Tema 3 - Sistemas operativos distribuidos

Procesos

Felipe Ortega, Enrique Soriano

GSyC, ETSIT. URJC.

Sistemas Distribuidos (SD)

3 de octubre, 2019

Contenidos

Procesos y threads

3.1 Procesos

Procesos

Repaso: ¿Qué es un proceso?

- **Transparencia de concurrencia.**
- El OS mantiene una tabla de procesos para almacenar el estado del proceso: los registros de la CPU, la tabla de páginas, ficheros abiertos, etc.
- Crear un proceso es caro: crear/copiar los segmentos, inicializar estructuras,
- Cambiar de contexto es caro también: salvar y cargar los registros de la CPU, invalidar entradas de la TLB, cambiar la tabla de páginas...

Procesos

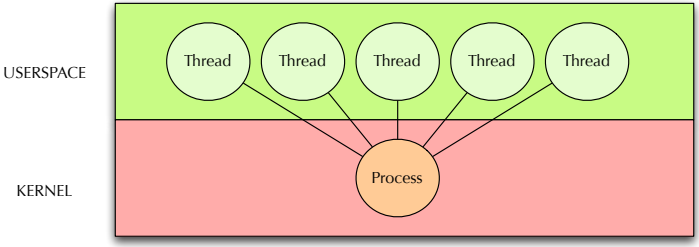
Threads: su contexto es básicamente el estado de la CPU.

- $\downarrow$ transparencia de concurrencia $\Rightarrow \uparrow$ complejidad.
- Dos tipos de threads:
  - Threads de biblioteca (usuario).
  - Threads de kernel.
- Son expulsivos (preemptive) cuando el sistema se encarga de cambiar el contexto cuando toque (planificación).
- Son no expulsivos (non-preemptive), o colaborativos, cuando ellos deciden cuándo dejan la CPU para que pueda entrar otro.

3.1 Threads de biblioteca

Threads de biblioteca

- Modelo N-1.
- Cada thread tiene su propio contador de programa y pila.
- El contexto del thread se gestiona área de usuario. El OS no sabe nada de los threads.



Threads de biblioteca

- Es barato crear, destruir y conmutar threads.
- Los threads no pueden ejecutar en paralelo en un multiprocesador.
- Si se bloquea el proceso se bloquean todos los threads. P. ej. I/O.

Mecanismo: longjmp y setjmp

```
int  setjmp(jmp_buf env);
void longjmp(jmp_buf env, int val);
```

- Lo que hay dentro de un `jmp_buf` depende de la arquitectura, no es portable (son detalles internos de la implementación de la Glibc).
- Hay que tener mucho cuidado: no saltar a una función que ha retornado, las variables locales deberían ser volatile[1], etc.
- Hay una versión especial para guardar además la máscara de señales: `sigsetjmp`.

[1] Si es volatile, el compilador no optimiza nada para esa variable, como usar un registro para ella, etc.

Mecanismo: getcontext y cia.

Si queremos modificar *a mano* el contexto de forma portable, debemos usar estas funciones en lugar de `setjmp/longjmp`:

```
int  getcontext(ucontext_t *ucp);
int  setcontext(const ucontext_t *ucp);
void makecontext(ucontext_t *ucp, void (*func)(), int argc, ...);
int  swapcontext(ucontext_t *oucp, const ucontext_t *ucp);
```

- Un `ucontext_t` es análogo a un `jmp_buf`, tiene los siguientes campos (entre otros):
  - `uc_link`: puntero al siguiente contexto.
  - `uc_stack`: información sobre la pila.
  - `uc_sigmask`: máscara de señales.
  - `uc_mcontext`: contexto, dependiente de máquina y debe ser opaco para el programador (para que sea portable).

Mecanismo: getcontext y cia.

