Taller de Sistemas Operativos

Procesos

Agenda

- Introducción
- PCB (task_struct)
- Agrupaciones de procesos
- Estado de un proceso
- Cambio de proceso (task switch)
- Creación y destrucción de un proceso
- Threads

Introducción

- Un proceso es un programa en ejecución
- Es una de las abstracciones fundamentales en un sistema Unix
- Incluye un conjunto de recursos como archivos abiertos, señales pendientes, espacio de memoria y uno o más hilos de ejecución
- Cada hilo de ejecución tiene su propio stack y program counter
- Veremos que para Linux un thread es solo un tipo particular de proceso

Introducción

- Identidad de los procesos:
  - Process ID (PID)
  - Credenciales (UID, GID)
- Entorno de ejecución
  - Argumentos
  - Entorno (lista con NOMBRE=VALOR)
  - Se guardan al principio del stack del proceso y se heredan en un fork
- En el código del kernel de Linux se les llama tasks

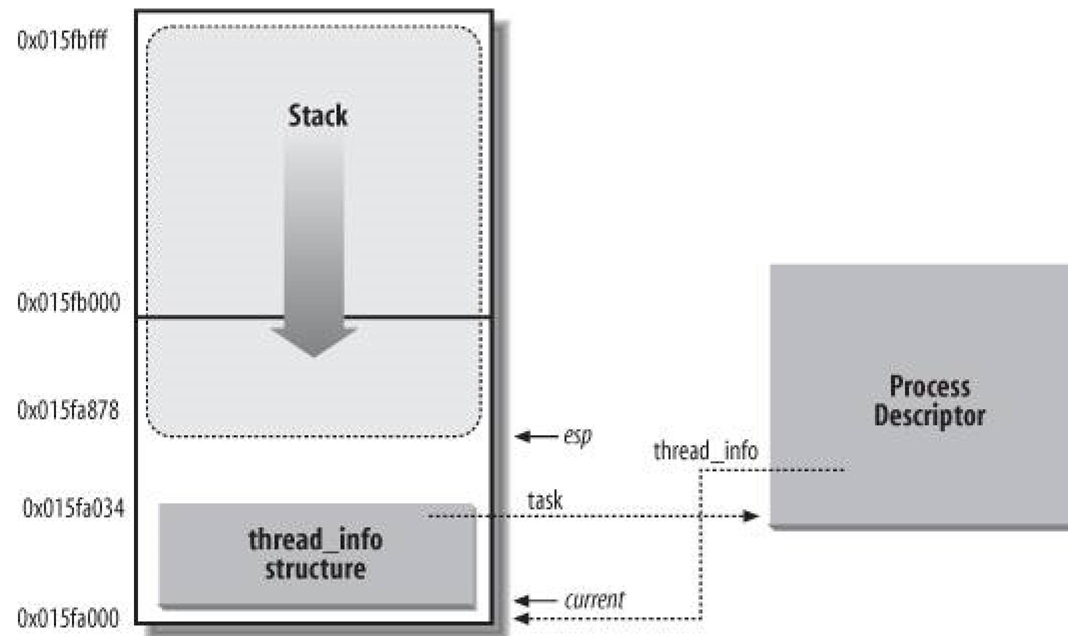
Introducción

- Contexto de los procesos
  - Contexto de planificación: copia de los registros, prioridad, clase de planificación (más adelante)
  - Recursos en uso y consumidos (accounting)
  - Tabla de archivos abiertos
  - Contexto en el file system (working directory)
  - Tabla de señales
  - Contexto de memoria virtual (describe el espacio de memoria del proceso)

