

# EE445M/EE380L.12

## Embedded and Real-Time Systems/ Real-Time Operating Systems

### Lecture 5: Semaphores, Deadlocks, Debugging, Testing

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

1

## Graduate Projects Ideas

1. Extend the OS with more features (do this if two students in group)
  - Efficient with 20 to 50 threads
  - Multiple cores (real-time scheduling algorithms & implementation)
  - Multiple Mailboxes, FIFOs
  - Multiple periodic/edge-triggered interrupts
  - Path expressions, bankers algorithm,
  - Semaphores with timeout, priority inheritance/ceiling (algorithms & implementation)
2. Make your Lab3 OS portable and port to another platform
  - First implement Lab3 on another architecture (each students does their own)
  - Rewrite OS into two parts: common OS.c (maximize), separate CPU.c per architecture (minimize)
3. Design and test a DMA-based eDisk driver for the LaunchPad board (one-person project)
  - Compare and contrast your Lab5 to FAT
4. Design and test a DMA-based camera driver for the LaunchPad board
  - See LM3S811 example [http://www.ece.utexas.edu/~valvano/arm/Camera\\_811.zip](http://www.ece.utexas.edu/~valvano/arm/Camera_811.zip) (one person project)
  - Implement object/lane detection/recognition and lane keeping (self-driving car) (two or more students)
5. Write your own memory management
  - Heap, malloc and free (one-person project)
  - Use of Memory Protection Unit (MPU) on LaunchPad
  - Virtual memory, paging on a different platform (two or more students)
6. Networking, Internet-of-Things (IoT)
  - Port a lightweight TCP/IP stack onto board (e.g. lwIP using external WiFi module via UART)
  - Robots communicate with each other/base station, control robot remotely (vehicle-to-vehicle / vehicle-to-)
7. Use of Launchpad/OS in other applications
  - Energy harvesting OS with checkpointing for frequent power loss
  - OS safety/security mechanisms

Due end of Feb

Level of complexity depends on size of group

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

2

# Semaphores

Edsger Dijkstra,  
UT Austin CS 1984-2000

- *P()* or *wait()*
  - Dutch word *proberen*, to test
  - *probeer te verlagen*, try to decrease
  - **OS\_Wait**      **OSSemPend**
- *V()* or *signal()*
  - Dutch word *verhogen*, to increase
  - **OS\_Signal**    **OSSemPost**

Reference Book, Chapter 4

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

3

# Semaphore Meaning

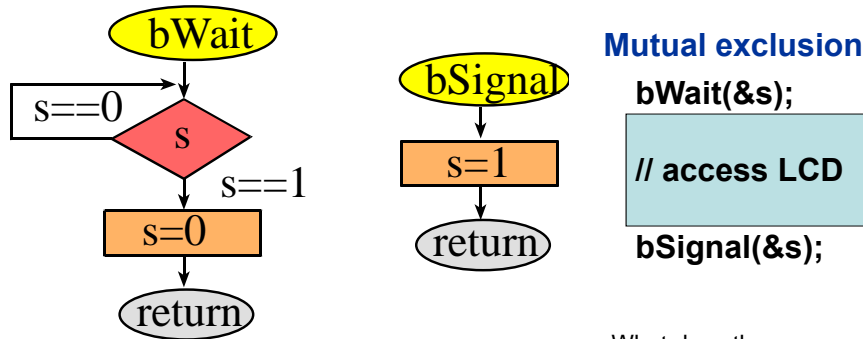
- Counting semaphore
  - Number of elements stored in FIFO
  - Space left in the FIFO
  - Number of printers available
- Binary semaphore (= mutex = flag)
  - Free (1), busy (0)
  - Event occurred (1), not occurred (0)

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

4

## Spin-Lock Binary Semaphore



How do we use this to solve critical sections?  
 Why is this a good solution for critical sections?

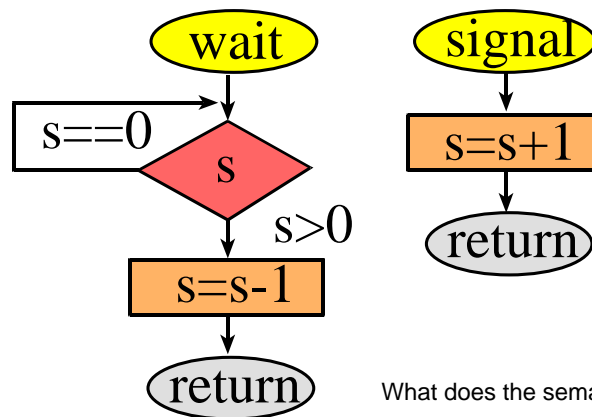
What does the semaphore mean?  
 What would be a better name for  $s$ ?  
 What about atomic?

Lecture 5

J. Valvano, A. Gerstlauer  
 EE445M/EE380L.12

5

## Spin-Lock Counting Semaphore



What does the semaphore mean?  
 What about atomic?

Lecture 5

J. Valvano, A. Gerstlauer  
 EE445M/EE380L.12

6

# Spin-Lock Semaphores

<pre> OS_Wait ;R0 points to counter LDREX  R1, [R0] ; counter SUBS   R1, #1 ; counter -1, ITTT   PL ; ok if &gt;= 0 STREXP R2,R1,[R0] ; try update CMPPL  R2, #0 ; succeed? BNE    OS_Wait ; no, try again BX     LR  OS_Signal ; R0 points to counter LDREX  R1, [R0] ; counter ADD    R1, #1 ; counter + 1 STREXP R2,R1,[R0] ; try update CMP    R2, #0 ; succeed? BNE    OS_Signal ;no, try again BX     LR         </pre>	<pre> void OS_Wait(long *s) {   DisableInterrupts();   while((*s) &lt;= 0){     EnableInterrupts();     DisableInterrupts();   }   (*s) = (*s) - 1;   EnableInterrupts(); }  void OS_Signal(long *s) {   long status;   status = StartCritical();   (*s) = (*s) + 1;   EndCritical(status); }         </pre>
---	--

**LDREX**  
**STREX**

Program 4.11

Cortex-M3/M4F Instruction Set, pg. 50

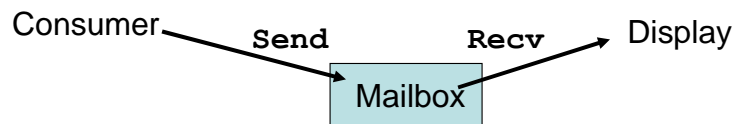
# Mailbox

## MailBox\_Send(...)

- **bWait(&BoxFree)**
- **Put data into Mailbox**
- **bSignal(&DataValid)**

## MailBox\_Recv(...)

- **bWait(&DataValid)**
- **Retrieve data from Mailbox**
- **bSignal(&BoxFree)**



What do the semaphores mean?

What are the initial values?

What if we remove **bWait(&BoxFree)** and **bSignal(&BoxFree)**?

# FIFO, Queue, or Pipe

## FIFO\_Put

Wait(&DataRoomLeft)  
 Disable Interrupts  
 Enter data into Fifo  
 Enable Interrupts  
 Signal(&DataAvailable)

## FIFO\_Get

Wait(&DataAvailable)  
 Disable Interrupts  
 Remove data from Fifo  
 Enable Interrupts  
 Signal(&DataRoomLeft)

## FIFO\_Put

Wait(&DataRoomLeft)  
 bWait(&Mutex)  
 Enter data into Fifo  
 bSignal(&Mutex)  
 Signal(&DataAvailable)

## FIFO\_Get

Wait(&DataAvailable)  
 bWait(&Mutex)  
 Remove data from Fifo  
 bSignal(&Mutex)  
 Signal(&DataRoomLeft)

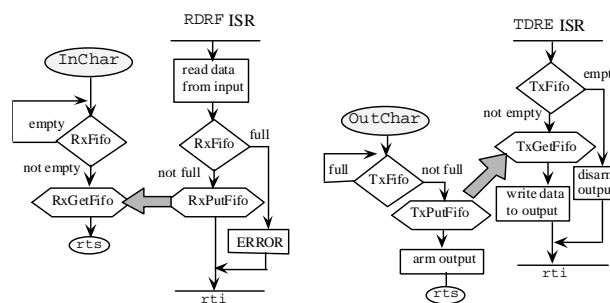
Lecture 5

What do the semaphores mean?  
 What if the FIFO never fills?

9

# No Background Wait

- Redo Mailbox if **Send** in background
- Redo Fifo if **Put** in background (RX)
- Redo Fifo if **Get** in background (TX)



Lecture 5

J. Valvano, A. Gerstlauer  
 EE445M/EE380L.12

10

## Cooperative Spin-Lock

**Cooperative spin-lock**

Could be implemented with a catch and throw

**Regular spin-lock**

Why would you want a timeout error?  
How would you implement timeout?

```

if (OS_Wait(&free, T100ms)) {
    // use it
    OS_Signal(&free);
} else {
    // error
}
    
```

Lecture 5 11

## Cooperative Semaphores

```

void OS_Wait(long *s){
    DisableInterrupts();
    while((*s) <= 0){
        EnableInterrupts();
        OS_Suspend();
        DisableInterrupts();
    }
    (*s) = (*s) - 1;
    EnableInterrupts();
}

void OS_Signal(long *s){
    long status;
    status = StartCritical();
    (*s) = (*s) + 1;
    EndCritical(status);
}
    
```

Let other thread run

Do an experiment of Lab 2 with and without cooperation

Lecture 5 12

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

## Blocking Semaphore (Lab 3)

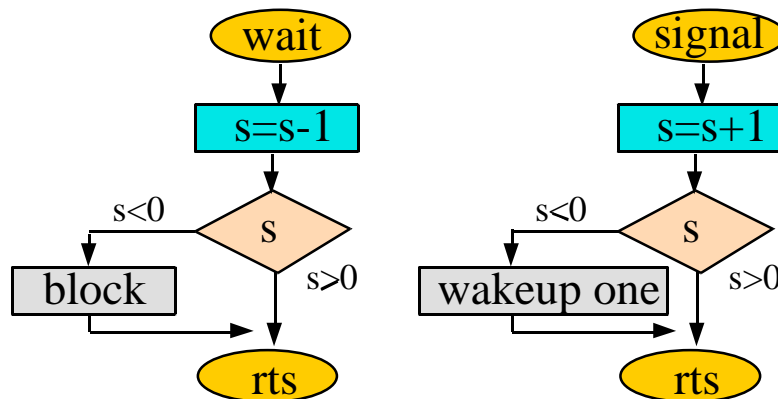
- Recapture time lost in the spin-lock
  - No spin operation, wakeup only on signal
  - Eliminate wasted time running threads that are not doing work (e.g., waiting)
- Implement **bounded waiting**
  - Once thread calls **Wait** and is not serviced,
  - There are a finite number of threads that will go ahead

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

13

## Blocking Semaphore



What does the semaphore mean?  
What about atomic?