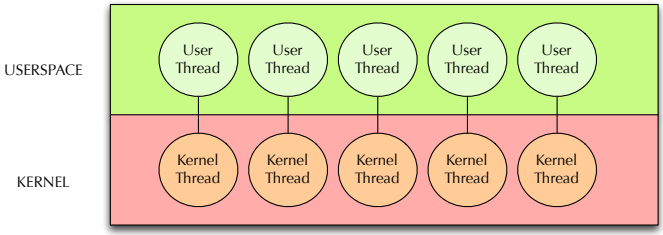
- `getcontext`: guarda el contexto en el `ucontext_t`.
- `setcontext`: restaura el contexto del `ucontext_t`.
- `makecontext`: modifica un contexto, que ha podido ser inicializado antes con una llamada a `getcontext()`. Pone el PC para empezar a ejecutar una función con ciertos parámetros.
- `swapcontext`: similar a `setcontext`, pero salva el estado antes, como haría `getcontext`. Cuando se restaura el contexto del llamador, retorna.

Todas retornan 0 en éxito, -1 en error.

3.3 Kernel threads

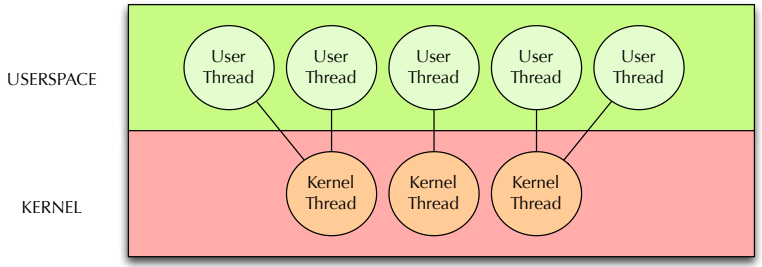
Threads de kernel. Modelo 1-1

- *Kernel-level threading*.
- Los threads son en realidad procesos que comparten memoria (y otras cosas): *procesos ligeros*.
- Crear, destruir y conmutar threads más caro: hay que entrar al kernel.
- P. ej. Linux 2.6 NPTL (Native Posix Thread Library).



Threads de kernel. Modelo N-M

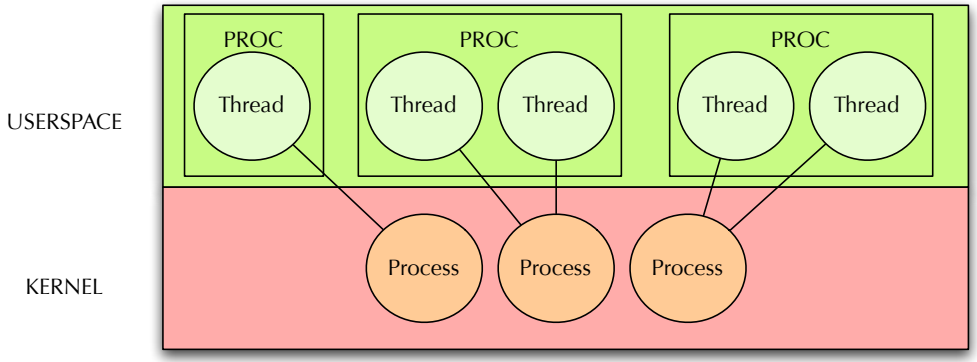
- Un proceso puede albergar uno o varios threads.
- Es un modelo híbrido que mezcla las dos aproximaciones anteriores.



Ejemplo M-N: *libthread* de Plan 9

- Una aplicación puede crear uno o más *procs*, que comparten memoria (segmento de datos y BSS).
- Hay un *proc* por proceso.
- Un *proc* puede albergar 1 o más threads.
- Los threads son *corrutinas colaborativas* (non-preemptive).
- Si un proceso se bloquea, se bloquean todos los threads de ese *proc*.

