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

Spring 2008 Anthony D. Joseph

Midterm Exam #2 *Solutions*

April 16, 2008
CS162 Operating Systems

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

General Information:
This is a **closed book and notes** examination. You have 90 minutes to answer as many questions as possible. The number in parentheses at the beginning of each question indicates the number of points given to the question; there are 100 points in all. 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!!

Problem	Possible	Score
1	30	
2	22	
3	23	
4	25	
Total	**100**	

1. (30 points total) Short answer questions:
   a. (12 points) True/False and Why?
      i) Small time slices always improve the average turnaround time of all the processes in a system.

TRUE FALSE

Why?

FALSE. *Small time slices will sometimes improve the average response of the system. If the slice is too small, the context switching time will start to dominate the useful computation time and everything (including response time) will suffer. A lot of students gave the example of a Round Robin Scheduler with equal-length processes having a greater TRT than a FIFO scheduler. Although the question did not specifically ask this, enough students (~50%) gave this as an answer so we gave credit for it.*

      ii) Shortest Job First (SJF) or Shortest Completion Time First (SCTF) scheduling is difficult to build on a real operating system.

TRUE FALSE

Why?

TRUE. *SJF/SCTF scheduling requires knowledge of how much time a process is going to take. This requires future knowledge. You might require a user to specify the maximum amount of time that a process could run (and kill it if it exceeds this amount), then use a variant on SJF/SCTF.*

iii) You run a workload using a fixed-size 100 MB (megabyte) buffer cache and notice that there are many disk references for files. If you repartition main memory to increase the buffer cache size to 200 MB, the workload may actually run significantly slower.

TRUE FALSE

Why?

***TRUE**. By decreasing main memory to increase the buffer cache, you may cause processes to need to swap more often, which can decrease performance. Doubling the buffer cache does not entail doubling the block size of the cache itself and no credit was given for such an assumption.*

iv) You profile your file system and notice that over the course of the day the system used 6,200 files, whose total size in bytes was 100 times larger than the size of the file system's buffer cache. A cache cannot provide much benefit for this type of workload.

TRUE FALSE

Why?

***FALSE**. It is possible that at any given time, only a small subset of the files are in use; in this case, that subset can be entirely cached throughout its use.*

b. (5 points) Roughly order stack, code, and heap segments as to how well you would expect them to perform in a heavyweight process, paged VM system that uses LRU page replacement. *State the intuitions for your ordering.*

An ideal replacement policy should always throw out the page in memory that will be used the farthest in the future; thus, we use this idea as a metric for gauging the performance of Stack, Code and Heap pages.

Heap accesses are extremely random, so even if an LRU policy throws out the least recently used heap page, probabilistically, each heap page has a uniform chance of being chosen next, so LRU is effectively as good as a random replacement policy in this context.

Code pages are far less random. They execute sequentially, so it is highly likely that the next instruction one chooses to execute is on the current page. However, Code Pages do have jumps to other functions/library calls that may not have been used in a while and may exist on other pages, so there is still a small chance that LRU evicts the code page that the user program would use in the near future.

Stack pages, by definition, work via LIFO (which is conducive to LRU). This means the top of the stack is the most recently used, while the bottom of the stack is the least recently used. It is unlikely that you have an instruction that directly references information that resides in several stack frames above the current one, so the chances of the LRU evicting a stack page that will be referenced next is minimal. If a stack page is evicted, it would be the one all the way at the top of the execution stack, which is likely to be the farthest stack page in the future that will be accessed (since we must return from the chain of function calls to access it).

-2 points for having the incorrect ordering
-1 point for having an incorrect explanation for Heap Pages
-1 point for having an incorrect explanation for Code Pages
-1 point for having an incorrect explanation for Stack Pages

c. (5 points) You're hired to develop the next-generation of high performance, electronic book file servers. One of the developers on your team proposes the idea of using disk compression to serve entire large e-books (storing data in compressed disk blocks), but your boss disagrees saying that disk storage is cheap and there's no reason to use compression. Do you agree with your teammate or your boss? What are the arguments (tradeoffs, benefits, and disadvantages) in favor of or against disk compression?