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

14

## Blocking Semaphore (V1)

- All threads exist on circular TCB list (active and blocked)
  - Each semaphore simply has a **Value**
  - No blocked threads if semaphore **Value  $\geq 0$** 
    - e.g., if **Value** is -2, then two threads are blocked
  - No information about which thread has waited longest
  - Add to TCB, a **BlockPt**, of type **Sema4Type**
    - initially, this pointer is **null**
    - **null** means this thread is active and ready to run
    - If blocked, this pointer contains the semaphore address
- New Scheduler
  - Find the next active thread from the TCB list
  - Only run threads with **BlockPt** equal to **null**

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

15

## Blocking Semaphore (V1)

**OS\_Wait(Sema4Type \*semaPt)**

1) Disable interrupts, I=1

2) Decrement the semaphore counter, S=S-1

`(semaPt->Value)--;`

3) If the Value<0 then this thread will be blocked  
specify this thread is blocked to this semaphore

`RunPt->BlockPt = semaPt;`

suspend thread;

4) Enable interrupts, I=0

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

16



## Blocking Semaphore (V1)

### OS\_Signal(Sema4Type \*semaPt)

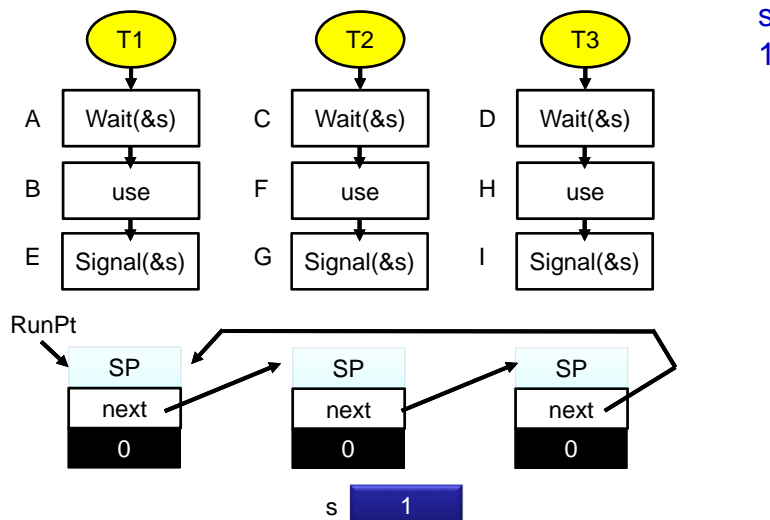
- 1) Save I bit, then disable interrupts
- 2) Increment the semaphore Value,  $S=S+1$   
`(semaPt->Value)++;`
- 3) If  $Value \leq 0$  then
  - wake up one thread from the TCB linked list  
(no bounded waiting)
  - search TCBs for thread with `BlockPt == semaPt`
  - set the `BlockPt` of this TCB to null
  - do not suspend the thread that called `OS_Signal`
- 4) Restore I bit

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

17

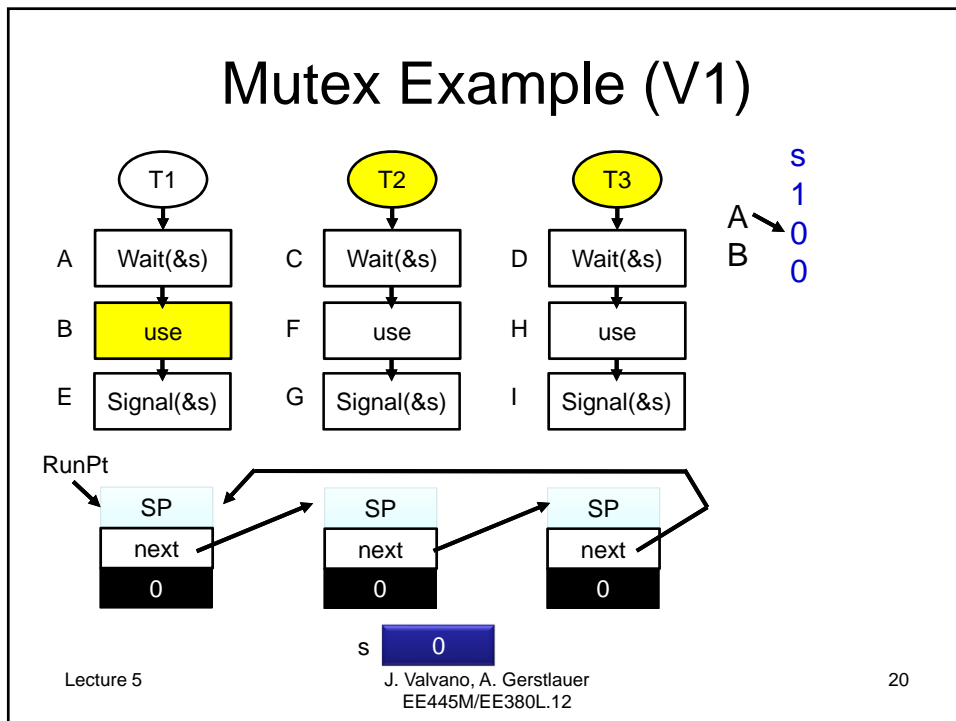
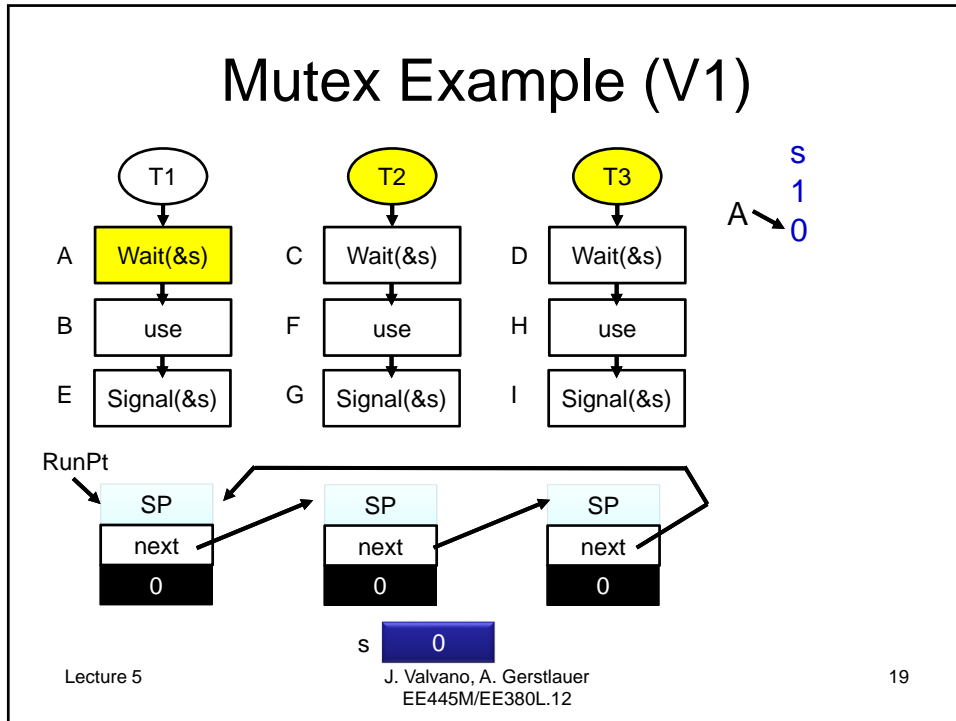
## Mutex Example (V1)

