**Inter-Process Communication In Linux systems**

**As linux is a wide subject in itself so we will focus on Inter process communication within linux system how they work and why they help achieve synchronization between processes**

**IPC stand for Inter-Process Communication, they** are **a powerful synchronization mechanism in Linux that allows processes to coordinate access to shared resources. They provide a way for multiple processes to communicate and synchronize their activities, ensuring that shared data is not corrupted and that operations are carried out in a consistent manner.**

****IPC where Semaphore are used:****

****Message Sharing:****

**It can be used to control access to message queues, preventing multiple processes from trying to read or write to the same queue simultaneously.**

**Pipes:**

**It can be used to synchronize the reading and writing of data from pipes, preventing race conditions and ensuring that data is not lost.**

****Memory Sharing****

**It can be used to protect access to shared memory regions, ensuring that only one process can modify the shared data at a time. This prevents conflicts in accessing shared memory.**

**Rsync is great example of an application which uses semaphores and the above mentioned IPC concepts to implement synchronization of files. it employs synchronization mechanisms to ensure data integrity and consistency during file transfers. It can also synchronization files between different machines, the transfer of file to the different machine will take place using sockets.**

**Test cases:**

**1) Transfering a single file using rsync**

**rsync will ensure that the file that you are transfering is locked so that no other process can write to the same file while rsync is writing to it**

**it will also handle the syncing of the file while transfering**

**2) Transfering multiple files using rsync**

**rsync will employ a counting semaphore technique to achieve the transfering of mulitple files this will use the same mechanism as when you are transfering a single file but it will do so in parallel**

**3) Transfering of files to a different computer**

**rsync will make use of sockets to communicate to the rsync server on the remote computer transfering the file through the socket, then when the file arives at the remote computer rsync will employ different ipc techniques to orchestrate the creation of that file**

**Sample code of how rsync might achieve transfer of multiple files using semaphores ( counting**

**semaphore )**

****PYTHON:****

**import** **os**

**import** **threading**

**import** **time**

**from** **ctypes** **import** \*

*# Semaphore operations*

SEM\_WAIT = 0

SEM\_TRYWAIT = 1

SEM\_POST = 2

*# Functions to implement semaphores*

*# Create a semaphore*

**def** sem\_create(name, value):

sem = sem\_open(name, O\_CREAT | O\_EXCL, 0666, value)

**if** sem == SEM\_FAILED:

**raise** **OSError**("Failed to create semaphore: **{}**".format(os.strerror(errno.errno)))

**return** sem

*# Open an existing semaphore*

**def** sem\_open(name, flags, mode, value):

sem = ctypes.CDLL("libsem.so").sem\_open(ctypes.c\_char\_p(name.encode()), flags, mode, value)

**if** sem == SEM\_FAILED:

**raise** **OSError**("Failed to open semaphore: **{}**".format(os.strerror(errno.errno)))

**return** sem

*# Post to a semaphore*

**def** sem\_post(sem):

result = ctypes.CDLL("libsem.so").sem\_post(sem)

**if** result == -1:

**raise** **OSError**("Failed to post to semaphore: **{}**".format(os.strerror(errno.errno)))

*# Close a semaphore*

**def** sem\_close(sem):

result = ctypes.CDLL("libsem.so").sem\_close(sem)

**if** result == -1:

**raise** **OSError**("Failed to close semaphore: **{}**".format(os.strerror(errno.errno)))

*# Unlink a named semaphore*

**def** sem\_unlink(name):

result = ctypes.CDLL("libsem.so").sem\_unlink(ctypes.c\_char\_p(name.encode()))

**if** result == -1:

**raise** **OSError**("Failed to unlink semaphore: **{}**".format(os.strerror(errno.errno)))

**def** transfer\_file(file\_path, semaphore):

**with** open(file\_path, 'rb') **as** f:

data = f.read()

*# ... (transfer data using a network socket or other method)*

sem\_post(semaphore)

**def** main():

semaphore\_name = "/transfer\_semaphore"

semaphore = sem\_create(semaphore\_name, 1) *# Initially available*

*# Create multiple threads to transfer files*

files\_to\_transfer = ["file1.txt", "file2.txt", "file3.txt"]

threads = []

**for** file **in** files\_to\_transfer:

thread = threading.Thread(target=transfer\_file, args=(file, semaphore))

threads.append(thread)

thread.start()

*# Wait for all threads to finish*

**for** thread **in** threads:

thread.join()

sem\_close(semaphore)

sem\_unlink(semaphore\_name)

**if** \_\_name\_\_ == "\_\_main\_\_":

main()

**C**

#include *<stdio.h>*

#include *<stdlib.h>*

#include *<pthread.h>*

#include *<semaphore.h>*

#include *<string.h>*

#define NUM\_FILES 5

#define BUFFER\_SIZE 1024

sem\_t file\_transfer\_sem;

void \*transfer\_file(void \*arg) {

char \*filename = (char \*)arg;

FILE \*fp\_in, \*fp\_out;

char buffer[BUFFER\_SIZE];

size\_t bytes\_read;

fp\_in = fopen(filename, "rb");

**if** (fp\_in == NULL) {

perror("fopen");

**return** NULL;

}

char new\_filename[strlen(filename) + 4];

strcpy(new\_filename, filename);

strcat(new\_filename, ".copy");

fp\_out = fopen(new\_filename, "wb");

**if** (fp\_out == NULL) {

perror("fopen");

fclose(fp\_in);

**return** NULL;

}

*// Acquire the semaphore to ensure exclusive access*

sem\_wait(&file\_transfer\_sem);

**while** ((bytes\_read = fread(buffer, 1, BUFFER\_SIZE, fp\_in)) > 0) {

fwrite(buffer, 1, bytes\_read, fp\_out);

}

*// Release the semaphore*

sem\_post(&file\_transfer\_sem);

fclose(fp\_in);

fclose(fp\_out);

printf("File %s transferred successfully.**\n**", filename);

**return** NULL;

}

int main(int argc, char \*argv[]) {

pthread\_t threads[NUM\_FILES];

int i;

*// Initialize the semaphore*

sem\_init(&file\_transfer\_sem, 0, 1);

*// Create threads to transfer each file*

**for** (i = 1; i < argc; i++) {

pthread\_create(&threads[i - 1], NULL, transfer\_file, argv[i]);

}

*// Wait for all threads to finish*

**for** (i = 0; i < argc - 1; i++) {

pthread\_join(threads[i], NULL);

}

*// Destroy the semaphore*

sem\_destroy(&file\_transfer\_sem);

**return** 0;

}