

Lab 5

Process/Threads Synchronization and Communication (IPC)

ITSC205: Operating Systems Internals

NAME: Ahmed Almass

Table of Contents

[Lab Outcome(s) 3](#_Toc472727817)

[Reading 3](#_Toc472727818)

[Introduction 3](#_Toc472727819)

[1.0 Race Condition and Synchronization 4](#_Toc472727820)

[2.0 Interprocess Communication (IPC) and Synchronization 5](#_Toc472727821)

[3.0 Windows Synchronization 7](#_Toc472727822)

4.0 IPC -Remote Procedure Call (RPC)……………………………………………………………8

5.0 DLL……………………………………………………………………………………………….10

**EVALUATION**:

|  |  |  |
| --- | --- | --- |
| Race Condition and Synchronization | 10 |  |
| IPC and Synchronization | 10 |  |
| Windows Synchronization | 5 |  |
| IPC – Remote Procedure Call (RPC) | 5 |  |
| DLL | 10 |  |
| TOTAL MARK | 40 |  |

Lab Outcome(s)

* Examine Windows communications models.
* Examine Windows synchronization methods.

Reading

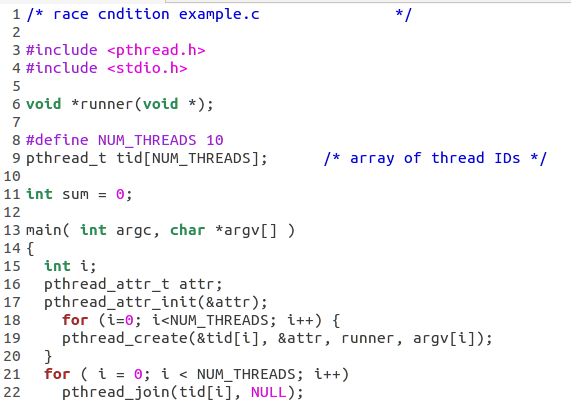
* Textbook sections synchronization 6.2 ( Critical Section problem) , 6.5 ( Mutex locks) 6.6( Semaphores), 7.3.1( Posix Mutex locks) and 7.3.2(POSIX Semaphores)

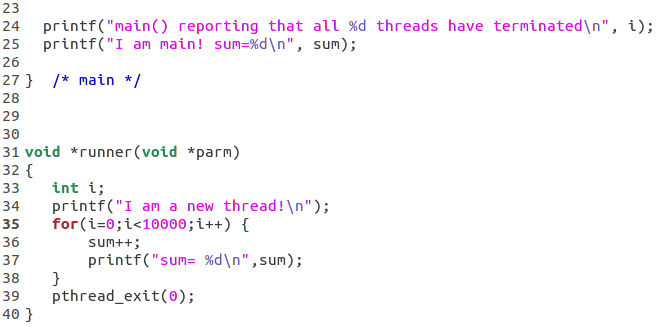
Introduction

Over many years of computer use, it has become clear that the greatest usefulness of computers comes with the use of multiple processes that communicate. Extending that idea to link computers together over a network, yields even greater usefulness. Whether this is 2 processes synchronizing their operation with a locking mechanism, a simple web application communicating over the Internet, or a complex network of powerful computers working together on a massive problem like weather forecasting, the reliable and efficient exchange of information among processes and computers has become essential. POSIX API is available for programmers at the user level and is not part of any particular operating system.

1.0 Race Condition and Synchronization

1. Run many times the following program and change the max value of the loop in the runner function from 10 to 10000 or more and analyze the results and the order of threads execution





1. Identify the critical section
2. Modify the program to solve race condition problem implementing semaphores :

sem\_t sem Declare the semaphore

sem\_init(&sem,0,1 ) Create semaphore and initialize it to 1

sem\_wait( sem ) Acquire mutex lock

sem\_post( sem ) Release mutex lock

1. Modify the program to solve race condition problem implementing mutex

pthread\_mutex\_t mutex Declare mutex

pthread\_mutex\_init(&mutex,NULL) Create and initialize mutex lock

pthread\_mutex\_lock(&mutex) Acquire mutex lock

pthread\_mutex\_unlock(&mutex) Release mutex lock

1. Explain and demo results to instructor

|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |

2.0 Inter-Process Communication and Synchronization

Processes on the same machine can communicate and share data by using share memory segment. Share memory segment has a name called **key** which is an integer. Processes may attach to a segment **shmat()** obtaining pointer to the segment.

