Name: KEY

Operating Systems
Fall 2014
Test 2
October 27, 2014

Closed books, notes, cell phones, PDAs, iPods, laptops, etc. No headphones, please. You may use a simple calculator (but you can do without).

You have 75 minutes to solve 6 problems. You get 10 points for writing your name on the top of this page. As with any exam, you should read through the questions first and start with those that you are most comfortable with. If you believe that you cannot answer a question without making some assumptions, state those assumptions in your answer.

Partial credit will be offered only for meaningful progress towards solving the problems.

Please read and sign below if you agree with the following statement:

In recognition of and in the spirit of the academic code of honor, I certify that I will neither give nor receive unpermitted aid on this exam.

Signature:_____

0	/10
1	/20
2	/10
3	/15
4	/35
5	/10
6	/10
Total	/110

1. (20 points) Short Answer Questions:

   a. The optimal page replacement policy results in the fewest number of page faults.

   b. When the system spends most of its time swapping pieces rather than executing instructions it leads to a condition known as thrashing.

   c. The term spin (busy) waiting refers to a technique in which a process can do nothing until it gets permission to enter its critical section but continues to execute an instruction or set of instructions that tests the appropriate variable to gain entrance.

   d. ~~*True or*~~ *False:* It is possible in a single-processor system to not only interleave the execution of multiple processes but also to overlap them.

      Note: I gave credit for "True" as well because of a possible (although not direct) explanation that the overlap is between the I/O of a process and the execution of another process. It is not really in the spirit of the problem, but is a meaningful explanation.

   e. ~~*True or*~~ *False:* Race condition is a situation in which two or more processes continuously change their states in response to changes in the other process(es) without doing any useful work.

   f. ~~*True or*~~ *False*: A process that is waiting for access to a critical section does not consume processor time when mutual exclusion is implemented with TestAndSet-based locks.

   g. A situation in which multiple threads or processes read and write a shared data item and the final result depends on the relative timing of their execution is a race condition.

   h. Critical section is a section of code within a process that requires access to shared resources and that must not be executed while another process is in a corresponding section of code.

   i. Atomic operation is when the sequence of instruction is guaranteed to execute as a group, or not execute at all.

   j. *True* ~~*or False*~~*:* Threads of the same process that share some memory need to be synchronized to enforce mutual exclusion.

2. (10 points) Short attention span:
    a. Once the operating system identifies a page to remove from memory, what must be done before the frame in which the page is stored can be reused?

        - Write the content of the frame back to swap space if modified (dirty bit is 1).
        - Mark the entry in the page table corresponding to the page that was stored in the frame as invalid.

    b. At one time, virtual memory system designers were advised to bias page replacement algorithms against modified pages in favor of those that have not been modified. The result of this suggestion was the unfortunate behavior of program code pages (which tend to be some of the only pages that are not modified) being kicked out of memory before other pages. Describe the rationale for the original suggestion of paging unmodified pages first.

        The attempt was to save on write-to-disk operations: the first step listed above (i.e., writing the dirty pages to the disk) would not be needed.

3. (15 points) A process has four frames allocated to it. (All the following numbers are decimal, and everything is numbered starting from zero.) The time of the last loading of a page into each frame, the last access to the page in each frame, and the virtual page number in each frame are as shown below (the times are in clock ticks from the process start at time 0 to the event).

Virtual Page Number	Frame	Time Loaded	Time Referenced
20	0	60	161
22	1	130	160
24	2	10	162
31	3	20	163

A page fault to virtual page 23 has occurred at time 164. Which **frame** will have its contents replaced for each of the following memory management policies?

- FIFO (first-in-first-out)

  Frame 2

- LRU (least recently used)

Frame 1

- Optimal. Use the following reference string: 23, 25, 23, 25, 20, 22, 31, 23, 24.

Frame 2

4. [35] Consider the following program.

```
#define N 64
int A[N, N], B[N, N], C[N, N];
int i, j;

for (j = 0; j < N; j ++)
     for (i = 0; i < N; i++)
          C[i, j] = A[i, j] + B[i, j];
```

Assume that the program is running on a system using on demand paging and the page size is 4KB. Each integer is 4 bytes long. It is clear that each array requires 4-page space. As an example, the following elements from matrix A will fit in one page: A[0,0]-A[0,63], A[1,0]-A[1,63], A[2,0]-A[2,63], … to A[15,0]-A[15,63]. A similar storage pattern can be derived for the rest of array A and for arrays B and C.

