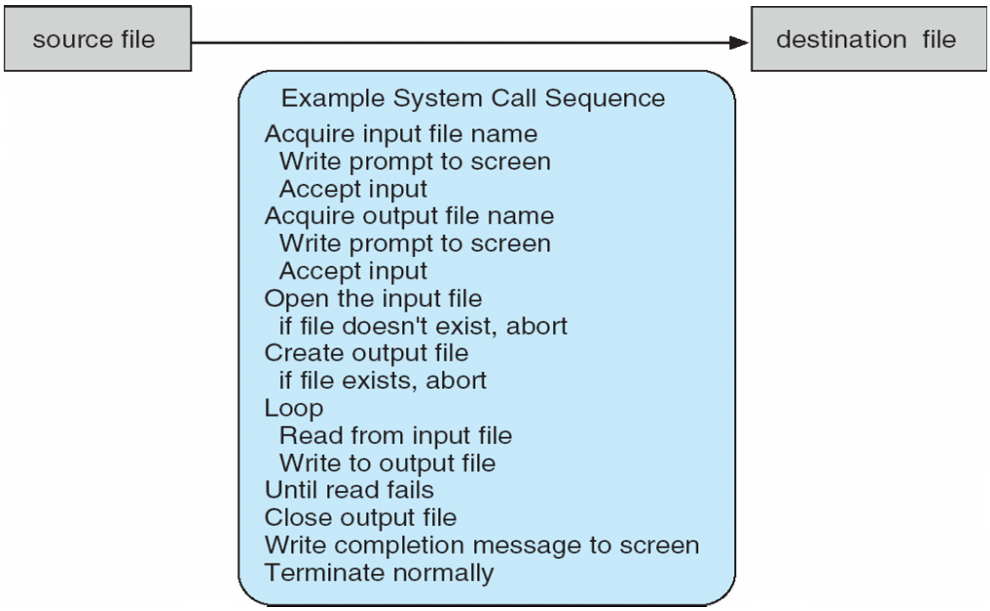
Guia de lectura - Actividades

1. Dele un vistazo al siguiente texto y determine:
   a. tema y subtema,
   b. información que brindan los cuadros y las viñetas.
2. Copie la definición de *system calls* y trasládela al español.
3. Indique en qué lenguajes se escriben estos llamados.
4. Puntualice cuáles podrían ser los pasos para copiar los contenidos de un archivo a otro en una secuencia de llamados.
5. Explique cómo se obtiene la API de la función `read ()` y qué debe incluir un programa que utilice esta función.
6. Describa los parámetros de esta función e indique qué pasa si ocurre un error.
7. Construya una grilla con los distintos tipos de llamados y sus funciones.

System Calls

System calls are programming interfaces that serve to invoke the services offered by the operating system. These calls are written in high-level languages such as C and C ++.

System call sequence to copy the contents of one file to another file:



EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t     read(int fd, void *buf, size_t count)
```

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer into which the data will be read
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

Types of System Calls

Process control:

- create process, terminate process
- end, abort
- load, execute
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory
- Dump memory if error
- **Debugger** for determining **bugs, single step** execution
- **Locks** for managing access to shared data between processes

File management:

- create file, delete file
- open, close file
- read, write, reposition
- get and set file attributes

Device management:

- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

Communications

- create, delete communication connection
- send, receive messages if message passing model to host name or process name
- From client to server
- Shared-memory model create and gain access to memory regions
- transfer status information
- attach and detach remote devices

Protection

- Control access to resources
- Get and set permissions
- Allow and deny user access

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

The following illustrates various equivalent system calls for Windows and UNIX operating systems.

	Windows	Unix

Process control	`CreateProcess()`
	`ExitProcess()`
	`WaitForSingleObject()`
		`fork()`
		`exit()`
		`wait()`

File management	`CreateFile()`
	`ReadFile()`
	`WriteFile()`
	`CloseHandle()`
		`open()`
		`read()`
		`write()`
		`close()`

Device management	`SetConsoleMode()`
	`ReadConsole()`
	`WriteConsole()`
		`ioctl()`
		`read()`
		`write()`

Information maintenance	`GetCurrentProcessID()`
	`SetTimer()`
	`Sleep()`
		`getpid()`
		`alarm()`
		`sleep()`

Communications	`CreatePipe()`
	`CreateFileMapping()`
	`MapViewOfFile()`
		`pipe()`
		`shm_open()`
		`mmap()`

Protection	`SetFileSecurity()`
	`InitlializeSecurityDescriptor()`
	`SetSecurityDescriptorGroup()`
		`chmod()`
		`umask()`
		`chown()`