TP3 – Ejercicio 2 – Sistemas Operativos

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Métrica | Caso 1  Archivos | Caso 2  FIFO | Caso 3  Memoria Compartida | Caso 4  Sockets localhost | Caso 5  Sockets IP local | Caso 6  Sockets Red |
| Tiempo reloj | 8.512 | 8.196 | 0.901 | 0.173 | 0.180 | 0.238 |
| Tiempo CPU sistema total | 0 | 0 | 0.004 | 0.008 | 0.008 | 0.124 |
| Tiempo CPU usuario total | 4.256 | 4.100 | 0 | 0 | 0 | 0.008 |
| Cantidad de Soft Page Faults | 12 | 12 | 8 | 5 | 6 | 7 |
| Cantidad de Hard Page Faults | 0 | 0 | 0 | 0 | 0 | 0 |
| Operaciones de Entrada (en bloques) | 0 | 0 | 0 | 0 | 0 | 0 |
| Operaciones de Salida (en bloques) | 8 | 0 | 0 | 0 | 0 | 0 |

* Analice el comportamiento y los datos obtenidos en cada uno de los casos, compare los resultados entre los mismos y explique a qué se deben las diferencias y similitudes en los comportamientos observados
* Indique cuál/es métodos pueden ser utilizados en entornos distribuidos (más de una computadora). Indique qué características tiene que tener la instalación para que esto suceda
* Indique que conceptos teóricos están involucrados en cada uno de los casos

Se puede ver que el primer caso de archivos (1) es muy lento dado por lo costoso en términos de tiempo que es escribir en el filesystem (en un archivo), además, el mecanismos de sincronización para ambos procesos por señales (según consigna no se podía usar semáforos) es lento, ya que cada señal enviada es una syscall que tiene overhead en el SO.

Por su parte, utilizando fifos (2) con un solo fifo bidireccional, se sigue utilizando como soporte el filesystem ya que el fifo es una archivo del filesystem que el SO administra para poder comunicar entre procesos, sin embargo logra una performance mejor que el caso de archivos, utilizando el mismo mecanismo de señales para sincronizar ambos procesos.

Al contrario, el caso de memoria compartida (3) es rápido, si bien se debe administrar la zona crítica con semáforos, lo cual hace un poco más lenta la ejecución, al compartir memoria sin escribir nada en filesystem entonces aprovecha la velocidad de la memoria del sistema,

Por último, los 3 casos de socket tienen performance similares, evidentemente el que se ejecuta de manera localhost (4) será más rápido comparado con los que se ejecutan de manera remota (6), por su parte, el caso especial que se ejecuta de manera remota pero con la dirección de ip de red (5), es más rápida que la que se ejecuta de manera remota (6) , ya que la información tarda de llegar de un host a otro, donde puede incidir el congestionamiento de la red y la perdida de paquetes. Si bien los casos de Socket se apoyan en estructuras de SO en memoria y no en el filesystem, toda la administración de la comunicación (controles de protocolo, chequeo de paquetes, reenvío de paquetes, etc) generan overhead pero los mecanismos de sincronización son más fáciles (no hace falta usar semáforos, ya que SO maneja los sokets), con lo cual son los que tienen mejor performance.

El único método que puede utilizarse en entornos distribuidos es el caso 6 Socket, ya que el resto de los casos utilizan estructuras de información que administra el SO y las comparten entre procesos de manera que los procesos deben estar en la misma PC. En cambio el uso de Sokets permite la conexión entre 2 PC, es decir 2 procesos cada uno en una PC distinta. Para utilizar Sockets, es necesario que ambas PC estén conectadas a la misma red.