Pomona College

Computer Science

Fall 2024: CSCI 181RT

Real-Time Systems in the Real World

Lecture 10

Thursday, September 26, 2024
Edmunds Hall 105
2:45 PM - 4:00 PM

Professor Jennifer DesCombes

Agenda

- Go Backs
- Discussion on Reading
- Discussion on Lab 4
- Use of Semaphores
- Look Ahead
- Assignment
- Action Items

Go Backs

- General?
- Action Item Status
  - AI240910-2: Find recommended book on computer architecture.
  - AI240924-1: At what point as a development team grows does it make sense to have dedicated software and integration testers?
  - AI240924-2: Is there a limit on the size of an Agile development effort before it becomes less efficient than other development approaches?
  - AI240924-3: Are ‘C’ type Handles (pointer to a pointer) similar in concept to Java Script Handles
  - AI240924-4: Send out presentation slides for Lecture 9. OK to Close?

Go Backs

- Discussion on Reading - The Mythical Man Month

- Discussion on Lab 4

Other Types of Semaphore Operations

- Counting Semaphores (we have been discussing this type)
- Binary Semaphores
- Locks and Mutexes
- Supervisor Mode / Interrupt Semaphore Calls
- Direct to Task Notifications

Use of Semaphores

- Synchronization of Tasks
- Protection of Shared Devices/Services
- Controlled Processing of Input Data
- Protection of Non-reentrant Code
- Guarantee Completion of Specific Operations

Pomona College

Use of Semaphores

- Semaphore Functions in Different RTOS

RTOS	P-Action	V-Action
POSIX	int **sem_wait**(sem_t *sem);	int **sem_post**(sem_t *sem);
FreeRTOS	pdStatus **xSemaphoreTake**(SemaphoreHandle_t xSemaphore, TickType_t xTicksToWait);	pdStatus **xSemaphoreGive**(SemaphoreHandle_t xSemaphore);
VxWorks	STATUS **semTake**(SEM_ID semId, int timeout);	STATUS **semGive**(SEM_ID semId);

- Code Snippets Not Complete
- Color Highlights Show Semaphore Control

Use of Semaphores - Task Synchronization

```
sem_t buttonSem;
sem_t timerSem;
bool buttonStateChanged( void );
void processButton( void );

// High Priority Task
void detectButtonPushTask (void)
{
   int myRet;
   while( true )
   {
     // wait for 5ms timer
      myRet = sem_wait( &timerSem );
     If ( buttonStateChange( ) )
     {
        myRet = sem_post( &buttonSem );
     }
     // other important things to do
   }
}
```

```
// Lower Priority Task
void processButtonTask (void)
{
   int myRet;
   while( true )
   {
      myRet = sem_wait( &buttonSem );
      processButton( );
   }
}
```

Use of Semaphores - Use of Shared Devices/Services

```
sem_t uartTxSem;
char uartChar;
void writeUART( char );

.
.
.
  int myRet;
  myRet = sem_wait( &uartTxSem );
  // Transmit "Hi!"
  writeUART( 'H' );
  writeUART( 'i' );
  writeUART( '!' );
  myRet = sem_post( &uartTxSem );
.
.
.
```

If the UART is available, execution will continue.

If the UART is busy, execution will be suspended (waiting) until UART is available. Code will resume execution when UART is no longer being used by other code/task.

Let the system/tasks know that the UART is now available

Use of Semaphores - Processing Input Data

```
sem_t uartRxSem;
char uartChar;
char readUART( void );
void rint( void );
void processChar( void );

// Interrupt
void serviceUARTInterrupt (void)
{
   int myRet;
   // get the character from UART
   uartChar readUART( );
   myRet = sem_post( &uartRxSem );
   rint( );
}
```

```
// Lower Priority Task
void processCharTask (void)
{
   int myRet;
   while( true )
   {
      myRet = sem_wait( &uartRxSem );
      processChar( );
   }
}
```

Use of Semaphores - Processing Input Data - More

```
sem_t uartRxL1Sem;
sem_t uartRxL2Sem;

// Interrupt
void serviceUARTInterrupt (void)
{
   int myRet;
   // get the character from UART
   uartChar readUART( );
   If (uartChar == 'H')
   {
      myRet = sem_post( &uartRxL1Sem );
   }
   else
   {
      myRet = sem_post( &uartRxL2Sem );
   }
   rint( );
}
```

```
// Lower Priority Task
void processCharTaskL1 (void)
{
   int myRet;
   while( true )
   {
      myRet = sem_wait( &uartRxL1Sem );
      processCharL1( );
   }
}
```

```
// Lower than Low Priority Task
void processCharTaskL2 (void)
{
   int myRet;
   while( true )
   {
      myRet = sem_wait( &uartRxL2Sem );
      processCharL2( );
   }
}
```

Use of Semaphores - Protection of Non-reentrant Code

```
sem_t codeScarrySem;
char uartChar;
uInt32 scarryCode( *uInt32 );

.
.
.
   int myRet;
   myRet = sem_wait( &codeScarrySem );
   uInt32 scarryResult;
   scarryResult = scarryCode( &someGlobal);
   myRet = sem_post( &codeScarrySem );
.
.
.
```

If no other task is currently executing the scarryCode method, execution will continue.