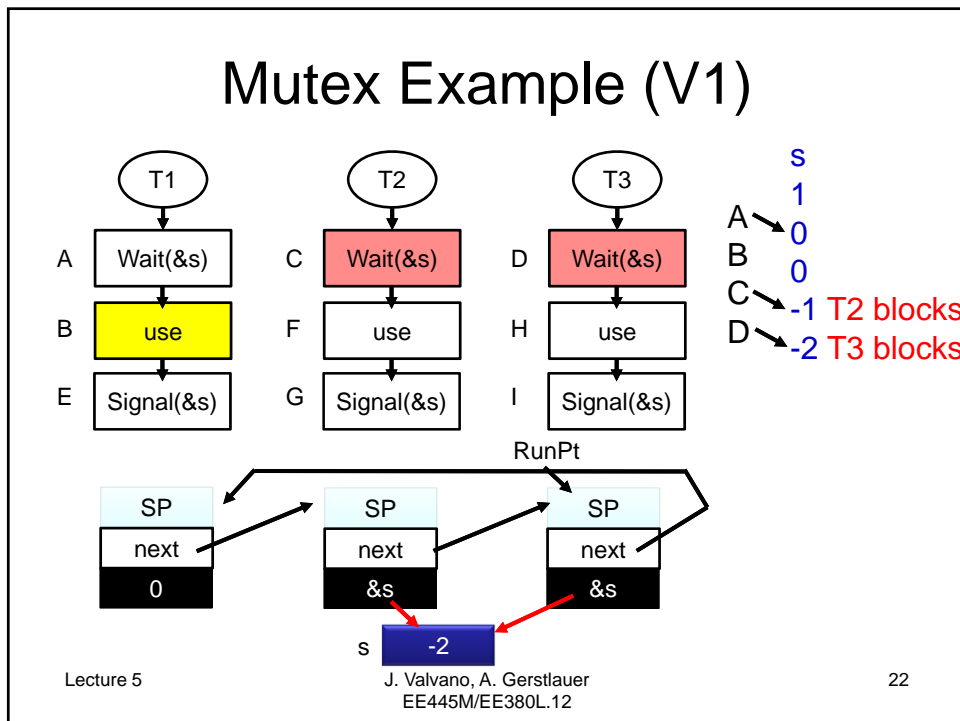
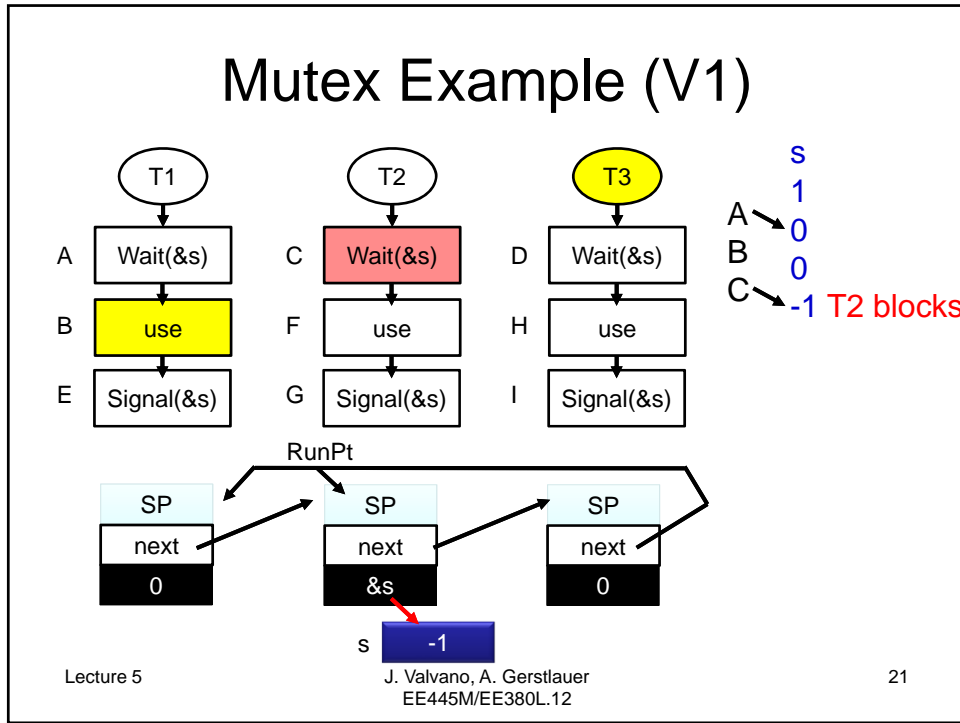


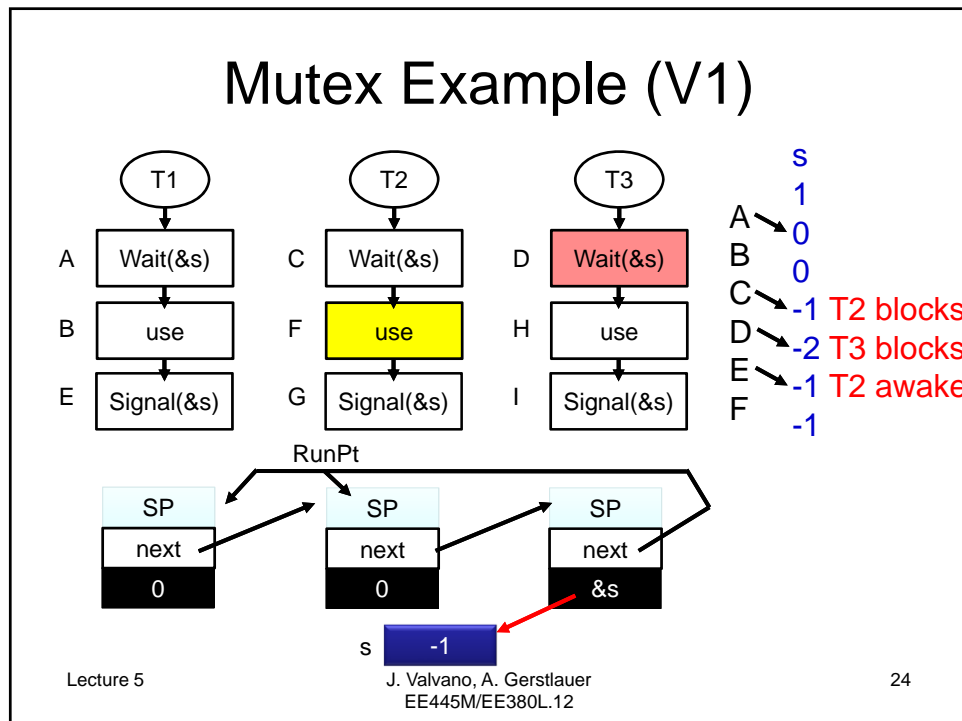
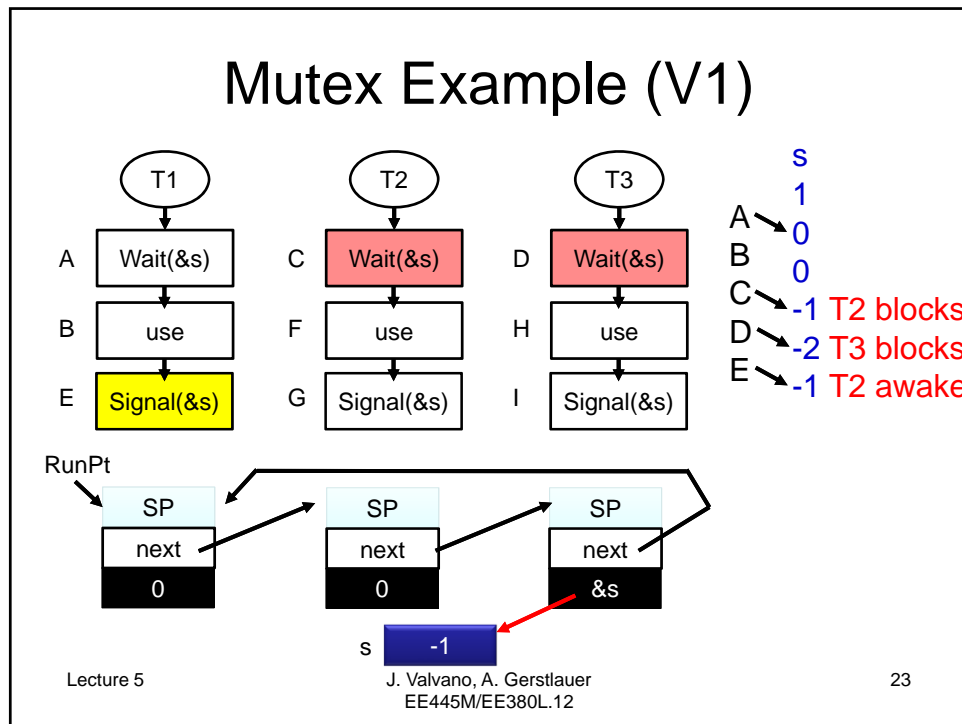
Lecture 5

