INFORME TRABAJO PRÁCTICO Nº1

Nivel y Área:

72.11 – Sistemas Operativos

1º Cuatrimestre 2018

Comisión: S - **Carrera:** Informática

Fecha: 4 de abril de 2018

Alumnos Expositores:

AQUILI, Alejo Ezequiel (57432)

BASSANI, Santiago (57435)

RITORTO, Bianca (57082)

SANGUINETI ARENA, Francisco Javier (57565)

Informe: Trabajo Práctico Nº1 - Inter Process Communication

Objetivo

El objetivo del trabajo práctico es desarrollar un sistema para el cálculo distribuido de los hashes md5 de un grupo de archivos recibidos por parámetro. Para esto se utilizaron diferentes procesos y mecanismos de comunicación y sincronización entre ellos.

Desarrollo e implementación

El sistema consta de un proceso Application, un proceso Slave y un proceso View.

El proceso Application recibe como parámetro los nombres de archivos a calcular su hash md5 y luego por medio de la utilización del IPC **POSIX Message Queue** distribuye la carga de archivos por cantidad de archivos entre los procesos Slave generados por Application tantos como la constante SLAVE_QTY y a través de **fork()** y **exec().** Cada proceso Slave desencola un archivo de la cola de "pedidos" común a todos los Slave y comienza a realizar el calculo del hash (Secuencialmente uno por vez). Cuando un Slave termina de calcular un hash, si no hay mas archivos en la cola de pedidos común, finaliza su ejecución. Es decir, Application envía una cierta cantidad de archivos a cada Slave, donde esa cierta cantidad es uno, cada Slave o server calcula de a un archivo por vez su correspondiente hash, repitiendo esto hasta agotar la cola de pedidos del client común a todos los servers.

Para obtener el hash md5 se utiliza el utilitario **md5sum**. El Slave para poder obtener la información del utilitario realiza un **fork()** junto con el uso de un IPC **pipe()** para leer la salida estándar de la ejecución del comando system(md5sum file). Una vez obtenido el hash del proceso hijo, el proceso padre acondiciona la información y la envía de regreso al proceso Application por medio de otra Message Queue que garantiza el comportamiento FIFO y por lo tanto garantiza el orden de llegada de los hash.

La utilización de la cola de pedidos común y la cola de hash permiten no sobrecargar el funcionamiento del proceso Application, permitiendo al mismo poder insertar los resultados de los cálculos en el buffer de llegada compartido sin tener que estar pendiente de los archivos, ya que anteriormente creo una cola de pedidos que se encarga de la distribución de carga por cantidad de archivos y de esta forma aprovechar la concurrencia en el calculo de los procesos Slave.

El proceso Application para evitar una espera activa en nuestro sistema distribuido y aprovechar la programación concurrente, utiliza sobre la cola de llegada de los hashes

md5, la función **select()** para monitorear la misma. Por otra parte, el proceso Application crear una región de memoria compartida haciendo uso de **POSIX Shared Memory** para la creación de un buffer de llegada compartido con un/unos posible/s proceso/s View. Para la sincronización de la lectura y escritura de dicho buffer lineal entre dichos procesos, se utiliza un mecanismo de sincronización con semáforos, utilizando **POSIX named Semaphores**.

El proceso View, obtiene acceso a la región de memoria compartida y el named semaphore gracias a recibir como parámetro el PID del proceso Application. El buffer compartido, dispone de un centinela (EOF) para indicar el final del buffer. Por otra parte, el proceso View evita nuevamente realizar una espera activa, utilizando **select()** para monitorear el buffer de llegada.

Por último, el proceso Application independientemente de la existencia de algún proceso View, garantiza el almacenamiento de los hashes obtenidos en un archivo en disco **output.txt.**

Decisiones importantes en el diseño

Durante el desarrollo e implementación del sistema se han tomado decisiones importantes desde el punto de vista del diseño. La primera decisión importante de diseño tomada fue intentar seguir un desarrollo del tipo TDD (Test Driven Development) el cual no fue llevado a cabo con éxito y si bien se intento perseguir la cultura de extremme programming, no se lo hizo de forma estricta. Una segunda decisión importante, fue apegarse al standard POSIX, por ello todos los mecanismos de IPC mencionados son los correspondientes al standard.