1. Download from D2L shmserver.c ( Server) and shmclient.c (client)
2. Open a terminal in Linux compile the server and run it in the background
3. Open a second terminal compile client program and run it three times
4. Observe server and client results.
   1. What is the purpose of this communication?
   * The purposed of this is so the client grabs data from the server.

What is been transferred?

* + The time from the server.
  1. Use Linux manual to identify the arguments of the following functions:

shmget(), shmat(), shmdt(), semctl() and semop()

* shmget: allocates a System V shared memory segment.

- Example: schmget(key,amountofmemory,createssharedmemory&u could give permissions)

* shmat: System V shared memory operations.

-Example: shmat(schmid, schmaddress(usally NULL), shmflg- bit-mask argument (0 is SHM\_RDONLY)) it attaches a memory address casting to a pointer

* shmdt: attaches the System V shared memory segment identified by shmid to the address space of the calling process.

Example: shmdt(detach by the address)

* semget: system call returns the System V semaphore set identifier associated with the argument

-Example: semget(key, numOfSemphores, semflag)

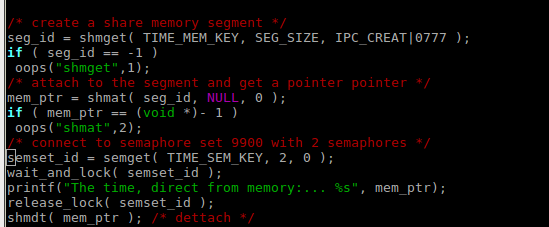
-The first argument, key, is a unique identifier that is used by different processes to identify this semaphore set. (This key will be generated using ftok())

-The next argument, nsems, is the number of semaphores in this semaphore set. When you access a pre-existing semaphre, you can set it to 0

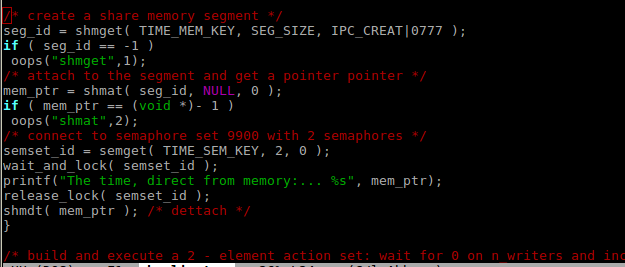
-The third argument, semflg, is used to specify access permission and/or special creation conditions. The flags IPC\_CREAT and IPC\_EXCL may be ORed with the permission value.

* Semop: performs operations on selected semaphores in the set indicated by semid.
  1. Analyze server and client code and identify the critical section

Server:

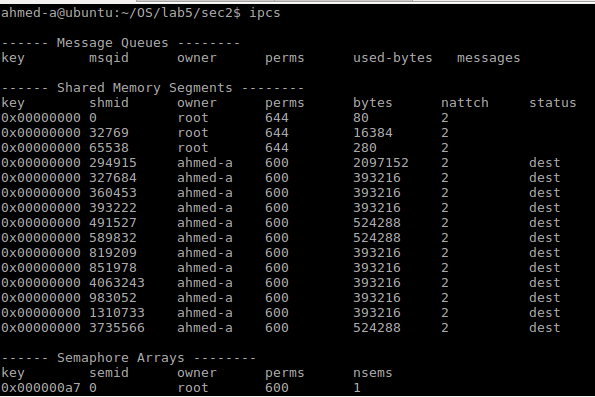


Client:



* 1. Compare wait\_and \_lock and release\_lock functions between server and client. What did you observed?

* The functions are the same for both client and server.
  1. Use ipcs command to display the IPCs information for the calling process.

****

1. **Demonstrate and explain** the results to the instructor

|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |

3.0 Windows Synchronization

Windows uses multiple data structures to control and manage access to a resource. These kernel objects are accessed by processes via a handle and will use one of the wait functions to synchronize with other threads. These objects are:

* Mutex: mutually exclusive lock, internally called mutant
* Semaphore: holds a count between 0 and a max value to limit access to shared resources
* Event: signal to one or more threads of an event
* Timer: signal to one or more threads when a specific time has arrived or when a particular time has elapsed

1. Read the article at <https://digital-forensics.sans.org/blog/2012/07/24/mutex-for-malware-discovery-and-iocs> and answer the following questions:
   1. How would you use procexp to find mutex objects?

