Tema 2.

El bloque de control de procesos

En Linux se denomina descriptor de proceso. La task\_struc es una estructura formada por una serie de superestructuras. Define:

-El estado del proceso

-Recursos del sistema que están siendo utilizados por el proceso. (Si se está usando un archivo,files\_struct). Todo proceso tiene al menos 3 archivos abiertos, la entrada estándar, la salida estándar y el error estándar. La file system struc almacena entre otras cosas el directorio de trabajo actual. Si al proceso se le da una ruta relativa, en el pcb encuentra el directorio de trabajo y ya de ahí busca con la relativa.

-La tty, descripción del dispositivo de entrada/salida asociado.

-Thread info no tiene nada que ver con los hilos, da información de bajo nivel para el proceso.

-Todo proceso tiene una relación con el proceso que lo creó y sus hermanos. Si todos están relacionados es más fácil organizarlos.

-El padre de un proceso es que lo creó, si el padre(parent) muere, a veces el creado se engancha a otro, conocido como real parent.

-En Linux hay un sistema de notificación de eventos, este evento se conoce como señal. Se identifican con un numerito.

-El separar el task struct en varias estructuras es que en la vida de un proceso hay estructuras que necesito parar, ejecutar.. Las que no necesites las puedes destruir, eliminarlas, así se ahorra memoria.

Una hebra kernel se ejecuta en espacio kernel, por lo que no necesita una descripción del espacio de usuario.

Puedo compartir las subestructuras entre las hebras, ahorrando así memoria. Los hilos comparten la misma memoria, el mismo espacio de direcciones, comparten la misma mmstruct.

La pila se divide en dos, usuario y kernel, por motivos de seguridad. Tendrá tantos niveles la pila como niveles de privilegios tenga el sistema, aunque lo habitual es usuario y kernel. La pila se almacena junto a la estructura thread\_info, no en ella. Se hace para optimizar la información.

Una de las funciones de thead info es permitir a la hebra saber en que cpu se esta ejecutando. En el kernel de Linux hay una macro definida que te dice la tarea actual, la macro current. Si miro el registro puntero de pila puedo calcular la base de la región, que nos lleva al descriptor de procesos(ver diapositiva).

Pasos a seguir en el cambio de contexto:

-Se guarda la información del proceso en el pcb (sobretodo los registros hardware)

-Seleccionar el siguiente proceso a ejecutar.

-Restauramos la copia de los registros de la cpu que se guardaron en su última ejecución.

-Conmutar entre procesos es conmutar entre pilas kernel.

-TIF\_NEED\_RESCHED es una bandera que indica cuando hay que llamar al planificador de procesos. Preempt count indica cuantas veces hemos metido el proceso en cpu.

Estados de los procesos

El estado bloqueado se divide en dos en Linux, en task interruptible o task uninterruptible. Esta diferenciación se debe hay que hay procesos que puede que no lleguen a ocurrir(una entrada de teclado..). Se almacenarían procesos bloqueados que estarían ocupando memoria. Con esta diferenciación, el administrador del sistema puede saberlo y eliminar el proceso.

Traceado es un estado en el que se ha producido un evento de ruptura, como una pausa del depurador.

Todos los procesos se crean a partir de uno, el primero se hace manualmente, cuando arranco la máquina. Todos se crean a partir de la llamada clone().

Transiciones entre estados

En Linux puedes usar clone() para crear un proceso, pero en el resto de Unix no, hay que usar fork(). Cuando Linux usa fork(), llama a clone().

Wakeup() hace la transición entre preparado y bloqueado.

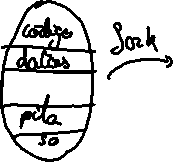
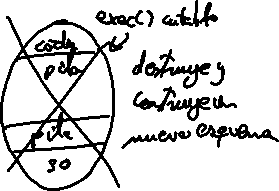
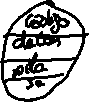
El init es el padre de todos los procesos.

Jerarquía de procesos

Actualmente systemd ha sustituido a init para la creación de hebras.

Manipulación de procesos

Un programa nuevo en un nuevo proceso necesita dos llamadas al sistema, una fork() y una exec(). En linux fork no tiene argumentos, en Linux no tiene argumentos, en Windows tiene 6 argumentos, tres de ellos son estructuras de datos.



Función hilo() {

..

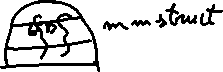
}

Main() {

…

Crear\_hilo(hilo, )

Un modelo 1:1 significa que un hilo a nivel usuario genera un hilo a nivel kernel. Todos los hilos de usuario son visibles para el kernel, por lo que son planificables de forma independiente.



Padre e hijo comparten los mismos recursos, la misma mmstruct.

Clone\_thread() significa que proceso padre e hijo pertenecen al mismo grupo. Si no se le pone este flag se le está diciendo al SO que lo ponga en un grupo diferente al del padre.

El pid de un proceso es un valor que indica dentro del sistema operativo cuál es el proceso. Cuando se pide el pid realmente da el identificador de grupo, para el identificador de hebra hay que pedir el tid. Se introdujo sys\_call que es capaz de generar el código en ensamblador para hacerlo.(ver ejemplo diapositiva).

Dos hebras son hermanas cuando comparten recursos.

Hilos kernel

Linux es monolítica, por lo tanto el código kernel se ejecuta como un proceso de usuario que demanda. Si no hay llamadas al sistema ni interrupciones el SO no se ejecuta.

Una hila kernel si necesita escribir algo lo hace en un fichero de log. No tienen terminal asociado, no leen ni escriben de pantalla.

Terminar un proceso

Si el programador no pone exit o return, el programa acaba, ya que el enlazador lo pone si no lo encuentra. Le pega la rutina correspondiente Parar terminar el proceso. La llamada al sistema en Linux se llama \_exit. Le doy la oportunidad al proceso de finalizar a nivel biblioteca.

Casi todas las señales finalizan el programa. Si el usuario no ha definido una señal por defecto se genera la correspondiente automáticamente

Matando un proceso

La terminación de un proceso se realiza en dos fases, la primera él mismo empieza a ponerse en fin y la segunda es el proceso padre finalizando el proceso.

El coredump es un proceso de volcado. Coge el espacio de direcciones de usuario y lo vuelca a un archivo.

RR significa rog robin, cíclica de no se qué.

En los sistemas actuales hay varias colas de bloqueado, una en función del evento que estamos ejecutando según me interese y depende del sistema.

Planificadores

Un planificador a medio plazo no es un planificador de la cpu directamente. El de largo plazo indirectamente se convierte en un gestor de planificación, por ejemplo cuando un programa decido que no se cargue en memoria principal directamente.

Si tengo un planificado apropiativo, puedo desde otra tarrea o componente del SO puedo apropiarme de la tarea, quitarle la cpu y usarla yo.