Circle the person you agree with:

TEAMMATE BOSS

Tradeoffs, benefits, and disadvantages:

The problem statement leads to the following conclusions:

- *We are serving entire large e-books which means that clients can only request whole books as opposed to requesting specific sections.*
- *Books don't change, so you assume that they remain at a fixed size for the remainder of its lifetime (whatever this may be).*
- *In general, text compression typically ranges from 80-90% (meaning that a 10 MB text file typically shrinks to 1 or 2 MB)*

Given the above restrictions, it makes perfect sense to compress the files.

Pros:

- *Smaller files have a higher probability of having their blocks allocated continuously.*
- *Smaller files have less blocks which means less I/Os. Even for the case where the blocks are continuous, an uncompressed file can span many cylinders and it takes valuable seek time for the head to move from one cylinder to the next (because it must accelerate and then decelerate).*
- *Smaller files take less time to transfer over the network. There is a reason *every* website / file farm stores files in some form of a compressed format. The time saved downloading a 100 MB vs. 500 MB file is not negligible. Media files such as MP3, JPEG or DIVX are implicitly compressed.*

Cons:

- *There is some CPU overhead to compression, but this is a ONE-TIME cost (i.e. the server compresses the file and stores it on the disk).*
- *There is some CPU overhead to decompression, but this is done at the client side and doesn't affect the performance of the server.*
- *Storing more books on a single hard drive increases the cost of a failed hard drive (i.e. more books are lost), although this can be offset via RAID.*
- *A single bit error in a compressed file potentially causes the loss of the entire book. A single bit error in the uncompressed file causes a malformed character.*

Erroneous Statements:

- *Some people assumed the compression was lossy. This doesn't make sense in the context of non-media files.*
- *Some people said compression is painfully expensive. CPU cycles vs. extra Disk I/O --> Disk I/O is orders of magnitude more expensive.*
- *Some people did not realize that decompression should occur at the client end.*

-1 points for not agreeing with your teammate
-1 points for not mentioning less disk I/Os
-1 points for not mentioning less network transfer time.
-1 points for not mentioning that there is CPU overhead with compression (an obvious con)
*-3 points for not mentioning *either* of the above two pros.*

No credit was given for saying that compression saves more disk space since this is implicitly part of the problem statement.

No credit was given for saying that compression is "complicated" and that the "easy way" out is to go with uncompressed data.

d. (8 points) Since high-performance scientific computing applications run on very expensive supercomputers, it is very important to optimize their performance. Consider the following scientific computing function:

```
MatrixAdd(int a[][], int b[][], int n) {
     for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
          a[j][i] += b[j][i];
}
```

The matrices a and b are allocated using the following code:

```
MatrixAllocate(int n) {
     a = malloc (n * size of *a);
     for (i = 0; i < n; i++)
        a[i] = malloc(n * sizeof a[i]);
     return a;
}
```

You observe that the program runs more slowly than expected when calling `MatrixAdd`, so you turn on profiling and notice that the function is causing large numbers of TLB flush operations – much more than expected. Explain the likely cause and implement a simple fix. Also, explain what you generally expect to happen to paging performance when your fix is implemented?

The inner addition loop will stride through memory in n sized pieces, which will really hurt performance. For example, if n >= page – size, then every access will touch a different page. The fix is to just swap the loops. You'd expect this to also help paging performance.
-2 correct, but extra incorrect stuff
-3 didn't comment on paging performance/got that part wrong
-5 didn't recognize the problem of the stride, but did something to improve spatial locality
-5 wrong fix
-7 didn't recognize the stride problem and fix wouldn't improve spatial locality
-8 mentioned something that could improve paging when n is large.

Some students decided that it is better to improve spatial locality by mallocing n2 elements in one row. I gave 7 out of 8 points for this explanation (and full credit if they recognized the striding issue with the TLB), unless they made bugs with their malloc statement (in which case I took an additional 2 points off).