If the scarryCode is currently being executed by some other task, execution will be suspended (waiting) the other task is done with scarryCode. Code will resume execution when UART is no longer being used by other code/task.

Let the system/tasks know that the scarryCode is now available

Use of Semaphores - Completion of Operations

```
sem_t accessGVectorSem;
char uartChar;
uInt32 scarryCode( *uInt32 );
.
.
.
   int myRet;
   myRet = sem_wait( &accessGVectorSem );
   gfRateVector.x = 30.45;
   gfRateVector.y = 0.003;
   gfRateVector.z = 10.62;
  myRet = sem_post( &accessGVectorSem );
.
.
.
```

```
.
.
.
  // Set gfRateVector to zero - OK??
  gfRateVector = cfZeroRateVector;

.
  // Set gfRateVector to zero - OK??
  int myRet;
  myRet = sem_wait( &accessGVectorSem );
  gfRateVector = cfZeroRateVector;
  myRet = sem_post( &accessGVectorSem );
.
.
.
```

Are these both OK?

Use of Semaphores - Mutex and Locks

- Mutual Exclusion Semaphores - Mutex
  Mutual Exclusion semaphores are used to protect shared resources (data structure, file, etc..).
- A Mutex semaphore is "owned" by the task that takes it. If Task B attempts to semGive a mutex currently held by Task A, Task B's call will return an error and fail.
- Similar Functionality to Binary Semaphores
- Mutex May Not Provide Suspend/Resume Feature - Will be System Dependent

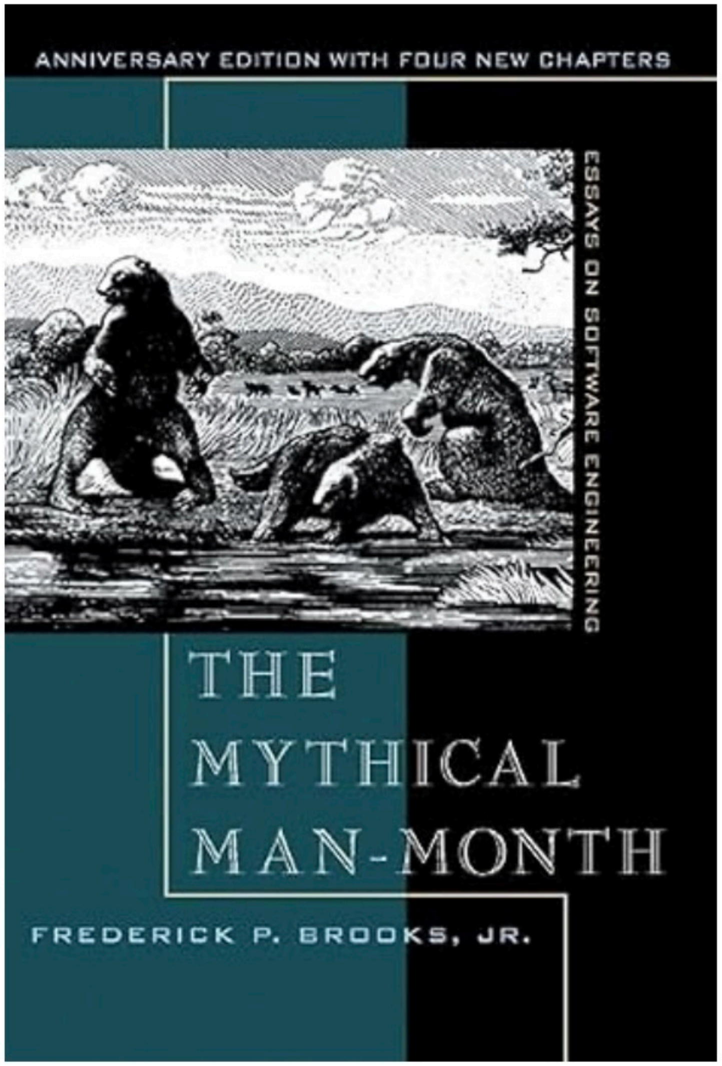
```
Thread A
   Take Mutex
      access data
      ...
      …
   Give Mutex
```

Look Ahead

- Review of Reading
- Interrupts and OS Control
- Discussion of Lab 4

Assignment - Readings

- The Mythical Man Month
  - Chapter 4, 5 & 6: Aristocracy, Democracy, and System Design - The Second System Effect - Passing the Word
- Send Me Discussion Topics by 10:00 on Tuesday, Oct. 1, 2024.


ANNIVERSARY EDITION WITH FOUR NEW CHAPTERS
ESSAYS ON SOFTWARE ENGINEERING
THE MYTHICAL MAN-MONTH
FREDERICK P. BROOKS, JR.

Pomona College

Action Items and Discussion

AI#:	Owner	Slide #	Document	Action