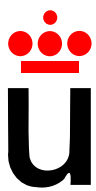
Tema 2: Programas y procesos

Enrique Soriano

Laboratorio de Sistemas,
Grupo de Sistemas y Comunicaciones,
URJC

15 de febrero de 2010



Universidad
Rey Juan Carlos

Procesos y programas

- Programa: conjunto de datos e instrucciones que implementan un algoritmo.
- Proceso: programa que está en ejecución, un programa vivo que tiene su propio **flujo de control**, su directorio de trabajo, y es independiente de los otros procesos.

Procesos

- Procesos concurrentes: varios procesos que están ejecutando al mismo tiempo.
- El sistema operativo crea la ilusión de que cada uno tiene su propia CPU.
- Ejecución paralela vs. ejecución pseudo-paralela $\longrightarrow$ para el programador es lo mismo.

Programas cargados

- Fuente → Compilación → Enlazado → Binario.
- El fichero binario contiene la información para crear un proceso, organizadas en secciones.
- En su cabecera se indica la arquitectura, el tamaño/**offset** de las distintas secciones que contiene y el punto de entrada (dirección para comenzar a ejecutar).

Programas cargados: secciones

- **Tabla de símbolos**: las strings con los tipos, ámbito, nombres, etc. de variables y funciones del código.
- Se usan para depurar el código. Pueden quitarse del binario, ya que el sistema no lo utiliza para nada.
- `strip` sirve para quitar la tabla de símbolos.

Programas cargados: secciones

- Un programa almacenado en un binario es distinto a un programa cargado en memoria mientras ejecuta.
- P. ej. las variables sin inicializar no se incluye en el fichero, sólo la información de que existen.

ver global.c

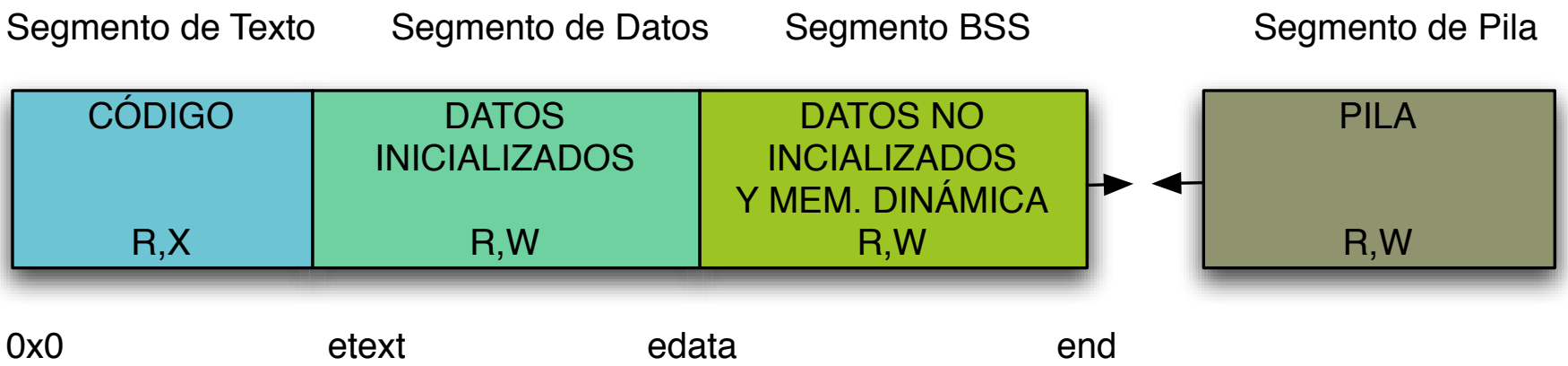
El cargador (loader)

- llamada al sistema (exec) $\longrightarrow$ el kernel carga el binario mediante el cargador.
  1. Mira en la cabecera el tamaño de las secciones del binario.
  2. Copia las instrucciones de código máquina (imagen).
  3. Copia las variables inicializadas.
  4. Reserva la memoria para las variables no inicializadas y se pone a cero.