Otra decisión importante tomada durante el desarrollo fue la utilización de message Queue como mecanismo de IPC principal para la comunicación entre Application y los Slaves. Esta decisión fue tomada en torno a conocer la existencia de este mecanismo de IPC por los contenidos teóricos de la cátedra, y fue impulsada por las curiosidad y ganas de conocer nuevos mecanismos de IPC por parte de los integrantes del grupo. Ya que los integrantes han trabajado con pipes y FIFOs anteriormente. Por lo tanto, se decidió trabajar con message Queue dado que además este mecanismo posee una estructura de datos subyacente de una cola de prioridades. Al ser utilizada con igual prioridad entre mensajes representa una cola y esta es la estructura de datos buscada para la asignación de tareas entre procesos. Si bien podría resultar más difícil la implementación utilizando este mecanismo de IPC frente a otros anteriormente mencionados como FIFOs, se decidió primando la innovación y no el principio de simplicidad.

En cuanto a las decisiones relacionadas con la espera activa, se primó evitar cualquier espera activa en el sistema mediante el uso de select().

En lo que respecta a las interfaces de POSIX se han desarrollado tipos abstractos de datos ADT que encapsulan el comportamiento de las interfaces de POSIX para independizar el sistema del standard con la idea que si en un futuro por algún motivo se desea remplazar ya sean las interfaces de messages Queues, Semáforos o memoria compartida de POSIX, por otras interfaces distintas. Por ejemplo, las interfaces de System V, basta con modificar dichos tipos abstractos de datos para que nuestro sistema trabaje con otras interfaces.

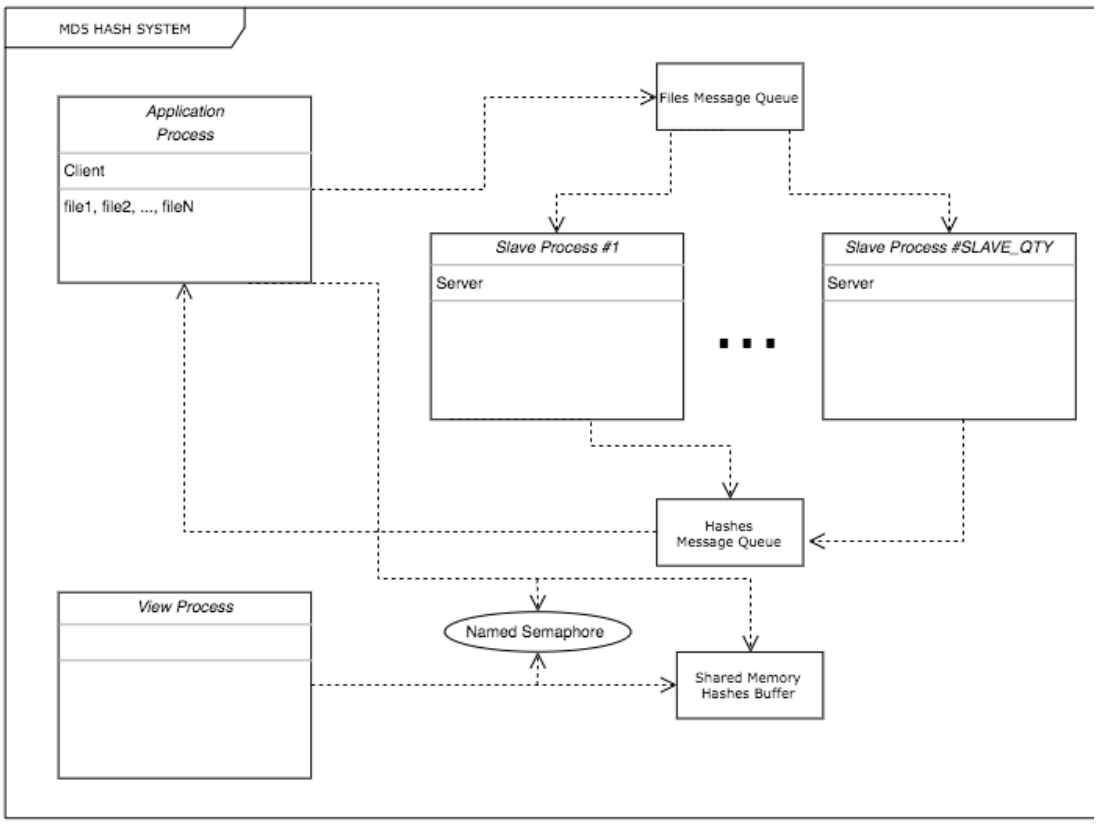

Figura 1 - Diagrama sintético del funcionamiento del sistema.