2. (22 points total) Multi-level Address Translation.

a. (10 points) You're hired by the HAL Corporation to develop their next generation machine, the HAL 2010. It will have a virtual memory architecture with the following parameters:

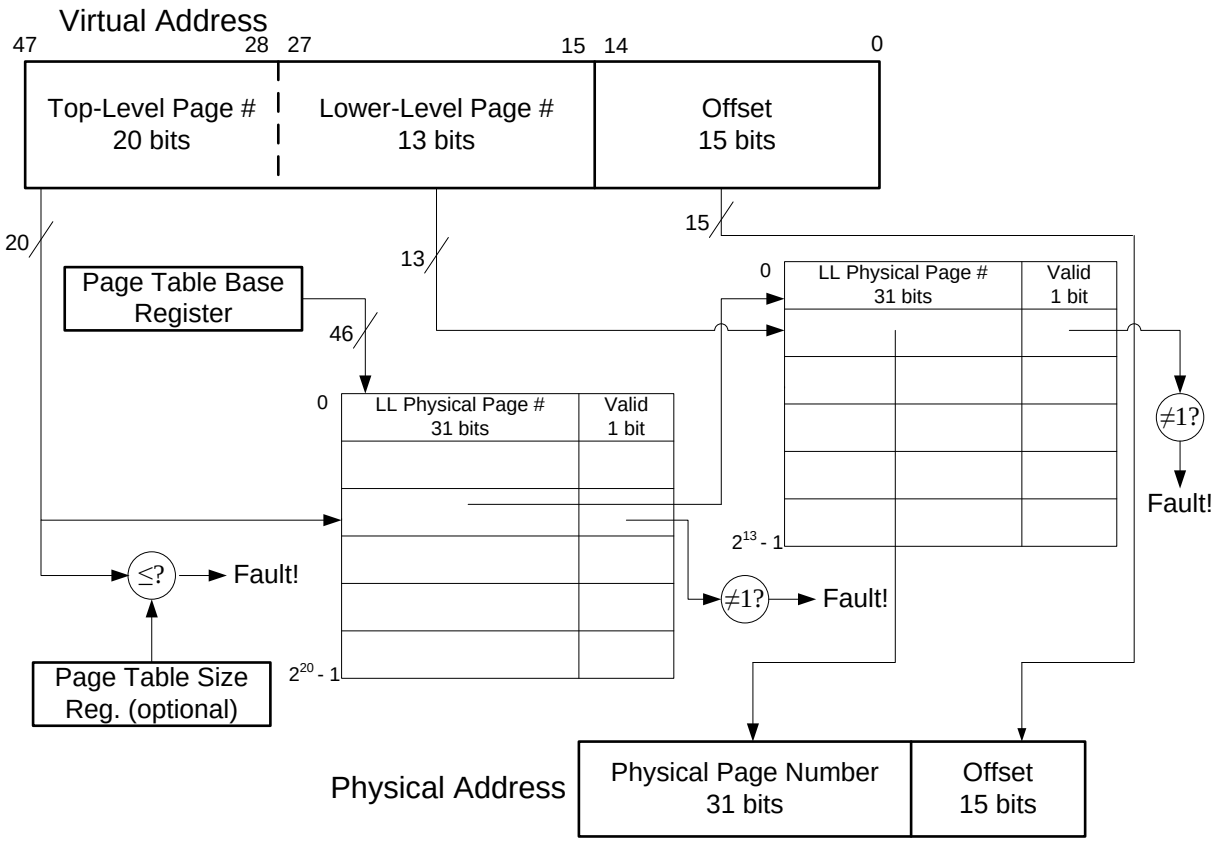
- Virtual addresses are 48 bits.
- The page size is 32K byte.
- The architecture allows a maximum 64 Terabytes (TB) of real memory (RAM).
- The first- and second-level page tables are stored in real memory.
- All page tables can start only on a page boundary.
- Each second-level page table fits exactly in a *single* page frame.
- **There are only valid bits and no other extra permission, or dirty bits.**

Your job is to draw and label a figure showing how a virtual address gets mapped into a real address. You should list how the various fields of each address are interpreted, including the size in bits of each field, the maximum possible number of entries each table holds, and the maximum possible size in bytes for each table (in bytes). Also, your answer should indicate where checks are made for faults (e.g., invalid addresses).