PCB

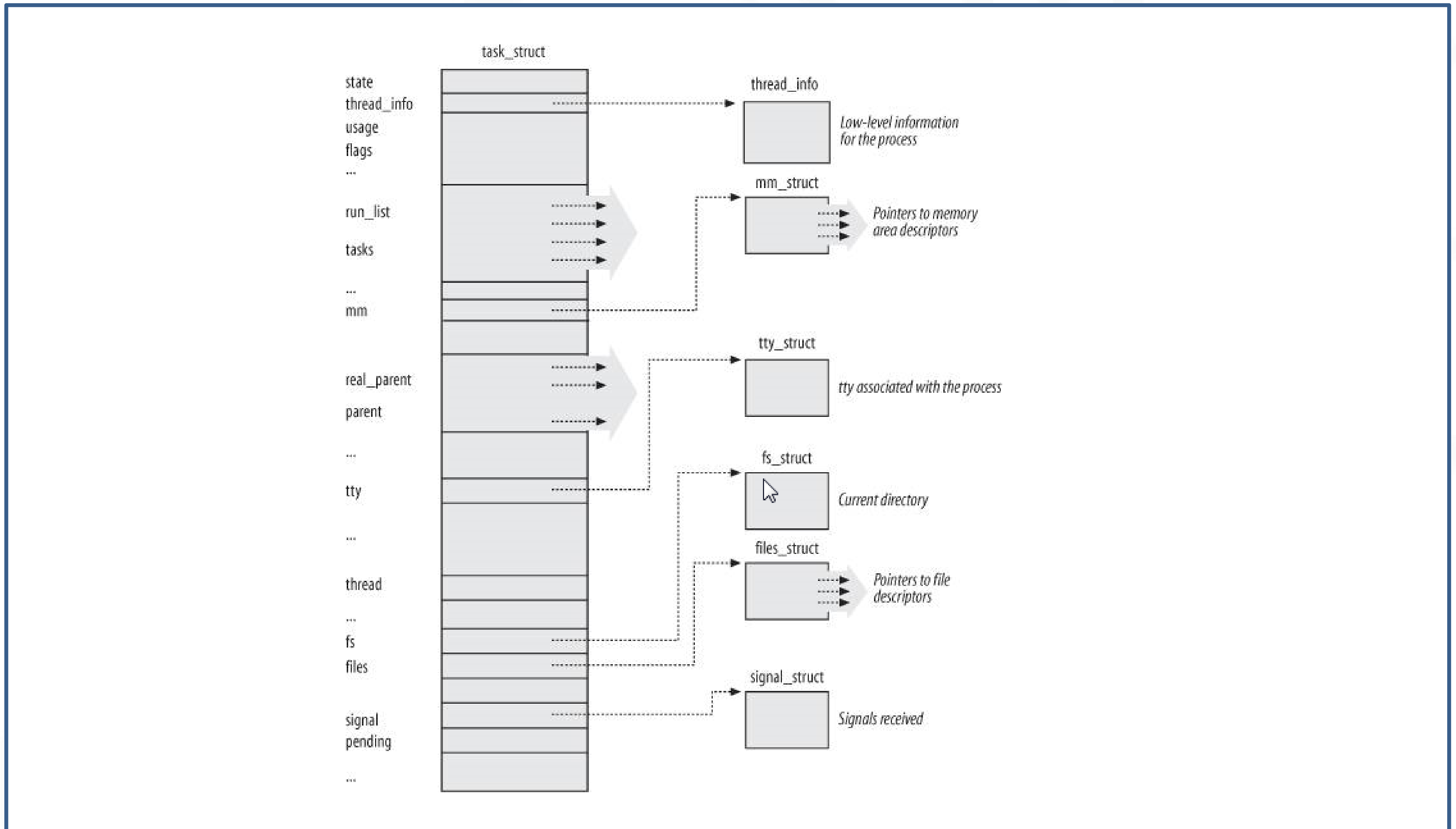
- Es el struct task_struct
- Es una estructura de cerca de 2 KB en una máquina de 32 bits
- Se puede acceder rápidamente al task_struct del proceso actual mediante la macro current
- Antes del kernel 2.6 se guardaba al final del stack del kernel de cada proceso
- Ahora se crea dinámicamente por el slab allocator y en su lugar se guarda el struct thread_info al final del stack
- El thread_info contiene un campo task que es un puntero al task_struct