Memoria virtual

- Memoria virtual: cada proceso tiene su espacio de direcciones virtuales (4 GB).
- El proceso cree que es el único programa cargado en memoria.
- Las direcciones virtuales son absolutas.
- `nm -n`

Segmentos


Segmento de Texto
Segmento de Datos
Segmento BSS
Segmento de Pila
CÓDIGO
R,X
DATOS INICIALIZADOS
R,W
DATOS NO INCIALIZADOS Y MEM. DINÁMICA
R,W
PILA
R,W
0x0
etext
edata
end

Memoria virtual

- El sistema operativo usa **paginación en demanda**: el programa se va cargando en memoria poco a poco según se demandan direcciones de memoria.
- Si se borra o se sobreescribe un binario, los procesos pueden fallar.
- BSS: la memoria física se reserva en realidad cuando se usa por primera vez.
- Hay segmentos que se pueden *compartir* entre procesos (texto).

Nacimiento

- El **cargador (loader)** pone a ejecutar una función que llama a `main`, y mete en la pila del proceso los argumentos (`argc,argv[]`).
- Un proceso tiene un identificador único llamado PID (*Proccess ID*). Llamada al sistema para conseguirlo: `getpid`.
- Los argumentos sirven para modificar el comportamiento del programa.
- Macros para procesar argumentos: `ARGBEGIN,ARGEND,ARGFE,ARGF`

ver argecho.c

Muerte

- Cuando `main` retorna, la función que la llamó realiza una llamada al sistema `exits` para acabar.
- El parámetro que se le pasa a `exits` determina el estado del proceso al acabar.
- `$status`

Errores en las llamadas al sistema

- **Hay que comprobar los errores.**
- Las llamadas al sistema suelen retornar negativo en caso de error (`nil` si son de tipo puntero). Los detalles se especifican en la correspondiente página de manual.
- Se actualiza la cadena de error del proceso *errstr(2)*.
- `rerrstr,werrstr,%r`

ver errs.c

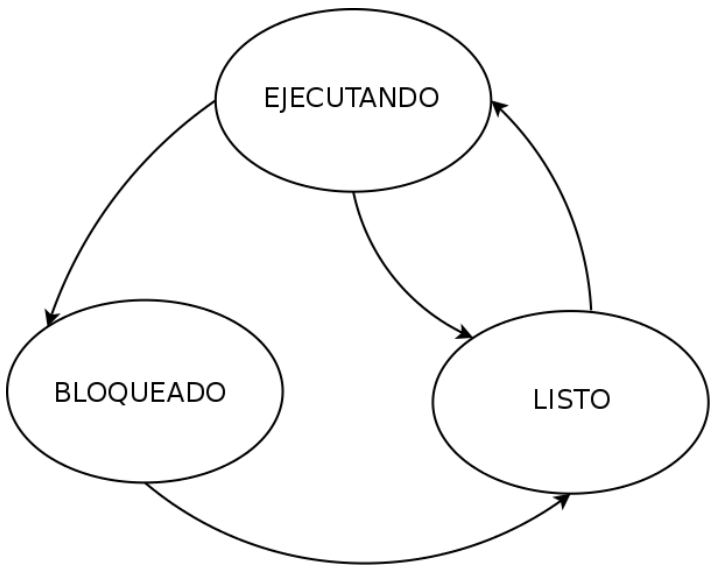
Entorno

- `name=value` (son strings)
- Se suelen *heredar* en programas que ejecutan en la misma ventana.
- Podemos definirlas para un nuevo proceso:
  `term% a=hola b=adios micomando`
- Llamadas al sistema: `getenv`,`putenv`

ver env.c

Procesos: estado

- Ejecutando: `Running`
- Listo para ejecutar: `Ready`
- Bloqueado: `Stopped, Pread, Pwrite, Sleep ...`
- Muerto: `Broken, Moribund`



Procesos y estado

- Tabla de procesos (PCBs):
  - Estado
  - Prioridad
  - Registros
  - PID
  - Segmentos: text, data, bss, stack
  - Descriptores de fichero
  - Espacio de nombres
  - Entorno
  - ...

Planificación

- **Planificador (scheduler)**
- **Política** vs **Mecanismo**
- **Cooperativa (non-preemptive)** vs **expulsiva (preemptive)**.

Planificación: mecanismo

- **Cambio de contexto (simplificado)**
  1. Salvar el estado de los registros del procesador en la tabla de procesos (entrada en la tabla de procesos para el proceso saliente).
  2. Cargar el estado de los registros del proceso entrante en el procesador (entrada en la tabla de procesos para el proceso entrante). El contador de programa al final.

Planificación: políticas

- Ya sé sacar un proceso y meter otro... ¿Cómo reparto la CPU?
- Los procesos se pueden ver como ráfagas de operaciones de CPU y operaciones de I/O.
- Algunos procesos están dominados por CPU y otros procesos están dominados por I/O.