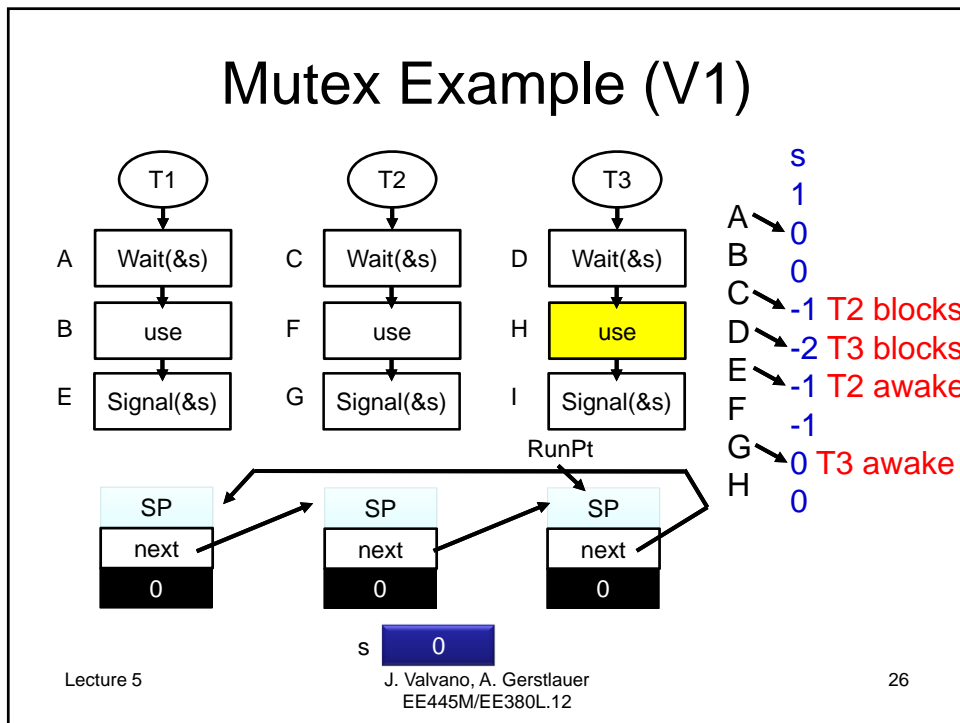
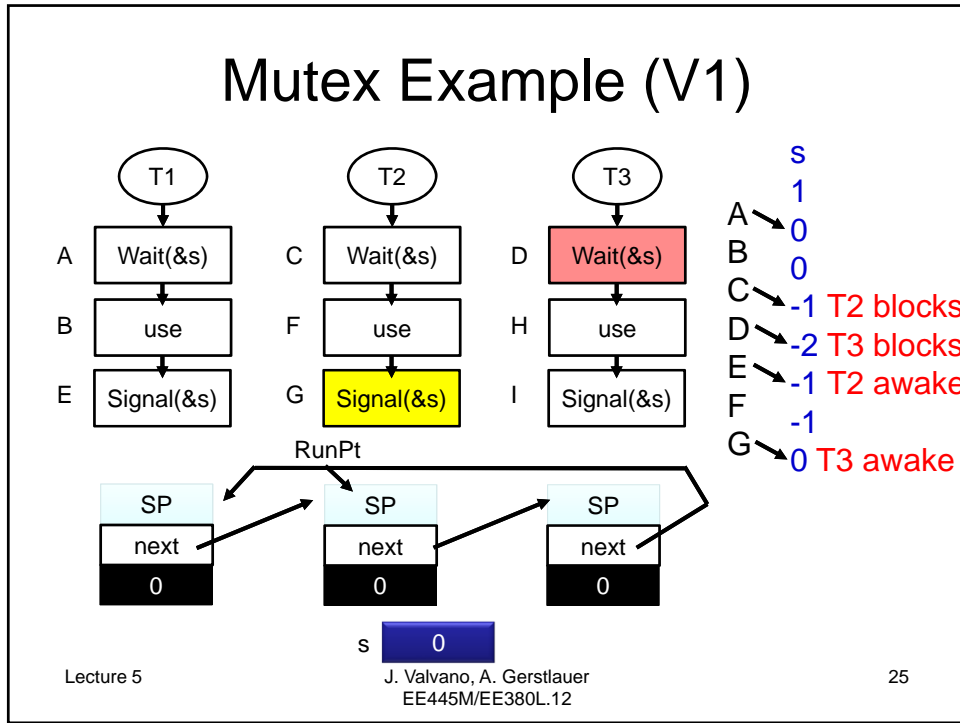
J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

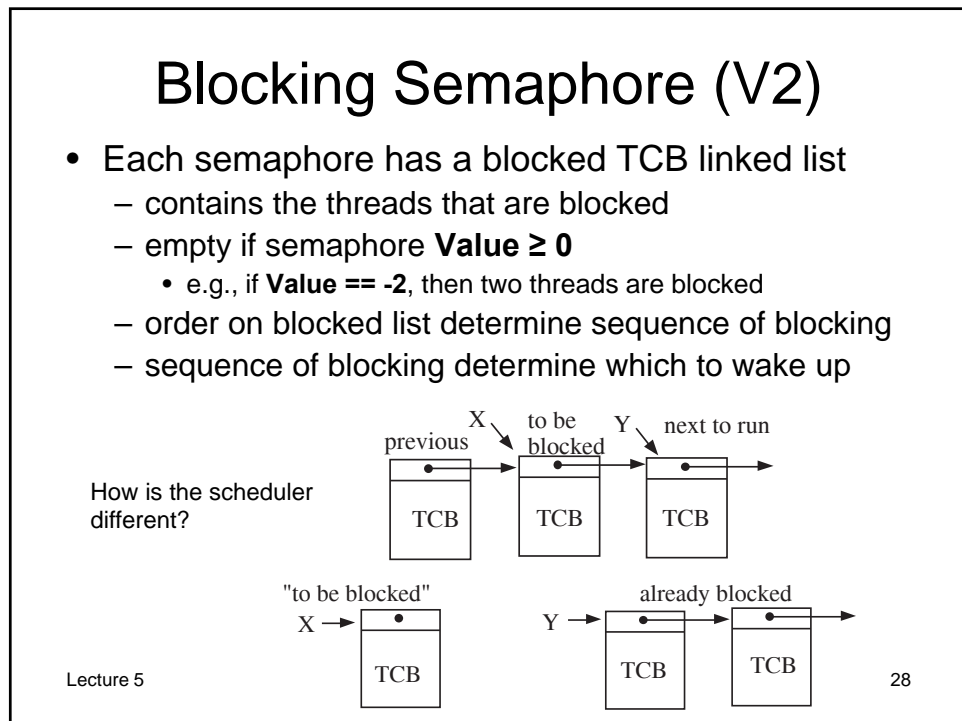
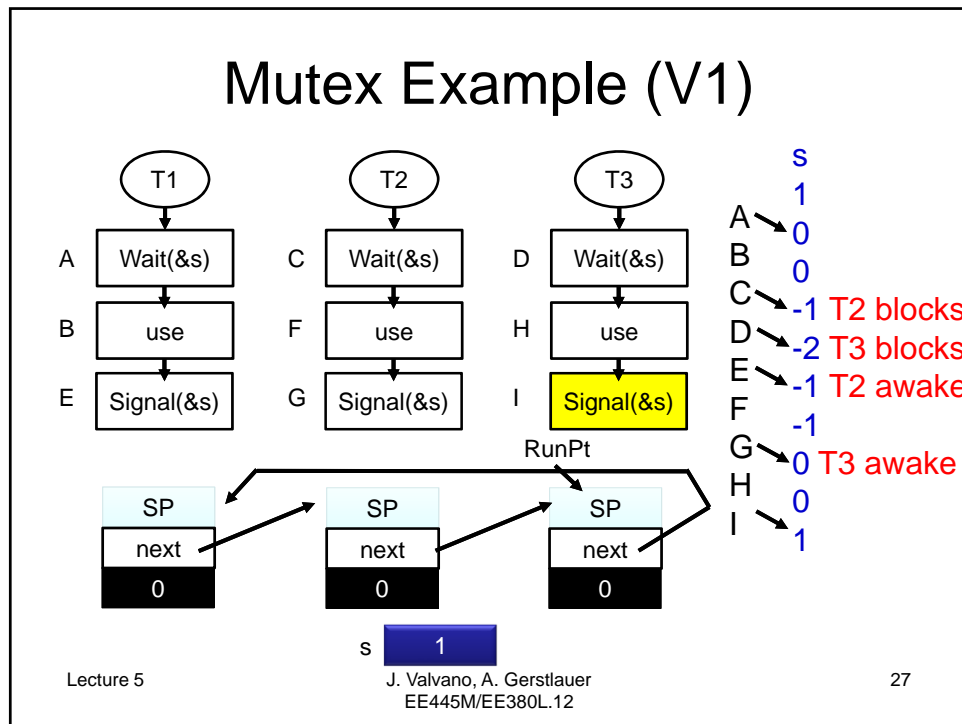
18











## Blocking Semaphore (V2)

### OS\_Wait(Sema4Type \*semaPt)

- 1) Save the I bit and disable interrupts
- 2) Decrement the semaphore counter,  $S=S-1$   
`(semaPt->Value)--;`
- 3) If the `value < 0` then this thread will be blocked  
 set the status of this thread to blocked,  
 specify this thread blocked on this semaphore,  
 suspend thread
- 4) Restore the I bit

