Sistemas Operativos

Pedro Cabalar

Depto. de Computación
Universidade da Coruña

TEMA III. PROCESOS.

Concepto de proceso

- El concepto central en un sistema operativo es la idea de proceso.
- Un programa es una colección de instrucciones. Ejemplo: el comando `ls` es un archivo ejecutable guardado en algún directorio, p.ej., `/bin/ls`.

Definición

Un proceso es una abstracción que hace referencia a cada caso de ejecución de un programa.

- Ojo: un proceso no tiene por qué estar siempre en ejecución. La vida de un proceso pasa por varias fases, incluyendo la ejecución.

Concepto de proceso

- Una analogía: receta de un pastel (el programa), ingredientes (los datos), cocinero (la CPU), la mesa de cocina (la memoria), cada pastel (un proceso), el horno (un dispositivo E/S).

- ¡Nuestro chef puede tener varios pasteles a medio hacer!

Concepto de proceso

- Prueba el ejemplo: abre 2 terminales y ejecuta en uno `ls -lR /` y en otro `ls -lR /usr`. Tenemos dos procesos ejecutando el mismo programa `/bin/ls`.
- En este caso (lo habitual) cada proceso es secuencial.
- Concurrencia: su ejecución parece simultánea pero, en realidad, la CPU salta de uno a otro (como nuestro sufrido cocinero). Es lo que llamamos sistema operativo multitarea.
- Mientras ejecuta un proceso, el otro está en espera.
- Cuando hay varias CPUs podemos tener ejecución paralela. Pero cada CPU sólo puede ejecutar un proceso a la vez. Normalmente número de procesos > número de CPUs.
- Más adelante veremos procesos que internamente también son concurrentes. Utilizan hebras (*threads*) de ejecución.

Tipos de procesos

Podemos clasificarlos en función de distintos criterios.

Según su diseño:

- Reutilizables: se cargan en memoria cada vez que se usan. Los programas de usuario suelen ser de este tipo.
- Reentrantes: se carga una sola copia del código en memoria. Cada vez que se usan se crea un nuevo proceso con su zona de datos propia, pero compartiendo el código.

Tipos de procesos

Según su acceso a CPU y recursos:

- Apropiativos: acceden a los recursos y sólo los abandonan de forma voluntaria (mediante instrucción CPU).
- No apropiativos: permiten a otros procesos apropiarse de los recursos que ahora poseen.

Tipos de procesos

Según su permanencia en memoria:

- Residentes: tienen que permanecer en memoria durante toda su evolución (desde creación hasta terminación).
- Intercambiables (*swappable*): es lo más normal. El SO puede decidir llevarlos a disco a lo largo de su evolución.

Tipos de procesos

Según su nivel de privilegio (no en todos los SO):

- Privilegiados: se ejecutan en modo supervisor.
- No privilegiados: los que normalmente ejecuta el usuario.

Tipos de procesos

Según su propietario:

- Procesos de usuario: son los diseñados por los usuarios. Se ejecutan en modo no protegido.
- Procesos del sistema: son los que forman parte del SO (de E/S, de planificación de otros procesos, etc).