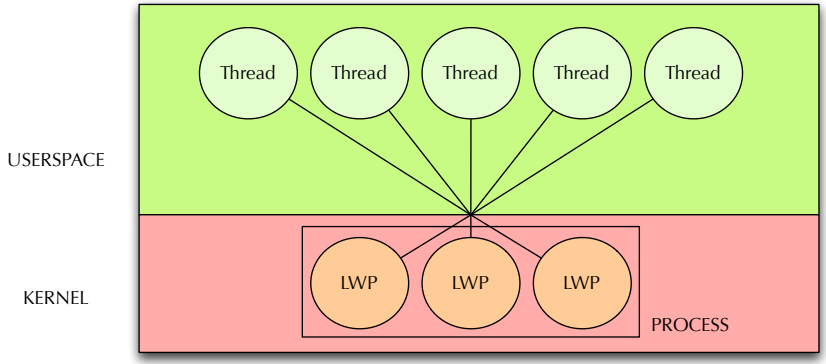
Ejemplo M-N: *libthread* de Plan 9



Ejemplo M-N: SunOS LWP

- Un proceso se compone de un espacio de direcciones y un conjunto de LWPs, que el kernel planificará por separado.
- Un thread está totalmente representado en espacio de usuario.
- Un LWP es como una *CPU virtual* para un thread.
- Los LWP planifican los threads sin necesidad de entrar al kernel.
- Si un LWP se bloquea, otro LWP puede ejecutar el resto de threads del proceso. Cada LWP puede hacer llamadas al sistema, tener fallos de página y ejecutar en paralelo independientemente.

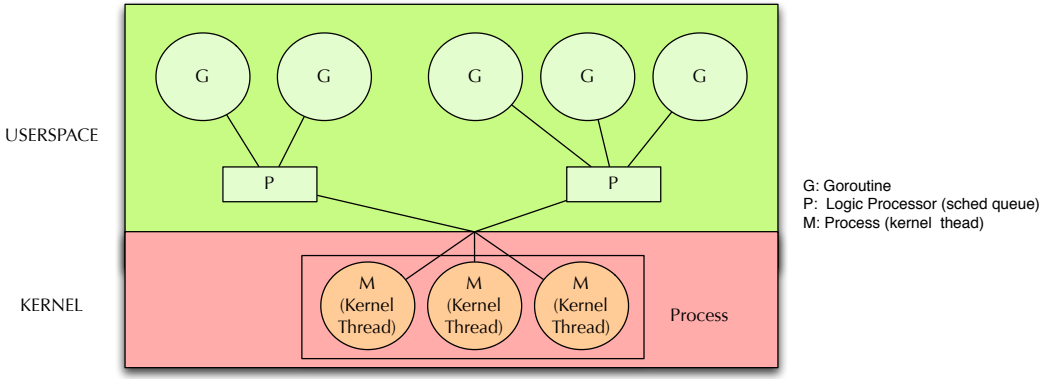
Ejemplo M-N: SunOS LWP



Ejemplo M-N: Go

- Las G (goroutines) son en realidad corutinas colaborativas (threads de usuario).
- Los M (worker thread, *machine*) son threads de kernel que van ejecutando Gs.
- Los P (*logic processor*) son colas de planificación de Gs sobre Ms.
- Los Ps cogen Gs de una cola global.
- Cuando un P se queda sin Gs que ejecutar, se los roba a otro P (balanceo).

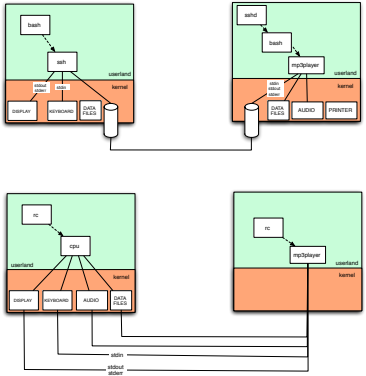
Ejemplo M-N: Go



Procesos: Ejecución remota

Migración de procesos $\neq$ ejecución remota de procesos:

- Clásica: comandos `rlogin`, `rexec`, `ssh`, etc.
- Plan 9: comando `cpu`.



