UNIVERSITY OF PORTSMOUTH

Operating Systems and Internetworking

OSI – M30233 (OS Theme)

Week 4 Appendix - Notification and Monitors

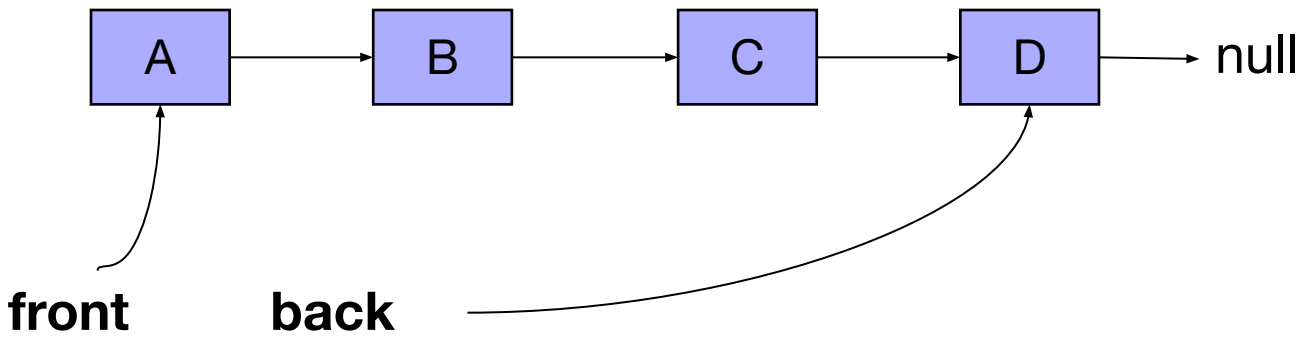
Dr Tamer Elboghdadly

Notification Detailed Example

- In an appendix to the week 2 lecture we considered a simple queue data structure, shared by threads, illustrating a race condition that may arise when multiple threads try to add items of data to the back of the queue simultaneously.
- In this week's appendix, we develop this example further to illustrate issues about *notification* between threads.

Basic Queue Data Structure

- Queue implemented as a linked list with pointers to front and back of list.
- Items added to back, removed from front.
- State of queue after four items – A, B, C, D – have been added:



Thread-safe queue

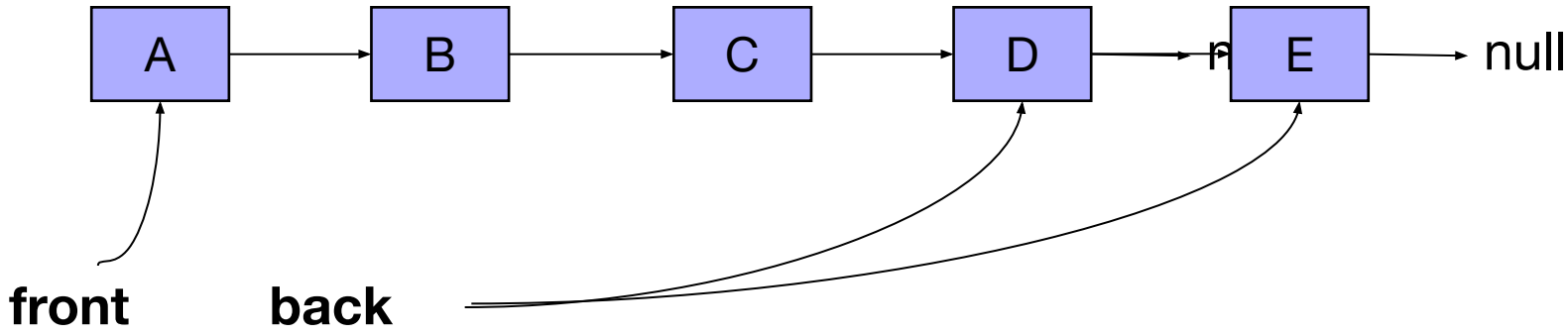
```
synchronized void add (Object data) {
    if (front == null) {  // queue empty
        front = new Node(data) ;
        back = front ;
    }
    else {
        back.next = new Node(data) ;
        back = back.next ;
    }
}
```

```
synchronized Object rem() {
    if (front == null)  {  // queue empty
        return null ;
     else {
        Object result = front.data ;
        front   = front.next ;
        return  result ;
    }
}
```