Assume that the system allocates a 4-page working set for this process. One of the pages will be used by the program and three pages can be used for the data. Also, assume that two index registers are assigned for i and j (so, no memory accesses are needed for references to these two variables).

a) Consider that the code fits entirely in page 0 of the process' address space and the matrices are stored starting from page 200, one after another (A first, then B, then C). Write the first 15-20 lines from a trace file as the one in Project 3 that would be generated by this program. (Make it as similar in format as you remember). [10]
b) Write the page reference string. [5]
c) How many page faults does this program generate? [10]
d) How would you modify the program to minimize the page fault frequency? What will be the frequency of page faults after your modification? [10]

a)
A is stored on pages 200—203; B is stored on pages 204—207; C is stored on pages 208—211.
Each row of every matrix takes 4 bytes per integer X 64 integers = 256 bytes.

Thus, A[0, 0] is stored at offfset 0 in page 200, A[1, 0] is stored at offset 256 in page 200; A[2,0] at offset 2x256 in page 200, etc. The same logic goes for B and C.

Note that the following do not include code (that's on page 0).

C8000	R	(read A[0,0], decimal address: 819200)
CC000	R	(read B[0,0], decimal address: 835584)
D0000	W	(write C[0,0], decimal address: 851968)
C8100	R	(read A[1,0], decimal address: 819456)
CC100	R	(read B[1,0], decimal address: 835840)
D0100	W	(write C[1,0], ...)
C8200	R	(read A[2,0])
CC200	R	(read B[2,0])
D0200	W	(...)
C8300	R	(...)
CC300	R	
D0300	W	
C8400	R	
CC400	R	
D0400	W	
C8500	R	
CC500	R	
D0500	W	

a) 0, 200, 204, 208, 0, 201, 205, 209, 0, 202, 206, 210, 0, 203, 207, 211, 0, 200, 204, 208, 0, 201, 205, 209, ... (repeated 64 times total)

b) Because 16 rows of each matrix fit on a page, and because the access pattern is to access first the first element of each row, then the second element of each row, then the third element of each row, etc., for every 16 iterations of the inner loop we have 3 faults, one for each matrix. Thus, 64/16 iterations of the inner loop that cause page faults X 3 page faults, one for each matric X 64 iterations of the outer loop = 768 is the number of page faults. (4x3X64).

c) Change the inner/outer loops as follows:

```
for (i = 0; i < N; i ++)
  for (j = 0; j < N; j++)
       C[i, j] = A[i, j] + B[i, j];
```

This will cause only 12 page faults.

5. [10 points] A computer with a 32-bit address uses a two-level page table. Virtual addresses are split into a 9-bit top-level page table field, an 11-bit second-level page table field, and an offset. How large are the pages and how many are there in the address space?

Page size: 2^{12} B= 4KB; Number of pages: 2^{20}

6. [10 points] Suppose the following functions foo and bar are executed concurrently and share the semaphore variables S and R (each initialized to 1) and the integer variable x (initialized to 0).

```
void foo( ) {
    while (1) {
      sem_wait(S);       //f1
      sem_wait(R);       //f2
      printf("foo\n");   //f3
      x++;               //f4
      sem_post(S);       //f5
      sem_post(R);       //f6
   }
}
```

```
void bar( ) {
     while (1){
       sem-wait(R);       //b1
       sem_wait(S);       //b2
       printf("bar\n");   //b3
       x--;               //b4
       sem_post(S);       //b5
       sem_post(R);       //b6
     }
}
```

a) Show all possible different paths of execution.
b) Can the concurrent execution of these two processes result in one or both being blocked forever? If yes, give an execution sequence in which one or both are blocked forever and explain your answer.

a) They can run one after another without interleaving their instructions, or they can run as in the following examples (the combinations in mirror -- i.e., bar first, then foo -- are not shown. Also, below are the most representative examples):

f1 f2 b1 [asleep on semaphore R] f3—f6, b2—b6;
f1 f2 b1 [asleep on semaphore R] f3—f6, b2, f1 [asleep on semaphore S], b3—b6;
f1 b1 b2 [asleep on semaphore S] f2 [asleep on semaphore R] – deadlock
f1 b1 b2 f2 [asleep on semaphore R] b3—b6;

b) Yes, the processes can deadlock in the following sequences:
   i. f1 b1 b2 f2
   ii. b1 f1 b2 f2
   iii. b1 f1 f2 b2
   iv. f1 b1 f2 b2