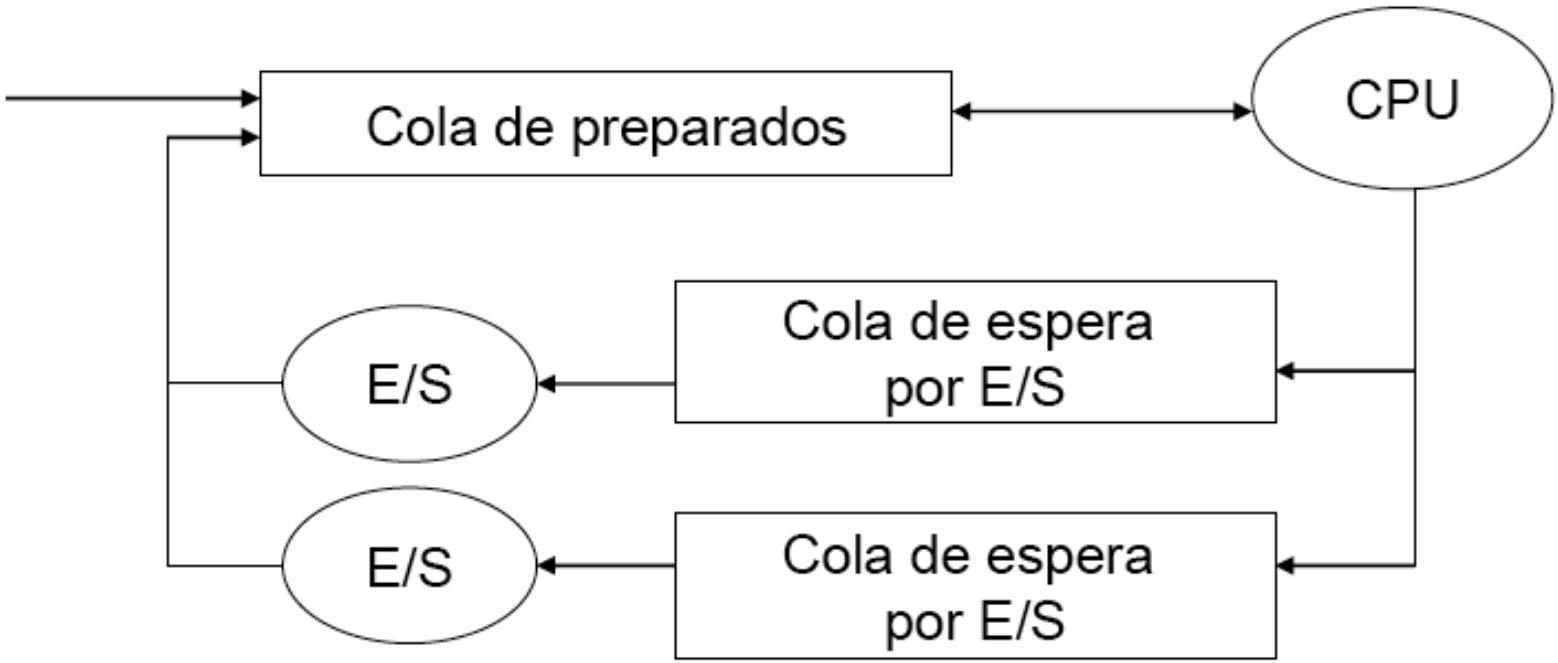
Estructuras de datos

¿Qué crees que necesita el SO para gestionar los procesos?

- Antes que nada, cargar el código en memoria
- El mismo programa puede dar lugar a varios procesos. Necesitará identificarlos: descriptor de proceso.
- Si no está siendo ejecutado, necesitará guardar información de ejecución: registros, pila, recursos que está usando, etc.

Estructuras de datos

- Necesitará saber la lista de procesos que tiene y el estado en el que están. Normalmente, usa colas de descriptores de procesos, una para cada estado o incluso para cada dispositivo E/S.



Veamos qué estructuras maneja típicamente el SO . . .

Bloque de Control del Sistema

Un primer grupo de datos es general para todos los procesos.

Definición (Bloque de Control del Sistema)

(en inglés System Control Block, SCB) es un conjunto de estructuras de datos que usa el SO para gestionar la ejecución de los procesos.

El SCB normalmente incluye:

- Lista de descriptores de procesos.
- Puntero al descriptor de proceso que está usando la CPU.
- Punteros a colas de procesos que se encuentran en distintos estados. Ejemplo: procesos en espera.
- Puntero a la cola de descriptores de recursos.
- Identificadores de las rutinas para tratar interrupciones hardware o software y errores indeseados.

Bloque de Control del Sistema

Las interrupciones permiten al SO tomar el control de CPU, por ejemplo, cuando:

- Se produce algún tipo de error
- Hay algún evento externo. P.ej.: finalización de operación E/S
- Reloj: se ha agotado algún tiempo límite

Bloque de Control del Sistema

Un ejemplo de valores de interrupción:

Nivel	**Evento Software**
0	Proceso de usuario
1	Planificación de procesos
2	Temporización
3 a 10	Drivers de E/S
11 a 15	Otros
Nivel	**Evento Hardware**
16 a 23	Interrupciones de dispositivos
24	Reloj interno
25 a 29	Errores de: - Procesador - Memoria - Buses
30	Fallo de tensión
31	Pila (stack) del núcleo errónea

Bloque de Control de Proceso

Además, el SO gestiona una tabla de procesos donde guarda la información de cada uno. Cada entrada de esa tabla se llama ...

Definición (Bloque de Control de Proceso)

(en inglés Process Control Block, PCB) son los datos particulares de cada proceso que usa el SO para gestionarlo.

Bloque de Control de Proceso

El PCB normalmente incluye información de:

- Identificación:
  - id. del proceso
  - id. del proceso padre (si hay jerarquía de procesos; ej. UNIX)
  - id. del usuario y grupo del propietario.
- Planificación:
  - Estado del proceso
  - Si estado=bloqueado, el evento por el que espera el proceso
  - Prioridad del proceso
  - Otra información usada por el algoritmo de planificación: contadores, colas, etc.

