Current Allocation				Max Need				Still Needs			
Process	**R1**	**R2**	**R3**	**R4**	**R1**	**R2**	**R3**	**R4**	**R1**	**R2**	**R3**	**R4**
P1	2	0	1	4	5	3	2	4	3	3	1	0
P2	2	0	0	3	2	4	3	3	0	4	3	0
P3	0	0	3	4	2	5	5	6	2	5	2	2
P4	1	0	1	0	2	2	3	2	1	2	2	2
P5	4	1	1	0	7	3	2	2	3	2	1	2

i) Is this system currently deadlocked? Why or why not? If not deadlocked, give an execution order. Consider the currently available resources below:

Currently Available Resources

R1	R2	R3	R4
1	2	2	2

ii) If a request from a process P1 arrives for (2, 1, 1, 0), should the request be immediately granted? Why or why not? If yes, show an execution order.

Currently Available Resources

R1	R2	R3	R4
3	4	1	0

c. (6 points) We discussed the Therac-25 in lecture and in an assigned reading.

i) Briefly explain the function of the Therac-25 and how it differs from the Therac-20?

ii) What problems did the Therac-25 introduce and what were the underlying causes.

d. (8 points total) Consider a demand paging system where a dedicated disk is used for paging, and all other filesystem activity uses other separate disks. Measured utilizations are:

Component	**Utilization of <u>each</u> component** (in terms of **time**, not space)
CPU	20%
Paging disk	99.7%
Other I/O devices	5%

For each of the following changes, *assume processes have similar memory usage* and CIRCLE what its likely impact will be on CPU utilization.

i) (2 points) Get a bigger paging disk.
Effect on CPU utilization (CIRCLE one of the following 3 choices):

Significantly decrease **Minimal effect** **Significantly increase**

ii) (2 points) Increase the number of running programs
Effect on CPU utilization (CIRCLE one of the following 3 choices):

Significantly decrease **Minimal effect** **Significantly increase**

iii) (2 points) Decrease the number of running programs
Effect on CPU utilization (CIRCLE one of the following 3 choices):

Significantly decrease **Minimal effect** **Significantly increase**

iv) (2 points) Get faster other I/O devices
Effect on CPU utilization (CIRCLE one of the following 3 choices):

