Sistemas Operativos

Threads - II

Sistemas Operativos

- ☑ Versión: Marzo 2020
- ☑ Palabras Claves: Threads, Hilos, ULT, KLT, Procesos, Concurrencia, Paralelismo, Multithreading, Linux,Solaris, Windows, Fibras, LWP, POSIX; clone

Algunas diapositivas han sido extraídas de las ofrecidas para docentes desde el libro de Stallings (Sistemas Operativos) y el de Silberschatz (Operating Systems Concepts)

Ejemplo - Linux (< 2.4)

- ☑ No considera el concepto de thread
- ☑ Es posible "clonar" un proceso para compartir recursos (archivos, memoria, etc.)
  - ✓ System Call clone(): Es una SC que crea un proceso hijo que comparte partes de su contexto de ejecución con el padre
- ☑ Comparte:
  - ☑ Espacio de memoria
  - ☑ Tabla de descriptores de archivos
  - ☑ Tabla de manejadores de señales (signal handler).

Ejemplo - Linux (< 2.4)

- System Call clone()
  - Versión modificada de fork()
  - Definida en **sched.h**
  - Permite especificar que recursos compartir con su "tareas hijas"

```
int clone(  int (*fn) (void *arg),
            void *child_stack,
            int flags,
            void *arg)
```

Ejemplo - Linux (< 2.4)

```
int clone(  int (*fn) (void *arg),
            void *child_stack,
            int flags,
            void *arg)
```

- **fn**: Puntero a una función. Retorna un entero que se corresponde con código de salida del proceso hijo (resultado de ejecución)
- **child_stack**: Especifica la posición de la pila usada por el proceso hijo. Ya que el proceso hijo y el padre pueden compartir la misma memoria, no es posible que el proceso hijo ejecute la misma pila que el proceso padre. El proceso padre (el que invoca la función clone) prepara un espacio de memoria para la pila del hijo y le pasa un puntero a este espacio mediante este argumento.
- **flags**: Permite definir que se comparte entre padre e hijo

Ejemplo - Linux (< 2.4)

- Posibles Flags:
  - CLONE_VM: Compartir espacio de memoria
  - CLONE_FS: Compartir información del File System (raiz del FS, dir. de trabajo, umask)
  - CLONE_FILES: Compartir la tabla de descriptores de archivos.
  - CLONE_SIGHAND: Compartir la tabla de manejadores de señal.

Ejemplo - Linux (> = 2.6)

- ☑ Soporta nativamente Hilos y adhiere a POSIX (Portable Operating System Interface)
  - ✓ Estandar de System Calls (IEEE) con la finalidad de generalizar interfaces y lograr portabilidad.
- ☑ Adopta el modelo NPTL (Native POSIX Threads Library)
  - ✓ Esquema 1:1 en las primeras versiones. Las versiones modernas soportan M:1
  - ✓ Multiples ULT compartiendo el mismo PID son mapeados a un KLT
  - ✓ Cada Thread de un proceso:
    - Tiene su propio identificador conocido como LWP
    - Para su representación utilizan la misma estructura de los procesos task_struct

```
UID          PID    PPID     LWP  C NLWP STIME TTY          TIME CMD
clamav     65283       1   65283  0    6 Mar24 ?        00:00:04 /usr/sbin/clamav-milter
clamav     65283       1   65284  0    6 Mar24 ?        00:00:00 /usr/sbin/clamav-milter
clamav     65283       1   65285  0    6 Mar24 ?        00:00:00 /usr/sbin/clamav-milter
clamav     65283       1   65286  0    6 Mar24 ?        00:00:19 /usr/sbin/clamav-milter
```