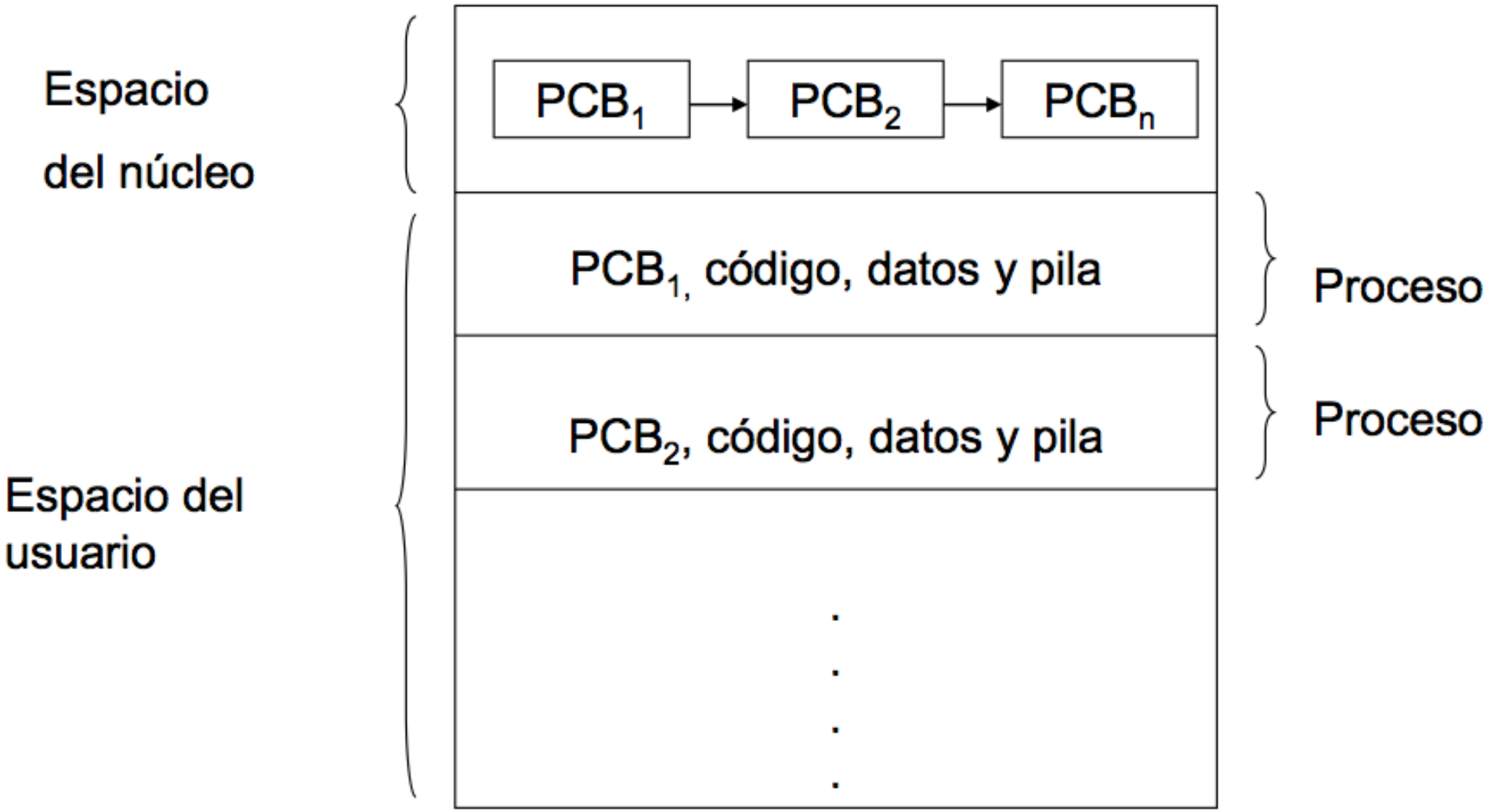
⋮

Bloque de Control de Proceso

⋮

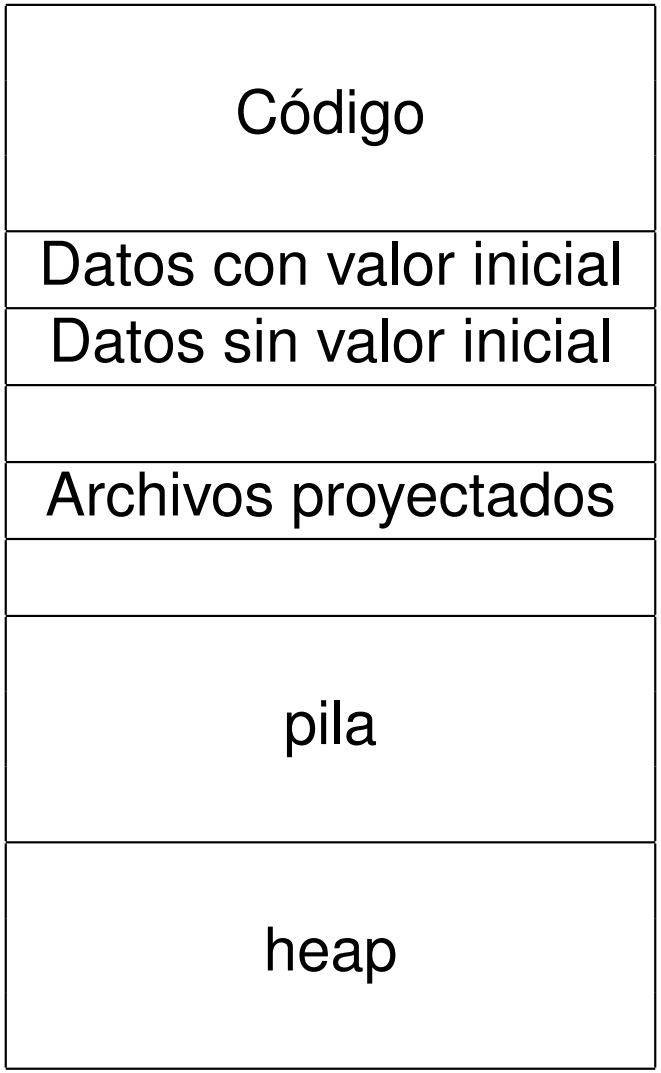
- Punteros a segmentos de memoria asignados:
  - Segmento de datos
  - Segmento de código
  - Segmento de pila
- Recursos asignados:
  - Archivos abiertos: tabla de descriptores o “manejadores” de archivos (*file descriptors* / *file handles*).
  - Puertos de comunicación asignados
- Punteros para organizar los procesos en colas.
- Información para comunicación entre procesos: señales, mensajes.