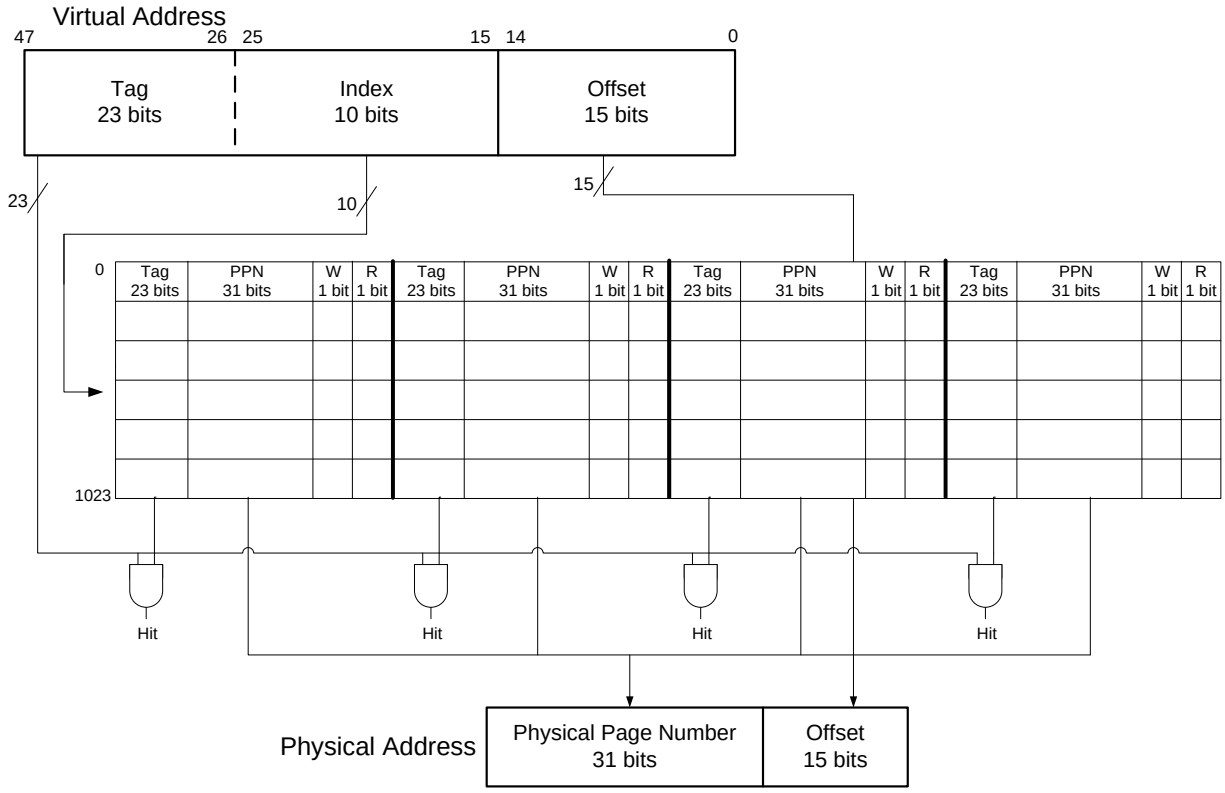
The first-level page table has 2^{20} (1 million) entries, each of which is 32 bits (31-bit PPN plus valid bit, or 4 bytes), so it has a maximum size of 4 MBytes.

Each second-level page table has 8K entries. The page size is 32KB and the second-level PTE has a PPN of 31 bits plus a valid bit, which fits nicely in a 32 bit word or 4 bytes. A 32KB page can thus hold 8K 4-byte entries (4 bytes times 8K is 32KB).

If you didn't draw a figure, we deducted 3 points. If you didn't check the valid bits, had the wrong page or offset field widths, your second level page table was the wrong size, you included extra bits in the page tables, or you did not specify the number of page table entries, we deducted 2 points (per error). If you used segments, we deducted five points. If you included extraneous checks, we deducted 1 point.


