CSE473/Spring 2008 - 2nd Midterm Exam

Th, April 17, 2008 — Professor Trent Jaeger

Please read the instructions and questions carefully. You will be graded for clarity and correctness. You have 75 minutes to complete this exam, so focus on those questions whose subject matter you know well. Write legibly and check your answers before handing it in.

Short Answer - some will be one or two words – no more than 3 sentences

1. (*3pts*) Define a *critical section* of a program.

   answer: A sequence of instructions where if one process/thread is executing those instructions, then no other process/thread may execute any instruction in that sequence.

2. (*4pts*) What is a fundamental difference between a *mutex* and a *semaphore*? How does this enable bounded buffer exclusion?

   answer: Semaphores enable counting with mutual exclusion. After a certain number of counts, then we can prevent further access. For example, for a bounded buffer, we can ensure mutual exclusion and prevent overflow and access to empty buffers.

3. (*3pts*) Name three of the necessary conditions for a deadlock?

   answer: (1) Mutual exclusion; (2) hold and wait; (3) no preemption; (4) circular wait

4. (*4pts*) What is fragmentation? What is the difference between *external* and *internal* fragmentation?

   answer: Fragmentation is a result of memory allocation where holes are formed between memory allocations (external) or within memory allocations (internal) which cannot be used for further allocations. External: total memory space exists to satisfy a request, but it is not contiguous. Internal: allocated memory is larger than requested, so memory internal to a region is not being used.

5. (*4pts*) Why is *page table size* a significant issue in the design of a memory management system? How does an *inverted page table* address this challenge?

   answer: A page table must store an entry for each *virtual* page (4K), so for 32-bit and 64-bit virtual address spaces *per process* these generates a large number of entries. An inverted page table is based on physical memory, so its size is limited by that of physical memory.

6. (*3pts*) What are three types of objects that comprise the file system interface?

   answer: Files, Directories, and Partitions (file systems)

7. (*3pts*) What is the problem with storing file data blocks in a linked list where the first few bytes of each data block refer to the next data block of the file?

   answer: To access the *nth* data block, you will have to make n reads from the disk. As disk reads are slow operations, this is unacceptable.

Long Answer - no more than 2 paragraphs

8. (*7pts*) What is *Belady's anomaly*? Suppose that we propose a page replacement algorithm that is a last-referenced, last-out algorithm, where the last referenced page is the last one replaced. Does this suffer from Belady's anomaly? Why or why not?

   answer: Belady's anomaly in a page replacement algorithm may occurs when the addition of more physical memory may still result in more page faults using this algorithm. The Belady's anomaly is not present for *stack algorithms* – algorithms that maintain the page references in a stack from most recent to least recent. The last-in, last-out replacement algorithm proposed places the last referenced page on the top of the stack – as it is the last one to be replaced. Thus, it is an example of a stack algorithm, so it does not suffer from Belady's anomaly.

9. (*7pts*) Show how a *strict alternation* mutual exclusion algorithm below either satisfies or fails to satisfy mutex, progress, or bounded waiting.

Strict Alternation

```
        turn = 0;

P0 {
        while (turn != 0);
        /*** access shared data ***/
        turn = 1;
}

P1 {
        while (turn != 1);
        /*** access shared data ***/
        turn = 0;
}
```

answer: Strict alternation satisfies mutex – only the thread whose turn it is (determined by the turn variable) can access the shared data. It does not satisfy progress, as the approach forces P0 to always run after P1 and vice versa. Should a thread want to enter the critical section consectutively, it will have to wait, even if the other thread is not interested in the critical section – violating progress. Also, the waiting time could be infinite. Finally, bounded waiting is satisfied because there is a bound on the number of other processes that are allowed to enter the critical section before this process (1).

10. (*7pts*) Consider the performance of disk reads and the choices of data block allocation (i.e., contiguous, linked, and indexed allocation). What type of allocation would be most efficient for small files of a bounded size? Why? What type of allocation would be most efficient for large, unbounded files accessed directly? Why?

    answer: Use contiguous allocation for small bounded files, because: (1) you can predict the size of memory, so no block movement is necessary; (2) because the file is small, direct and sequential access is fast and easy. Use indexed allocation for large, unbound files because: (1) direct access can be done more quickly than linked; (2) no movement of blocks is required because it uses non-contiguous allocation; (3) access latency to data early in the file is good for direct access and not bad even near the end (a few blocks).

