REAL TIME OPERATING SYSTEM PROGRAMMING-I: μC/OS-II and VxWorks

Lesson-6:

μC/OS-II Semaphore Functions

1. Semaphore Functions

μC/OS-II Semaphore functions

- Provides for using same semaphore functions as an event signaling flag or mutex or counting semaphore.

Semaphore for event flag functions

- When a semaphore created is used as a an event signaling flag or as counting semaphore, semaphore value at start = 0, which means event yet to occur and 1 will mean event occurred.

Semaphore as mutex functions

- When a semaphore created is used as a resource acquiring key, Semaphore value at start = 1, which means resource available and 0 will mean not available.

OSSemCreate (semVal)

- OS_Event OSSemCreate (unsigned short *semVal*)
  - To create and initialize semaphores

Refer examples 9.16 to 9.18 Steps7, 2 and 6 for event-flag, key and count, respectively.

OSSemPend (*eventPointer, timeOut, *SemErrPointer)

- void OSSemPend (OS_Event *eventPointer, unsigned short timeOut, unsigned byte *SemErrPointer)

To check whether semaphore is pending or not pending (0 or >0). If pending (=0), then suspend the task till >0 (released). If >0, decrement the value of semaphore and run the waiting codes (Example 9.16 Step 31)

OSSemAccept (*eventPointer)

- unsigned short OSSemAccept (OS_EVENT *eventPointer)
  - To check whether semaphore value > 0 and if yes, then retrieve and decrement. Used when there is no need to suspend a task, only decrease it to 0 if value is not already zero

OSSemPost (*eventPointer)

- unsigned byte OSSemPost (OS_EVENT*eventPointer)
  - SemVal if 0 or more, increment. Increment makes the semaphore again not pending for the waiting tasks. (Examples 9.16, 9.17 and 9.18 Steps 19, 10, 16, respectively)

OSSemQuery (*eventPointer)

unsigned byte OSSemQuery (OS_EVENT *eventPointer, OS_SEM_DATA *SemData)

–To get semaphore information

2. Macros for semaphore functions to find status after execution of OS semaphore Functions

Macros for semaphore functions

- OS_NO_ERR,

if semaphore signaling succeeded. [SemVal > 0 or 0.] or when when querying succeeds;

- OS_ERR_EVENT_TYPE,

if *eventPointer is not pointing to the semaphore.

- OS_SEM_OVF,

when semVal overflows (cannot increment and is already 65535.)

3. Example of semaphore function Applications as event signaling flag

Example of semaphore function Applications

- Programming Example of two application tasks in a chocolate vending machine. One task is to Read and get coins (ReadTask) and other to deliver the chocolate (DeliveryTask)

Step *i*: Initiating the Semaphores

```
#define OS_MAX_EVENTS 8
/*When total number of IPCs needed in an
application = 8*/
#define OS_SEM_EN 1
/*When the use of semaphores is
contemplated */
```

Step j: Global IPC functions and their parameters declarations

```
OS_EVENT *SemFlagChocolate;
/* When Sem is to be used as flag for the
delivery of chocolate, initial value = 0 (means
not released) */
 SemFlagChocolate = OSSemCreate (0);
```

Step k: Main function

```
void main (void) {
OSInit ();
/* Create First task */
OSTaskCreate (FirstTask, void (*)
0,(void *)&FirstTaskStack[
FirstTaskStackSize], FirstTaskPriority);
OSStart ( );
}
```

Step l: First task

```
static void FirstTask (void *taskPointer) {
/*System clock time set */
/* Create apllication related highest prio task */
OSTaskCreate (ReadTask, void (*) 0,(void
*)&ReadTaskStack [ReadTaskStackSize],
ReadTaskPriority);
```

Step l: First task

```
OSTaskCreate (DeliveryTask, void (*) 0,(void
*)&DeliveryTaskStack
[DeliveryTaskStackSize], DeliveryTaskPriority);
OSTimeSet (presetTime);
OSTickInit (); /* Initiate system timer ticking*/
```

Step l: First task suspend indefinitely after creating application tasks and initiating system timer ticks…

```
static void FirstTask (void *taskPointer) {
.
.
.
while (1) {
OSTaskSuspend (FirstTaskPriority);
}
```

Step m: ReadTask releasing semaphore on finding appropriate coins

```
static void ReadTask (void *taskPointer) {
.
.
while (1) {...; ...; ...;
.
OSSemPost (SemFlagChocolate);
/* after this instruction executes
SemFlagChocolate = 1 (released state)*/
```