Virtual Address
47
28
27
15
14
0
Top-Level Page #
20 bits
Lower-Level Page #
13 bits
Offset
15 bits
20
13
15
Page Table Base Register
46
0
LL Physical Page #
31 bits
Valid
1 bit
0
LL Physical Page #
31 bits
Valid
1 bit
≠1?
Fault!
2^13 - 1
≤?
Fault!
≠1?
Fault!
2^20 - 1
Page Table Size Reg. (optional)
Physical Address
Physical Page Number
31 bits
Offset
15 bits

b. (6 points) To improve the HAL 2010's performance, you decide to add a TLB to the HAL 2010's virtual memory architecture. The TLB will be 4-way set associative and have 1024 rows in each set. Draw and label a diagram of the TLB, showing the size of each field in the TLB. Indicate how bits of the virtual address are used as input to the TLB, and describe the outputs from the TLB. Include a read enable bit and write enable bit for each page in your TLB.



Since the TLB has 1,024 rows in each set, and the offset is 15 bits, we need 10 bits to index the appropriate row, and we have 23 bits as the tag field.
If your answer was missing bits in the virtual address (e.g., no tag field) or the size of the tag was wrong, we deducted one point. If your answer didn't include a figure, your index and tag scheme was incorrect or missing, or your solution didn't include four sets, we deducted three points for each error. If your answer didn't include a reasonable index scheme or you incorrectly used a 4-way set associative cache, we deducted two points.

c. (3 points) If your Page Table Entries contain R/W enable bits, why do they also need to be included in the TLB?

If an address hits in the TLB, it will bypass looking in the page table. If the R/W bits are only in the page table, we wouldn't be able to check the permissions on pages whose addresses were in the TLB.

d. (3 points) Your manager is concerned that the page tables for the HAL 2010 memory system might become very large, especially if they are *very* sparsely populated. Describe an alternate technique for the virtual memory system to more efficiently store virtual to physical address mappings, and explain how this technique would improve the storage efficiency of the page tables.

Use an inverted hash table. Size is proportional to the total amount of physical memory and the amount of virtual memory space in use. The correct answer was worth one point, and the explanation was worth two points. Note that we did not accept other schemes as they don't handle ***very*** *sparse virtual address spaces efficiently.*

No Credit – **Problem X** (000000000000 points)

Degrees of Matriculation

(April 1, 2008, adapted from NY Times Op-Ed Contributor ANDY BOROWITZ)

FROM: Office of Admissions, Stanford University

To: Double Legacy Applicant, Class of 2012

Here at Stanford's Office of Admissions, we have some very exciting news for you. While your SAT scores and grade point average fall below the threshold for acceptance to Stanford's class of 2012, your Stanford parents' dogged participation in our annual fund-raising appeals — including their generous contributions to Stanford's recombinant DNA lab and IMAX theater — have gained you admission to a unique new program called LegacyPlus™.

With LegacyPlus™, you, the Stanford double legacy, will enjoy all of the perks of students who actually got into Stanford — except for the education part.

As a LegacyPlus™ attendee, you will live on the Stanford campus for the next four years in special LegacyPlus™ housing in historic East Palo Alto. You will walk, talk and — on four special "Common Nights" a year — dine with real Stanford undergraduates.

What will you and your fellow LegacyPlus™ enrollees be doing while the actual Stanford Class of 2012 is going to lectures, choosing majors and taking exams? A better question would be, "What won't you be doing?" With a jam-packed schedule of specially organized LegacyPlus™ pub crawls, you won't have time to think about all of those classes you won't be taking.

And the benefits of LegacyPlus™ don't end in June 2012, when, in a ceremony scheduled to coincide with Stanford commencement, you will receive an authentic-looking Stanford College Certificate of Participation™. This diploma-like document, produced by an expensive laser printer, allows you to tap into Stanford's "old boy network" of jobs, fellowships and government positions that have been turned down by actual Stanford graduates.

