Name_____ ID#_____

# MID TERM EXAMINATION

ICS 143 Principles of Operating Systems
Winter Quarter
February 16, 2012

Name_____

In recognition of and in the spirit of the University of California Honor Code,
I certify that I will neither give nor receive un-permitted aid on this exam.

Signature_____

---

***Instructions:***

1. **Do not** start this test until the proctors say you can begin.
2. There are 100 points possible on this test.
3. Your exam should have 6 pages. Please check to see if they are all there.
4. Place all class materials under your desk. This is a **CLOSED** book exam.
5. Quickly browse each problem before beginning the test. Note that some questions are worth more points than others.
6. Show only the work for which you would like to receive partial credit on the exam. You may use the reverse sides of sheets if you need more room.
7. If you have a question about any problems, raise your hand until a proctor can help you.
8. Be brief in your answers.

The exam consists of the following three parts:

- Part A: 15 true/false questions worth 30 points. For each of these questions:
  - A correct answer will get you **2** points.
  - An incorrect answer will get you **0.5 negative** points.
  - A question left blank will get you **0** (zero) points
- Parts B, C, and D: There is no negative grading for these questions.
- The time limit for the exam is 1 hour.

| Part | Max Points | Score |
|---|---|---|
| A | 30 | |
| B | 15 | |
| C | 25 | |
| D | 30 | |

Name_____ ID#_____

## Part A [True/False, 30 pts]

Mark the following (T-F) questions using T for True, and F for False.
***Write your answers in the table provided on the next page.***

1. Multiprogramming makes efficient use of the CPU by overlapping the demands for the CPU and I/O devices from various users.

2. Privileged instructions cannot be executed in monitor mode.

3. An interrupt vector contains the saved program counter values of interrupted user programs.

4. Direct memory access (DMA) requires a special controller that facilitates the transfer of blocks between the I/O device and main memory.

5. If a kernel is single-threaded, system calls from any thread can block the entire task.

6. We can prevent users from accessing other users programs and data by introducing base and limit registers that hold the smallest physical memory address and the size of range of the user program respectively.

7. An operating system can have both microkernel and virtual machine structures.

8. There are solutions to synchronization problems that can be implemented using monitors, but cannot be implemented using semaphores.

9. Long-term scheduler (or job scheduler) selects which processes should be brought into the ready queue and balances load for better throughput.

10. When designing a multithreaded application, you must use synchronization primitives to make sure that the threads do not overwrite each other's registers.

11. Small time slices always improve the average turnaround time of all the processes in a system.

12. An OS uses multilevel feedback queues for process scheduling and a process in this OS needs 40ms for execution. If the first queue uses 2ms time quantum and for each level time quantum of the level is time quantum of previous level plus 5ms, a process will finish processing in fifth ($5^{th}$) level. (Suppose that the level with 2ms quantum time is the first queue).

Name_____ ID#_____

13. This pseudo-code for a process and TestAndSet function, without any other assumptions, solves the critical section problem between N processes.

```
function TestAndSet(boolean lock)
{
        boolean initial = lock;
        lock = true;
        return initial;
}

Initial value of lock = FALSE;
```

```
function Process(int N)
{
        Repeat
                while(TestAndSet (lock)) no-op;
                critical section
                lock = TRUE;
                remainder section
        until FALSE
}
```

14. Priority Inheritance is a scheme to circumvent priority inversion where the higher priority process temporarily assumes a lower priority.

15. Consider the following pseudo-code. This code solves the critical section problem for two processes: True or False?

```
shared boolean flag[2];
flag[0]=flag[1]=false;
proc(int i){
        while(true){
                while(flag[(i+1) mod 2]==true);
                flag[i]=true;
                critical-section;
                flag[i]=false;
        }
}
```

***Fill in the answers for true/false questions into this table.***

| | | | |
|---|---|---|---|
| **1** | T | **9** | T |
| **2** | F | **10** | F |
| **3** | F | **11** | F |
| **4** | T | **12** | T |
| **5** | T | **13** | F |
| **6** | T | **14** | F |
| **7** | T | **15** | F |
| **8** | F | | |

Name_____ ID#_____

## Part B [Processes and Threads 15 pts]

**Question 1: Mapping of Kernel/User Threads (6 pts).**

Which of the following instructions should be allowed only in kernel mode? State whether it should be kernel-only or could be executed in User-mode (circle one) and why.

I. Disable all interrupts.

Kernel User

Why?

II. Multiply 2 matrices.

Kernel User

Why?

III. Set the time-of-day clock.

Kernel User

Why?

**Question 2: Concurrent Execution of Processes (9 pts).**

a) State and briefly explain the necessary and sufficient conditions for a deadlock(4 pts)

mutual exclusion, hold & wait, no preemption, circular wait.
--see slides

Name_____ ID#_____

b) What could the output of the concurrent execution of process A and process B be? (State all possible outputs) (5 pts)

```
int x=0;                          "initialization"
int y=0;
```

| Process A | Process B |
|---|---|
| while(x==0) {do-nothing}; | printf("b"); |
| printf("a"); | x=1; |
| y=1; | while(y==0){do-nothing}; |
| y=0; | printf("c"); |
| printf("d"); | |
| y=1; | |

**badc**
**bacd**

Name_____ ID#_____

## Part C [CPU Scheduling 25 pts]

]

**Question C.1: Short Answer CPU Scheduling Questions (10 pts).**

a) Briefly describe one way to approximate the information required to implement the shortest-remaining-time-first algorithm. What characteristic of program behavior makes your proposed strategy result in a good approximation of the future?

exponential averaging
near-future behaviour similar to near-past behaviour

b) Consider the execution of two processes P1 and P2, with the following CPU and I/O burst times.

| P1 | P2 |
|---|---|
| CPU - 3 | CPU - 4 |
| Net - 4 | Disk - 3 |
| CPU - 2 | CPU - 3 |
| Disk - 3 | Net - 3 |

Each row shows the required resource for the process and the time that the process needs that resource. For example "Net 3" in forth row says that P2 needs network card for 3 time units.