Bloque de Control de Proceso

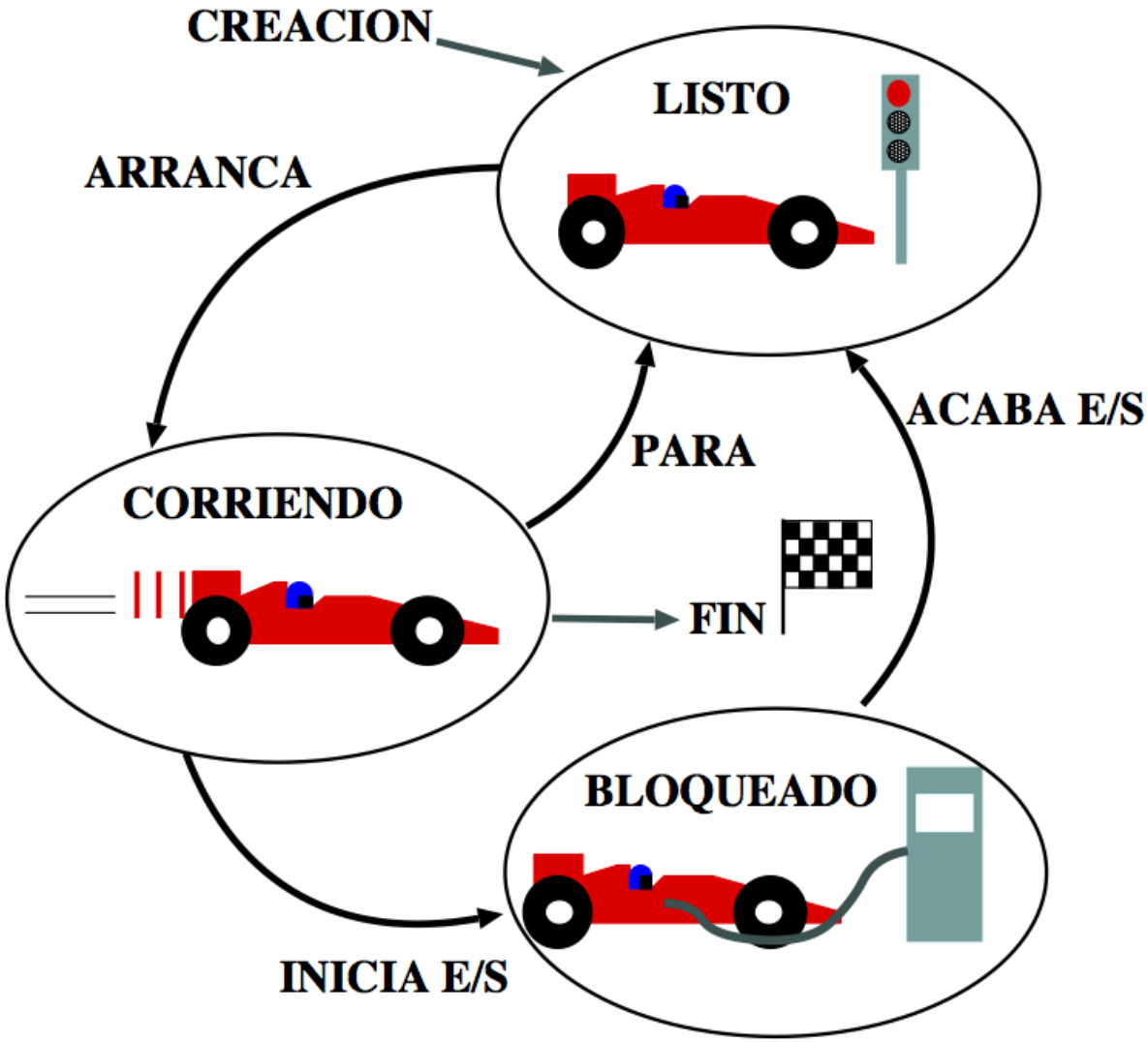


Mapa de memoria de un proceso