Dificultades

Dado que se decidió respetar el standard POSIX para la implementación se presentaron varias dificultades asociadas a esta decisión. Primero, y gracias al desarrollo guiado por test surgió la necesidad de hashear más de 10 archivos lo cual traía problemas con las messages Queues del standard. Por el hecho de que este mecanismo de IPC no es muy utilizado en los sistemas que adhieren a POSIX, nos encontramos que Mac OS no posee esta interfaz mientras que Linux sí, pero al no ser utilizadas por desarrolladores han quedado en el olvido y Linux tiene un archivo indicando un máximo relativo de elementos que puede encolar una message Queue. Este máximo relativo es 10 y es excesivamente menor al máximo absoluto tolerado por el sistema operativo. Para solucionarlo es necesario correr el sistema con permisos de sudo para poder cambiar el archivo de **/dev/mqueue** con el máximo relativo y de esta forma poder encolar más de 10 archivos y el máximo absoluto definido nos permite encolar tantos archivos como sean necesarios ya que este ultimo es un número de grandes magnitudes.

También se presentaron problemas de permisos para acceder a **/dev/shm** donde se almacenan las referencias a las named shared memory y los named semaphore, lo cual imposibilitaba poder abrir el semáforo necesario para la sincronización entre el proceso View y el proceso Application desde el proceso View. Esto ocurre al intentar correr el proceso Application con un proceso View teniendo permisos de sudo en el primero de ellos y no en el segundo. Para solucionar este problema, también fue necesario correr el proceso View con igualdad de permisos que el proceso Application, es decir ambos con permisos de sudo.

Instrucciones de compilación y ejecución

Para la compilación y linkedición existe un Makefile, basta con correr:

```
:~/SO-TP01$ make all
```

Para la ejecución del proceso Application se debe realizar:

```
:~/SO-TP01$ sudo ./Application/applicationProcess.out <file1> <file2> … <fileN>
```

Donde <fileI> es el nombre del i-ésimo archivo a calcular el hash. Los archivos se deben encontrar en la carpeta S0-TP01.

Para la ejecución del proceso View se debe realizar en otra terminal:

```
:~/SO-TP01/View$sudo ./viewProcess.out <PID_APP>
```

Donde <PID_APP> es el PID del proceso Application, el cual el mismo proceso Application muestra en pantalla durante su ejecución.

Para la ejecución de los Test se debe realizar:

```
:~/SO-TP01$ make test
```

Debugging

Para el debugging se utilizaron las herramientas de **valgrind** y **cppcheck**, de la siguiente manera:

Valgrind:

:~/Desktop/SO-TP01$ valgrind --leak-check=full --show-leak-kinds=all -v ./Application/applicationProcess.out b1 b2

Fragmentos de la salida generada por la invocación previa:

```
==9439== LEAK SUMMARY:
==9439==    definitely lost: 0 bytes in 0 blocks
==9439==    indirectly lost: 0 bytes in 0 blocks
==9439==      possibly lost: 0 bytes in 0 blocks
==9439==    still reachable: 72 bytes in 2 blocks
==9439==         suppressed: 0 bytes in 0 blocks
==9439==
==9439== Use --track-origins=yes to see where uninitialised values come from
==9439== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
==9439==
==9439== 1 errors in context 1 of 1:
==9439== Syscall param write(buf) points to uninitialised byte(s)
==9439==    at 0x50524A0: __write_nocancel (syscall-template.S:84)
==9439==    by 0x5051414: sem_open (sem_open.c:255)
==9439==    by 0x402101: createSemaphore (sharedMemory.c:99)
==9439==    by 0x401DDC: sharedMemoryCreator (sharedMemory.c:36)
==9439==    by 0x40119E: main (applicationProcess.c:33)
==9439==  Address 0x1fff0002ec is on thread 1's stack
==9439==  in frame #1, created by sem_open (sem_open.c:141)
==9439==
==9439== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