11. (*7pts*) Suppose that you have two UNIX threads that are writing to the same file's data blocks. Why would it be more efficient for concurrency control between these two threads to be implemented on the block-writes by the file system, rather than application-level semaphores? Why would the file system not be able to enforce such concurrency control if the file is memory-mapped? Why might file system concurrency control be insufficient for the application requirements?

    answer: The file system could synchronize block writes more efficiently because this could be done within the `write` system call, rather than requiring more system calls for mutexes or semaphores. However, a memory-mapped file is modified without executing a `write` system call, so the file system has no way to synchronize such memory operations. Finally, the application's critical section may require that multiple writes be performed as an atomic unit. Such OS-level synchronization will not be aware of the synchronization requirements of a set of writes.

Word Problems - take your time and answer clearly and completely.

12. (*10pts*) Consider a system with 4 processes (P_1 through P_4) and 3 resource types A, B, C. Suppose at time T_0 the following snapshot has been taken:

Process ID	Allocated (A/B/C)	Max (A/B/C)
P_1	1/3/2	9/4/3
P_2	3/2/0	4/9/6
P_3	4/1/1	5/1/5
P_4	1/4/2	6/7/3

    (a) (2pts) Draw the *Needs* table for the system.

    (b) (2pts) Suppose the number of free resources are: $A = 1$, $B = 2$, $C = 4$. Find a process sequence that will prove that this system is in a *safe state* or identify that it is not in a *safe state*.

    (c) (2pts) Suppose that process P_2 makes a request for $B = 1$, $C = 2$. Would this result in a safe state (if so, show how)?

(d) (2pts) Suppose that the request in step (c) is granted by the system (from the free resources in (b)). Will the following request set result in a *deadlock*? $P_1 = (4, 1, 1)$, $P_2 = (0, 0, 4)$, $P_3 = (3, 0, 6)$, $P_4 = (0, 1, 1)$. If there is no deadlock, show a legal process sequence.

(e) (2pts) Specify two ways to recover from a deadlock.

answer: (a)

Needs (A/B/C)
8/1/1
1/7/6
1/0/4
5/3/1

(b) P3, P4, P2, P1 or P3, P4, P1, P2
(c) No, not safe
(d) No deadlock. P4, P2, P1, P3

after P4: 2/5/4 only P2 can go after P2: 6/7/6 either P3 or P1

(e) two of preempt resources, kill the process, checkpoint and roolback

13. (*10pts*) Answer the questions regarding the page table of one process specified below. Assume 4K pages.

Virtual Page	Frame Number	Valid Bit	Reference Bit	Dirty Bit
0	3	v	0	1
1	-	i	0	0
2	0	v	1	1
3	-	i	0	0
4	9	v	1	0
5	-	i	0	0
6	-	i	0	0
7	12	v	0	0

(a) (2pts) How many pages of this process are resident in physical memory? What happens when one of the other pages is accessed?

(b) (2pts) What physical memory address is accessed when the process accesses the virtual address 0x432f?

(c) (2pts) Suppose there is a reference to virtual page 6 by the process and that there is no free frame. If we use a *Not Recently Used* page replacement strategy, which page would be selected for

replacement? What operation(s) would need to occur before freeing that page?

(d) (2pts) Write the resulting page table entries for virtual page 6 and the page that was replaced. Assume that the reference to virtual page 6 was to write to the page.

(e) (2pts) Suppose the memory access time is 100 nanoseconds and the average page-fault service time is 1 millisecond. What is the page fault probability for an *effective access time* of 1 microsecond (a close approximation is acceptable – show work)?

answer: (a) four frames. Page fault.
(b) Physical address would be 0x932f.
(c) 7. Nothing – the page is clean.
(d) for 6: v:6, f:12, vb:v, rb:1, db:1; for 7: v:7, f:-, vb:i, rb:0, db:0
(e) 1 = (1-p).1 + 1000p = .1 - .1p + 1000p 9 in 10000

14. (*10pts*) Answer questions about concurrency control for the following code.