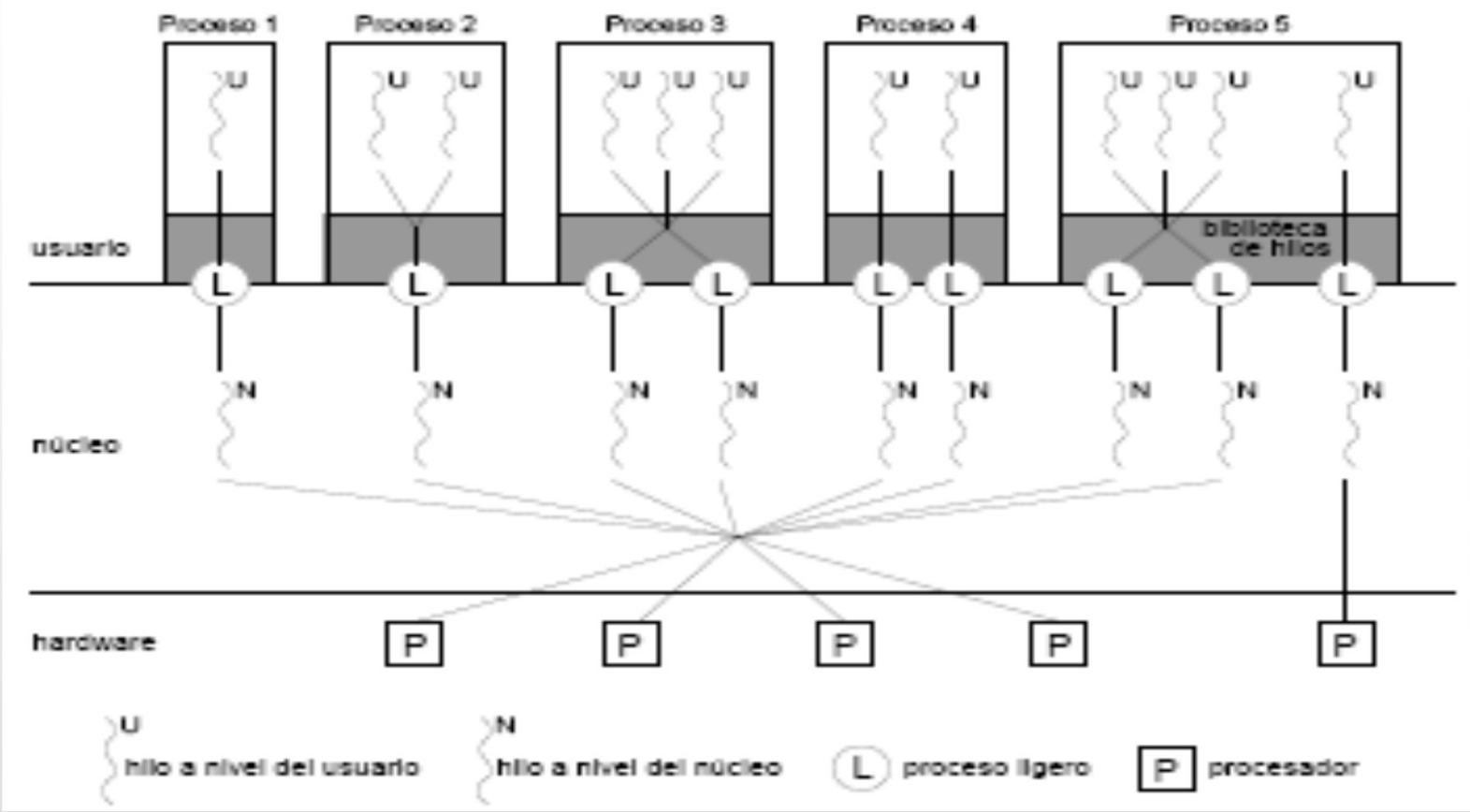
Linux

- Mas Información
  - http://en.wikipedia.org/wiki/Native_POSIX_Thread_Library (04/2018)
  - https://computing.llnl.gov/tutorials/pthreads/ (04/2018)
  - http://en.wikipedia.org/wiki/POSIX_Threads (04/2018)
  - http://www.ibiblio.org/pub/Linux/docs/faqs/Threads-FAQ/html/Accessability.html (04/2018)

Ejemplo - Solaris

- ☑ Maneja un esquema combinado de ULT y KLT
- ☑ Esquema pensado para aprovechar al máximo el uso de las CPU pensando en arquitectura SPARC (luego portado a x86)
- ☑ Considera 3 tipos de Thread:
  - ✓ ULT: Soportados a través del uso de librerías. Son invisibles al Kernel
  - ✓ KLT: Planificación en el/los procesador/es. Pensados para ejecutar tareas del SO y disminuir los Context switches en este ámbito
  - ✓ LWP:
    - Cada proceso posee como mínimo un LWP
    - La librería de hilos combina ULT con LWPs
    - Se utilizan para mapear ULT con KLT
    - Un LWP puede dar soporte a varios ULT y se corresponde con un 1 KLT
    - Los KLT se planifican en las múltiples CPU
    - Permite que ULTs se ejecuten en múltiples procesadores
    - Planificados independientemente por el Kernel (el kernel solo ve LWPs)

Ejemplo - Solaris



Ejemplo - Solaris

- ☑ Los ULT pueden estar
  - ✓ Ligados
    - ◆ Asociado permanentemente a un LWP
    - ◆ Aplicaciones en Tiempo Real
  - ✓ No ligados (opción por defecto)
    - ◆ No están permanentemente asociados a un LWP
    - ◆ Múltiplexación entre LWPs disponibles
  - ◆ Este esquema permite priorizar la ejecución de ULTs en base al modo de asociación
  - ◆ Los ULT se schedulean dentro de los LWP sin que intervenga el kernel (no es necesario el context switch). Lo que los hace muy eficientes.

Ejemplo - Solaris

- La biblioteca de hilos realiza además estas actividades:
  - Ajusta el número de LWPs para mejorar el rendimiento de la aplicación.
  - Si todos los LWPs de un proceso están bloqueados y hay más ULTs, la biblioteca crea otro LWP para asignar a un ULT que espera.
  - Elimina los LWP cuando no se utilizan por cierto tiempo

Ejemplo - Solaris

- ☑ Cada KLT tiene:
  - ☑ Una pequeña estructura de datos que incluye:
    - ☑ Copia de registros del kernel
    - ☑ Apuntador al LWP al que está relacionado
    - ☑ Información de prioridades y planificación
  - ☑ Un stack
- ☑ Cada LWP tiene (en espacio de direcciones del Kernel):
  - ☑ Registros para el ULT con el que está ligado
  - ☑ Información de memoria y de administración
- ☑ Cada ULT tiene (en espacio de direcciones de usuario):
  - ☑ Identificador de hilo
  - ☑ Conjunto de registros (incluye PC y apuntador de pila
  - ☑ Pila
  - ☑ Prioridad

Ejemplo - Solaris

- ☑ Analizar:
  - ✓ Supongamos que se quiere trabajar con 5 archivos y hay 5 read que deben ocurrir simultáneamente.
  - ✓ Debe haber 5 LWP donde cada ULT haga el request para que se canalicen como requerimiento de I/O simultáneos que puedan llevarse a distintas CPUs (ideal: que esten en distintos discos...).
  - ✓ Si fueran 4 LWP, el 5to debería esperar el retorno de uno de los LWP para ejecutarse.

Solaris

- Mas Información:
  - http://www.cs.cf.ac.uk/Dave/C/node29.html (04/2018)

Otras Referencias

- http://www.usenix.org/publications/library/proceedings/usenix-nt98/full_papers/zabatta/zabatta_html/zabatta.html (03/2016)