Sistemas Operativos

TP2: Construcción del Núcleo de un Sistema Operativo

Integrantes:

Nicolás Kuyumciyan	55165
Magdalena Vega	55206
Lucas Casagrande	55302

Aplicaciones

Previas de Arquitecturas:

Clear - Borra toda la pantalla.
Time - Muestra la hora y la fecha
Help - Muestra todos los comando disponibles
Change time - Te permite cambiar el tiempo y la fecha
Whoami - Te dice quien sos.
Keyboard - Muestra gráficamente la distribución del teclado
Colors - Permite cambiar los colores de la consola.
Screen time - Permite cambiar el tiempo de inactividad hasta que aparezca el salva pantallas
Piano - Le permite al usuario generar sonido en forma de un piano
Beep - Genera un sonido predefinido

Nuevas Aplicaciones:

ipcs – Muestra todos los IPC's existentes
ps – Muestra todos los procesos
game – Ejecuta el juego
draw – Como un paint

Juego

IDEA:

La idea del juego es que manejas una víbora por la pantalla, que va dejando el camino pintado. Y al chocarte con alguno de los caminos perdes. El juego se puede jugar de hasta 6 personas.

DIVISION:

El juego está dividido en 3 partes. La impresión y pantalla y la lógica, por un lado. El input por el otro y el sonido en tercer lugar. Debido a la falta de tiempo no se pudo separar el input de la lógica, pero el sonido se reproduce en otro proceso, los cuales se comunican mediante el IPC.

PREVIAS DE ARQUITECTURAS:

- ✓ WRITE – PERMITE ESCRIBIR EN PANTALLA
- ✓ ERASE_SCR – BORRA LA PANTALLA
- ✓ GET_STR - PERIMITE TRAER UN STRING DEL BUFFER DE TECLADO LIMPIO (AHORA CON LA IMPLEMENTACION DE MULTITASKING ES BLOQUEANTE)
- ✓ GET_CHAR – TRAE UN CHAR DEL BUFFER DE TECLADO LIMPIO. (AHORA TAMBIÉN BLOQUEANTE)
- ✓ RTC_READ – LEE EL TIEMPO
- ✓ RTC_WRITE – PERMITE MODIFICAR EL TIEMPO
- ✓ COLORS – PERMITE CAMBIAR LOS COLORES DE LA CONSOLA (Y AHORA TAMBIÉN EL SCREENSAVER)
- ✓ SCR_TIME – CAMBIA EL TIEMPO DE INACTIVIDAD HASTA QUE SE ACTIVA EL SCREENSAVER
- ✓ BEEP - EMITE UN SONIDO
- ✓ PIANO – ACTIVA EL PIANO
- ✓ SONGS - PERMITE REPRODUCIR UNA CANCION

NUEVAS SYSTEM CALLS:

MEMORY MANAGEMENT SYSTEM CALLS

- ✓ MALLOC – PERMITE RESERVAR PÁGINAS DE A 4K (PUDIENDO RESERVAR MÁS DE UNA)
- ✓ FREE – LIBERA LAS PAGINAS RESERVADAS

IPC SYS CALLS

- ✓ MKFIFO – PERMITE CREAR UN FIFO (DEVUELVE EL FD)
- ✓ OPENFIFO – PERMITE ABRIR EL FIFO (DEVUELVE EL FD)
- ✓ CLOSEFIFO – CIERRA EL FIFO
- ✓ WRITEFIFO – PERMITE ESCRIBIR EN EL FIFO
- ✓ READFIFO – PERMITE LEER EL FIFO SIN BLOQUEARSE
- ✓ READFIFOBLOQ – LEE EL FIFO PERO BLOQUEA SI NO HAY NADA
- ✓ SHOWIPCS – MUESTRA EN PANTALLA LA LISTA DE IPCS

GRAPHIC SYSTEM CALLS

- ✓ SET_SCREENSAVER - SETEA LA IMAGEN DEL SCREENSAVER
- ✓ COLORS_GRAPHIC – SETEA LOS COLORES DEL SCREENSAVER
- ✓ CLEAR – BORRA LA PANTALLA
- ✓ DRAW_CIRCLE – DIBUJA UN CIRCULO EN LA POSICION Y DEL COLOR ESPECIFICADOS

- ✓ DRAW_IMAGE – DIBUJA UNA IMAGEN EN PANTALLA
- ✓ DRAW_TEXT – DIBUJA TEXTO EN LA POSICIÓN CORRESPONDIENTE
- ✓ ENTER_DRAW_MODE – PERMITE ENTRAR EN MODO DIBUJO DONDE NO SE IMPRIME LO QUE EL USUARIO ESCRIBE
- ✓ EXIT_DRAW_MODE - SALE DEL MODO VIDEO
- ✓ DRAW_ERASABLE_CIRCLE – PERMITE DIBUJAR UN CIRCULO QUE DESPUÉS SE PUEDE BORRAR Y RESTAURA LO QUE TENÍA ABAJO
- ✓ UNDRAW_ERASABLE_CIRCLE – BORRA EL CIRCULO CORRESPONDIENTE
- ✓ GET_CHAR_FROM_BUFFER – PERMITE TRAER UN CHAR DESDE EL BUFFER

MULTITASKING SYSTEM CALLS

- ✓ NEW_PROCESS – CREA UN NUEVO PROCESO Y LO COLOCA EN EL SCHEDULER
- ✓ END_PROCESS – FINALIZA UN PROCESO
- ✓ GET_ALL_PROCESS – IMPRIME TODOS LOS PROCESOS
- ✓ SLEEP – DUERME EL PROCESO POR X MILISEGUNDOS

Funcionamiento

IPC:

SE DECIDIÓ IMPLEMENTAR NAMED PIPES (FIFO) PARA LA COMUNICACIÓN ENTRE PROCESOS. ESTO FUE POR LA SIMPLICIDAD Y EFECTIVIDAD QUE PROVEE ANTE LOS OBJETIVOS PROPUESTOS.