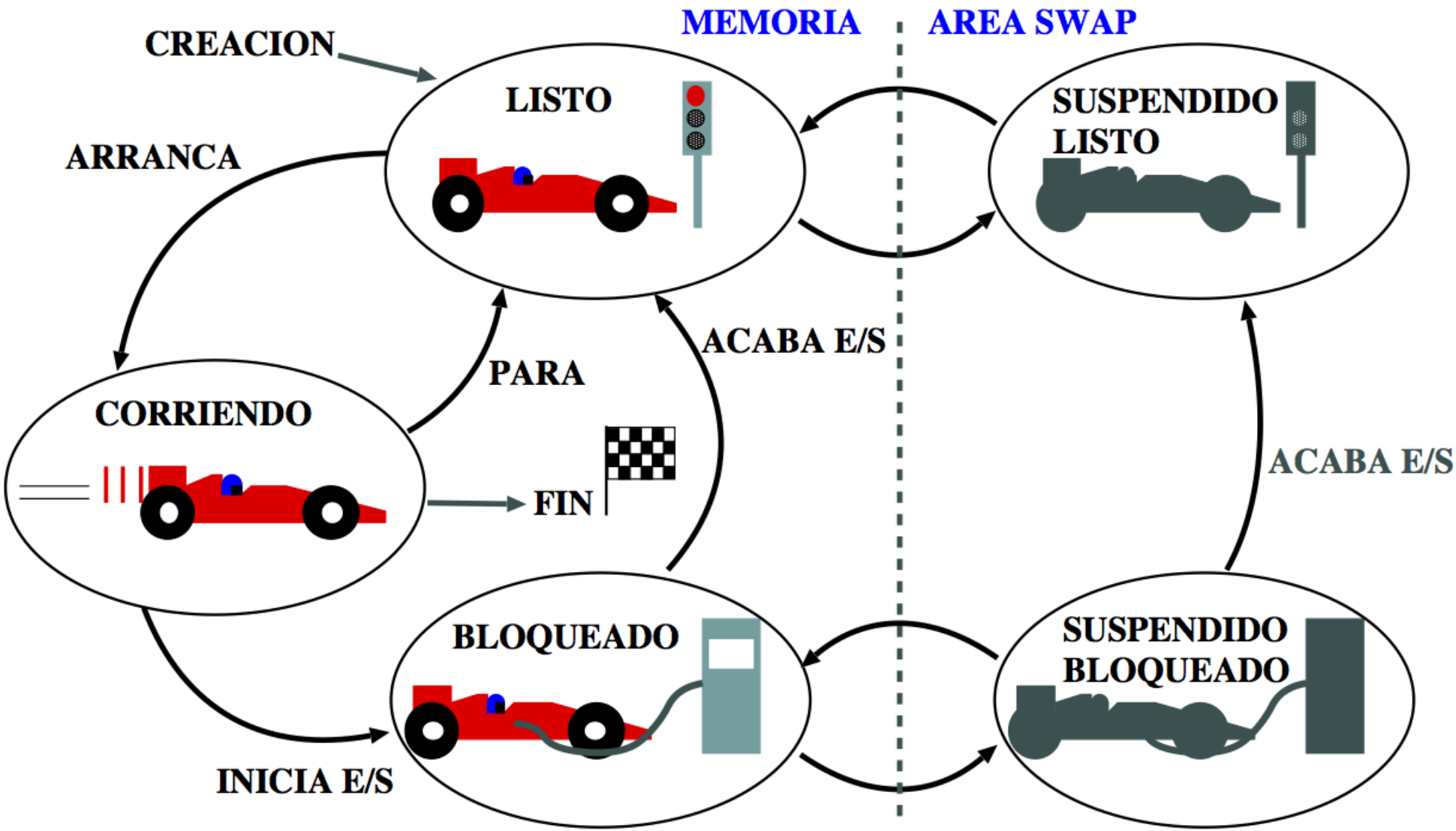
Posible mapa de memoria de un proceso:



Estados de un proceso



Estados de un proceso



Estados de un proceso

Terminología:

- preparado o listo (*ready*)
- corriendo o en ejecución (*running*)
- bloqueado (*blocked*) o dormido (*asleep*) o en espera (*waiting*).
- suspender (*swap out*) = enviar el proceso a área de intercambio (*swap*)
- reanudar (*swap in*) = traer el proceso desde área de intercambio a memoria

Transiciones de estado

- Paso a preparado: puede haber 4 causas
  - Creación: acaba de cargarse el programa en memoria e iniciamos el proceso
  - Desde ejecución: porque la CPU va a pasar a ejecutar otro proceso (cambio de contexto).
  - Desde bloqueado: porque acabó una operación E/S por la que estaba esperando.
  - Desde suspendido-listo: porque el SO decide traérselo a memoria (reanudarlo) y ya estaba preparado antes.
- Paso a ejecución: se toma el primero de la cola de preparados cuando el reloj haya interrumpido el que estaba en ejecución.

Transiciones de estado

- Paso a bloqueado: bien desde ejecución, al solicitar E/S o bien desde suspendido-bloqueado porque volvió a memoria (reanudación) pero no acabó E/S.
- Paso a suspendido-listo o suspendido-bloqueado: el SO puede decidir suspender un proceso parado (ya sea listo o bloqueado), pasando al estado correspondiente en cada caso.

Creación de un proceso

- Al crear un proceso (suponemos que el código ya está en memoria) el SO debe:
  1. asignarle un identificador
  2. crear e inicializar su PCB.
  3. actualizar el SCB para llevar cuenta de él.
  4. pasarlo a la cola de preparados.

Creación de un proceso

Un proceso puede ser creado por distintas causas:

- Inicialización del sistema: al iniciar se crean muchos procesos:
  - procesos primer plano (*foreground*) que interactúan con el usuario (ej: terminal, entorno gráfico, etc).
  - procesos segundo plano (*background*): actúan por detrás y la mayor parte del tiempo están a la espera (impresión de documentos, TCP/IP, recepción de correo, etc). Se denominan demonios (*daemons*).

  Prueba: comando `ps` en UNIX o el *task manager* en Windows.
- Llamada al sistema realizada por otro proceso para crear uno nuevo. Ejemplo: nuestro código crea otro proceso para meter datos en un buffer.
- Petición del usuario: ejemplo, desde el *shell* o la interfaz gráfica.
- Inicio de un proceso por lotes (*batch*).

En realidad en todos ellos se realiza una llamada al sistema.

Creación de un proceso en UNIX

- En UNIX tenemos dos formas de crear procesos: la llamada `fork` y la familia de llamadas `exec*`.
- `fork`: crea un clon (proceso hijo) del proceso que hace la llamada (proceso padre).
- `fork` copia todo: contador de programa, valor de las variables, estado de la pila, heap, archivos abiertos, etc.

Creación de un proceso en UNIX

- Problema de identidad: después de un `fork` ¿cómo saber dentro del código si soy el padre o el hijo?

- Respuesta: `fork` devuelve:
  - El valor 0 si estamos en el hijo
  - Un valor $>$ 0 (el pid del hijo) si estamos en el padre.
- Variante `vfork`: idem que `fork` pero no copia la pila ni los datos.