Current en x86



Fuente: Understanding the linux kernel, 3ra edición

PCB

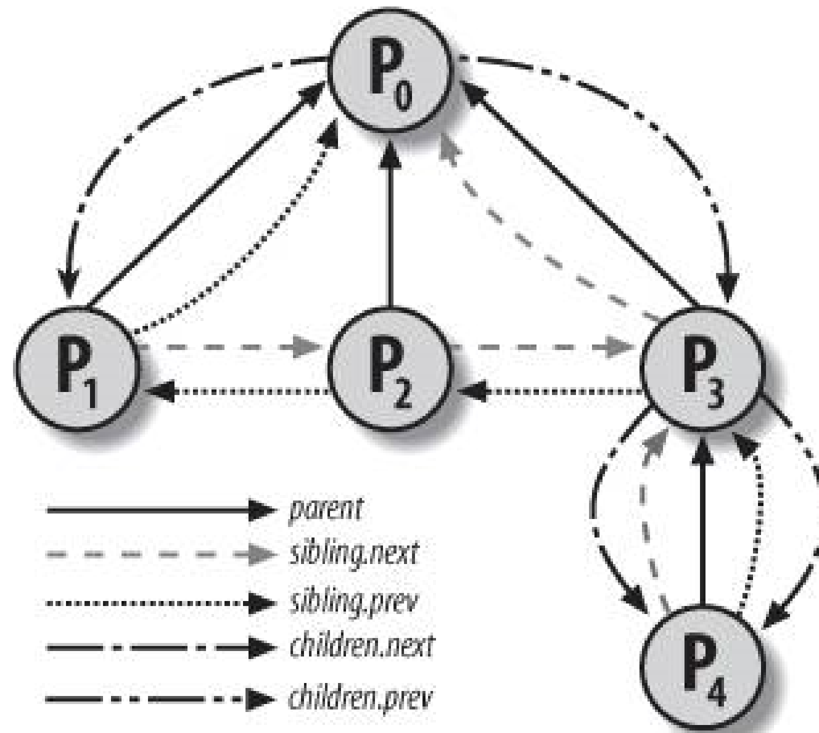


Fuente: Understanding the linux kernel, 3ra edición

Agrupaciones de procesos

- Todas las listas son doblemente encadenadas con un encabezado (struct list_head)
- Lista global de procesos
- Listas de procesos listos
- Jerarquía: parent, children, sibling
- pid_hash: para encontrar un proceso a partir del PID
- Listas de espera por eventos

Agrupaciones de procesos



Fuente: Understanding the linux kernel, 3ra edición

list_head

- Es una implementación de listas doblemente encadenadas y circulares provista por el kernel
- Puede contener elementos de cualquier tipo y funciona en todas las plataformas soportadas por el kernel
- A diferencia de las listas comunes funciona agregando un elemento de tipo list_head a cualquier struct
- Se pueden tener varios elementos de este tipo en el mismo struct para armar varias listas (como en task_struct)
- El tipo es:

```
struct list_head {
        struct list_head *next;
        struct list_head *prev;
}
```

list_head

- Se proveen varias funciones para
  - Insertar al principio y al final
  - Recorrer la lista
  - Obtener y borrar elementos
- Para obtener un elemento se usa la macro (#define)
  - list_entry(elemento, struct lista, variable)
  - elemento: es un puntero a un list_head
  - struct lista: es el tipo de la lista
  - variable: es el elemento de tipo list_head dentro del struct
- Se obtiene el puntero al struct a partir del puntero al list_head restando al puntero elemento el offset de variable dentro del struct

Estado de un proceso

- El estado de un proceso está dado por el flag state del PCB
- TASK_RUNNING
  - El proceso está corriendo o en una lista de espera por la CPU
- TASK_INTERRUPTIBLE
  - El proceso está bloqueado esperando por alguna condición. El proceso puede pasar a running si recibe una señal
- TASK_UNINTERRUPTIBLE
  - Igual que el anterior solo que no se despierta si recibe una señal. En general se usa menos que el otro.

