Lab 5

Critical Section

Exercise 1:

Write a C program that demonstrates the use of a semaphore to control access to a critical section. Use the provided code template and complete the missing part to achieve the desired synchronization.
Ensure that the program correctly synchronizes the critical section using the semaphore, allowing only one process to enter the critical section at a time.
Compile and run your code to verify its correctness. Provide the complete code solution as your answer.

- The **CreateSemaphore** function is used to create a semaphore object. It is a Windows-specific function that allows you to create a semaphore and obtain a handle to that semaphore.

- **The syntax of the CreateSemaphore function is as follows:**

```
HANDLE CreateSemaphore(
  LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,
  LONG                  lInitialCount,
  LONG                  lMaximumCount,
  LPCTSTR                 lpName
);
```

lpSemaphoreAttributes: This parameter is used to specify the security attributes for the semaphore object. It can be set to NULL for default security settings.
lInitialCount: This parameter sets the initial count value for the semaphore. It determines the number of resources or permits initially available. In your code example, the initial count is set to 1, meaning there is one resource available initially.
lMaximumCount: This parameter sets the maximum count value for the semaphore. It defines the upper limit of resources or permits that can be available. In your code example, the maximum count is set to 2, indicating that the semaphore can have a maximum of two permits.
lpName: This parameter is used to provide a name for the semaphore object. It can be set to NULL for an unnamed semaphore.

- The **WaitForSingleObject** function is used to wait for a synchronization object, such as a semaphore, mutex, or event, to become signaled or available. It is a Windows-specific function that allows a thread to wait until the specified object is in a signaled state.

- **The syntax of the WaitForSingleObject function is as follows:**

```
DWORD WaitForSingleObject(
  HANDLE hHandle,
  DWORD  dwMilliseconds
);
```

hHandle: This parameter is the handle to the synchronization object that you want to wait for. In the context of your code example, WaitForSingleObject(semaphore, INFINITE);, the semaphore variable holds the handle to the semaphore object created with CreateSemaphore.
dwMilliseconds: This parameter specifies the time-out interval, in milliseconds, for waiting on the synchronization object. It determines how long the thread should wait for the object to become signaled. There are several possible values:
INFINITE (represented by -1): The thread waits indefinitely until the object becomes signaled.
0: The thread does not wait and immediately returns the result of the wait operation.
A positive value: The thread waits for the specified number of milliseconds before returning.

- the **ReleaseSemaphore** function is used to release or increment the count of a semaphore. It is a Windows-specific function that allows you to signal that a resource or permit is available, allowing other threads or processes to access the critical section or shared resource protected by the semaphore.

- **The syntax of the ReleaseSemaphore function is as follows:**

```
BOOL ReleaseSemaphore(
  HANDLE hSemaphore,
  LONG   lReleaseCount,
  LPLONG lpPreviousCount
);
```

hSemaphore: This parameter is the handle to the semaphore object that you want to release. It is the same handle obtained when the semaphore was created using CreateSemaphore.
lReleaseCount: This parameter specifies the number of resources or permits to release. It determines how many resources or permits are made available for other threads or processes to access.
lpPreviousCount: This optional parameter is a pointer to a LONG variable that receives the previous count of the semaphore. If you are not interested in the previous count, you can pass NULL for this parameter.

Code:

```
#include <stdio.h>
#include <windows.h>

HANDLE semaphore;

void criticalSection() {
   // Simulate some work in the critical section
   printf("Process enters the critical section.\n");
   printf("Process is performing some work.\n");
   Sleep(2000); // Simulate work for 2 seconds
   printf("Process exits the critical section.\n");
}

int main() {
   semaphore = CreateSemaphore(NULL, 1, 1, NULL);

   // Wait (decrement) the semaphore
   WaitForSingleObject(semaphore, INFINITE);

   // Enter the critical section
   criticalSection();


   // Release (increment) the semaphore
   ReleaseSemaphore(semaphore, 1, NULL);

   // Close the semaphore handle
   CloseHandle(semaphore);

   return 0;
}
```