Trabajo Práctico 1
Sistemas Operativos (72.11)

Juan Godfrid - 56609
Pablo Radnic - 57013
Joaquín Ormachea -
Francisco Delgado -

2 de abril de 2018

Índice general

Capítulo 1

Diseño

1.1. Diseño del Sistema

dshfdshfdshfds

1.2. IPCs

1.2.1. Message Queue

El sistema utiliza dos *message queues* del tipo POSIX (son más modernas que sus contrapartes SYSTEM V) en la intercomunicación de los procesos maestro y esclavo. El primero se utiliza para enviar los archivos que se requieren *hashear* hacia los esclavos, mientras que el segundo se utiliza para enviar los *hashes* ya procesados devuelta al proceso maestro. La utilidad de dicho IPC es una multicausalidad. Al ser una cola, el *message queue* asegura el comportamiento FIFO, por lo que es ideal para enviar los *hashes* de vuelta al proceso maestro, ya que asegura que el orden de llegada corresponde al orden de procesamiento. El *message queue* también resulta adecuado para enviar los nombres de los archivos a los esclavos ya que los mensajes sirven como instrucciones atómicas: Los esclavos leen las instrucciones de la cola, cuando terminan, leen otra instrucción, y cuando está vacía la cola significa que el procesamiento concluyó. Todo esto se realiza sin gastar un solo ciclo de procesador del proceso maestro.

1.2.2. Señales

Capítulo 2

Utilización

2.1. Compilación y Ejecución

El sistema posee un archivo de tipo *GNU make* que discrimina en la compilación del código fuente en base a qué proceso debería pertenecer, de este modo logra compilar separadamente los archivos binarios de manera autónoma. El programa debe recibir como argumento por línea de comandos una lista de archivos a procesar, también puede recibir una expresión *bash* para los archivos (por ejemplo: ./*.c). Un ejemplo de uso del programa es el siguiente:

```
/SO–TP1$ make
/SO–TP1$ ./Binaries/run main.c
```

2.2. Proceso Vista

El *make* compila también de manera separada el proceso vista, para ejecutarlo se debe enviar el siguiente comando:

Capítulo 3

Problemas y Resoluciones

3.1. Proceso Maestro

3.1.1. Problema de espera activo

msq recieve con block asi no hace espera activa sino que el SO lo despierta cuando la queue cambia de empty a nonempty

3.1.2. *Hasheo* de archivos

Nos vimos ante la necesidad de utilizar la función de Linux md5sum para conseguir el hash de los archivos. Sin embargo este función era únicamente accesible a través de línea de comandos de Linux y no en el entorno de biblioteca de funciones de c. Al no tener las herramientas para crear una función propia debimos encontrar otra solución para el problema.

Luego de investigación en Internet se propuso utilizar la función popen() con parámetro cmd que ejecuta un proceso de línea de comandos de Linux y a través de él ejecutar la función md5sum con los parámetros correspondientes. Luego la salida de la ejecución de las funciones es traída al entorno de la aplicación con la función fgets.

3.2. Proceso Esclavo

sdfasdf

3.3. Proceso Vista

asdfadsf Dummy text

Capítulo 4

Testing

4.1. Sistema de Testing

asdfasldjfasd

4.2. Fugas de Memoria

Era posible la presencia de fugas de memoria en cualquiera de los tres procesos, las fugas de memoria son errores de software que ocurren cuando los bloques de memoria reservados no son liberados antes de la finalización del programa. Para asegurar la ausencia de fugas de memoria se tomo el recaudo de utilizar la herramienta *Valgrind*. A continuación se encuentran los registros del programa para distintos parámetros.

Registros de Valgrind:

```
>>valgrind ./Binaries/run ./*

==7295==
==7295== HEAP SUMMARY:
==7295==     in use at exit: 0 bytes in 0 blocks
==7295==   total heap usage: 11 allocs, 11 frees, 17,528 bytes allocated
==7295==
==7295== All heap blocks were freed -- no leaks are possible
==7295==
==7295== For counts of detected and suppressed errors, rerun with: -v
==7295== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)


>>valgrind ./Binaries/view 1234

==7550==
==7550== HEAP SUMMARY:
==7550==     in use at exit: 0 bytes in 0 blocks
==7550==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==7550==
```

```
==7550== All heap blocks were freed -- no leaks are possible
==7550==
==7550== For counts of detected and suppressed errors, rerun with: -v
==7550== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

```
>>valgrind ./Binaries/slave a a
```

```
==8156==
==8156== HEAP SUMMARY:
==8156==     in use at exit: 0 bytes in 0 blocks
==8156==   total heap usage: 4 allocs, 4 frees, 8,704 bytes allocated
==8156==
==8156== All heap blocks were freed -- no leaks are possible
==8156==
==8156== For counts of detected and suppressed errors, rerun with: -v
==8156== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Podemos afirmar que para los casos vistos nuestros programas no presentan fugas de memoria.

Capítulo 5

Limitaciones

5.1. Limitaciones del Sistema

asdfasldjfasd

5.2. Limitaciones de la Implementación

asdfasdf Dummy text

Capítulo 6

Bibliografía

6.1. Extractos de código

1. https://stackoverflow.com/questions/3395690/md5sum-of-file-in-linux-c
2. https://stackoverflow.com/questions/4553012/checking-if-a-file-is-a-directory-or-just-a-file