If P2 arrives 2 time units after P1 and the scheduling policy is non-preemptive SJF (Shortest Job First), then calculate the finish time for each process and the CPU idle time in that duration.

finish for P1: 12
finish for P2: 16
idle: 4 or 1 accepted

Name_____ ID#_____

**Question C.2: CPU Scheduling (15 pts).**

Consider the following set of processes:

| Process ID | Arrival Time | CPU Burst Time |
|---|---|---|
| P1 | 0 | 2 |
| P2 | 2 | 6 |
| P3 | 1 | 1 |
| P4 | 3 | 4 |
| P5 | 3 | 3 |

a) Show the scheduling order for these processes under three policies: First Come First Serve (FCFS), Optimal Waiting Time (OWT), Round-Robin (RR) with time slice quantum = 1 time unit. Provide schedules in following table by writing process ID for each CPU time slot.

| Time Slot | FCFS | Round Robin (q==1) | OWT |
|---|---|---|---|
| 0 | | | |
| 1 | | | |
| 2 | | | |
| 3 | | | |
| 4 | | | |
| 5 | | | |
| 6 | | | |
| 7 | | | |
| 8 | | | |
| 9 | | | |
| 10 | | | |
| 11 | | | |
| 12 | | | |
| 13 | | | |
| 14 | | | |
| 15 | | | |

b) For each process in each schedule above, calculate the wait time.

| Scheduler | P1 | P2 | P3 | P4 | P5 |
|---|---|---|---|---|---|
| FCFS Wait | | | | | |
| | | | | | |
| OWT Wait | | | | | |
| | | | | | |
| RR Wait | | | | | |
| | | | | | |

Name_____ ID#_____

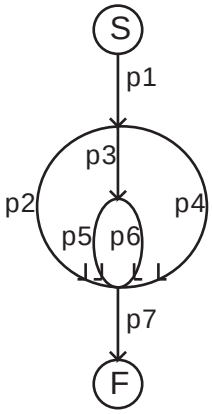
## Part D [Synchronization, 30 pts]

**Question 1: Basic Synchronization (15 pts).**

a) Consider the following flow graph (representing 7 processes)

Use the P(), V() representation to describe the synchronization. You can denote the code executed by the process using the notation "body".

*Example: A process Pi that waits on semaphore s1, executes and finally signals another semaphore s2 is represented by "Pi : P(s1) ; body ; V(s2)"*




P1: body; v(s1); v(s1); v(s1)
P2: p(s1); body; v(s2)
P3: p(s1); body; v(s3); v(s3)
P4: p(s1); body; v(s4)
P5: p(s3); body; v(s5)
P6: p(s3); body; v(s6)
P7: p(s2); p(s4); p(s5); p(s6); body

b) Consider the (following) Bakery Algorithm, a solution of the Critical Section Problem for n processes. Fill in the blanks.

```
repeat
    choosing[i] := true;
    number[i] := max(number[0], number[1],…,number[n-1]) +1;
    choosing[i] := false;
    for j := 0 to n-1
      do begin
            while (choosing[j])
                  do ……no-op……
                  ……
            while number[j] <> 0 and …(number[j],j) < (number[i],i)……
                  do no-op;
      end;
       critical section
     ……number[i] = 0;……..
       remainder section
until false;
```

Name_____ ID#_____

**Question 2: Semaphores (15 pts).**

a) Consider a coke machine that has 10 slots. The producer is the delivery person and the consumer is the student using the machine. We use the following three semaphores:

- semaphore mutex
- semaphore fullBuffer /* Number of filled slots */
- semaphore emptyBuffer /* Number of empty slots */

The following operations are available on the semaphores: wait(semaphore s), signal(semaphore s).

Given functions (see code) delivery_person() and student():

- What will be the initial values of the semaphores?
- Write a solution that guarantees mutual exclusion and no deadlocks. (Part of the code for student () function has been written)

```
/* Initialize */
#define NUM_SLOTS  10

semaphore mutex       = 1
semaphore fullBuffer  = x      //x+y must equal 10
semaphore emptyBuffer = y      //x+y must equal 10
```

```
delivery_person() {

    wait(emptyBuffer)
    wait(mutex)

    put_1_coke_in_machine();

    signal(mutex)
    signal(fullBuffer)

}
```

```
student() {

    wait(fullBuffer);
    wait(mutex);
    take_1_coke_from_machine();

    signal(mutex)
    signal(emptyBuffer)

}
```

b) If the two wait() functions inside the student() section are interchanged [i.e., wait(fullbuffer) and wait(mutex) are interchanged], will your solution to the previous question will still be correct ? If not, explain your reason?

**No. Deadlock.**

**Good Luck**