DADO QUE EL TRABAJO NO TIENE IMPLEMENTADO UN FILE SYSTEM LOS PIPES SON DE TAMAÑO FIJO (4KB), SON ALOCADOS EN TIEMPO DE CREACIÓN Y LIBERADOS AL CERRARLOS.

PARA LOGRARLO SE IMPLEMENTÓ UN ARRAY DE ESTRUCTURAS QUE CONTIENEN EL FILE DESCRIPTOR ASOCIADO CON EL PIPE, SU NOMBRE Y UN PUNTERO A LA DIRECCIÓN DE MEMORIA DONDE SE ENCUENTRAN LOS DATOS.

PARA MAYOR VERSATILIDAD SE IMPLEMENTARON DOS TIPOS DIFERENTES DE READ: BLOQUEANTE Y NO BLOQUEANTE. EL BLOQUEANTE ES IMPORTANTE PARA LOS PROCESOS QUE NO TIENEN NADA QUE HACER HASTA QUE PUEDAN LEER ALGO, LO CUAL PERMITE MARCARLOS COMO INACTIVOS Y NO CONSUMEN TIEMPO DE PROCESADOR. UNA VEZ QUE SE ESCRIBE EN EL PIPE DONDE SE QUEDARON BLOQUEADOS SE LOS MARCA NUEVAMENTE COMO ACTIVOS Y CONTINUAN CON SU EJECUCIÓN.

SCHEDULER:

EL SCHEDULER UTILIZA UN ALGORITMO SENCILLO DE ROUND-ROBIN.

ADEMÁS LOS PROCESOS PUEDEN ESTAR EN ESTADO READY (LISTOS PARA EJECUTARSE) O WAITING (QUE ESTÁN ESPERANDO ALGUNA ACCIÓN ESPECIAL PARA DESBLOQUEARSE).

Adicionalmente hay un proceso inactivo del sistema que se ejecuta cuando todos los otros procesos están inactivos. Dicho proceso es un hlt que evita que se use el CPU cuando no es necesario.

Los procesos guardan la siguiente información: su nombre, estado, id y el id del proceso que lo creó, además de todos los datos necesarios para realizar el contex switch y si el proceso tiene el foreground o no.

Para crear un proceso simplemente se necesita llamar a la system call correspondiente pasándole el nombre y el puntero a función donde el programa va a ejecutarse.

Los procesos se pueden bloquear mediante un sleep por una cantidad determinada de tiempo, además de bloquearse cuando se pide un recurso que no está disponible (por ejemplo cuando la consola pide el texto ingresado por el usuario, pero el usuario no ingreso nada todavía).

Memory Management:

decidimos para que sea más sencillo que al dar memoria al usuario se le van a dar múltiplos de 4kb.

Para ello se implementó un vector con la cantidad de posiciones como bloques de 4kb haya, y luego de ese vector, hay otro igual donde por cada dirección dice cuantos bloques se dieron (esto es para poder saber cuantos se tienen que liberar). cuando un usuario pide memoria, se marca las posiciones que se llevó como "usadas" y se le da la dirección de donde empieza el primero.

para el free lo que se hace es deducir a partir de la dirección que se pasa por parámetro qué bloque es y se marca ese bloque como "libre".

se implementó una estructura para poder hacer esto que contiene el vector, la cantidad de bloques que va a haber y un int que dice cual fue el último bloque que se alocó (Esto se hace para no tener que recorrer todo el vector, asumiendo que nada de lo anterior se liberó)

Asumpciones y decisiones de diseño

- En el scheduler se decidió crear un proceso inactivo para que cuando todos los procesos estén en estado waiting el CPU no esté en uso
- La dirección donde se reservan las direcciones en el malloc están definidas por nosotros
- Debido a que no se implementó un file-system los fifos están en memoria y no en disco
- El tamaño de cada fifo esta fijo y es 4KB
- Solo se puede tener hasta un máximo de 64 procesos
- Solo puede haber un proceso esperando para leer un fifo (en el caso del read bloqueante)
- Las imágenes que se utilizan están en un nuevo módulo en la dirección 0x7100000
- Por falta de tiempo no se pudo separar el input del juego
- Se decidió no poner una música de fondo en el juego ya que era sumamente molesta
- Se decidió que haya una instrucción que lee del fifo bloqueante y otra que no bloquee, para sumarle versatilidad.
- Cuando un fifo se cierra todos los procesos que estaban bloqueados a ese fifo salen del estado recibiendo un 0 como el tamaño del dato leído.
- Si en el malloc se pide un tamaño más grande que una página, en caso de no encontrarse la suficiente cantidad de páginas contiguas para satisfacer el pedido se retorna un error

Referencias

https://bitbucket.org/RowDaBoat/wyrm/wiki/Home (Context Switching y scheduler)