**SYNCHRONIZATION**

|  |  |
| --- | --- |
| NAME | S.SHANTHAN REDDY |
| ROLL No. | CB.EN.U4CSE19459 |

#include <windows.h>  
#include <stdio.h>  
  
#define MAX\_SEM\_COUNT 4  
#define THREADCOUNT 4  
  
HANDLE ghSemaphore;  
int num1, num2, sum;  
  
DWORD WINAPI ThreadProcSemaphore( LPVOID lpParam )  
{  
  
 // lpParam not used in this example  
 UNREFERENCED\_PARAMETER(lpParam);  
  
 DWORD dwWaitResult;   
 BOOL bContinue=TRUE;  
  
 while(bContinue)  
 {  
 // Try to enter the semaphore gate.  
  
 dwWaitResult = WaitForSingleObject(   
 ghSemaphore, // handle to semaphore  
 INFINITE); // zero-second time-out interval  
   
 switch (dwWaitResult)   
 {   
 // The semaphore object was signaled.  
 case WAIT\_OBJECT\_0:   
 // *TODO: Perform* printf("\nThread with %d id is executing...", GetCurrentThreadId());  
 printf("\nEnter 1st number: ");  
 scanf("%d", &num1);  
   
 printf("Enter 2nd number: ");  
 scanf("%d", &num2);  
   
 sum = num1 + num2;  
 printf("Sum of %d and %d : %d\n", num1, num2, sum);  
   
   
 bContinue=FALSE;   
  
 // Release the semaphore when task is finished  
  
 if (!ReleaseSemaphore(   
 ghSemaphore, // handle to semaphore  
 1, // increase count by one  
 NULL) ) // not interested in previous count  
 {  
 printf("ReleaseSemaphore error: %d\n", GetLastError());  
 }  
 break;   
  
 // The semaphore was nonsignaled, so a time-out occurred.  
 case WAIT\_TIMEOUT:   
 printf("Thread %d: wait timed out\n", GetCurrentThreadId());  
 break;   
 }  
 }  
 return TRUE;  
}  
  
DWORD WINAPI ThreadProc( LPVOID lpParam )  
{  
  
 // lpParam not used in this example  
 UNREFERENCED\_PARAMETER(lpParam);  
 int i;  
   
 for( i=0; i < 5; i++)  
 {  
 Sleep(2);  
 printf("Thread with %d id is executing..... \n", GetCurrentThreadId());  
 }  
  
 /\*printf("\nEnter 1st number: ");  
 scanf("%d", &num1);  
   
 printf("\nEnter 2nd number: ");  
 scanf("%d", &num2);  
   
 sum = num1 + num2;  
   
 printf("Sum : %d", sum);\*/  
   
 return 0;  
}  
  
int main( void )  
{  
 HANDLE aThread[THREADCOUNT];  
 DWORD ThreadID;  
 int i;  
  
 // Create a semaphore with initial and max counts of MAX\_SEM\_COUNT  
  
 ghSemaphore = CreateSemaphore(  
 NULL, // default security attributes  
 1, // initial value  
 1, // no.of resources  
 NULL); // unnamed semaphore  
  
 if (ghSemaphore == NULL)   
 {  
 printf("CreateSemaphore error: %d\n", GetLastError());  
 return 1;  
 }  
  
 // Create worker threads  
  
 printf("\nTHREAD SYNCHRONIZATION WITH SEMAPHORES\n");  
 for( i=0; i < 2; i++ )  
 {  
 aThread[i] = CreateThread(   
 NULL, // default security attributes  
 0, // default stack size  
 &ThreadProcSemaphore, //starting address of the thread  
 NULL, // no thread function arguments  
 0, // default creation flags  
 &ThreadID); // receive thread identifier  
  
 if( aThread[i] == NULL )  
 {  
 printf("CreateThread error: %d\n", GetLastError());  
 return 1;  
 }  
 }  
  
 // Wait for all threads to terminate  
  
 WaitForMultipleObjects(  
 2, // number of object handles in the array  
 aThread, // array of object handles  
 TRUE, // function returns when the state of all objects in the handles array is signaled  
 INFINITE); // time-out interval, in milliseconds  
  
 // Close thread and semaphore handles  
  
 for( i=0; i < 2; i++ )  
 CloseHandle(aThread[i]);  
  
 CloseHandle(ghSemaphore);  
   
   
 printf("\nTHREADS WITHOUT SEMAPHORES\n");  
 for( i=2; i < 4; i++ )  
 {  
 aThread[i] = CreateThread(   
 NULL, // default security attributes  
 0, // default stack size  
 (LPTHREAD\_START\_ROUTINE) ThreadProc, //starting address of the thread  
 NULL, // no thread function arguments  
 0, // default creation flags  
 &ThreadID); // receive thread identifier  
  
 if( aThread[i] == NULL )  
 {  
 printf("CreateThread error: %d\n", GetLastError());  
 return 1;  
 }  
 }  
   
 WaitForSingleObject(  
 aThread[2], // handle to semaphore  
 INFINITE); // zero-second time-out interval  
   
 WaitForSingleObject(   
 aThread[3], // handle to semaphore  
 INFINITE); // zero-second time-out interval  
   
   
 return 0;  
}

**Output :**

