

Instructor: Luan Duong, Ph.D.

Acknowledgement: By doing these practice questions, you agree to the followings: Practice questions are **ONLY** for your practice. This set of practice questions **ONLY** serve **additional exercises** for you better prepare for your final. It is NOT a template as well. Also, the content of these questions **may not cover** the whole content of the final that you need to prepare for. Please follow the final review for the coverage. Please try out the questions on your own first before asking the TAs/. If you need additional questions, one thing you could do is to try changing the numbers from the previous questions or the lecture notes.

Q0: Important concepts: Again, please refer to the final review slides for what to expect for the finals. The **important** topics covered:

- + Memory structure: Virtual Memory: Dynamic Relocation, Segmentation, Paging and Swapping

- + Memory Hierarchy (Locality, Free memory management, Page replacement, Caching: Remember how to do direct mapped cache, E-way associative cache)

- + Concurrency: Locks, Condition Variables, Semaphores, Deadlocks (RAG), Classical Concurrency problems (Bounded Buffer, Reader-Writer, Dining Philosophers, Sleeping Barber)

- + I/O Devices: Disk, Disk Access time, Disk Scheduling, RAIDs, I/O devices (Interrupt, DMA)

- + Process/Thread Scheduling (FCFS/FIFO, STF, STCF, Preemptive, Non-preemptive, Round-robin, Multi-level Feedback Queue)

Q1: List all conditions for deadlock and write a one-line description of each.

Q2: Complete the correct implementation of dining philosophers using semaphores. Your answers should either be down() or up().

```
Initialize semaphores chopstick[3] to 1 //Each chopstick is available
```

```

function philosopher(id):
    left = id                // Philosopher's left chopstick
    right = (id+1)%3         // Philosopher's right chopstick
    while not done eating:
        print("Philosopher", id, "is thinking")
        //Pick up chopsticks
        if id is even then:
            _____
            _____
        else:
            _____
            _____
        Priont("Philosopher", id, "is eating")
        up(chopstick[left])
        up(chopstick[right])

```

Q4: What is swapping? Explain it in a few sentences and explain why we need it and how it works.

Q5: Look at the following code snippets. Tell me if there is an error and if there is then give a brief description of how you may fix it:

Code 1:

```

pthread_mutex_t m = ...

pthread_cond_t c = ...

...

Thread0:                                Thread1:

```

<code>pthread_mutex_lock(&amp;m)</code>	<code>pthread_mutex_lock(&amp;m)</code>
<code>pthread_cond_signal(&amp;c)</code>	<code>pthread_cond_wait(&amp;c, &amp;m)</code>
<code>...</code>	<code>...</code>
<code>pthread_mutex_unlock(&amp;m)</code>	<code>pthread_mutex_unlock(&amp;m)</code>

Code 2:

Thread0:	Thread1:
<code>pthread_mutex_lock(&amp;m)</code>	<code>pthread_mutex_lock(&amp;d)</code>
<code>pthread_mutex_lock(&amp;d)</code>	<code>pthread_mutex_lock(&amp;m)</code>
<code>...</code>	<code>...</code>
<code>pthread_mutex_unlock(&amp;m)</code>	<code>pthread_mutex_unlock(&amp;m)</code>
<code>pthread_mutex_unlock(&amp;d)</code>	<code>pthread_mutex_unlock(&amp;d)</code>
<code>...</code>	<code>...</code>

Code 3:

```
pthread_mutex_t m = ...
pthread_cond_t c = ...
int trueOrFalse TF = ...
...

Thread1:
    pthread_mutex_lock(&m)
    if (TF!=1){
        pthread_cond_wait(&c, &m)
    }...
```

Q6: Define each of the following components/terms

Sector:

Track:

Platter:

Head:

Seek Time:

Starvation:

Q7: Name and describe any deadlock prevention method.

Q8: Giving the following different cases, please state whether deadlock can happen, or not. If yes, draw the resource allocation graph (RAG) to describe and show the “cycle” for deadlock. Assume 1 lock = 1 type of resource

Case 1: [Code shown]

**Process 0:**

```
lock1.acquire();  
lock2.acquire();  
lock1.release();  
lock2.release();
```

**Process 1:**

```
lock1.acquire();  
lock2.acquire();  
lock1.release();  
lock2.release();
```

P0: try to grab lock 1 and lock 2 in order: 1 then 2.

P1: try to grab lock 1 and lock 2 in order: 1 then 2.

Case 2:

**Process 0:**

```
lock1.acquire();  
lock2.acquire();  
lock1.release();  
lock2.release();
```

**Process 1:**

```
lock2.acquire();  
lock2.release();  
lock1.acquire();  
lock1.release();
```

P0: try to grab lock 1 and lock 2 in order: 1 then 2.

P1: try to grab lock 2, then release it, then try to grab lock 1, then release it.

Q9 [Scheduling]: Giving the following processes with length and arrival time as follows:

Process	Length	Arrival time
P1	8	0
P2	4	0.4
P3	1	1.0

- What is the turnaround time for each process with FCFS scheduling algorithm? What is the average turnaround time?
- What is the response time for each process with SJF scheduling algorithm? What is the average response time?
- What is the average response time for process with FCFS and SJF?
- Do you notice any problem with this? Try to do b. again, but this time we choose to idle the CPU for the first unit and then schedule SJF again for those processes. Recalculate the average turnaround time and the average response time in this case.

Q10: Consider the following page reference string: 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

How many page faults would occur for the following replacement algorithms, assuming one, two, three, four, five, six and seven frames (respectively)?

Remember that all frames are initially empty, so your first unique pages will cost one fault each.

1. LRU Replacement
2. FIFO Replacement
3. Optimal replacement.