Creación de un proceso en UNIX

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
    pid_t pid, val=2;

    switch (pid=fork()) {
       case -1: printf("Error de fork\n");
                exit(0);
       case 0:  val--; break;  /* proceso hijo  */
       default: val++;         /* proceso padre */
    }
    printf("proceso %d: val=%d\n", getpid(), val);
}
```

Creación de un proceso en UNIX

- En las llamadas `exec*` el código ejecutable se toma de un archivo.
- Con `exec*` el nuevo proceso reemplaza al que llama. El que hace la llamada desaparece.
- El proceso nuevo reusa el espacio virtual creado para el antiguo. De cara al SCB, apenas se producen cambios.

Creación de un proceso en UNIX

Ejemplo con `execve`:

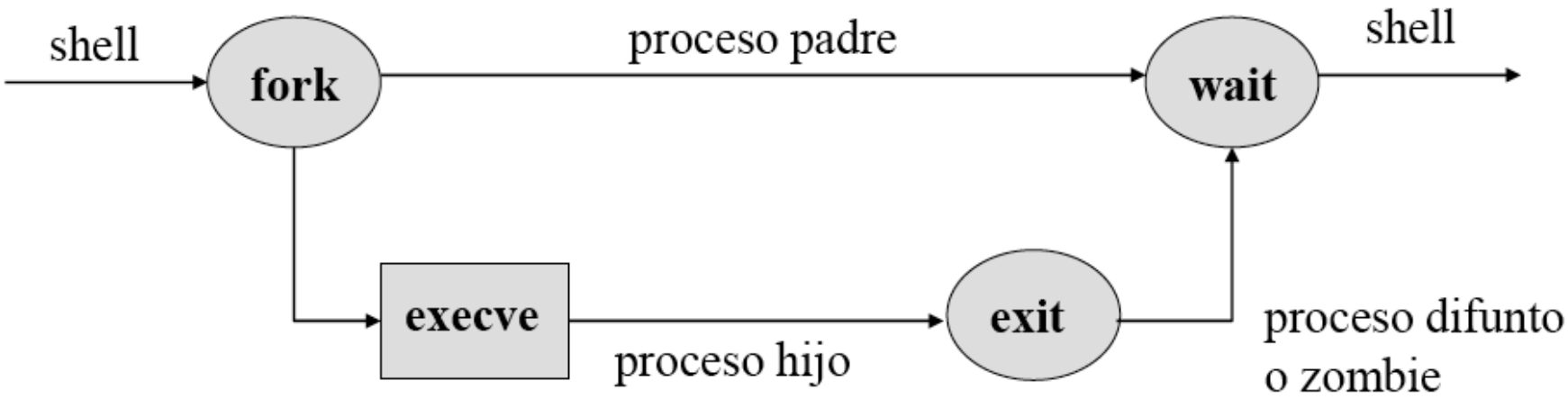
```
#include <stdio.h>
#include <unistd.h>

int main() {
  char *arg[]={"ls","-lR","/",NULL}; /*argumentos*/
  char *argp[]={NULL}; /* variables de entorno */