```
:~/Desktop/SO-TP01$ sudo valgrind --leak-check=full --show-leak-kinds=all -v ./View/viewProcess.out 2872
```

Fragmentos de la salida generada por la invocación previa:

```
==3130== LEAK SUMMARY:
==3130==    definitely lost: 0 bytes in 0 blocks
==3130==    indirectly lost: 0 bytes in 0 blocks
==3130==      possibly lost: 0 bytes in 0 blocks
==3130==    still reachable: 72 bytes in 2 blocks
==3130==         suppressed: 0 bytes in 0 blocks
==3130==
==3130== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Se indagó acerca del error y al parecer en el archive sem_open.c en la línea 255 se escribe contenido no inicializado. Este problema va más allá de el desarrollo del sistema y puede considerarse también consecuencia de la utilización de los POSIX named Semaphores.

En cuanto a los memory leaks se puede observar en el leak summary la ausencia de estos.

Cppcheck:

```
:~/Desktop/SO-TP01$ sudo cppcheck  --enable=all .
```

Salida generada por la invocación previa:

```
Checking Application/applicationProcess.c...
1/11 files checked 9% done
Checking Slave/slaveProcess.c...
2/11 files checked 18% done
Checking Test/AllTests.c...
3/11 files checked 27% done
Checking Test/CuTest.c...
[Test/CuTest.c:310]: (style) The scope of the variable 'i' can be reduced.
[Test/CuTest.c:311]: (style) The scope of the variable 'failCount' can be reduced.
4/11 files checked 36% done
Checking Test/CuTestSO.c...
5/11 files checked 45% done
Checking Test/CuTestTest.c...
```

```
[Test/CuTestTest.c:18] -> [Test/CuTestTest.c:18]: (style) Same
expression on both sides of '||'.

[Test/CuTestTest.c:137] -> [Test/CuTestTest.c:137]: (style) Same
expression on both sides of '=='.

[Test/CuTestTest.c:175] -> [Test/CuTestTest.c:175]: (style) Same
expression on both sides of '=='.

6/11 files checked 54% done

Checking Utils/errorslib.c...

7/11 files checked 63% done

Checking Utils/messageQueue.c...

8/11 files checked 72% done

Checking Utils/processlib.c...

9/11 files checked 81% done

Checking Utils/sharedMemory.c...

10/11 files checked 90% done

Checking View/viewProcess.c...

11/11 files checked 100% done

[Test/CuTest.c:217]: (style) The function
'CuAssertDblEquals_LineMsg' is never used.

[Test/CuTest.c:208]: (style) The function
'CuAssertIntEquals_LineMsg' is never used.

[Test/CuTest.c:227]: (style) The function
'CuAssertPtrEquals_LineMsg' is never used.

[Test/CuTest.c:183]: (style) The function
'CuAssertStrEquals_LineMsg' is never used.

[Test/CuTest.c:255]: (style) The function 'CuSuiteDelete' is
never used.

[Utils/processlib.c:25]: (style) The function
'childFactoryWithArgs' is never used.

[Utils/sharedMemory.c:50]: (style) The function 'getId' is never
used.

[Utils/messageQueue.c:74]: (style) The function
'setMQAttributes' is never used.

(information) Cppcheck cannot find all the include files (use --
check-config for details)
```

Testing

Como se mencionó anteriormente, se intentó seguir la filosofía de extreme programming y llevar a cabo TDD, dado que ninguno de los integrantes del grupo ha trabajado con TDD con anterioridad, se decidió abandonar dicha estrategia, ya que, si bien se comenzó realizando test unitarios para el desarrollo guiado por testing, no se lograba dejar de lado la consigna y abstraerse de los requerimientos específicos para la realización del trabajo, que indirectamente influenciaban nuestro desarrollo.

Finalmente, se decidió planificar nuevamente el enfoque del testing en nuestro desarrollo, pero de **ninguna** manera se lo descartó del mismo. Se decidió implementar test unitarios de los distintos componentes del sistema. Para ello, se utilizó el Framework CUTest (Copyright (c) 2003 Asim Jalis.) (https://github.com/aiobofh/cutest.git).