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

29

## Blocking Semaphore (V2)

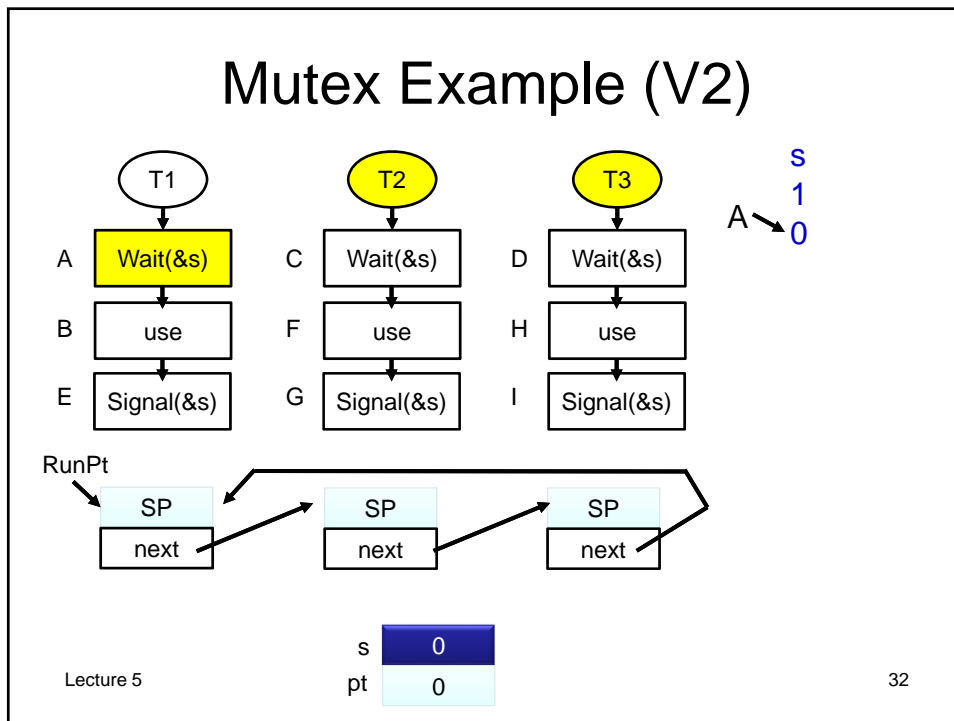
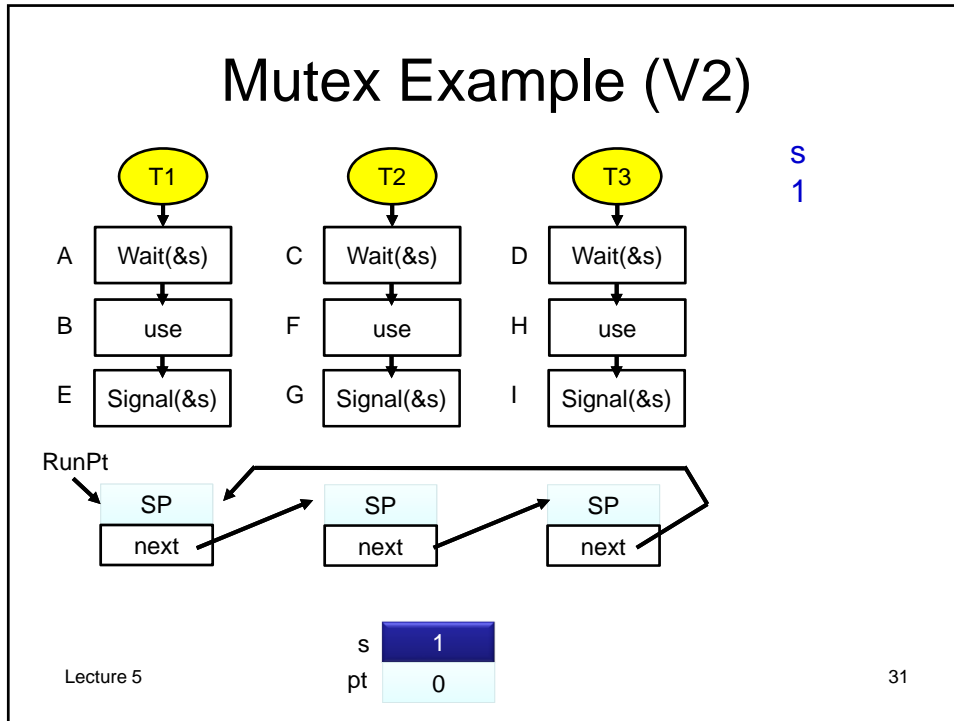
### OS\_Signal (Sema4Type \*semaPt)

- 1) Save I bit, then disable interrupts
- 2) Increment the semaphore counter,  $S=S+1$   
`(semaPt->Value)++;`
- 3) If the `value ≤ 0` then
  - Wake up one thread from the TCB linked list**
    - Bounded waiting -> the one waiting the longest
    - Priority -> the one with highest priority
  - Move TCB of the "wakeup" thread**  
**from the blocked list to the active list**
  - What to do with the thread that called OS\_Signal?***
    - Round robin -> do not suspend
    - Priority -> suspend if wakeup thread is higher priority
- 4) Restore I bit

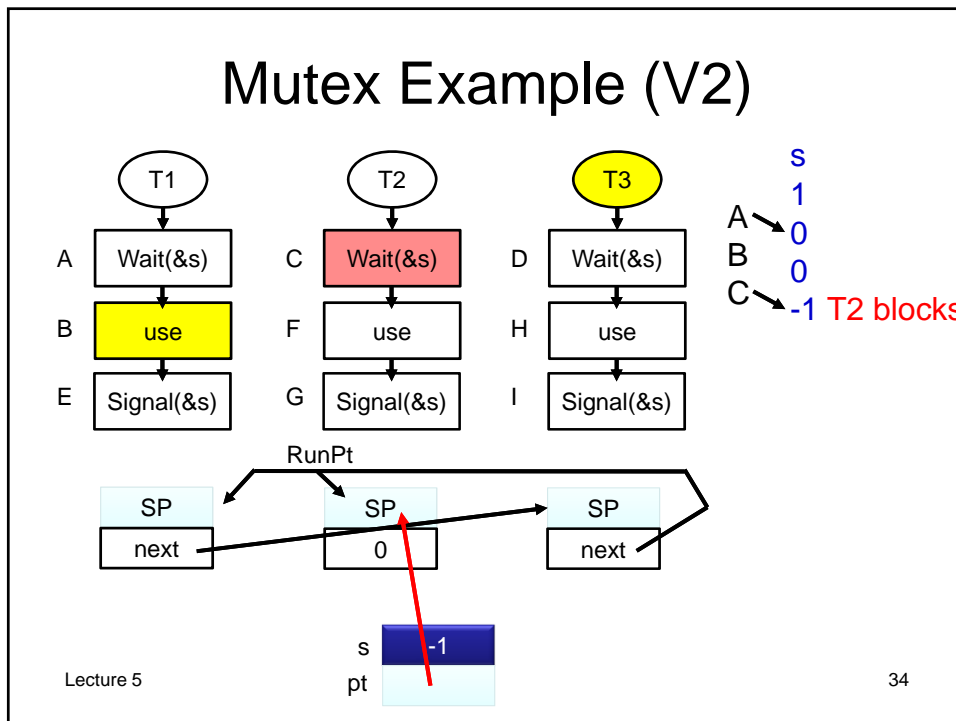
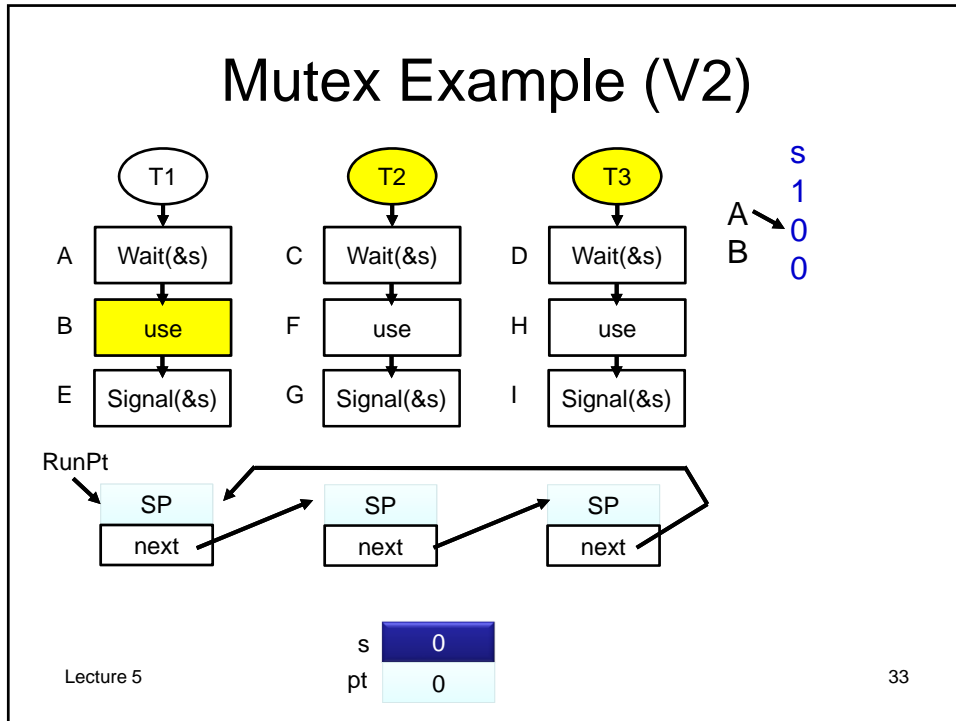
Lecture 5

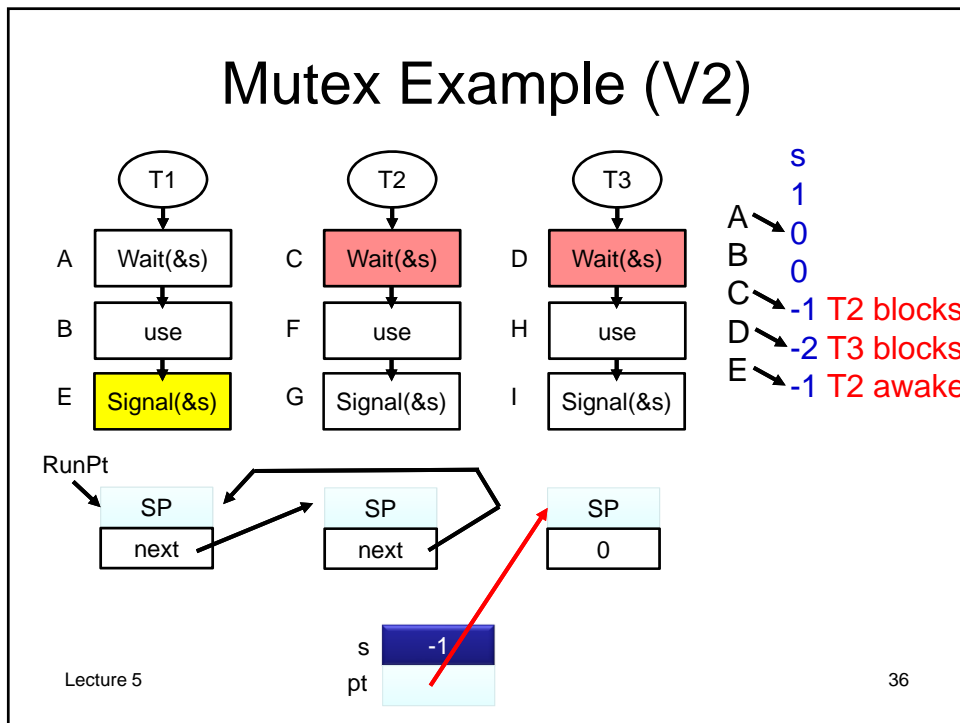
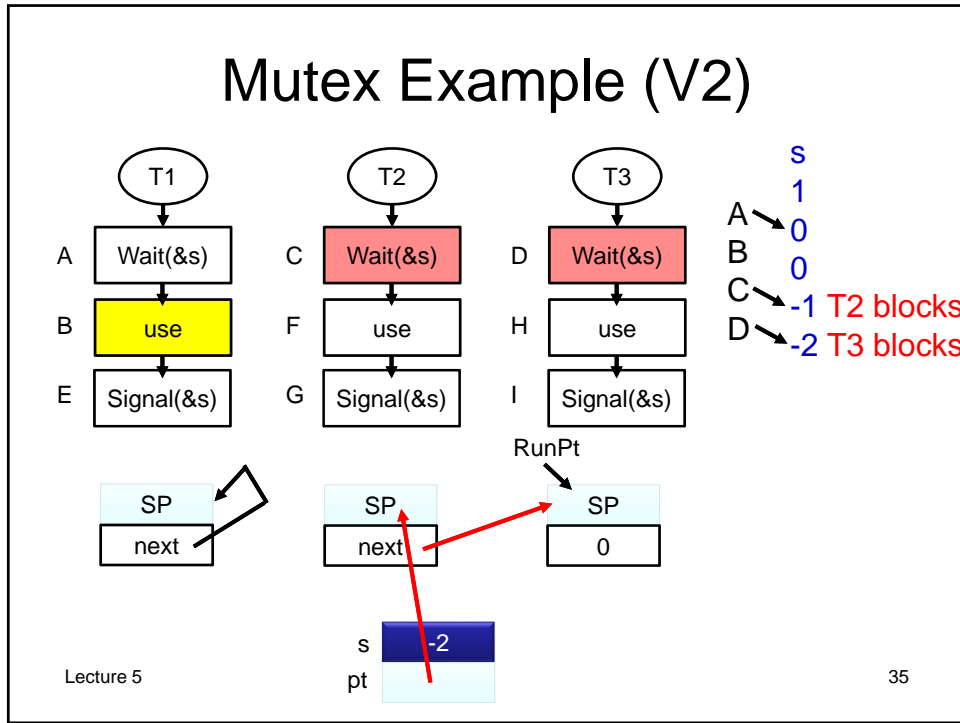
J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