  execve("/bin/ls",arg,argp);
  printf("\n execve produjo un error\n");
}
```

Creación de un proceso en UNIX

- Habitualmente llamamos a `fork` para crear un hijo e inmediatamente hacemos `execve` en él para ejecutar lo que queramos. El padre se queda en espera: llamada `wait`.
- Un ejemplo: al hacer un `ls` desde el shell ocurre lo siguiente



Creación de un proceso en UNIX

Internamente, el código sería similar a:

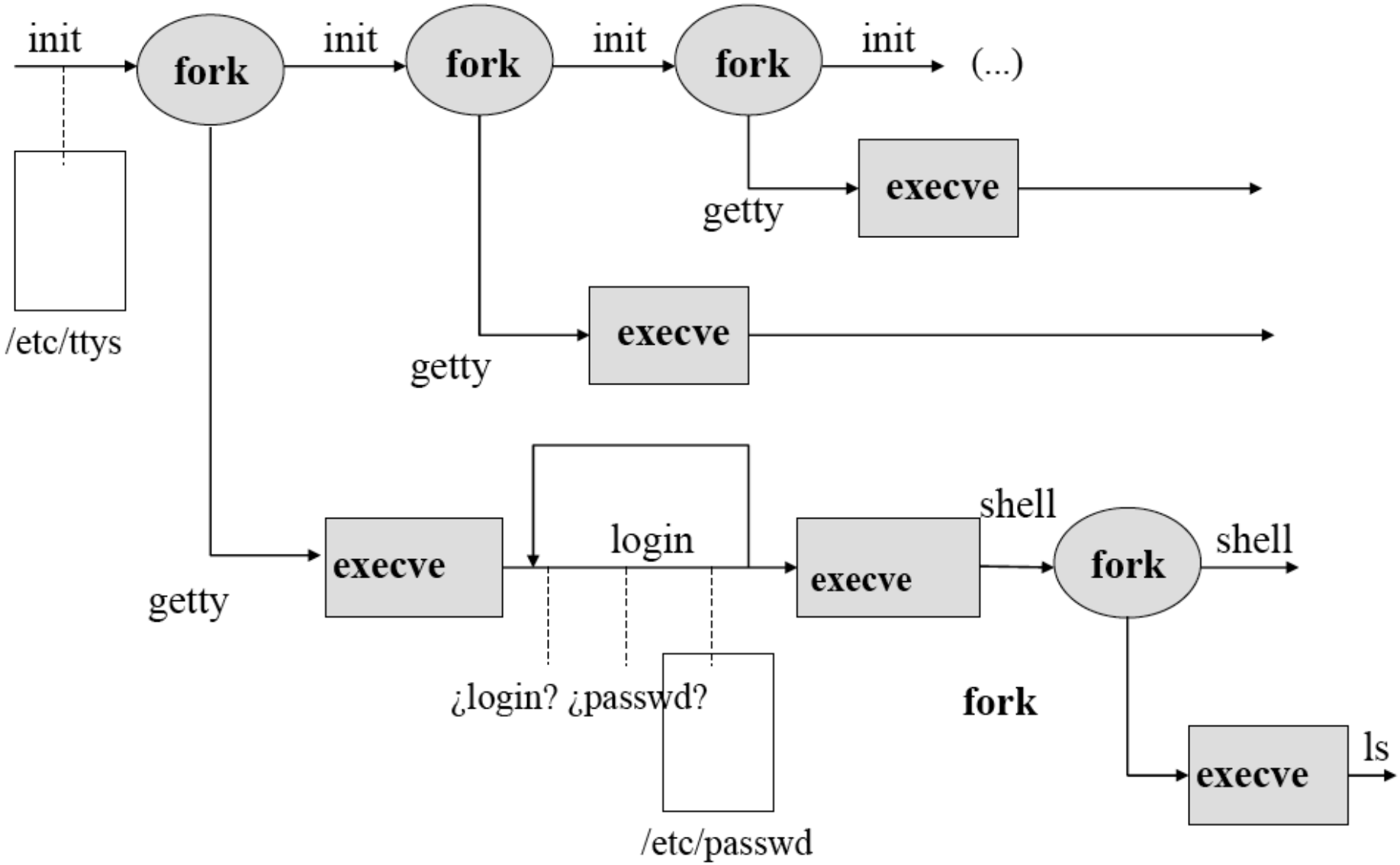
⋮

```
int main(int argc, char *argv[]) {
    pid_t pid;      time_t t;     int status;
    t=time(NULL); printf("Hago fork %s\n",ctime(&t));
    switch (pid=fork()) {
      case -1: printf("Error de fork\n"); exit(0);
      case 0: execv("/bin/ls",argv);
          printf("Error execv\n");
          exit(0);
      default:
          waitpid(pid,&status,0);
          t=time(NULL);
          printf("Termina el hijo %s\n",ctime(&t));
    }
}
```

Creación de un proceso en UNIX

- Al iniciar el sistema se lanza un proceso especial `init` presente en la imagen de arranque.
- El proceso `init` se encarga de ir lanzando otros procesos (demonios, interfaz gráfica, etc).
- En UNIX la jerarquía de procesos es un árbol cuya raíz es el proceso `init`.

Creación de un proceso en UNIX



Destrucción de un proceso

- Destruir proceso: se elimina la entrada PCB.
- Si hay procesos hijos: puede tener que esperar por ellos o finalizarlos forzosamente.

Destrucción de un proceso

Formas de terminación

1) Normal: es voluntaria y causada por el fin esperado del proceso.
2) Por error: también es voluntaria, pero causada por una situación anormal (ejemplo: no existe fichero, error en `fork`, etc). En UNIX usaríamos `perror`+`exit`.

Destrucción de un proceso

Formas de terminación (cont.)

3) Por error fatal: es involuntaria (no contemplada en el código) y causada por operación no posible (fallo de segmento, división por cero, etc).

4) Por señal de terminación (`kill`) enviada desde otro proceso con permisos para ello. Ejemplo: el padre hace `kill` ...

Ejercicio en UNIX

- Ejercicio: piensa cómo crear un timeout para `ls`.
- Pista: lanza dos hijos, uno con el `ls` y otro con el contador de timeout `sleep`+`kill`.