3.4 Migración

Procesos: Migración

- Tipos de movilidad:
  - Débil (weak mobility): migración de código.
  - Fuerte (strong mobility): migración de procesos.
- Motivos:
  - Balancear carga (cpu, red, etc.).
  - Acercarse a una fuente de datos.
  - Tolerar el fallo parcial de la máquina.
  - Computación móvil.
- Puede ser transparente para la aplicación/usuario (transparencia de migración/relocalización).
- Antes (1990s), se usaba esta aproximación (P. ej. Sprite).
- Ahora la aproximación es distinta: pausar, migrar y reanudar máquinas virtuales o contenedores.

Procesos: Migración

¿Por qué es complicado?

Parar $\rightarrow$ **Capturar proceso (checkpoint)** $\rightarrow$ **Mover** $\rightarrow$ **Continuar**

¿Qué hay que mover?

- **Código:** instrucciones. ¿Entorno heterogéneo?
- **Estado:** pila, registros, datos privados, PC, etc. Si la infraestructura es distinta, es un problema.
  - Endianness.
  - Tipos de datos.
  - ...

Procesos: Migración

¿Qué podemos hacer con la memoria?

Coste inicial vs. Coste en ejecución

- **Eager (all):** copiar toda la memoria.
- **Eager (dirty):** copiar sólo las páginas sucias, el resto copiarlas de almacenamiento (no se hace en demanda).
- **Copy On Reference (COR):** paginación en demanda, reclamando las páginas a la máquina origen.
- **Flushing:** las páginas del proceso van a área de intercambio (*swap*) antes de migrar.
- **Precopy:** se van copiando las páginas sucias de memoria de forma especulativa a otra máquina, aunque el proceso no haya migrado todavía.

Procesos: Migración

¿Qué hay que mover?

- **Recursos:** referencias a dispositivos, conexiones abiertas, descriptores de ficheros, etc. ← difícil.
  La **vinculación** puede ser más fuerte o menos:
  - Por identificador (p.ej. URL)
  - Por valor (p.ej. libsec.1.12.so)
  - Por tipo (p.ej. /dev/mouse)

Es más sencillo mover una VM completa y hoy nos lo podemos permitir.

Bibliografía I

- *A. S. Tanenbaum.* Operating Systems, design and implementaiton. Pearson Prentice Hill.
- *A. S. Tanenbaum.* Distributed Systems. Pearson Prentice Hill.
- *A. S. Tanenbaum.* Modern Operating Systems. Pearson Prentice Hill.
- *A. Silberschatz.* Operating Systems. Wiley.
- *M. L. Powell , S. R. Kleiman , S. Barton , D. Shah , D. Stein , M. Weeks.* SunOS Multi-thread Architecture. Winter 1991 USENIX Conference.
- *D. S. Milojičić, F. Douglis, Y. Paindaveine, R. Wheeler, S. Zhou*. 2000. Process migration. ACM Comput. Surv. 32, 3, 241-299.
- *John K. Ousterhout, Andrew R. Cherenson, Frederick Douglis, Michael N. Nelson and Brent B. Welch*, The Sprite Network Operating System, IEEE Computer, 21. 1988.
- *N. Deshpande et al.. Analysis of the Go runtime scheduler*

Bibliografía II


[van Steen & Tanenbaum, 2017] van Steen, M., Tanenbaum, A. S.
Distributed Systems.
Third Edition, version 01. 2017.


[Colouris et al., 2011] Colouris, G., Dollimore, J., Kindberg, T., Blair, G.
Distributed Systems. Concepts and Design.
Pearson, May, 2011.


[Ortega et al., 2005] Ortega, J., Anguita, M., Prieto, A.
Arquitectura de Computadores.
Ediciones Paraninfo, S.A. 2005.