Trabajo Práctico 2:

Núcleo de un SO con mecanismos de administración de recursos

Integrantes:

- Mateo Reino
- Jonatan Blankleder
- Agustin German Ramirez Donoso

1 Introducción

El objetivo del tp se puede dividir en 4 objetivos principales: Crear un manejador de memoria, un manejador de procesos, una sincronización dentro de dichos procesos y diferentes aplicaciones de UserSpace pedidas por la cátedra. Fuimos capaces de implementar las 4 funcionalidades aunque tuvimos un problema principal el cual vamos a explicar más adelante

2. Manejo de memoria

2.1 Implementación

Solo se implementó una de las dos maneras que se pedía de manejar memoria (a elección). Esta usa bump allocator con free.

Pese a no tener un buddyManager, igual se configuró los makefiles y archivos bash para poder elegir entre cual manager usar en el proyecto: si se quiere usar el simpleManager, ejecutar compile.sh. Si se quiere buddyManager, compileBuddy.sh pero como buddyManager no está hecho, dará error. Se configuró igualmente para aprender a cómo se hace si se quiere compilar ciertos archivos o no

2.1 Limitaciones

Hemos utilizado un bloque de 1 MiB para el heap.

3. Procesos

3.1 Implementación

PCB: Array fijo de 256 entradas (struct pcb); campos : PID, estado, prioridad, punteros de stack, regs salvados.

Scheduling basado en el algoritmo **Round-Robin** con prioridades.

Además, concluimos que la prioridad máxima en nuestro proyecto iba a ser de 7 ya que consideramos que la diferencia entre prioridades 1 y 7 es lo suficiente. Esta diferencia se puede ver corriendo test_priority.

3.2 Limitaciones

Puede haber hasta 256 procesos al mismo tiempo.

4. Semáforos

4.1 Implementación

Buscamos hacer que la implementación de los semáforos sea lo más parecida a la vida real viéndolo desde el lado del usuario. Creamos syscalls para sem_init, sem_post, sem_wait y sem_destroy. 2 procesos se pueden conectar al mismo semáforo sabiendo su nombre.

Cuando un proceso se bloquea pasa a estar en una cola. Siempre que se hace un sem_post se liberan todos los procesos en la cola de ese semáforo.

Se asegura la atomicidad de los semáforos mediante el uso de spinLocks (aunque sea innecesario dentro de un SO unicore)

4.2 Limitaciones

Puede haber hasta 128 procesos esperando en un semáforo.
Puede haber hasta 32 semáforos.
La longitud máxima del nombre de un semáforo es de 32 caracteres.
Se le tiene que dar nombre si o si.

5. Pipes

5.1 Implementación

La implementación de los pipes también se buscó que sea lo más parecida a la vida real posible (vista desde el usuario). Además, 2 procesos se pueden conectar a una misma pipe sabiendo su nombre.

5.2 Limitaciones

Tamaño máximo del buffer de la pipe: 1924.
Longitud máxima del nombre: 32 caracteres.
Máxima cantidad de pipes: 32.
Se le tiene que dar nombre si o si

6. Aplicaciones

6.1 Implementación

Se tiene una shell (llamada nanoShell) dentro de UserLand para la ejecución de los diferentes programas para ejecutar, sin incluir a phylo que no se pudo terminar a tiempo. Se puede ver una lista de ellos escribiendo help.

Además, la shell tiene el mecanismo de ejecutar en segundo plano algún programa o, añadiendo & al final del comando, o hacer que el output de un programa sea el input de otro, escribiendo | entre medio de los comandos para cada programa. Ejemplos:

- echo hola como estas? | cat (cat recibe hola como estas?)
- loop 1& (loop se va a ejecutar en segundo plano con 1 segundo entre mensajes)
- echo hola | cat& (aunque no tenga sentido, cat recibe hola y se ejecuta en segundo plano)

Por último, se puede escribir el carácter de EOF mediante **ctrl + c** y se puede matar al proceso en foreground con **ctrl + d**.

7 Compilación y ejecución

Para compilar y ejecutar, se encuentran hechos archivos bash para crear el contenedor (create-container.sh), compilar (compile.sh o compileBuddy.sh, dependiendo que memory Manager se quiera), limpiar los archivos .o (clean.sh) y ejecutar los binarios/imágenes (run.sh). Si está usando WSL probablemente sea necesario hacer dos2unix a los mismos para que se ejecuten correctamente

Comandos a ejecutar (en orden):

7.1 Crear contenedor

./create-container

7.2 Compilar kernel y userland

./compile.sh

o

./compileBuddy.sh (no compila)

7.3 Ejecutar con QEMU

./run.sh