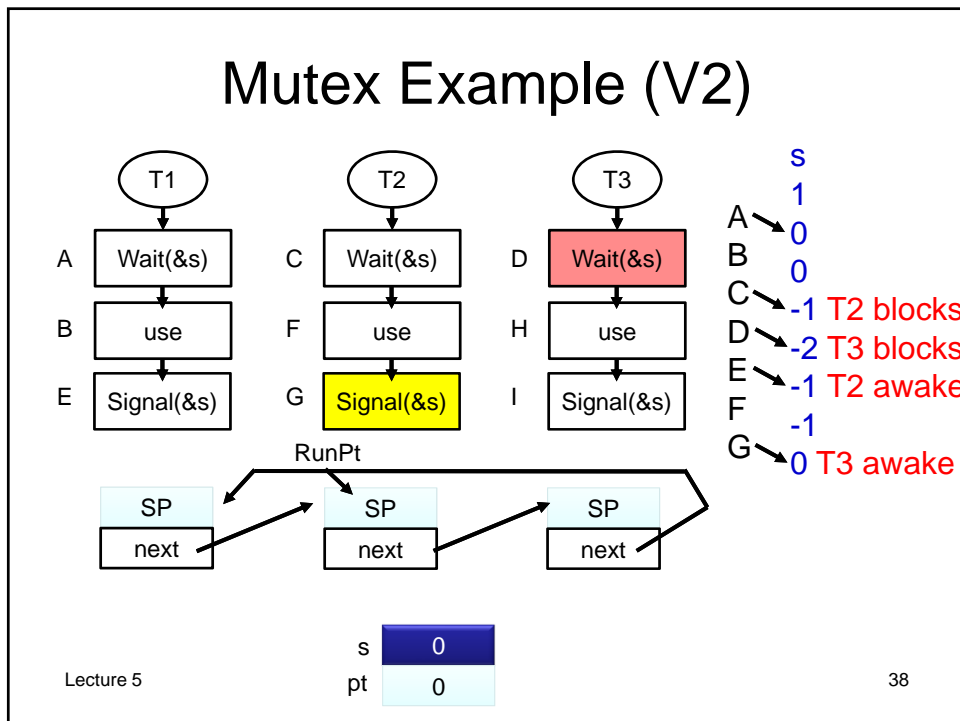
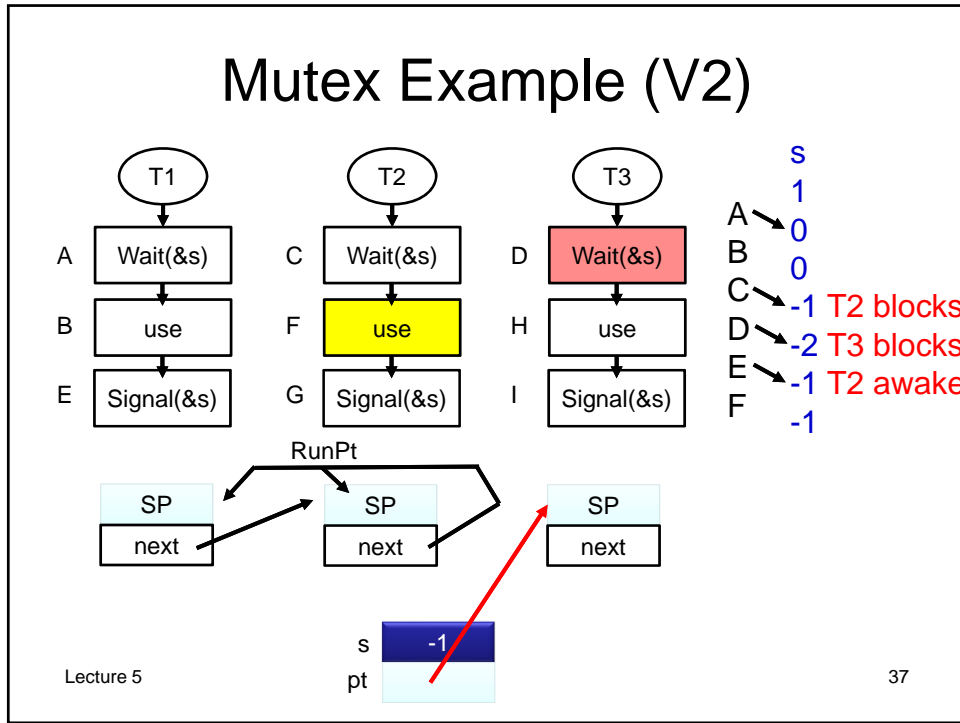
30

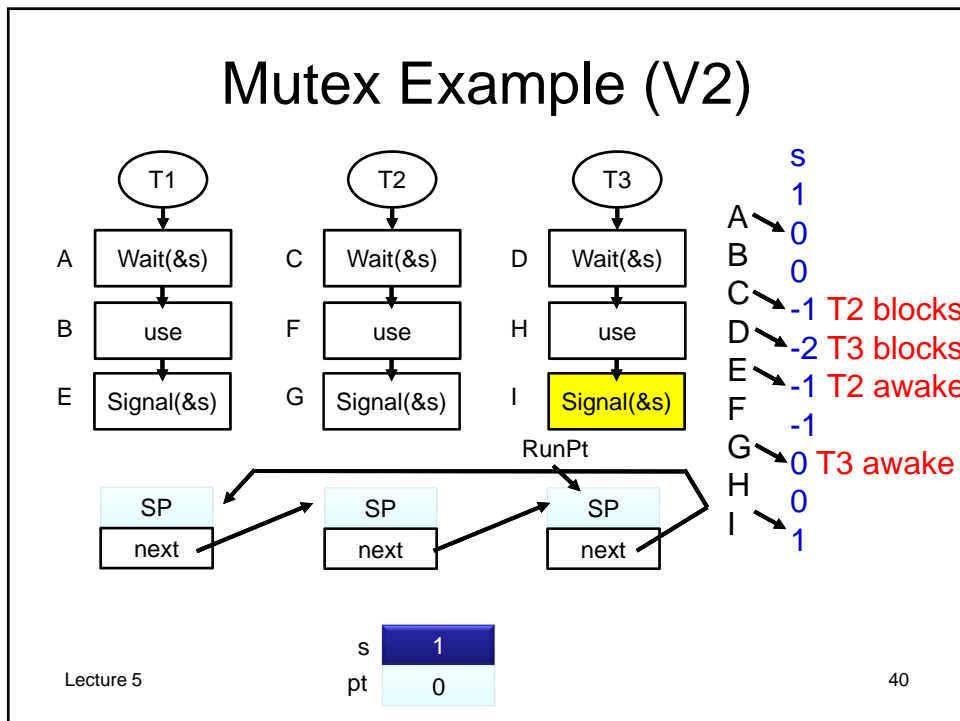
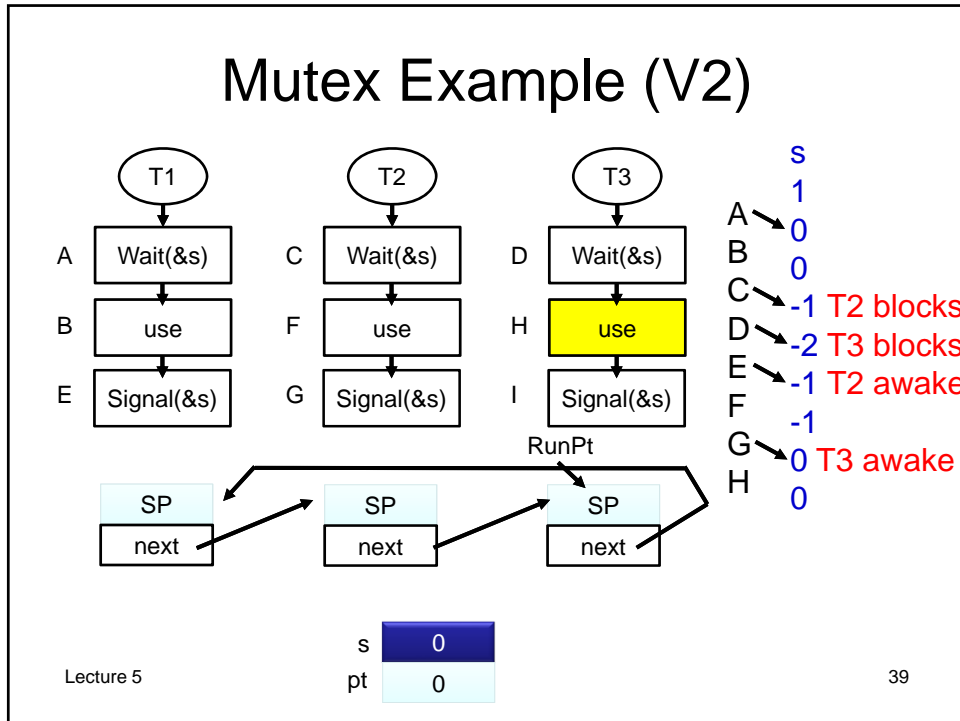












