Trabajo Práctico Nº 2

KERNEL (GNA)

Primer cuatrimestre de 2025

Comisión S - Grupo 4

Integrantes:

Cuneo Gima, Matias	64492
Buela, Mateo	64680
Prado, Nahuel Ignacio	64276

Introducción

El siguiente informe detalla el desarrollo del trabajo práctico número dos de la materia de sistemas operativos. Este trata de la implementación de un kernel propio. En particular, uno que provee un API (Application Programming Interface) de tipo POSIX, tenga estructura monolítica y sea de sesenta y cuatro bits. Algunas de las funcionalidades con las que cuenta este son drivers de teclado y de pantalla, syscalls como read, write y exit, un sistema de manejo de procesos con un algoritmo de scheduling de tipo round robin con prioridades, dos memory managers y, por último, provee la capacidad de tener comunicación entre procesos (IPC).

Desarrollo

Decisiones tomadas durante el desarrollo

En primer lugar, se decidió que se retomaría el trabajo realizado en la materia de Arquitectura de computadoras con la diferencia de que no se utilizaría modo gráfico si no que se realizó el trabajo con modo texto. Esto llevó a una reducción drástica de código y a alteraciones al comportamiento previo, resultando en tener que reimplementar algunas funcionalidades.

Otra de las decisiones tomadas fue con respecto a la mecánica de round robin en el scheduler. A continuación se detalla el algoritmo:

Suponiendo que ya están estos procesos en ejecución: init (PID 0), shell (PID 1) y p_ejemplo (PID 2) cada uno con prioridades 0, 0 y 1 respectivamente, se almacenan con sus contextos y atributos en un arreglo circular y, a su vez, se guarda la referencia a estos en una cola (queue) dependiendo de la prioridad de este. El scheduler procederá a generar un número pseudo-aleatorio y luego dependiendo de este se calcula probabilisticamente de qué cola tomar el próximo procesos a ejecutar. De esta manera se garantiza que se respeten las prioridades como también que no hayan inaniciones.

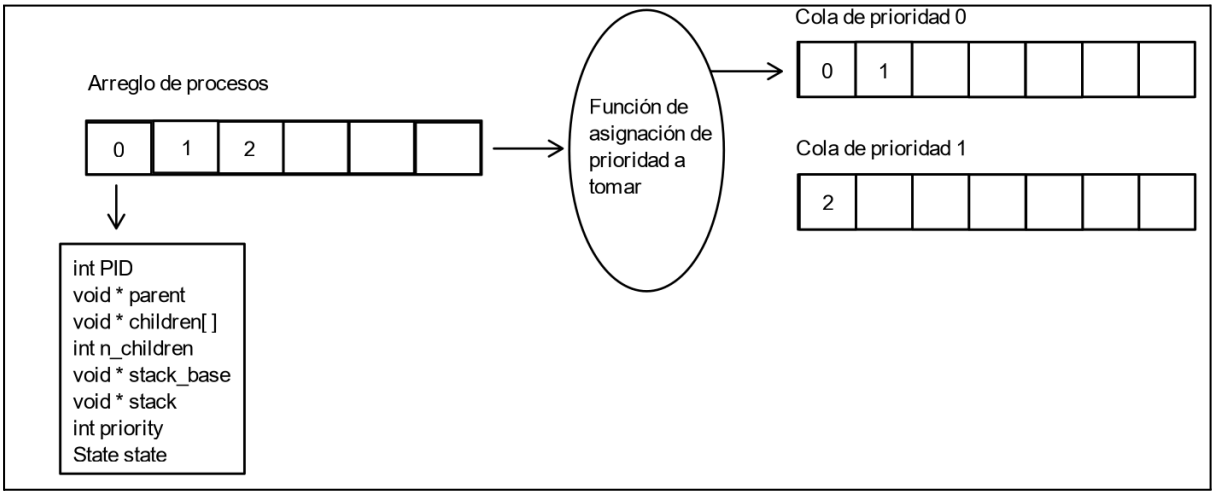

Figura 1: Diagrama del scheduler.

Luego para el caso de cambios de prioridades se continúa con este algoritmo:
Se añade la referencia del proceso a la cola correspondiente y se actualiza la prioridad en la estructura del en el arreglo. Luego si se llamase a la anterior referencia del proceso, esta no se correría pues no coinciden la prioridad de la cola con la del proceso.

También, se decidió que todos los procesos terminen en una llamada a exit() en vez de return.

En cuanto al memory manager, se eligió usar el siguiente algoritmo además del buddy:

Cuando se solicita memoria se busca entre la memoria ya alocada un bloque libre con el tamaño pedido, y en caso de no haberlo, se aloca un nuevo segmento del tamaño de la memoria solicitada. Además, proveemos la funcionalidad de liberar la memoria.

Instrucciones de compilación y ejecución

Para la correcta ejecución del proyecto se deben seguir los siguientes pasos:

- Primero se clona el repositorio de GitHub utilizando el comando $> git clone <repositorio>
- En segundo lugar, correr el archivo “./init.sh” para clonar la imagen.
- Por último, correr el script de “compAndRun.sh” para compilar el proyecto y correrlo.

Instrucciones para demostrar el funcionamiento de los requerimientos

Para probar el trabajo y ver todas la funcionalidades requeridas se ofrecen algunos comandos de terminal:

- **help**: Imprime un listado de los comandos implementados
- **mem**: Muestra el estado actual de la memoria
- **loop**: Corre un proceso que imprime su PID cada cierto tiempo
- **kill**: Mata un proceso dado su PID
- **nice**: Cambia la prioridad de un proceso dado su PID
- **block**: Bloquea/Desbloquea un proceso dado su PID
- **ps**: Imprime todos los procesos y sus propiedades (nombre, PID, prioridad, stack y base del stack y si está en foreground o background) incluyéndose a sí mismo pues, a su vez, es un proceso
- **phylo**: Es una implementación del problema de los filósofos comensales. Partiendo de cinco filósofos (se puede añadir uno más introduciendo ‘a’ y eliminar uno usando ‘r’ en runtime) se puede ver el estado de la mesa con los filósofos que están comiendo.
- **cat**: Imprime lo que reciba en entrada estándar
- **wc**: Imprime la cantidad de líneas, palabras y caracteres que reciba por entrada estándar (ver wc –help o -h)
- **filter**: Elimina las vocales que reciba por entrada estándar

- **test**: Prueba algunas de las funcionalidades del kernel (ver test --help)

Además, se cuenta con la posibilidad de añadir al final de cada llamada el símbolo “&”, el cual indica que el proceso a correrse ha de hacerlo en background.
Por último, se cuenta con el carácter especial “|” el cual une la salida estándar del proceso escrito a la izquierda de esto con la entrada estándar del derecho. Cuenta con la limitación de sólo permitir conectar dos procesos en una instrucción.

Limitaciones

El trabajo está limitado por algunos factores. A continuación se las describen.
Se cuenta con 2.097.152 bytes de memoria dinámica alocable, sólo pueden existir 1024 procesos, 256 pipes y 512 posibles instancias de semáforos.
Luego, para el programa de filósofos comensales, se cuenta con un mínimo y máximo de filósofos siendo 5 y 13 respectivamente.

Problemas encontrados durante el desarrollo

A lo largo del proyecto, el mayor factor limitante fueron las dependencias que tienen ciertos puntos de la consigna con respecto a otros. Por ejemplo, para testear al memory manager se necesita de procesos. Para correr procesos dinámicamente se necesita la shell. Luego, para usar la shell se necesitan syscalls bloqueantes (get_char()). Por último, para implementar las syscalls bloqueantes se necesita la misma lógica de bloqueo que los semáforos. Esto llevó a tener que pensar muy por adelantado todo antes de su implementación, como también a conflictos y cuellos de botella.

Conclusión

En conclusión, se cree haber realizado el trabajo correctamente. Fue un placer poder realizar este proyecto. Hacer un memory manager, scheduler, procesos e IPC para el proyecto arrancado en Arquitectura de computadoras realmente le da un cierre al trabajo y resulta satisfactorio. Finalmente, se agradece por esta oportunidad y de haber podido reforzar los conocimientos adquiridos durante la cursada de la materia de manera práctica y didáctica.