Trabajo Práctico Especial
Sistemas Operativos

Diego Bruno Cilla - 57028
Oliver Balfour - 55177
Sebastian Favaron - 57044
Ximena Zuberbuhler - 57287

Introducción

El objetivo del trabajo práctico especial es añadirle nuevas funcionalidades al trabajo práctico especial realizado para la materia Arquitectura de Computadoras (72.08) en base al *kernel* booteable por Pure64 que implementamos en https://bitbucket.org/acalatayud/arqui/src se incluyeron mecanismos de IPC, Memory Management y Scheduling.

Diseño

System Calls:

Read	*0*
Write	*1*
Clear	*2*
SetTimeZone	*3*
GetTime	*4*
GetDate	*5*
Echo	*6*
Run	*7*
Malloc	*8*
DrawPixel	*9*
GetResolution	*10*
CreateProcess	*11*
Exit	*12*
Ps	*13*
GetPid	*14*

KillProcess	*15*
GetMutex	*16*
CloseMutex	*17*
LockMutex	*18*
FreeMutex	*19*
Receive	*20*
Send	*21*
AllocBlock	*22*
DeallocBlock	*23*
CreateSemaphore	*24*
OpenSemaphore	*25*
CloseSemaphore	*26*
WaitSemaphore	*27*
SignalSemaphore	*28*

Manejo de la memoria física:

El sistema cuenta con un *buddy page allocator* con las funciones *allocBlock* y *deallocBlock* para poder reservar y liberar páginas respectivamente. Se resolvió a fines prácticos que el tamaño de las páginas sea de 4096 bytes y la cantidad máxima de bloques de 1024 indicados con las constantes *BLOCK_SIZE* y *NUMBER_OF_BLOCKS* respectivamente (total 4MB).

Procesos:

Los procesos son unidades de actividad caracterizados por un código ejecutable, un estado actual, y un conjunto de recursos administrados por el sistema.
La estructura de los procesos que administramos incluye:

- El ID del proceso
- El ID del proceso padre
- El estado del proceso (puede variar entre corriendo, listo, bloqueado y terminado)
- Un flag que indica si corre en primer plano
- La descripción del proceso
- Un puntero al código
- Un heap y su tamaño correspondiente
- Un stack

Se definió la syscall CreateProcess para crear dichos procesos. Se decidió que la solución más práctica fuese que los procesos no tengan múltiples hilos para que sea más sencillo de implementar, al ser procesos *monothreads* la información de cada hilo se encuentra incluida en el proceso.

Planificador:

La tarea del *scheduler* es repartir el tiempo disponible en el microprocesador entre los procesos que se encuentran listos para su ejecución. El sistema cuenta con un *multitasking* en el que cada proceso tiene un quantum de tiempo igual al intervalo de un timer tick para ejecutarse. El algoritmo de planificación elegido para el sistema es el Round-Robin que permite seleccionar de manera equitativa y ordenada a los procesos con la misma prioridad. Se comienza por el primer elemento de una lista definida como *current* hasta llegar al último y comenzando de nuevo desde el primer elemento.

Cambio de contexto:

La rotación del Round-robin se produce en el handler del Timer Tick, cuando se cambia de un proceso a otro la información se *pushea* al stack y cuando vuelve a correr el proceso se *popea* dicha información.

Mensajes entre los procesos:

Pipes:

Los mensajes se comunican con una estructura que se decidió ponerle de nombre pipes por convención ya que no respeta todas las características de un pipe, siendo más parecido a una message queue. Cuenta con un buffer circular en el que si un proceso quiere escribir y está lleno se impide la escritura y se bloquea el proceso (escritura bloqueante), en tanto, para la lectura si se desea realizar la lectura y el buffer se encuentra vacío, se bloquea al proceso hasta que se reconozca algo para leer (lectura bloqueante).

Mutex:
Los mutex al igual que las colas de mensajes tienen un nombre identificatorio. Un lock de un mutex solo puede ser liberado por el mismo proceso que tiene el lock, siendo ignorados los free de los procesos que no poseen el lock. Al liberar el lock de un proceso, se lockea el siguiente proceso en la cola del mutex. Son creados mediante la syscall *GetMutex* y son bloqueados y desbloqueados mediante las syscall *LockMutex* y *FreeMutex* respectivamente. Pueden eliminarse mediante la syscall *CloseMutex*.

Semáforos:
Se implementaron semáforos para manejar el ipc, y se utilizaron los mutex para garantizar la exclusión de la sección que hace incremento y decremento del semáforo. Son creados mediante la syscall *SemCreate,* son obtenidos mediante la syscall *SemOpen,* y el incremento y decremento se hace mediante *SemSignal* y *SemWait* respectivamente. Pueden eliminarse mediante la syscall *SemClose*.

Userland:

$> ps: Lista los procesos corriendo y su información.

$> kill [pids]: Remueve los procesos enviados como argumentos.

$> ipc [message]: Demo del ipc entre 2 procesos. Ofrece un comando para crear un proceso enviante y otro para crear un proceso receptor.

$> prodcons: Demo del problema del productor-consumidor resuelto utilizando semáforos. Ofrece comandos para agregar y remover procesos productores y/o consumidores.

$> while1: Crea un proceso con un ciclo sin final. Se puede utilizar para probar correr un proceso en background y el kill.

Agregando & antes del nombre de cada módulo se puede correr el proceso en background.
Con ^C (ctrl + C) se llama al sysCall *Exit* que remueve al proceso actual.

Tests:

Se incluye un test básico del buddy memory allocator, en el que se prueba el alloc, 2 alloc consecutivos y un dealloc.

Compilación:

El repositorio cuenta con un archivo makefile para la compilación del código. Se recomienda ejecutar el código sobre un sistema operativo linux que cuente con gcc version 5 y qemu. Asimismo, se incluye un *dockerfile* para facilitar la compilación del mismo.

$> make all

Ejecución:

$> ./run
$> ./runtests