## Semaphore Applications

- Sequential execution
  - **Run-A** then **Run-B** then **Run-C**
- Rendezvous
- Event trigger
  - **Event-A** and **Event-B**
  - **Event-A** or **Event-B**
- Fork and join
- Readers-Writers Problem

Look at old exams

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

41

## Readers-Writers Problem

### Reader Threads

- 1) Execute **ROpen(file)**
- 2) Read information from **file**
- 3) Execute **RClose(file)**

### Writer Threads

- 1) Execute **WOpen(file)**
- 2) Read information from **file**
- 3) Write information to **file**
- 4) Execute **WClose(file)**



Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

42

## Readers-Writers Problem

**ReadCount=0**: number of Readers that are open  
**mutex=1**: semaphore controlling access to **ReadCount**  
**wrt=1**: semaphore is true if a writer is allowed access

### ROpen

```
wait(&mutex);
ReadCount++;
if(ReadCount==1) wait(&wrt)
signal(&mutex);
```

### WOpen

```
wait(&wrt);
```

### RClose

```
wait(&mutex);
ReadCount--;
if(ReadCount==0) signal(&wrt)
signal(&mutex);
```

### WClose

```
signal(&wrt);
```

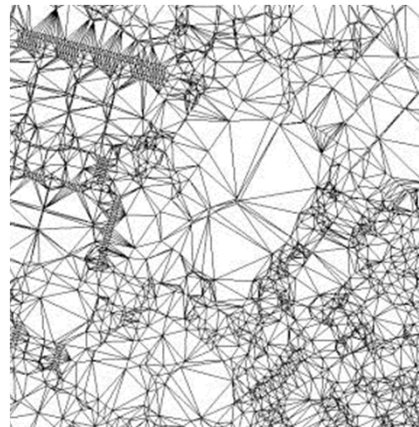
Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

43

## Deadlock

- Conditions
  - Mutual exclusion
  - Hold and wait
  - No preemption of resources
  - Circular waiting



Where is the deadlock?

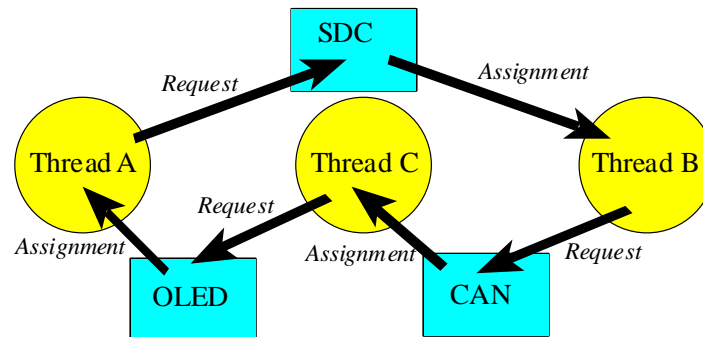
Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

44

## Resource Allocation Graph

Thread A	Thread B	Thread C
<code>wait(&amp;bOLED); //1</code>	<code>wait(&amp;bSDC); //2</code>	<code>wait(&amp;bCAN); //3</code>
<code>wait(&amp;bSDC); //4</code>	<code>wait(&amp;bCAN); //5</code>	<code>wait(&amp;bOLED); //6</code>
<code>use OLED and SDC</code>	<code>use CAN and SDC</code>	<code>use CAN and OLED</code>
<code>signal(&amp;bSDC);</code>	<code>signal(&amp;bCAN);</code>	<code>signal(&amp;bOLED);</code>
<code>signal(&amp;bOLED);</code>	<code>signal(&amp;bSDC);</code>	<code>signal(&amp;bCAN);</code>



Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

45

## Deadlock Prevention

- No mutual exclusion
- No hold and wait
  - Ask for all at same time
  - Release all, then ask again for all
- No circular waiting
  - Number all resources
  - Ask for resources in a specific order

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

46

## Prevention

- No hold and wait

Thread A	Thread B	Thread C
wait(&bOLED,&bSDC); use OLED and SDC signal(&bOLED,&bSDC);	wait(&bSDC,&bCAN); use CAN and SDC signal(&bSDC,&bCAN);	wait(&bCAN,&bOLED); use CAN and OLED signal(&bCAN,&bOLED);

- No circular wait

Thread A	Thread B	Thread C
wait(&bOLED); wait(&bSDC); use OLED and SDC signal(&bSDC); signal(&bOLED);	wait(&bSDC); wait(&bCAN); use CAN and SDC signal(&bCAN); signal(&bSDC);	wait(&bOLED); wait(&bCAN); use CAN and OLED signal(&bOLED); signal(&bCAN);

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

47

## Deadlock Avoidance

- Is there a safe sequence?
- Tell OS current and future needs
  - Request a resource
  - Specify future requests while holding
  - Yes, if there is one safe sequence
- OS can say no, even if available
  - Google search on Banker's Algorithm

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

48



# Deadlock Detection

- Add timeouts to semaphore waits
- Detect cycles in resource allocation graph
- Kill threads and recover resources
  - Abort them all, and restart
  - Abort them one at a time until it runs

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

49

# Advanced Topics (Grad Students)

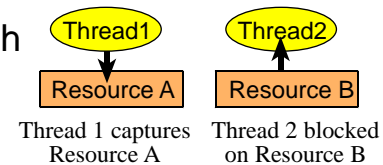
- Bounded waiting
- Time-out
- Deadlock avoidance
  - Banker’s algorithm
- Deadlock detection
  - Wait-for-graph
  - Resource allocation graph

Two names for the same thing

Works for single instance resources

- Two types of boxes  
Threads, resources
- Two types of arrows  
Assignment, request

Assignment edge      Request edge



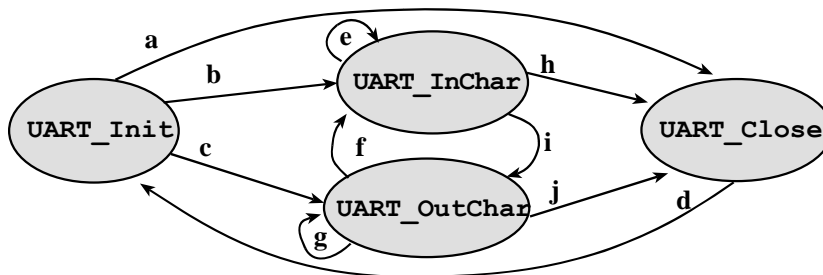
Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

50

## Path Expressions (1)

- Specify and enforce correct calling order
  - A group of related functions/tasks (task graph)
    - E.g. I/O, initialize before use



Book Section 4.6.2

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

51

## Path Expressions (2)

Each arrow is  
a '1' in matrix

```

int State=3; // start in the Closed state
int const Path[4][4]={ /* Init  InChar  OutChar  Close */
/*      column  0      1      2      3 */
/* Init   row 0*/ {  0  ,  1  ,  1  ,  1  },
/* InChar row 1*/ {  0  ,  1  ,  1  ,  1  },
/* OutChar row 2*/ {  0  ,  1  ,  1  ,  1  },
/* Close  row 3*/ {  1  ,  0  ,  0  ,  0  }};
void UART_Init(void){
    if(Path[State][0]==0) OS_Kill(); // kill if illegal
    State = 0;                       // perform valid Init
    // xxxx regular stuff xxxx
}
char UART_InChar(void){
    if(Path[State][1]==0) OS_Kill(); // kill if illegal
    State = 1;                       // perform valid InChar
    // xxxx regular stuff xxxx
}
  
```

Final exam 2004, Q9

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

52

## Testing (1)

- How long do you test?
  - $n$  = number of times T1 interrupts T2
  - $m$  = total number of assembly instructions in T2
  - Run test until  $n$  greatly exceeds  $m$
- Think of this corresponding probability question
  - $m$  different cards in a deck
  - Select one card at random, with replacement
  - What is the probability after  $n$  selections (with replacement) that a particular card was never selected?
  - Similarly, what is the probability that all cards were selected at least once?

