Sistemas Operativos

Informe TP 2

Grupo 9

Lucas David Perri - 62746

Santiago Rivas Betancourt - 61007

Franco Panighini - 61258

Índice

Decisiones

Instrucciones de compilación y ejecución

Se encuentran en el readme de la branch “main” en el repositorio de github del proyecto.

El último commit tiene hash: d31c0f425b7b70e7c10c5bb969d5270f85c3c2a1

Instrucciones de uso para tests

Para utilizar los tests se pueden llamar desde la shell con sus respectivos nombres:

$> test-mm *mem_size*
$> test-prio
$> test-sync *iterations 1*
$> test-sync iteration 0 (test-no-sync)
$> test-processes *proc_number*

Cada test recibe dichos parámetros.
Se puede correr en background (como el resto de los procesos) utilizando el símbolo de “&”.

Limitaciones

La principal limitación con la que nos encontramos fue lo difícil que fue debuggear el sistema operativo. Varios errores que parecían simples a primera vista terminaron contándonos varias horas y bastante frustración. Además otra limitación fue el tiempo, con algunos días más podríamos haber cambiado y mejorado el código. En particular, nos quedaron algunos archivos ‘lib’ muy extensos que podrían ser subdivididos.

Una limitación importante del sistema operativo es que no soporta (crashea) el uso de dos pipes.

Problemas y Soluciones

Memory manager

Los memory manager implementados son:

- Buddy: con la memoria organizada en bloques de 256 bytes como un árbol binario aplanado y haciendo uso de un poco de matemática para ubicar los nodos rápidamente, pero tiene la desventaja de usar una gran cantidad de memoria para el árbol y mucho desperdicio con malloc() de menor tamaño.
- Free list: se almacena en dos listas (alloc list y free list) en forma de headers en la misma memoria que se utiliza, tiene la desventaja de que su complejidad temporal es de O(n) y no puede dar la totalidad del bloque de memoria asignado debido a que también se usa para almacenar las listas.

Scheduler

El scheduler es un priority based round robin. La implementación elige solo los procesos que están ready y les permite correr una cantidad de quantum determinada por su prioridad. La prioridad más alta es indicada por un número entre el 0 y el 5, siendo el número más bajo la mayor prioridad. Los procesos son elegidos igualitariamente, pero a cada uno se le da más cantidad de quantums para correr dependiendo de su prioridad.

Shell

Para la shell se utiliza un parser de strings que divide el input en un array de strings. El primer valor de ese array se interpreta como el comando. El resto se pasan como argumentos a los procesos. El arreglo está *null terminated*.

Citas de código

La gran mayoría del código del trabajo práctico es de autoría propia o perteneciente a librerías del estándar gnu-11 de C. Se utilizó como guía, código de semáforos e ipc de repositorios públicos, pero se trabajó bastante sobre los mismos para corregir errores preexistentes y ‘portear’ las partes que nos servían a nuestro sistema operativo.

Modificaciones a los tests

Para el test_prio se utilizaron las prioridades 0, 3 y 5 para que se note claramente la diferencia de prioridad.

Para bloquear y desbloquear procesos están los comandos *block* y *unblock* a diferencia del propuesto *block* que funciona como un *toggle* para bloqueo y desbloqueo de procesos.

Bibliografía

- Manual de Linux (man)

- OSDev