a. mutex - Mutexes happens to critical sections of code where only one process can touch at a time.

b. critical section – a piece of code that accesses a shared resource that must not be accessed by more than one thread of execution

c. semaphore - var that provides useful abstraction for controlling access by multiple processes to a common resource.

d. deadlock – situation which two or more competing actions are each waiting for the other to finish.

e. starvation – situation where a low priority action may never be executed.

f. thread-safe – code is thread-safe if it functions correctly during simultaneous execution by multiple threads.

g. race condition – when system attempts two or more operations at same time but needs them to be done in a sequential order.

2. Why would you use process-based (rather than thread-based) parallelization for building a web system? Give two examples.

1) Because processes are independent they contain their own state info, address spaces. This keeps each call for different users separate. 2) Threads typically do not need to be performed in a linear manner, this could be bad when dealing with user requests from a web server.

3. Is worrying about "race conditions" necessary in your average single-threaded application?

No, because in a single threaded application everything is done sequentially, you wont need to worry about multiple operations done at the same time.

4. Why do modern OS not worry too much about deadlock?

Because the OS preempts the CPU eliminating one of the four preconditions for deadlock.

5. What is an "atomic" operation? Why are these necessary?

Operation which a processor can simultaneously read a location and write it in the same bus. This prevents any other processor from writing or reading memory until it is complete.

6. Why would we want to keep critical sections as short as possible?

To avoid long delays for servicing other interrupts

7. How could you prevent deadlock in the "Dining Philosophers" problem?

If we have a mediator which all philosophers must request permission, he can oversee how many forks are given out and make the other wait until the previous is done.

8. Give a simple way to keep starvation from being an issue in a multithreaded system.

Implement a Fair Lock which utilizes a queue, allowing the top of queue to use the CPU, and the back waits