Estado de un proceso

- TASK_STOPPED
  - El proceso no está ejecutando y tampoco se puede ejecutar.
  - Se llega a este estado luego de recibir una señal SIGSTOP, SIGTSTP, SIGTTIN o SIGTTOU
- TASK_TRACED
  - La ejecución se detuvo por acción de un debugger
- EXIT_ZOMBIE
  - El proceso terminó pero el padre no ha ejecutado wait todavía
- EXIT_DEAD
  - El proceso terminó y está siendo eliminado del sistema

Switching de procesos

- El contenido de los registros de la CPU (hardware context) se guarda en parte en el PCB y en parte en el stack del kernel
- Los cambios de proceso ocurren siempre en la función schedule()
- Básicamente consiste en:
  - Cambiar el directorio de páginas para apuntar a un nuevo espacio
  - Cambiar el stack del kernel y recuperar el contexto de hardware
  - Es una de las operaciones más dependientes del hardware de todo el sistema operativo

Creación y destrucción

- Se crean con fork()
- En cualquier momento se puede cambiar el programa ejecutado por un proceso con execve (exec*)
- Se terminan con exit()
- Tradicionalmente los sistemas UNIX copian todo el espacio de memoria del padre al hijo en el fork
- Linux usa el mecanismo copy-on-write:
  - Inicialmente se comparte (read only) toda la memoria y al escribirla se copia
  - Si se hace un exec() inmediatamente no se copia nada

Fork en detalle

- fork crea procesos a partir de la primitiva clone que permite especificar que recursos van a compartir el padre y el hijo (threads)
- clone llama a do_fork() que a su vez llama a copy_process() quien hace casi todo el trabajo:
  - Crea nuevas estructuras thread_info y task_struct copiando las del padre
  - Chequea que la creación del hijo no exceda la cantidad máxima de procesos del usuario
  - Inicializa algunos valores del PCB (el resto permanecen copiados)
  - Pone el estado del nuevo proceso en TASK_UNINTERRUPTIBLE

Fork en detalle

- setea algunas flags del task_struct
- llama a get_pid() para obtener un nuevo PID para el proceso
- dependiendo de los parámetros de clone copia o comparte los archivos abiertos, manejadores de señales y espacio de direcciones
- divide el time slice del padre entre dos y le asigna la mitad al hijo
- termina y retorna

• Si todo funcionó bien se ejecuta primero al hijo y después al padre (para evitar cows innecesarios)

• También existe la primitiva vfork() que se usaba antes de que existiera el cow.

Fin de un proceso

- Se produce al invocar la system call exit() que se encarga (entre otras cosas) de
  - Liberar la memoria del proceso (si no era compartida)
  - Quitar al proceso de las listas de espera por semáforos IPC
  - Cerrar los archivos en uso (si no los usa alguien más)
  - Guardar el código de salida en el PCB.
  - Enviar SIGCHLD al padre y dejar al proceso en estado EXIT_ZOMBIE
  - Llamar a schedule()

Fin de un proceso

- El PCB del proceso se mantiene en el sistema luego del exit() (junto con el thread_info y el stack)
- Solo se elimina luego del que el padre llame a wait() mediante al system call wait4()
- Al borrar:
  - Se resta uno a la cantidad de procesos del usuario
  - Se borra el PCB del pid_hash y la lista de procesos
  - Se borra el task_struct y las páginas con el stack del kernel

Threads

- Linux no maneja los threads como objetos separados de los procesos
- Son procesos que comparten ciertas estructuras con otros
- Se crean también con clone
- Se comparten el espacio de direcciones, los descriptores de archivos y los manejadores de señales (tampoco se copia la tabla de páginas)
- También existen los kernel threads que, a diferencia de los threads normales, no tienen un espacio de direcciones propio (trabajan solo en el espacio del kernel)

Threads

- Ejemplos de threads del kernel
  - swapper (pid 0): Se ejecuta cuando la CPU no tiene nada para hacer (uno por CPU)
  - migration: se encarga de mover procesos de una CPU a otra cuando la carga es despareja
  - kswapd: recupera memoria pasando páginas a disco
- El estándar POSIX exige que todos los threads de un proceso tengan el mismo PID, pero Linux les asigna uno distinto a cada uno
- Para lograr esto Linux usa grupos de threads con un líder cuyo PID representa a todos