Sistemas Operativos

Trabajo Práctico N°3

Estructuras de Administración de Recursos

24 de Octubre de 2018

Agustina Osimani 57526

Nicolás Barrera 57694

Ezequiel Keimel 58325

Marcos Lund 57159

Pipes

Se implementaron unnamed pipes en el sistema de archivos que operan de la misma manera que lo hacen en el sistema operativo de Linux. Al crearlos se establece una punta de escritura y una de lectura. A partir de estos unnamed pipes, se incorporó la posibilidad de redireccionar la salida estándar de cualquier programa a la entrada estándar de otro utilizando '|' en la consola. De esta forma, la salida estándar del primer programa se redirecciona a la punta de escritura del unnamed pipe, y la punta de lectura del unnamed pipe se redirecciona a la entrada estándar del segundo programa. Este comportamiento se generalizó para permitir el uso de múltiples pipes en un solo comando.

También se implementaron named pipes haciendo uso del sistema de archivos implementado en el tp anterior.

Mejoras al Physical Memory Management

Se implementó Buddy Allocation para poder reservar varias páginas físicas contiguas en el Trabajo Práctico N°2. Los detalles técnicos de esto fueron explicados en el informe de dicho trabajo.

Mejoras al Scheduler

Se mantuvo el esquema de scheduling multi cola, y se implementó un nuevo tipo de cola, la cola de prioridades. Esta cola soporta hasta 5 niveles de prioridad, e implementa una lista de threads por cada uno.

Al comenzar a recorrer esta cola, si la misma no está vacía, el scheduler buscará un proceso en la lista de máxima prioridad. Si no encuentra nada allí, buscará en la de prioridad siguiente, y así sucesivamente hasta encontrar un thread. Cuando encuentre un thread, guardará en qué nivel de prioridad lo encontró para no empezar de cero y correr riesgo de inanición en el próximo uso. Virtualmente, la cola de prioridades de comporta, justamente, como una priorityQueue de Java, pero circular; Como un round robin común, con la diferencia de que el quantum por el que correrán depende de la prioridad (nice + 3), y están ordenados en la inserción por prioridad.

Si un thread intenta ser agregado a la cola pero tiene nice 0, se lo envía a la cola de round robin tradicional, de quantum 3.

Para garantizar que todos los procesos dispondrán de un tiempo de procesamiento justo, se limita el timelapse de cada cola. En nuestra implementación, procesos de la cola de prioridad correrán durante un máximo de 8 ticks, mientras los de la cola RoundRobin lo harán durante 3.

Esta implementación garantiza que, si un thread corre, deberá esperar a que todos los demás threads *ready* corran durante la cantidad de quantum que les corresponda según su prioridad antes de volver a correr. La inanición es, bajo este esquema, imposible.

Filósofos Comensales

Para realizar la simulación de los filósofos comensales se tiene un semáforo que representa la cantidad de filósofos que pueden acceder a la mesa donde se come. Debido a que en cualquier momento dado, hay la misma cantidad de filósofos que tenedores, si se restringe el acceso a la mesa para que haya la cantidad total de filósofos menos uno, nunca

se producirá un deadlock ya que siempre algún filósofo podrá comer. Cuando finalice, dejará los dos tenedores que utilizó y se irá de la mesa para ponerse a pensar. Esto permite que si un filósofo quedó afuera de la mesa, pueda ingresar para esperar por su turno para utilizar dos tenedores.

Cada vez que nace un filósofo, se aumenta la cantidad de tenedores (presentes en forma de semáforos) y los lugares en la mesa en uno. Al morir, estas cantidades disminuyen en uno.

Aplicaciones de Userspace

Pipes

Para demostrar el correcto funcionamiento de los pipes, se desarrolló un programa “prueba” que recibe argumentos de un pipe y los imprime luego de un mensaje, intercalando 1s entre los caracteres recibidos. Por lo tanto, si se corre el comando

```
Terminalator>  echo hola | prueba
```

desde la línea de comandos el resultado que se observa es:

```
Jelou1h1o1l1a1
```

De esta manera, se comprueba que la redirección de las entradas y salidas de ambos procesos funciona según lo esperado.

Nice

Recibe el número de PID cuya prioridad se desea modificar y el valor de la prioridad en cuestión, comprendido entre 0 y 5. Posteriormente, y si los valores ingresados son correctos, se ajusta la prioridad del proceso y se mueven todos sus threads a la correspondiente cola de prioridades.

```
Terminalator>  nice 2 4
```

La invocación exhibida modifica el valor de prioridad del proceso de PID 2 a 4.

Para demostrar el correcto funcionamiento de la rutina, se provee el comando priorityTest, que recibe dos valores de prioridad entre uno y cinco, y ejecuta dos programas, cada uno con su respectiva prioridad. Los programas son idénticos, con la diferencia de que uno imprime la letra A y el otro la letra B. Diferencias importantes entre las prioridades, por ejemplo 0 y 5, harán evidente que el proceso más prioritario acapara más tiempo de procesamiento.

```
Terminalator>  priorityTest 0 5
ABBBABBBABBABBBABBBAAAAAAA
Terminalator>  priorityTest 3 3
ABABABABABABABABABABABABAB
```

Filósofos Comensales

Para correr la implementación del clásico problema explicado anteriormente se debe correr el comando:

```
Terminalator> philosophers
```

siguiendo a continuación las instrucciones en pantalla.

Problemas encontrados durante el desarrollo

- Fue bastante complejo implementar el parseo de comandos con pipes ya que existía la posibilidad de contar con más de uno en un solo comando.
- La arquitectura ideal del scheduler requirió reprogramarlo casi por completo, lo cual dio lugar a varios errores que debieron ser resueltos en consecuencia.

Instrucciones de compilación y ejecución

Se debe navegar al directorio que contiene el código fuente y ejecutar el siguiente comando por consola:

```
user@linux:$ cd Toolchain
user@linux:$ make all
```

A continuación se debe regresar al directorio fuente y correr el mismo comando:

```
user@linux:$ cd ..
user@linux:$ make all
```

Finalmente, se debe correr el programa ejecutando:

```
user@linux:$ ./run.sh
```