Criterios, ¡no se puede todo a la vez!

- **Justicia (fairness)**: todos los procesos tienen su parte.
- **Eficiencia (efficiency)**: sacar máximo partido a la CPU.
- **Respuesta interactiva**: es importante el tiempo de espera en la cola hasta que se empieza con él.
- **Respuesta (turnaround)**: tiempo de entrega.
- **Rendimiento (throughput)**: número de trabajos acabados por unidad de tiempo.

Planificación cooperativa

- **FCFS**: Cola FIFO de llegada.
  - Efecto *convoy*.
  - Tiempo promedio de espera alto.
- **SJF**: Se elige el proceso con ráfaga más corta.
  - Tenemos que saber el volumen del trabajo de antemano.
  - Mejora el tiempo de espera promedio.

Planificación expulsiva: SRTF

- **SJF expulsivo (SRTF)**: Cuando entra en la cola un proceso con una ráfaga más corta que lo que le queda al actual, se expulsa al actual.

Planificación expulsiva: Round-Robin

- Se asigna la CPU en *cuantos*.
- Se rota por los procesos que están listos para ejecutar.
- Cuando se agota el *cuanto*, se expulsa al proceso.
- El proceso puede dejar la CPU antes de que acabe su *cuanto*.
- Los procesos nuevos entran por el final de la cola.

Planificación de procesos: Round-Robin

Pros y contras:

- Aumenta la respuesta interactiva.
- Reduce el rendimiento (throughput) por los cambios de contexto.

Problema: ¿Cómo se elige el *cuanto*?

- Si es pequeño, se desperdicia mucha CPU en los cambios de contexto.
- Si es grande, las aplicaciones interactivas sufren.

Planificación de procesos: prioridades

- No todos los procesos tienen la misma importancia.
  - P. ej. un reproductor de video vs. un cliente de correo.
- Prioridades **estáticas** vs. **dinámicas**.
- Problema: inanición (*starvation*). Solución: tener en cuenta la edad del proceso (*aging*).

Planificación de procesos: colas multinivel con retroalimentación

En general:

- Número de colas.
- Algoritmo para cada cola.
- Método para subir el proceso a una cola de mayor prioridad.
- Método para bajar el proceso a una cola de menor prioridad.
- Método para determinar la cola en la que empieza un proceso.

Planificación de procesos: colas multinivel con retroalimentación

Ejemplo particular: Round-Robin con prioridades dinámicas.

- Múltiples colas, según prioridad.
- R-R con los procesos de cada cola.
- Si hay procesos listos en una cola, no se atienden las colas de menor prioridad.
- Cuando un proceso agota su *cuanto*, se baja su prioridad.
- Cuando un proceso no agota su *cuanto*, se sube su prioridad.

Prioridades: caso de estudio, Plan 9

- 20 prioridades (0-19).
- Los procesos de usuario empiezan en 10 por omisión.
- Los procesos de kernel empiezan en 13 por omisión.
- Nunca pueden subir de su prioridad base.
- Bajan de prioridad si agotan sus *cuantos*.

Threads

- Flujo de control.
- Procesos "ligeros" (LWP).
- Se comparte todo menos la pila y los registros.

Threads

- Threads de usuario.
- Threads de kernel.

Threads

- Many-to-one (N-1).
- One-to-one (1-1).
- Many-to-many (N-M).

Mecanismo para planificación en área de usuario

Funciones de la `libc`:

- `setjmp` guarda el estado de los registros en una estructura de datos (`jump buffer`).
- `longjmp` carga el estado de los registros de una estructura de datos (`jump buffer`).
- `man 2 setjmp`

Depuración

- Trazar con `print`.
- `src`
- `acid: stk(), lstk()`
- Para abortar un programa (dejarlo *Broken*) : `abort()`
- `leak`

ver fail.c

Sistemas de ficheros como interfaz

- Plan 9 te ofrece sus abstracciones mediante una interfaz de **sistemas de ficheros sintéticos** $\longrightarrow$ no son ficheros *de verdad* (bloques en el disco duro). P. ej:
  - /dev vs. time(), etc.
  - /proc vs. kill, etc.
  - /env vs. putenv(), etc.

Sistemas de ficheros como interfaz

```
cat /proc/$pid/segment # ver segmentos
echo kill > /proc/$pid/ctl # matar proceso
lp /dev/screen # imprimir un screenshot
cp /dev/text > $home/rcsession
                        #salvar histórico de shell
```