Data Base Access Code

```
        database_t db;

void *thread1( void *arg )
{
        query_t *q = (arg ? (input_t *) arg→query : NULL); /* get query from arg */

        update_query( q, arg→more_stuff ); /* change q use "more stuff" */
        update_db( db, q ); /* query updates database */

        return NULL;
}

void *thread2( void *arg )
{
        query_t *q = (arg ? (input_t *) arg→query : NULL); /* get query from arg */

        verify_query( q, arg→test ); /* read q to compare to "test" */
        read_db( db, q ); /* query reads from database */

        return NULL;
}
```

(a) (2pts) Which variable(s) are definitely shared between two threads, one running *thread1* and the other running *thread2*?

(b) (2pts) State how you would rewrite the code of *thread1* to enforce mutual exclusion to these variable(s). Use `mutex_lock` and `mutex_unlock`.

(c) (2pts) State how you would rewrite the code of *thread2* to enforce mutual exclusion to these variable(s). Use `mutex_lock` and `mutex_unlock`.

(d) (2pts) Suppose many threads run the code of *thread2*, and few threads run *thread1*. What is the problem with using a simple, mutual exclusion solution between *thread1* and *thread2*?

(e) (2pts) How would you rewrite the code (in abstract terms – just describe) in *thread2* to solve this problem?

answer: (a) db
(b) add mutex_locks around db in thread1
(c) add mutex_locks around db in thread2
(d) readers-writes problem – blocking progress of a reader in thread2 where no problem would occur
(e) Use mutex to protect number of readers, before and after mutex – write semaphore to protect db in thread2. Other answers are possible

15. (*8pts*) Answer the following questions about process creation and memory management.

(a) (2pts) Suppose a process has stack of 2 pages, a heap of 4 pages, and a code segment of 4 pages. When the process is created how many physical frames will it consume?

(b) (2pts) Suppose that the first thing that this process does is execute a `malloc` instruction from

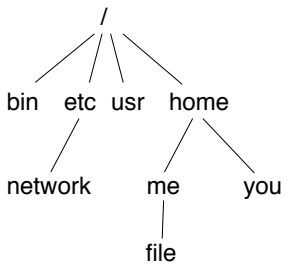
the code segment that allocates 2 pages of memory and assigns the address of this memory to a stack variable. How many physical frames will be allocated for executing this operation?

(c) (2pts) A new process is created by forking this process. If *copy-on-write* is used how many new physical frames must be allocated for the new process?

(d) (2pts) What happens when the original process writes to a page after the fork operation described in (c)?

answer: (a) None.
(b) 2. The code page must be mapped in and the stack page for the reference.
(c) None.
(d) A memory exception is taken (because the pages are marked readonly) and a new frame is allocated for this page for the original

16. (*10pts*) Consider this file system.



(a) (2pts) Specify the *absolute path name* to the file *file* and the *relative path name* from the directory `network`.

(b) (2pts) Suppose that one process opens *file* using the absolute path name and another process opens *file* using the relative path name. How many entries are added to the respective process's open file descriptor tables? How many inodes are added to the system-wide open file table?

(c) (2pts) How many disk accesses (assume nothing is cached) must be made to create an in-memory representation for `file` using a UNIX file system? Specify them. Assume one data block per directory to find the specified entry.

(d) (2pts) How are the first and second data blocks in a file added to a File Allocation Table (FAT)?

(e) (2pts) How are the first and second data blocks in a file added to a UNIX *inode*?

answer: (a) /home/me/file and ../../home/me/file.
(b) Per-process: one entry each for the newly opened file; only one entry in the system-wide open file descriptor table.
(c) 8. (1) the partition control block for the file system; (2-4) the root, home, and me directory data blocks; (5) the file control block for file. Plus, the directory inodes and data blocks are separate in UNIX. So, we access 2 blocks per directory.
(d) The FAT stores a table where each block number stores the next block number in the file. For the first block, we would find a block, and have the file control block reference this block. For the second block, it would be referenced by the FAT entry for the first block.
(e) An inode has several direct blocks for specifying data. The first two data blocks in the file would be assigned to the first two direct blocks in the file control block (inode).