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

53

## Testing (2)

Entered 486,736,253 times

```

Rx_Fifo_Get
0 24996444 0x000009B4 4601 MOV    r1,r0      ;int RxFifo_Get(rxDataType *datapt){
1 47909851 0x000009B6 481D LDR    r0,[pc,#116] ; if(RxPutPt == RxGetPt ){
2 12498221 0x000009B8 6800 LDR    r0,[r0,#0x00]
3 71517599 0x000009BA 4A1B LDR    r2,[pc,#108]
4 14581259 0x000009BC 6812 LDR    r2,[r2,#0x00]
5 11109532 0x000009BE 4290 CMP    r0,r2
6 10415182 0x000009C0 D101 BNE    0x000009C6
7 12498220 0x000009C2 2000 MOVNS r0,#0x00    ; return(RXFIPOFAIL);
8 4860416 0x000009C4 4770 BX     lr          ; }
9 694345 0x000009C6 4818 LDR    r0,[pc,#96] ; *datapt = *(RxGetPt++);
10 4860416 0x000009C8 6800 LDR    r0,[r0,#0x00]
11 1388690 0x000009CA 7800 LDRB  r0,[r0,#0x00]
12 694345 0x000009CC 7008 STRB  r0,[r1,#0x00]
13 4166071 0x000009CE 4816 LDR    r0,[pc,#88]
14 1388690 0x000009D0 6800 LDR    r0,[r0,#0x00]
15 2777381 0x000009D2 1C40 ADDS  r0,r0,#1
16 1388691 0x000009D4 4A14 LDR    r2,[pc,#80]
17 0 0x000009D6 6010 STR    r0,[r2,#0x00]
18 1388692 0x000009D8 4610 MOV    r0,r2
19 1388690 0x000009DA 6802 LDR    r2,[r0,#0x00]
20 2777380 0x000009DC 4811 LDR    r0,[pc,#68]
21 1388690 0x000009DE 4282 CMP    r2,r0      ; if(RxGetPt==&RxFifo[RXFIFOSIZE])
22 0 0x000009E0 D102 BNE    0x000009EA
23 0 0x000009E2 3820 SUBS  r0,r0,#0x20 ; RxGetPt = &RxFifo[0];
24 1388691 0x000009E4 4A10 LDR    r2,[pc,#64]
25 1388690 0x000009E6 6010 STR    r0,[r2,#0x00]
26 2083035 0x000009E8 2001 MOVNS r0,#0x01
27 0 0x000009EA E7EA B     0x000009C4 ; return(RXFIPOSUCCESS);}

```

FIFO\_4C123

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

54

## Performance Measures

- Maximum time running with  $l=1$
- Percentage of time it runs with  $l=1$
- Latency  $t_{action} - t_{trigger}$
- Time jitter  $\delta t$  on periodic tasks
 
$$T_i - \delta t < t_n - t_{n-1} < T_i + \delta t \quad \text{for all } n$$
- CPU utilization
  - Percentage time running idle task
- Context switch overhead
  - Time to switch tasks

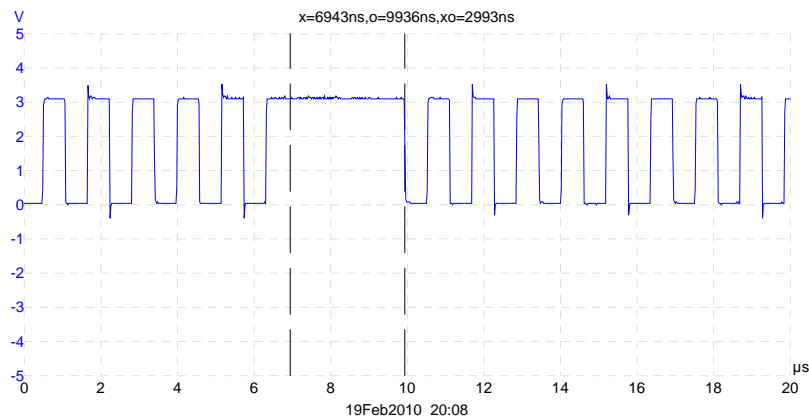
Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

55

## Context Switch Time

- Just like the Lab 1 measurement



Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

56

## Running with I = 1

```
#define OSCRITICAL_ENTER() { sr = SRSave(); }
#define OSCRITICAL_EXIT() { SRRestore(sr); }
```

- Record time  $t1$  when I=1

```
#define OSCRITICAL_ENTER() {sr = SRSave(); t1=OS_Time();}
```

- Record time  $t2$  when I=0 again
- Measure difference

```
#define OSCRITICAL_EXIT() {dt=OS_TimeDifference(OS_Time(),t1);
SRRestore(sr); }
```

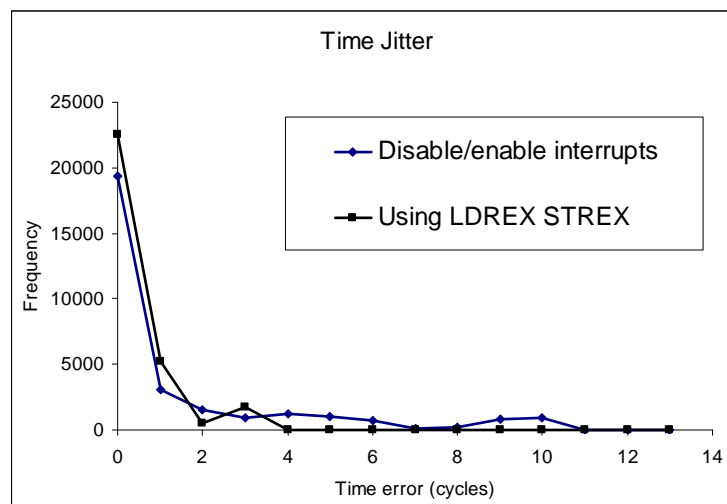
- Record maximum and total

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

57

## Time Jitter



Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

58

## Semaphore Drawbacks

- Shared global variables
  - Can be accessed from anywhere
- No connection between the semaphore and the data being controlled by the semaphore
  - Used both for critical sections (mutual exclusion) and coordination (scheduling)
- No control or guarantee of proper usage

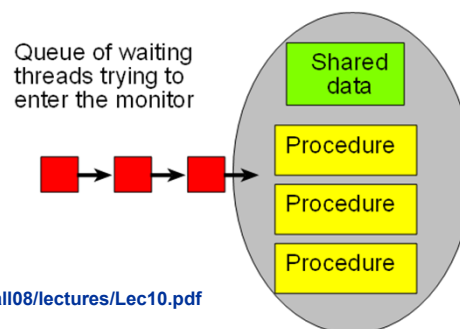
Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

59

## Monitors

- Proper use is enforced
- Synchronization attached to the data
- Removes hold and wait
- Threads enter
  - One active at a time



<http://lass.cs.umass.edu/~shenoy/courses/fall08/lectures/Lec10.pdf>

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

60

## Monitors

- Lock
  - Only one thread active at a time
  - Must have lock to access condition variables
- One or more condition variables
  - If cannot complete, leave data consistent
  - Threads can sleep inside by releasing lock
  - Wait (acquire or sleep)
  - Signal (if any waiting, wakeup else NOP)
  - Broadcast

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

61

## FIFO Monitor

**Put(item):**

- 1) lock->Acquire();
- 2) put item on queue;
- 3) conditionVar->Signal();
- 4) lock->Release();

**Get():**

- 1) lock->Acquire();
- 2) while queue is empty  
conditionVar->Wait(lock);
- 3) remove item from queue;
- 4) lock->Release();
- 5) return item;

<http://lass.cs.umass.edu/~shenoy/courses/fall08/lectures/Lec10.pdf>

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

62

## Hoare vs. Mesa Monitor

- Signal() switches immediately vs. later

**Hoare wait:**  
**if(FIFO empty)**  
**wait(condition)**

**Mesa wait:**  
**while(FIFO empty)**  
**wait(condition)**

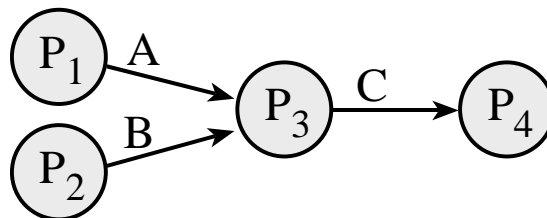
Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

63

## Kahn Process Network (KPN)

- Parallel programming model
  - Blocking read
  - Non-blocking writes (never full)
  - Tokens are data (no time stamp)



Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

64



## Kahn Process Network (KPN)

- Deterministic
  - Same inputs result in same outputs
  - Independent of scheduler
- Non-blocking writes (never full)
- Monotonic
  - Needs only partial inputs to proceed
  - Works in continuous time

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

65

## Kahn Process Network (KPN)

```
void Process3(void){
long inA, inB, out;
while(1){
  while(AFifo_Get(&inA)){};
  while(BFifo_Get(&inB)){};
  out = compute(inA,inB);
  CFifo_Put(out);
}
}
```

```
void Process3(void){
long inA, inB, out;
while(1){
  if(AFifo_Size()==0){
    while(BFifo_Get(&inB)){};
    while(AFifo_Get(&inA)){};
  } else{
    while(AFifo_Get(&inA)){};
    while(BFifo_Get(&inB)){};
  }
  out = compute(inA,inB);
  CFifo_Put(out);
}
}
```

EE445M/EE380L.12

66

## Kahn Process Network (KPN)

- Strictly bounded?
  - Prove it never fills (undecidable!)
  - Dependent on scheduler
- Termination
  - All processed blocked on input
- Scheduler
  - Needs only partial inputs to proceed
  - Works in real time

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

67

## KPN Boundedness

- Try to find a mathematical proof
- Experimentally adjust FIFO size
  - Needs a realistic test environment
  - Profile/histogram DataAvailable for each FIFO
  - Leave the profile in delivered machine
- Dynamically adjust size with malloc/free
- Use blocking write (not a KPN anymore)
- Discard the data

Lecture 5

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

68

## Summary

- Use the logic analyzer
  - Visualize what is running
- Learn how to use the debugger
  - Breakpoint inside ISR
    - Does not seem to single step into ISR
- What to do after a thread calls Kill?