Lecturas 31 de octubre

-

Concepto de tarea en Linux

Concepto de tarea en Linux (procesos): Stalling pp. 193-196

Tareas Linux

Una tarea (o proceso) en Linux se representa por una estructura de datos `task_struct`, que contiene información sobre varias categorías: * **Estado**: estado de ejecución del proceso (ejecutando, listo, suspendido, detenido, zombie). * **Información de planificación**: información necesitada por Linux para planificar procesos. Los procesos se planifican según su tipo (los procesos de tiempo real antes que los normales), dentro de cada tipo puede haber prioridades. Además, hay un contador que calcula el tiempo que un proceso se ha estado ejecutando. * **Identificadores**: cada proceso tiene un identificador único de proceso, así como identificadores de usuario y grupo. * **Comunicación entre procesos**: Linux soporta el mecanismo IPC encontrado en UNIX SVR4 (colas de mensajes, semáforos y memoria compartida). * **Enlaces**: cada proceso incluye un enlace a sus padres, a sus hermanos (procesos con el mismo padre), y a todos sus hijos. * **Tiempos y temporizadores**: tiempo de creación del proceso y cantidad de tiempo de procesador consumido por el proceso.Un proceso puede tener asociado uno o más temporizadores. Para ello, se realiza una llamada al sistema, cuando finaliza el temporizador se manda una señal al proceso. Un temporizador puede ser periódico o de un solo uso. * **Sistema de archivos**: punteros a los archivos abiertos por este proceso, a los directorios actual y raíz para este proceso. * **Espacio de direccione**: espacio de direcciones virtual asignado a este proceso. * **Contexto específico del procesador**: información de los registros y de la pila que conforman el contexto de este proceso. * **Ejecutando**: un proceso Ejecutando puede estar ejecutándose o estar listo para ejecutar. * **Interrumpible**: es un estado bloqueado. El proceso está esperando por un evento (finalización de una operación de E/S, disponibilidad de un recurso, una señal de otro proceso,...). * **Ininterrumpible**: otro estado bloqueado. El proceso está esperando por un estado hardware, así que no manejará ninguna señal. * **Detenido**: el proceso ha sido parado y sólo puede ser reanudado por la acción de otro proceso. * **Zombie**: el proceso se ha terminado pero, por algún motivo, su estructura de tarea debe estar en la tabla de procesos.

Hilos Linux

Los sistemas UNIX tradicionales no proporcionaban soporte multihilo. Las aplicaciones se escribían con un conjunto de funciones de biblioteca de nivel de usuario (más conocida: pthread, POSIX thread), donde se asociaban todos los hilos en un único proceso a nivel de núcleo.

Las versiones actuales de UNIX ofrecen soporte para varios hilos a nivel de núcleo. La solución que Linux facilita no distingue entre hilos y procesos. Los hilos de nivel de usuario se asocian con procesos de nivel de núcleo. Si múltiples **hilos** de nivel de usuario constituyen un único **proceso** de nivel de usuario, se asocian con procesos Linux a nivel de núcleo y comparten el mismo ID de grupo. Así, estos procesos pueden compartir recursos y evitan cambiar de contexto cuando el planificador cambia entre procesos del mismo grupo.

Para crear un nuevo proceso, en Linux, se copian los atributos del proceso actual. Un nuevo proceso se puede clonar compartiendo sus recursos (archivos, manejadores de señales,memoria virtual,...). Si ambos procesos comparten la misma memoria virtual, funcionan como hilos de un solo proceso. Sin embargo, no se define ningún tipo de estructura de datos independiente para un hilo. En lugar del mandato normal `fork()`, los procesos se crean en Linux usando el mandato `clone()`, este mandato incluye un conjunto de flags como argumentos, definidos en la tabla a continuación:

Flag	Descripción
CLONE_CLEARID	Borrar el ID de la tarea

Flag	Descripción
CLONE_DETACHED	padre recibe la señal SIGCHLD (señal de finalización del proceso hijo)
CLONE_FILES	Comparte la tabla que identifica los archivos abiertos

Flag	Descripción
CLONE_FS	Compartir la tabla que identifica al directorio raíz y al directorio actual de trabajo, así como el valor de la máscara umask

Flag	Descripción
CLONE_IDLETASK	Establecer el PID a cero, que se refiere a la tarea idle (utilizada cuando todas las tareas disponibles están bloqueadas esperando por recursos)

Flag	Descripción
CLONE_NEWNS	Crear un nuevo espacio de nombres para el hijo
CLONE_PARENT	Compartir el proceso padre
CLONE_PTRACE	Si el proceso padre está siendo trazado, el proceso hijo también lo será

Flag	Descripción
CLONE_SETTID	Escribir el TID en el espacio de usuario
CLONE_SETTLS	Crear un nuevo TLS para el hijo
CLONE_SIGHAND	Compartir la tabla que identifica los manejadores de señales
CLONE_SYSVSEM	Compartir la semántica SEM_UNDO de System V

Flag	Descripción
CLONE_THREAD	Insertar este proceso en el mismo grupo de hilos del padre (fuerza de forma implícita a CLONE_PARENT)

Flag	Descripción
CLONE_VFORK	padre no se planifica para ejecución hasta que el hijo realiza la llamada al sistema execve()

Flag	Descripción
CLONE_VM	Compartir el espacio de direcciones (descriptor de memoria y todas las tablas de páginas)

La llamada al sistema tradicional `fork()` se implementa en Linux con la llamada al sistema `clone()` sin ningún flag.

Cuando el núcleo de Linux realiza un cambio de un proceso a otro, comprueba si la dirección del directorio de páginas del proceso actual es la misma que la del proceso a ser planificado. Si lo es, están compartiendo el mismo espacio de direcciones, por lo que el cambio de contexto consiste en saltar de una posición del código a otra. Los procesos clonados que forman parte del mismo grupo de procesos pueden compartir el mismo espacio de memoria, sin embargo, no pueden compartir la misma pila de usuario. Por tanto, la llamada `clone()` crea espacios de pila separados para cada proceso.

Procesos e hilos en otros sistemas

Otros sistemas sí que distinguen entre procesos e hilos, los procesos están asociados con la propiedad de recursos y los hilos con la ejecución de programas.Este enfoque podría mejorar la eficiencia y la conveniencia del código. En un sistema

multihilo, se pueden definir múltiples hilos concurrentes en un solo proceso, se podría hacer utilizando tanto hilos de nivel de usuario como hilos de nivel de núcleo. Los hilos de nivel de usuario se crean y gestionan por medio de una biblioteca de hilos que se ejecuta en el espacio de usuario de un proceso. Los hilos de nivel de usuario son muy eficientes porque no se requiere ningún cambio de contexto para cambiar de un hilo a otro. Sin embargo, sólo se puede ejecutar al mismo tiempo un único hilo de nivel de usuario y, si un hilo se bloquea, el proceso entero se bloqueará. Los hilos a nivel de núcleo son hilos de un proceso que se mantienen en el núcleo. Como son reconocidos por el núcleo, múltiples hilos del mismo proceso se pueden ejecutar en paralelo en un multiprocesador y el bloqueo de un hilo no bloquea al proceso completo. Sin embargo, se requiere un cambio de contexto para cambiar de un hilo a otro.