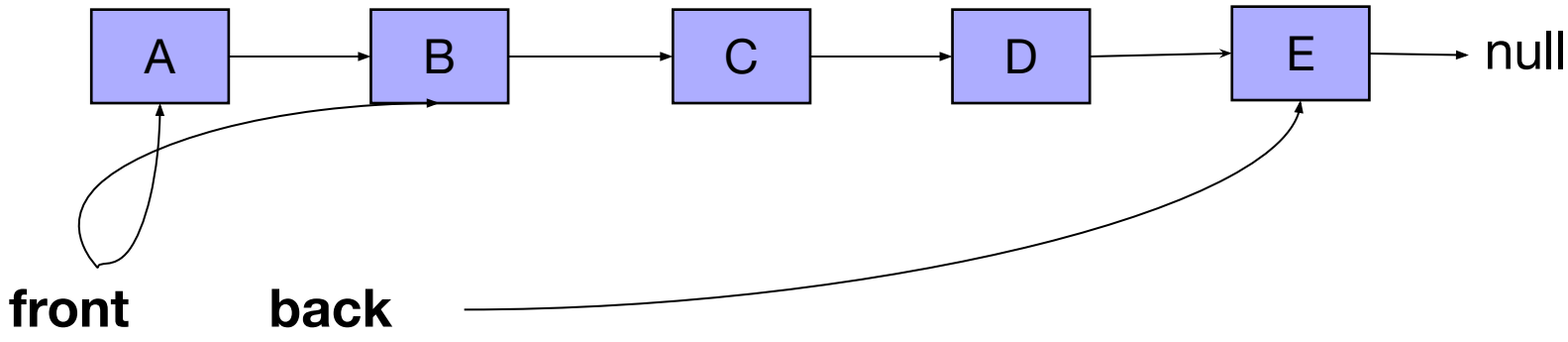
UNIVERSITY OF PORTSMOUTH

Operations on Queue

- Adding item to back of queue:



- Removing item from front of queue:



Methods on queue

```
void add (Object data) {
    if (front == null) {  // queue empty
        front = new Node(data) ;
        back = front ;
    }
    else {
        back.next = new Node(data) ;
        back = back.next ;
    }
}
```

```
Object rem() {
    if (front == null)  {  // queue empty
        return null ;
     else {
        Object result = front.data ;
        front   = front.next ;
        return  result ;
    }
}
```

UNIVERSITY OF PORTSMOUTH

Comments

- Note the rem() method removes the first node from the queue, and returns the data object contained in the node.
  - If the queue is empty, it returns null.
- If add() or rem() were called by different threads, we would have potential *race conditions*.
- Easily fixed if system supports monitors. In Java we just make methods *synchronized*.

Blocking Queue

- In a multi-threaded context, rather than rem() on an empty queue returning null, it makes sense for the rem() operation to *block* (wait) until another thread adds an item to the queue.
  - Our queue then becomes a simple *message queue*.
- For a first attempt we might try to use a semaphore that is increased at the end of each add() and decreased at the start of each rem().
- Introduce semaphore, *S*, initialized to zero.

Attempt at Message Queue

```
synchronized void add (Object data) {
    if (front == null) {  // queue empty
        front = new Node(data) ;
        back = front ;
    }
    else {
        back.next = new Node(data) ;
        back = back.next ;
    }
    V(S) ;
}
```

```
synchronized Object rem() {
    P(S) ;
    Object result = front.data ;
    front   = front.next ;
    return result ;
}
```

A Deadlock!

- If queue is empty and a thread calls rem(), statement P(0) blocks.
- Thread is now stuck inside a synchronized method. No other thread can "enter the monitor" (i.e. call another synchronized method) till first thread exits rem().
- But the only way P(S) will unblock is if another thread can execute add() to increase the semaphore – can now never happen!
- Simple example of a *deadlock* situation.

“Condition Variables”

- Languages like Java and C# that implement the *monitor paradigm* provide alternate primitives for notification.
- These include wait() and notify() (or notifyAll()) methods that behave something like the P() and V() operations on a semaphore used for notification.
- Besides blocking a thread, the wait() method *releases* the mutual exclusion lock it holds, so other threads can rectify the condition causing the wait, then notify any waiting threads.

Message Queue in Java

```
synchronized void add (Object data) {
    if (front == null) {  // queue empty
        front = new Node(data) ;
        back = front ;
        notifyAll() ;
    }
    else {
        back.next = new Node(data) ;
        back = back.next ;
    }
}
```

```
synchronized Object rem() {
     while (front == null)  { // queue empty
         wait() ;
     }
    Object result = front.data ;
    front   = front.next ;
    return result ;
}
```

UNIVERSITY OF PORTSMOUTH

Questions?