Significantly decrease **Minimal effect** **Significantly increase**

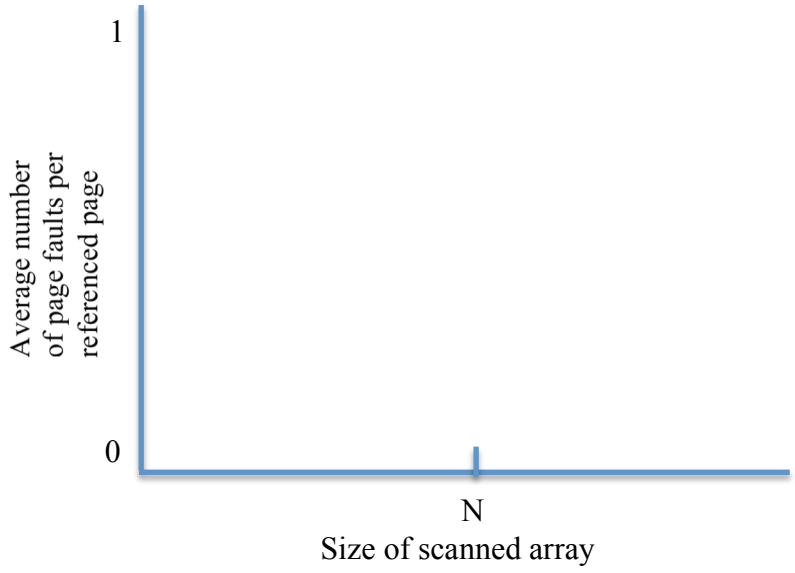
NAME: _____

2. (9 points total) Paging

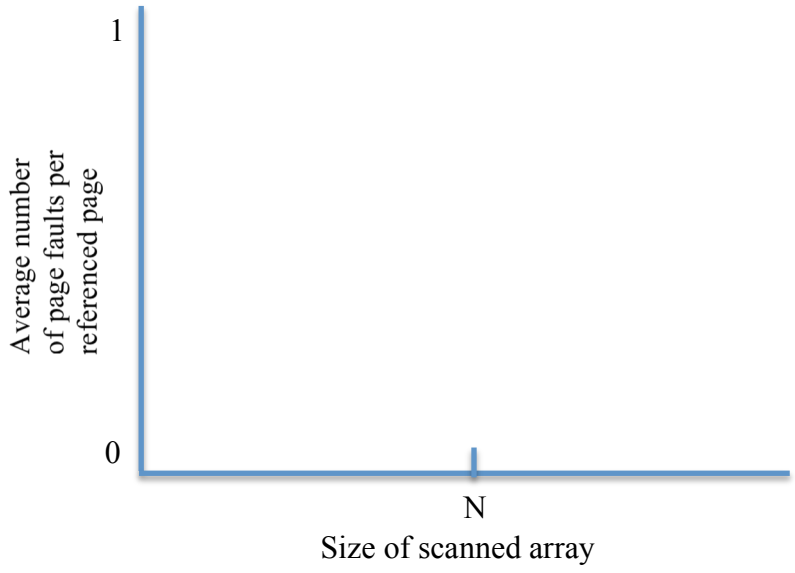
Suppose a program that **repeatedly** scans through a *very large* array is running in virtual memory. In other words, if the array is 4 pages long, its page reference pattern is ABCDABCDABCD...

For each of the following page replacement algorithms, sketch a graph showing the paging behavior. The y-axis of the graph is the number of page faults per referenced page (ranging from 0 to 1) and the x-axis is the size of the array being scanned (ranging from smaller than physical memory to much larger than physical memory).
Assume the size of memory is N pages.
Label any interesting points on each graph on both the x and y axes.

a. (3 points) FIFO:

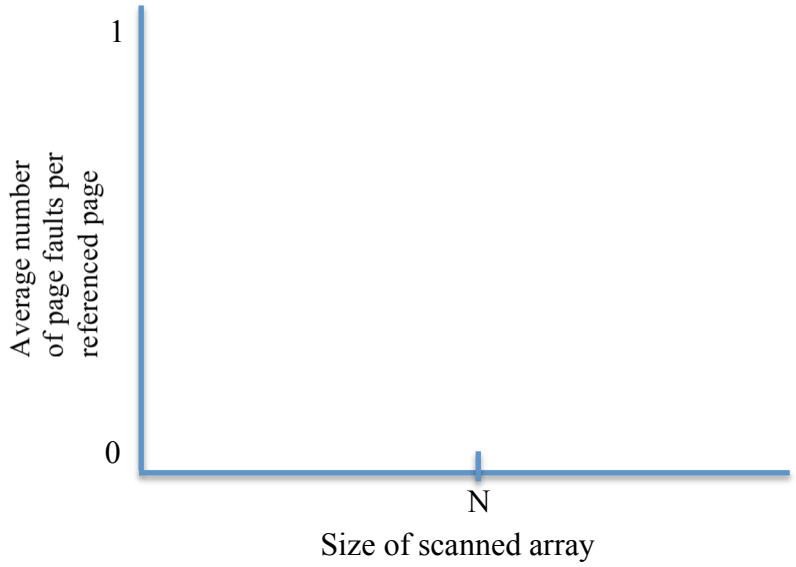


b. (3 points) LRU:



NAME: _____

c. (3 points) MIN:



NAME: _____

3. (15 points total) Memory management

Consider a multi-level memory management scheme with the following format for virtual addresses:

Virtual Page # (10 bits)	Virtual Page # (10 bits)	Offset (12 bits)

Virtual addresses are translated into physical addresses of the following form:

Physical Page # (20 bits)	Offset (12 bits)

Page table entries (PTE) are 32 bits and contain the 20-bit physical page number and OS bookkeeping bits (e.g., valid, dirty, used, etc.).

a. (2 points) How big is a page? Explain your answer.

b. (2 points) What is the maximum amount of memory (in bytes) in a single virtual address space? Explain your answer.

c. (2 points) What is the maximum amount of physical memory (in bytes) that this memory management scheme supports? Explain your answer.

d. (4 points) Sketch the format of the page table for the multi-level virtual memory management scheme. Illustrate the process of resolving an address as well as possible. Assume there is no TLB or cache.

e. (2 points) What is the maximum size in bytes of the *entire* multi-level data structure that maps virtual addresses to physical addresses for this scheme? Explain your answer. You may leave your answer in non-simplified form (e.g., powers of 2).

f. (3 points) Suppose that we have a process with a virtual address space with one physical page at the top of the address space, one physical page in the middle of the address space, and one physical page at the bottom of the address space. How big would the page table be (in bytes)? Explain your answer.