Some of the additional perks of LegacyPlus™ include:

- Official Stanford TuitionBill™ identical to those paid by parents of the Stanford Class of 2012.
- Permission to root for the Stanford football, hockey and lacrosse teams; complimentary tickets to swim meets.
- 15 percent discount on LegacyPlus™ Stanford insignia items, including a car window sticker reading, "My Child Is at Stanford."
- Special LegacyPlus™ "Don't ask, don't tell" policy regarding your use of the words "Stanford College" on your résumé.
- Invitations to LegacyPlus™ class reunions, including such events as the "Big Game" attended by the actual Stanford Class of 2012, which you will view from your hotel via closed-circuit TV.
- And finally, the greatest privilege of all: a lifelong relationship with Stanford's dedicated network of fund-raising cold-callers.

So join us, won't you? Take advantage of this rare opportunity by wiring funds today. (Prices double after May 1.)

LegacyPlus™: The Class, Without the Classes.

3. (23 points total) Filesystems.

a. (12 points) For each of the following file descriptor data structures, describe: how efficiently this structure handles sequential access of large files, and why.

i) Contiguous Allocation:

-4 for saying that this is not efficient (for accessing).
-4 for saying the file blocks are fragmented on disk.
*-3 for not acknowledging that the file is allocated as a contiguous portion on **disk**. Some students confused disk with memory allocation. Memory access time is uniform (at least in the context of our class), so putting a*
-1 for not explaining that there is little seek penalty. You only need 1 seek + rotate.
-1 for incorrect terminology (i.e. pages instead of blocks, logically contiguous on disk, and etc.)

ii) Linked Allocation (*not* a FAT):

-4 for saying that this is more efficient than contiguous allocation.
-3 for not acknowledging that the file blocks are fragmented on disk.
-1 for not explaining that there is potentially a seek + rotate for reading each block.
-1 for incorrect terminology.

iii) UNIX 4.2BSD inodes:

-4 for incorrect description of inodes.
-3 for not _explicitly_ saying that the 4.2 version attempts to allocate data blocks contiguously, and it tries to allocate the ptr blocks close together. (A lot of students explained what the inode structure is and how it works but failed to realize the placement of blocks on disk).
-1 for not realizing that there may be some seeks and rotates.
-1 for not realizing that seeks and rotates may be avoided (because blocks may be allocated contiguously).

b. (11 points) UNIX filesystems.

Consider a UNIX filesystem with the following components:

- Disk blocks are 4096 bytes. Sectors are 512 bytes long.
- All metadata pointers are 32-bits long.
- An inode has 12 direct block pointers, one indirect block pointer and one double-indirect block pointer. The total inode size is 256 bytes.
- Both indirect and double indirect blocks take up an entire disk block.

i) (6 points) How much disk space, including metadata and data blocks, is needed to store a 4 GB DVD image file? You can leave the answer in symbolic (e.g., 6MB + 3KB) form– show your calculation for partial credit.

Requires 2^{32} bytes of actual data: 1048576 data blocks.
1024 block ptrs per indirect block.
1,048,564 blocks/1024 = 1024 indirect blocks needed.
1 double-indirect block needed.
256 bytes for inode.
Total: 4,299,165,952 bytes.
-2 for not including the actually data (4GB)
-1 for not including the inode (256B)
*-1 for not including the doubly pointer block (1 * 2^12B = 4KB)*
-1 for not including the singly (indirect) pointer blocks:
*(210 * 2^12B = 4MB)*
-1 for each additional term.
-1 for not showing steps.

ii) (5 points) Assuming that there is not a buffer cache (i.e., no filesystem structures or data are cached), starting from the *inumber*, how many disk accesses will be required to read *only* the last byte in this file? To instead overwrite it? Show your calculations for partial credit.

To read: 1 read required for the inode, 1 to read the double indirect block, 1 to read indirect, 1 to read the data: 4 disk accesses.

To write: 4 reads, plus 1 write = 5 disk accesses.

-3 for incorrect # of I/O's for reading.
-2 for incorrect # of I/O's for reading but showed somewhat reasonable steps.
-2 for not acknowledging that there is an extra I/O for overwriting the last byte (s.t. you have to read the block, and then write back the whole block).

