Exam Monitor-problem examples

Q2. (35 points)
Memory Allocation Monitor: suppose there are two operations: ***request(amount)*** and ***release(amount)*** where *amount* is a positive integer. When a process calls *request*, it delays until at least *amount* free pages of memory are available. Processes return *amount* pages to the free pool by calling *release*. Each time amount is generated randomly.

The memory has a capacity of **C** equal to1024 pages. A process shouldn't require more than **allowed_limit** (initialized to 64) pages. If amount is greater than allowed_limit, then the process should execute a new **request(amount)**.
a) Develop a monitor to synchronize processes. You can use two methods named:
request(amount) and **release(amount)** (draw the monitor picture, give the thread execution pseudo-code, give service methods' pseudo-code, initialization, condition variables)
Use the concept of monitors: cond var have names, are implemented as queues with FIFO policy.You can use any of the two signal policies but specify which one you use. Don't use ***notifyAll****.*

Q3. (35 points) Monitors
3.1 Synchronization of automobile traffic through a one-way tunnel.
Suppose a two-way (two-lane) north-south road contains a one-lane tunnel. A southbound (or northbound) car can use the tunnel only if when arrives at the entrance of the tunnel, there are no oncoming cars in the tunnel. When a car approaches the tunnel, it notifies the controller computer by calling a method **northboundArrival** or **southboundArrival** depending on the direction of travel of the car. When a car exits the tunnel, it notifies the tunnel controller computer by calling a function named **depart.** The direction of the traffic should be mentioned.

a) Develop a monitor to synchronize the automobile traffic (draw the monitor picture, give the thread execution sequence, give service methods' pseudo-code, initialization, condition variables).

Use the concept of monitors: cond. var. have names, are implemented as queues with FIFO policy. You can use any of the two signal policies but ***specify which one you use****. Don't use* ***notifyAll****. Briefly comment your implementation.*

b) Is it possible for the pseudo-code of your service methods to support the other (not chosen) signal policy? If no, explain why not.

Q1: (35 points)

Modified Producer Consumer problem:

Assume one producer process and **n** consumer processes share a bounded buffer having **b** slots. The producer **deposits** messages in the buffer; consumers **fetch** them. Every message deposited by the producer is to be eventually received by all **n** consumers. Furthermore, each consumer is to receive the messages in the order they were deposited. However, consumers can receive messages at different times. For example, one consumer could receive up to **b** more messages than another if the second consumer is slow.

a) Develop a monitor to synchronize the two types of threads (draw the monitor picture, give the thread execution pseudo-code, give service methods' pseudo-code, initialization, condition variables). Extensively comment your implementation.

*Use the concept of monitors: condition variables have names, are implemented as queues with FIFO policy. You can use any of the two signal policies but specify which one you use. Don't use **notifyAll**.*

b) Is it possible for the pseudo-code of your service methods to support the other (not chosen) signal policy? If no, explain why not.