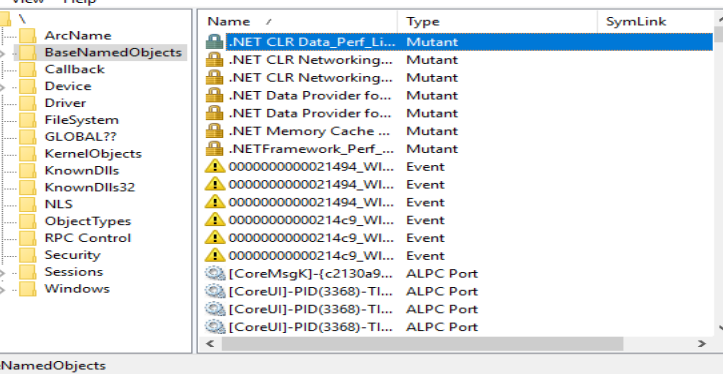
Using the filter

* 1. What’s another Sysinternals utility mentioned in the article to find mutex objects? What is the command to list only mutex objects using this tool?

EnumerateMute

handle -a | findstr Mutant

1. Use System internals tool WindObj  BaseNamedOBjects to identify mutex and semaphores created.

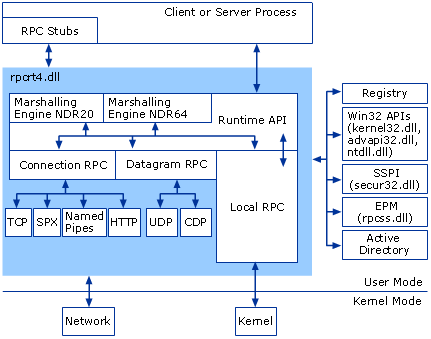


1. Demo results to instructor

|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |

4.0 IPC - Remote Procedure Call

RPCs allows a process to shared data and call functions from another process, the processes can be on different computers. Windows rely heavily on the server/client model and uses RPC for all server/client communications.



Source: <https://technet.microsoft.com/en-us/library/cc738291(v=ws.10).aspx>

Synchronization in RPC is managed using an event-pair object – a pair of event objects to signal the arrival of new messages from the client thread and the server thread. A process uses these objects to communicate by obtaining a handle to the client event object, the server event object, or both.

1. Open procexp and examine the procexp process. Find the event-pair objects. What are the names of the client and server event objects?
2. Use WinObj tool  KnownDlls and identify rpcrt4.dll.
   1. What is the purpose of this library?

* is the Remote Procedure Call (RPC) API, used by Windows applications for network and Internet communication.
  1. These libraries are type **section**. What is the meaning of **section**? (Reference : Windows Internals Six Edition part 1- Table 3-8 Executive Objects exposed to Windows API – page 143-144)
* A region of shared memory (known as a file-mapping object in Windows).

*Pro-tip: There has been several serious security issues with Microsoft’s RPC services. Attackers can listen on one of the RPC ports and initiate DoS attacks or compromise the system via privilege escalation.*

5.0 DLLs

Multiple processes can share code and resources in Windows via Dynamic Link Libraries (DLL). They usually have the file extension of .dll and are designed to run inside a host process. This gives the DLL access to all of the process’ resources – its threads, handles, and the full range of the process’ memory.

*Note: the operating system runs many DLLs as services through the* *svchost process.*

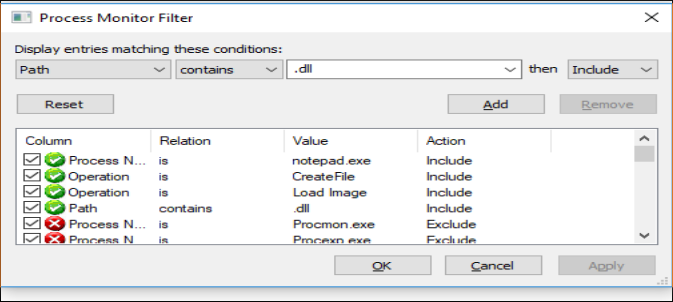
1. Open procexp. The DLL view of the System process shows all the images mapped into kernel memory, including the kernel and all the loaded device drivers.
2. Examine the list of DLLs mapped to the System process in procexp. What is the path for the Hardware Abstraction Layer DLL?