4. (25 points total) CPU Scheduling. Here is a table of processes and their associated arrival and running times.

Process ID	**Arrival Time**	**Expected CPU Running Time**
Process 1	0	4
Process 2	2	3
Process 3	4	5
Process 4	6	2

a. (9 points) Show the scheduling order for these processes under First-In-First-Out (FIFO), Shortest-Job First (SJF), and Round-Robin (RR) with a quantum = 1 time unit. *Assume that the context switch overhead is 0 and new processes are added to the **head** of the queue except for FIFO.*

Time	**FIFO**	**SJF**	**RR**
0	*1*	*1*	*1*
1	*1*	*1*	*1*
2	*1*	*1*	*2*
3	*1*	*1*	*1*
4	*2*	*2*	*3*
5	*2*	*2*	*2*
6	*2*	*2*	*4*
7	*3*	*4*	*1*
8	*3*	*4*	*3*
9	*3*	*3*	*2*
10	*3*	*3*	*4*
11	*3*	*3*	*3*
12	*4*	*3*	*3*
13	*4*	*3*	*3*

b. (12 points) For each process in each schedule above, indicate the queue wait time and turnaround time (TRT).

Scheduler	Process 1	Process 2	Process 3	Process 4
FIFO queue wait	*0*	*2*	*3*	*6*
FIFO TRT	*4*	*5*	*8*	*8*
SJF queue wait	*0*	*2*	*5*	*1*
SJF TRT	*4*	*5*	*10*	*3*
RR queue wait	*4*	*5*	*5*	*3*
RR TRT	*8*	*8*	*10*	*5*

The queue wait time is the *total* time a thread spends in the wait queue. The turnaround time is defined as the time a process takes to complete after it arrives.

Part a) 3 points per column, part b) 2 points per row. We deducted one point per error up to the maximum score for the column/row. We deducted one point for a wrong overall arrival time. We graded part b) based on your answer for a).
Part a):

- *SJF: -1 preempt poorly. we told everyone non-preemptable SJF to avoid confusion, but even if you preempt, you should have done shortest remaining time (SJF-preempt is SRTF), and you get the same answer.*
- *SJF: -1 for other errors or swapping processes 3 and 4*
- *RR: -1 for poor queue maintenance (not putting new items at the head) or an erroneous swap/typo.*
  -2 if it was really bad and there was no evidence of a reasonable queue.

Part b):

- *If we couldn't tell how you got an answer, we deducted .5 points per error (and rounded up).*
- *If there were systemic problems, such as an off-by-one error or forgetting to subtract the arrival time, then we deducted 1 point per row or 2 points overall.*
- *If you used runtime for TRT, we deducted 3 points overall for this systemic mistake.*
- *If you swapped two processes within a scheduling type (like 3 and 4 within SJF) we deducted 1 point.*

c. (4 points) A FIFO page replacement algorithm replaces the page that was first referenced the longest time ago. Under a workload with temporal reference locality, choosing to replace a page accessed a long time ago seems like a good idea, since it is not likely to be within the current locality being accessed. Does this mean that a FIFO replacement algorithm should approximate a LRU algorithm for workloads with strong locality? Justify your answer with an example memory reference pattern or string.

There are cases when LRU and FIFO act the same way. However, FIFO does not automatically approximate LRU. Suppose you have four page frames and 6 pages.

For the access pattern A B C D E F, FIFO and LRU do the same thing. However, for the access pattern ABACADAEAF, FIFO would kick out A to bring in E while LRU would kick out B to bring in E.

Hence, the two techniques are not synonymous. The first touched item is not necessarily the item touched longest ago.
-4 for nothing or saying Yes, but not making a lot of sense
-3 for saying just No, or saying Yes and expressing an understanding of the issues
-2 for a Yes answer that presented an argument supporting "No," as if there was a misunderstanding about the term "approximate."
-2 for a No answer with no example or a really bad example
-1 for a No answer for a bad explanation.

This page intentionally left blank