USING A PUBLIC MUTEX TO GUARD A SHARED RESOURCE

Protect a shared data region among tasks in multiple memory spaces. A shared data region should already be created and tasks sharing the region should have already opened it to gain access. The Wind River® Task Guide, Creating and Using a Shared Data Object, provides details. The code is taken from the video demonstration, Creating and Using of VxWorks® Public Objects.

TASK GUIDE

	Action	Example
1	Create a project for each participant sharing the data region and its associated mutex.	

2	For the Writer, the first task to execute, write code that does the following:
	
	a) Creates the public mutex and captures its ID
	b) Loops, assigning to V1 and V2 (shared variables) the value of the variable COUNT (using offsets to the shared data region), then increments COUNT; note the framing of the critical region with semTake() and semGive()
		The code will look something like this:
		`int *pSharedData; /* pointer to shared data */`
		`SEM_ID mutexId;`
		`mutexId = semOpen("/mutexSem",SEM_TYPE_MUTEX,0,`
		`SEM_Q_PRIORITY,OM_CREATE,NULL);`
		
		`FOREVER{`
		`semTake(mutexId, WAIT_FOREVER);`
		`* (pSharedData + V1_OFFSET) =* (pSharedData +`
		`COUNT_OFFSET);`
		`* (pSharedData + V2_OFFSET) =* (pSharedData +`
		`COUNT_OFFSET);`
		`semGive(mutexId);`
		`(* (pSharedData + COUNT_OFFSET))++;`
		`}`

3	For the Reader, write code that does the following:
	
	a) Attaches to (gains access to) the Receiver task and captures the ID
	b) Loops, comparing the values of the shared variables V1 and V2; note the framing of the shared variables with semTake() and semGive()
	c) Compares the variables, printing an error message if they are not equal; remove the framing to examine the effect
		The code will look something like this:
		`SEM_ID mutexId;`
		`int *pSharedData; /* pointer to shared data */`
		`int v1, v2, same; /* count variables */`
		`mutexId = semOpen("/mutexSem", 0, 0, 0, 0);`
		`FOREVER{`
		`semTake(mutexId, WAIT_FOREVER);`
		`v1 = * (pSharedData + V1_OFFSET);`
		`v2 = * (pSharedData + V2_OFFSET);`
		`semGive(mutexId);`
		`same=(v1==v2);`
		`if (!same)`
		`printf("v1 = %u, v2 = %u\n", v1,v2);`
		`taskDelay (sysClkRateGet() / 5);`
		`}`

Key Points

- The semOpen () call is used either to create the public task initially or to gain access to it. The only difference is the OM_CREATE option.
- This facility may easily be extended to incorporate addition tasks that participate in the sharing.
- The code is for illustration only; you'll apply the mutex protection in a similar way in your application.