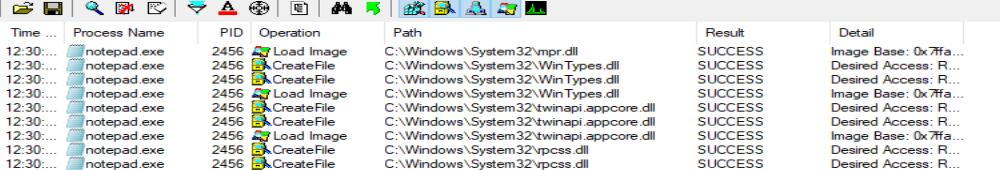
*Pro-tip: DLLs can be hijacked! Malicious code are placed in a different folder but has the same name as the real DLL. Always check the path of the DLL! You can double-click on a DLL to display the Properties dialog box to verify its digital signature.*

DLLs can be loaded as part of a process’s initialization routines, or a thread explicitly calling a library during run-time. DLLs may also depend on other DLLs which are also loaded into the process’s address space.

1. Open procmon and set the following filters:
   * Process Name is notepad.exe
   * Operation is CreateFile
   * Operation is LoadImage
   * Path contains .dll



1. Capture the events as you start a notepad instance.



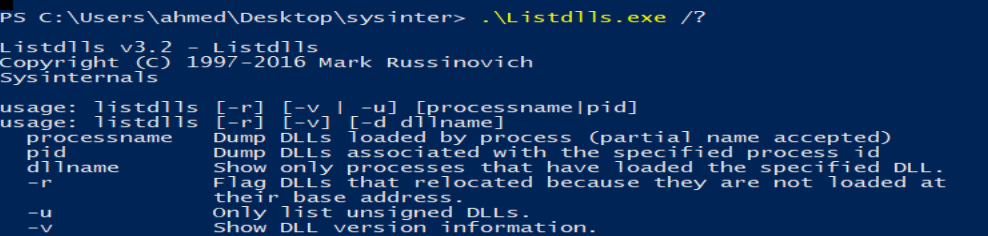
**Demonstrate and explain** how you can use this to find vulnerabilities in a particular application to the instructor.

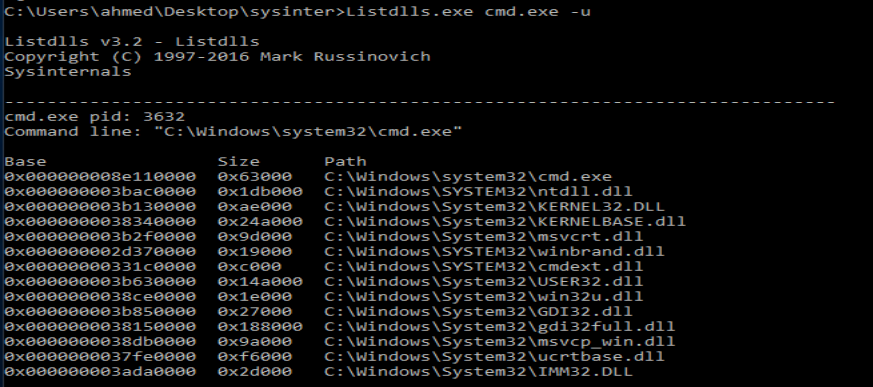
* Checking dll location and hashing the file checking virus total.

|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |

ListDLLs is another Sysinternals utility that displays information about DLLs loaded in processes in the system. It can show you all DLLs in use throughout the system or in specific processes and it allows you to search for processes that have a specific DLL loaded. The most powerful aspect of this tool is the ability to search for unsigned DLLs in use and flag DLLs that were relocated or replaced after they were loaded.

1. Start a console window from the Sysinternals folder. Use the /? switch to find out the exact command to list unsigned DLLs loaded by the cmd process along with their version information. Demo results to instructor





|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |