TPE 2
Sistemas Operativos

Alumnos:
54623 Juan P. Dantur
52477 Damian Rizzotto
57629 Sofia Johansson

Scheduler y procesos:

Inspirado en el scheduler del sistema Wyrm hecho por el profesor Rearden.

El sistema cuenta con un scheduler implementado de la siguiente forma:
Se tiene una lista circular con nodos, donde cada nodo representa un proceso que contiene:

- Su estado (activo o durmiendo)
- Su process ID
- El process ID del proceso que lo creo (ppid)
- Su nombre (string)
- Las paginas reservadas para hacer backup de los registros
- El stack.

El scheduler contiene:

- Una referencia al proceso que se está ejecutando en el presente instante
- Un int que representa el process ID del proceso que tiene el “foreground” (solo a este proceso se le permite imprimir mediante system calls y leer caracteres del teclado)
- Un int que representa el pid del proceso que se durmió mediante la system call “sleep”, asi como la cantidad de tiempo que le queda hasta ser despertado. Esta implementación está abierta a mejoras en el futuro dado que solo permite un proceso durmiendo a la vez.
- Una direccion de memoria en la cual se encuentra el stack y los registros del kernel, los cuales son accedidos cuando se hace el cambio de proceso.

Además de lo previamente mencionado el scheduler contiene métodos que permiten hacer el switch user-kernel y kernel-user, como asi también funciones que ayudan al manejo del mismo como por ejemplo:

new_process(name, entry)
Crea un proceso, le asigna el nombre dado, y el entry_point dado. El pid se lo asigna el scheduler, y es retornado en el nombre de la función.

set_fore(pid)
Le da el foreground al proceso pedido

kill(pid)
elimina el proceso pedido, no puede ser el actual, ni “init”(pid 0)

sleep(time)
Duerme al proceso actual por n segundos

list()
Imprime en pantalla la lista de procesos en la cola, indicando si están despiertos o no, su nombre, su pid, e indica el pid del proceso que tiene el foreground.

Los procesos no se pueden hacer kill a si mismos dado que eso genera fallas en el sistema, por lo cual de esto se debe encargar otro proceso.
Cuando un proceso muere, todos los procesos hijos de éste pasan a ser hijos del padre del mismo.
El timertick, además de incrementar el contador si hay algún proceso dormido, se encarga de ceder el procesador al próximo proceso no dormido en la cola.

Comandos involucrados en la shell:

ps
Muestra una lista de todos los procesos abiertos junto con su estado y su ppid

dummy
El comando dummy se encarga de escribir ‘’Dummy”” en pantalla. Se utiliza para probar el argumento “&” que indica si el proceso debe o no estar en foreground.

kill [pid]
Elimina el proceso que tiene asignado el pid indicado.

IPC: Semaforo y Memoria Compartida

El sistema cuenta con un mecanismo de exclusión mutua para comunicar procesos, el cual esta implementado asi:
Se tiene una variable tipo in, la cual representa un semáforo, una zona de memoria accesible mediante una system call, y una cola de process IDs. La variable sem se inicializa en 1, la cola de pids inicialmente es nula.

El protocolo de acceso a la memoria compartida es:

1. Hacer down() para verificar que ningun otro proceso está modificando la zona de memoria compartida.
2. Manipular la zona de memoria compartida como se desee;
3. Hacer up() para habilitar el semáforo y despertar al próximo proceso que desea usarlo en caso de haber alguno.

Se cuenta con los siguientes métodos:
down(): Si sem vale 1, lo hace 0 y retorna. Si vale 0, agrega al final de la cola el pid del proceso actual, lo duerme por tiempo indefinido, y cede el procesador al siguiente proceso.
up(): Si la cola esta vacia, incrementa sem y retorna. Si hay elementos en la cola de espera, despierta al primer proceso y lo remueve de la misma.
Esto garantiza que:

- Un proceso no puede acceder a la zona de memoria cuando otro la está manipulando, siempre y cuando se respete el protocolo.
- El orden de acceso de los procesos a la memoria compartida sea por orden de llegada.

System calls involucradas:

get_mem()
Recupera zona de memoria compartida.

down()
Explicado previamente

up()
Explicado previamente

Comandos involucrados en la shell:

ipc
Describe la existencia del semáforo, e indica cómo probarlo.

semtest
Crea 2 procesos al mismo tiempo, los cuales tienen la función de modificar el valor que hay en la zona de memoria compartida (ambos respetan el protocolo de acceso). Pone en evidencia que no puede haber mas de un proceso modificando al mismo tiempo la zona de memoria compartida y que los demás van a esperar a que el otro salga. Uno de los procesos hace un sleep de 5 segs entre el down y el up, para poder apreciar el tiempo de espera del otro proceso entre que hace down y accede a la memoria.

Paginación:

Implementacion A (paging.c) :
En nuestra implementación original de Paginación realizamos un paginado dinámico de la siguiente manera:
Al iniciar el kernel se mapea toda la memoria (512 MiB) y se guardaban punteros a paginas entre los MiBs 10 y 12. Los cuales no eran modificables por los procesos.
Al crear un proceso se creaba una tabla L4 para el mismo, reservando paginas desde el comienzo del stack (512 MiB) para abajo (Por lo tanto habría una limitación de procesos dado que eventualmente las paginas llegarían a los 12 MiBs donde se encontraban los punteros. Pero dado el scope del TPE esta condición es imposible de cumplir).
El proceso tiene una ubicacion virtual en 30 MiBs, de los cuales se utilizaron 8 MiBs por debajo como stack (que el procesador utiliza automáticamente) y 2 MiB por arriba como heap. El Page Fault Handler recibe las excepciones y, de buscar una pagina que no estaba presente pero que se encontraba dentro de los limites del stack/heap la reserva.
Si bien el sistema estaba testeado y funciona, dadas una serie de complicaciones que se tuvo a la hora de implementar el scheduler se tuvo que recurrir a la implementación B.

Implementacion B:
Se realiza un paginado de toda la memoria disponible (512 MiB). Luego se procede a reservar memoria comenzando desde 6 MiB. Si bien este approach no es tan elegante como la implementación A se garantiza que los procesos no van a interferir con el Kernel Space.