NAME: _____

4. (18 points total) Scheduling. Consider the following processes, arrival times, waiting, priority, and CPU processing requirements:

Process Name	Arrival Time	Priority	Waits on Lock Held by	Processing Time
1	0	1	None	4
2	2	20	1	3
3	5	1	None	3
4	6	10	3	2

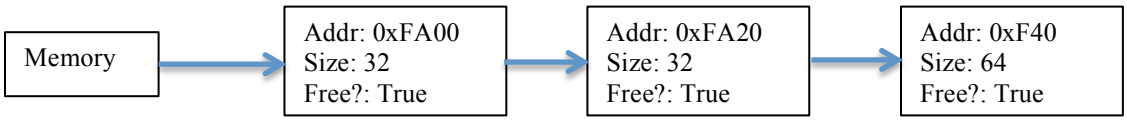
For each scheduling algorithm, fill in the table with the process that is running on the CPU (for timeslice-based algorithms, assume a 1 unit timeslice). Notes:

- For RR and Priority, assume that an arriving thread is run at the beginning of its arrival time, if the scheduling policy allows it.
- If a thread tries to acquire a lock, it will do so at the end of its first time slice. If thread A waits for a lock (formerly) held by thread B and B has already finished, it does not wait and acquires the lock immediately.
- Assume the currently running thread is not in the ready queue while it is running.
- Turnaround time is defined as the time a process takes to complete after it arrives

Time	**FIFO**	**RR**	**Priority**
0	**1**	**1**	**1**
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
Average Turnaround Time			

5. (24 points total) Synchronization – a thread safe memory allocator

A memory allocator is a userspace library that helps user processes acquire memory using methods called `malloc` and `free`. In this problem we assume the allocator only returns chunks sized in powers of two (i.e., it gives out 32 (2^5) bytes, 64 (2^6) bytes, 128 (2^7) bytes, etc.). To manage the allocated and unallocated virtual memory, the allocator maintains a linked list of nodes where each node corresponds to a block of memory, which may be free or used. Here is an example of the linked list:



Note that adjacent nodes in this linked list are also adjacent in memory. To avoid wasting memory, the allocator will split a node into smaller nodes when a smaller size is asked for than available chunks – if a thread requests 32 bytes and only one 64 byte node is available, the allocator will split that node into two 32 byte chunks and give the user one 32 byte chunk.

What we want is an algorithm that combines (from left to right) adjacent free blocks, whose sizes add up to at least the requested size. The following simple algorithm solves this problem, but is not thread safe (guaranteed to operate correctly in the presence of requests from multiple threads).

```
def malloc(size):
   for node in malloc_list:
         if (node.size >= size and node.free):
             # break node into multiple nodes if necessary
             break_apart_node(node, size)
             node.free = false
             return node.address

def free(ptr):
   for node in malloc_list:
         if (node.address == ptr):
             node.free = true

def combine_left_to_right(node, begin_size):
   if (not node->free):
         return false
   if (combine_left_to_right(node->next, node->next->size
                                         + begin_size)):
         node->size += node->next->size
         node->next = node->next->next
         return 1

   return is_power_of_two(node->next->size + begin_size)
```

Note that a background thread periodically iterates through the linked list and calls `combine_left_to_right(node, 0)` for every node.

NAME: _____

Make the above code thread safe by using locks.

```
def malloc(size):
   for node in malloc_list:
         if (node.size >= size and node.free):
             # break node into multiple nodes if necessary
             break_apart_node(node, size)
             node.free = false
             return node.address
```

a. (5 points) Thread safe `malloc`:

```
def free(ptr):
   for node in malloc_list:
         if (node.address == ptr):
             node.free = true
```

b. (5 points) Thread safe `free`:

```
def combine_left_to_right(node, begin_size):
   if (not node->free):
         return false
   if (combine_left_to_right(node->next, node->next->size
                                                + begin_size)):
         node->size += node->next->size
         node->next = node->next->next
         return 1

   return is_power_of_two(node->next->size + begin_size)
```

c. (8 points) Thread safe `combine_left_to_right`:
Note that you may want to restructure the code.

d. (3 points). Your friend Alice suggests you use **recursive locks**, which are like normal locks except that the same thread can acquire a recursive lock multiple times *without blocking* (any other thread will block as usual). However, a recursive lock must be released as many times as it is acquired. Make the original `combine_left_to_right` thread safe using recursive locks:

e. (3 points). Your friend Charlie says that **recursive locks** are just like Semaphores and you can simply replace all the calls to `Acquire` with `P()` and `Release` with `V()`. Is Charlie correct? If so, justify his claim, if not explain why.

Correct?

Justification (in four sentences or less):

This page intentionally left blank as scratch paper

Do not write answers on this page

NAME: _____

This page intentionally left blank as scratch paper

Do not write answers on this page