CE-6
August 2014 QE

1. (45 pts, 3 pts each) For each of the statements below, indicate whether the statement is true or false. Explanations may allow partial credit for wrong answers.

   - A system call is triggered by the OS.

   - The child process will print the value of "local" as 2.

```
void main() {
   int local = 1;
   int pid = fork();
   if (pid == 0) {
      local = 2;
      execlp("/bin/ls", "ls", NULL);
      printf("The child process at after exec: local = %d \n", local);
   } else {
      local = 3; wait(NULL);
      exit(0);
   }
 }
```

   - Assume synchronization primitives are correctly supported by the OS. If a multithreaded user program using the synchronization primitives is known to execute correctly on a time-shared (i.e. preemptively scheduled) uniprocessor, it may not execute correctly on a multi-processor.

   - A binary semaphore can only be used by two processes/threads for synchronization between them.

   - Condition variables alone can be used to solve synchronization problems, for example, the producer-consumer problem.

- The use of a log stored in fast, non-volatile RAM increases the reliability but not the performance of a file system.
- When a file is being accessed by a process, i.e., read or written, the current position within the file where it is being accessed is stored in the on-disk copy of its inode.
- A multi-level indexed file descriptor permits faster random access than a file descriptor with a single level of index.
- In the Unix File System, user processes can direcly write directories using the `write()` system call just like ordinary files.
- In the Unix File System, the name of a (non-directory) file is stored in its parent directory's inode.
- In the Unix File System, the name of a directory is stored in its inode.
- In the UNIX file system, after file system initialization, there is a fixed upper limit on how large files can be.
- In the Unix File System, the operation "ls -l" is potentially more expensive and hence slower than "ls".

- In File Systems, the buffer cache is implemented purely in software, unlike demand paging, which is implemented jointly in software and with hardware support (i.e. MMU).

- It is slower to write 1 block in a RAID Level 5 organization with 5 disks than in a RAID Level 1 organization with 2 disks (i.e., mirroring only).

2. (Deadlock - 20 pts)

   (a) (4 pts) List the four conditions required for deadlock to occur. Briefly explain each.

   (b) (5 pts)
   Consider the three processes below, (proc1 through proc3,) each of which competes for six shared resources, (A through F):

```
proc1()                 proc2()                 proc3()
{                       {                       {
   while (1)               while (1)               while (1)
   {                       {                       {
     lock(&D);               lock(&C);               lock(&A);
     lock(&E);               lock(&F);               lock(&B);
     lock(&B);               lock(&D);               lock(&C);
     // Use D, E,            // Use C, F,            // Use A, B,
     //   and B              //   and D              //   and C
     unlock(&D);             unlock(&C);             unlock(&A);
     unlock(&E);             unlock(&F);             unlock(&B);
     unlock(&B);             unlock(&D);             unlock(&C);
   }                       }                       }
}                       }                       }
```

   Could the three processes enter into deadlock? If so draw the resource allocation graph with the cycle that represents this deadlock. (Remember: Processes are circles, resources rectangles.)

Write in Exam Book Only

(c) (5 pts) (Deadlock preventation) Can you think of a way of reordering the statements in the above processes to prevent deadlock from happening? Note that (1) the resouce usage of each process should not be changed – each process still needs to use its corresponding TWO or THREE resources simultaneously in each of its critical section; and (2) each process should only hold the locks for the resources used in each critical section before enter that critical section.

(d) (6 pts) (Deadlock avoidance) Consider the version of the dining-philosophers problem in which the chopsticks are placed at the enter of the table and any two of them can be used by a philosopher. Assume that *requests for chopsticks are made one at a time.* Describe a simple rule for determining whether a particular request can be satisfied without causing deadlock given the current allocation of chopsticks to phisolophers. Assume there are N philosophers and a total of N chopsticks. Note you solution should not only allow one phisolopher to eat at a time.

3. (Page Replacement - 20%)

   (a) (5%) A page replacement algorithm belongs to the class *stack algorithms* if it can be shown that given any sequence of virtual page references, the set of pages that would be in memory with n physical pages is always a subset of the set of pages that would be in memory with $n+1$ physical pages. Which of the following page replacement algorithms are stack algorithms? (1) OPT (2) LRU (3) FIFO (4) LIFO (5) LFU (Least Frequently Used)

   (b) (5%) Consider the following virtual page reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5, 1, 2. How many page faults will there be under the LRU replacement algorithm on Tom's PC which has 4 physical pages? How many page faults will there be on Jerry's machine which has 3 physical pages?

   (c) (5%) Assuming the LRU replacement algorithm again. Tom has upgraded his machine to 16 MB (4 thousand pages), and Jerry has upgraded his machine to 12 MB (3 thousand pages). Can you come up with a reference string that contains more than 4000 references and still exhibits the above relative performance behavior, i.e., the ratio of the numbers of page faults on the two machines remain the same?

(d) (5%) Going back to the reference string in (b). What is the minimum number of page faults for an optimal page replacement strategy, assuming 4 physical pages and assuming 3 physical pages, respectively?

4. (File System Interface - 15 pts)

   (a) (8 pts) Compare the read/write model of file access, and the memory-mapped file model of file access. What potential drawbacks of the read/write model are addressed by the memory-mapped model?

   (b) (7 pts) The generic version of the system call used to set up a memory-mapped file looks like this:
   `mmap(void *start_address, size_t length, int protection, int flags, int fd, off_t offset)`
   After an invocation of mmap, with length = 16385 bytes, successfully returns, how many physical pages have been allocated to the virtual address segment being mapped?