Step m: ReadTask delaying to enable lower priority DeliveryTask to run

```
OSTimeDelay (3000)
/* Write delay function 3000 ticks to let the
control transfer to deliver task (next priority
task)*/
….; }
}
```

Step n: Chocolate delivery task waiting to take the required semaphore

```
static void DeliveryTask (void *taskPointer)
{
.
while (1) {
OSSemTake (SemFlagChocolate);
/* after this instruction executes
SemFlagChocolate = 0 (taken state)*/
...; ...; ...; /*Deliver Chocolate*/
```

Step n: Chocolate delivery task resuming delayed TaskRead …

```
…; …; …;
OSTimeDlyResume (ReadTaskPriority);/ *
Resume ReadTask for next read*/}; }
```

4. Example of semaphore function Applications as mutex

Programming Example

- Mutex *m* used to let a running task critical section 1 using a resource (for example, print buffer) not used by another task critical section 2 waiting to take same *m*

Step A: Creating a Mutex Semaphore

```
#define OS_MAX_EVENTS 8
/*When total number of IPCs needed in an
application = 8*/
#define OS_SEM_EN 1
 /*When the use of semaphores is
contemplated */
```

Step B: Global IPC functions and their parameters declarations

```
OS_EVENT *Sem_mCoin;
/* Declare a pointer to data structure for
semaphore and other IPC events */
```

Step B: Global IPC functions and their parameters declarations…

SemFlagChocolate = OSSemCreate (0);

/* When Sem is to be used as a event signaling flag, initial value = 0 (means not yet released) */

Sem _mCoin = OSSemCreate (1);

/* When Sem is to be used as a mutex initial value = 1 (means is in released state) */

Step C: Main function

```
void main (void) {
OSInit ();
/* Create First task */
OSTaskCreate (FirstTask, void (*)
0,(void *)&FirstTaskStack[
FirstTaskStackSize], FirstTaskPriority);
OSStart ( );
}
```

Step D: First task

```
static void FirstTask (void *taskPointer) {
/* Create apllication related highest prio
task */
OSTaskCreate (ReadTask, void (*) 0,(void
*)&ReadTaskStack [ReadTaskStackSize],
ReadTaskPriority);
```

Step D: First task…

```
OSTaskCreate (DeliveryTask, void (*)
0,(void *)&DeliveryTaskStack
[DeliveryTaskStackSize],
DeliveryTaskPriority); /*System clock time
set */
OSTimeSet (presetTime);
OSTickInit (); /* Initiate system timer
ticking*/
```

Step D: First task…

```
static void FirstTask (void *taskPointer) {
.
.
.
while (1) {
OSTaskSuspend (FirstTaskPriority);
}
```

Step E: Read task taking the mutex-semaphore before critical section starts

```
static void ReadTask (void *taskPointer)
{...
while (1) {...
OSSemtake (Sem_mCoin); /* after this
instruction executes the amount can be
incremented by ReadTask as Sem_mCoin =
0 (not in released state)*/
.....
```

Step E: Read task posting the mutex-semaphore…

/* Critical section run for collecting the coins and releasing the semaphore for permitting a reset later on the coin-amount after delivery of chocolate by other task critical section*/

- OSSemPost (Sem_mCoin); /* after this instruction executes the next task section can reset the amount */
- OSSemPost (SemFlagChocolate);
- ……. } }

Step F: Delivery task taking the required semaphores

```
static void DeliveryTask (void
*taskPointer) {..
while (1) {
OSSemTake (SemFlagChocolate);...; ...; ...;
OSSemTake (Sem_mCoin);
 /* after these instruction executes Sem_mCoin
and SemFlagChocolate = 0 (taken state) Critical
section 2 starts */
```

Step F: Delivery task taking the required semaphores

.../*Reset *amount* by setting *amount* = 0 after the chocolate delivery*/

OSSemPost(Sem_mCoin);

/* Release mutex to let read task section increment the amounts */

...;

/* code for resuming Delayed ReadTask*/ }}

Summary

We learnt

- Use of the OS Functions
- Initial setting of system-time, starting system ticks,
- Initiating OS Event semaphore IPC, initialing semaphore, taking and releasing a semaphore when using it as event flag.

We learnt

- Initiating OS Event semaphore IPC, initializing semaphore, taking and releasing a semaphore, when using it for a *mutex* fro exclusive access to run one of the two task sections .

We learnt

- A simplicity feature of μC/OS-II is that same semaphore functions are used for binary semaphore, for event signaling flag, resource key (mutex) and counting.

End of Lesson-6 Chapter 9 on μC/OS-II Semaphore Functions