# Procesos

Adaptación

Juan Felipe Muñoz Fernández

# Preguntas

- ¿Qué es un proceso?
- ¿Por qué es importante el concepto de proceso en los sistemas operativos?

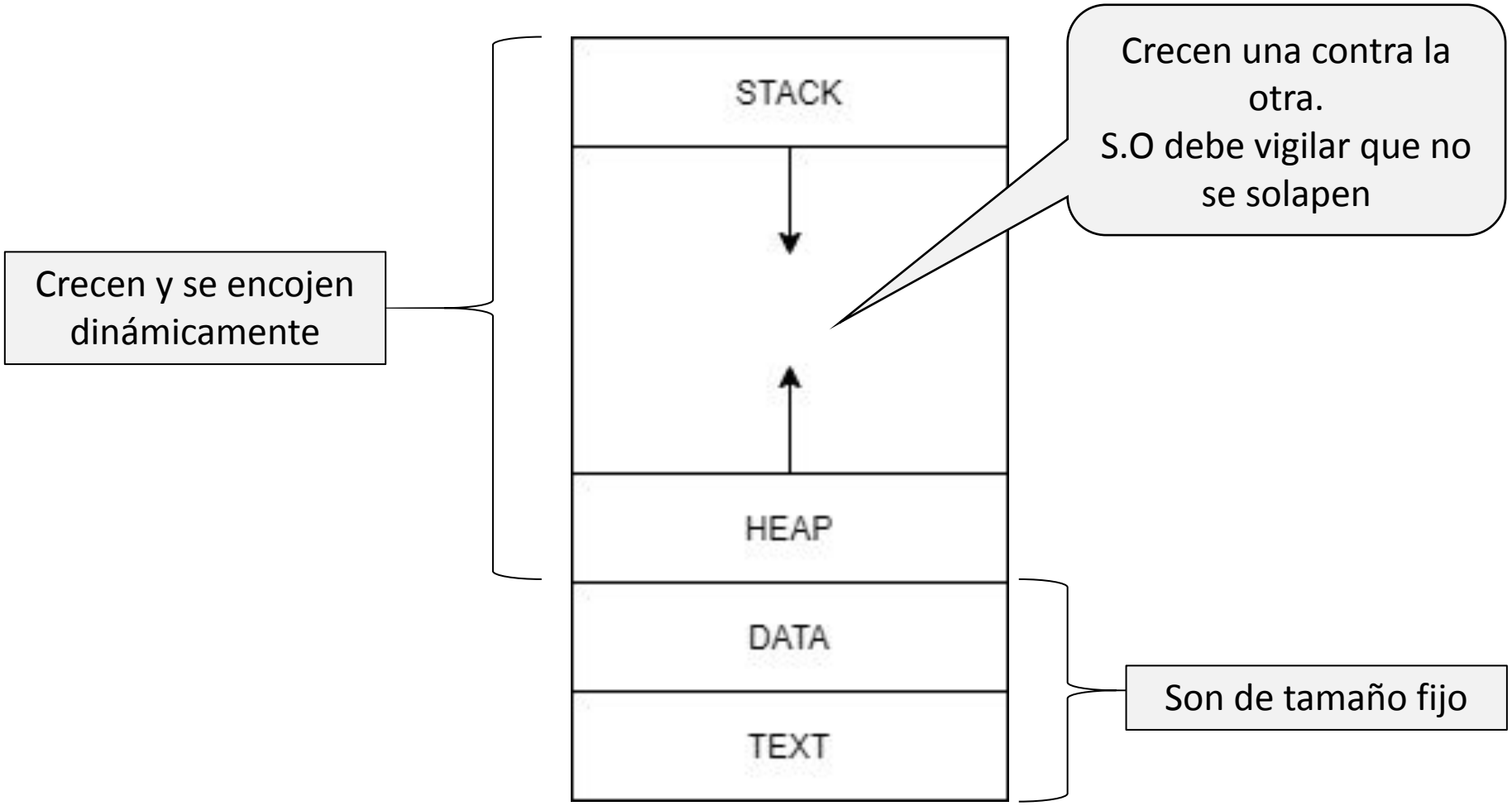
# Definición

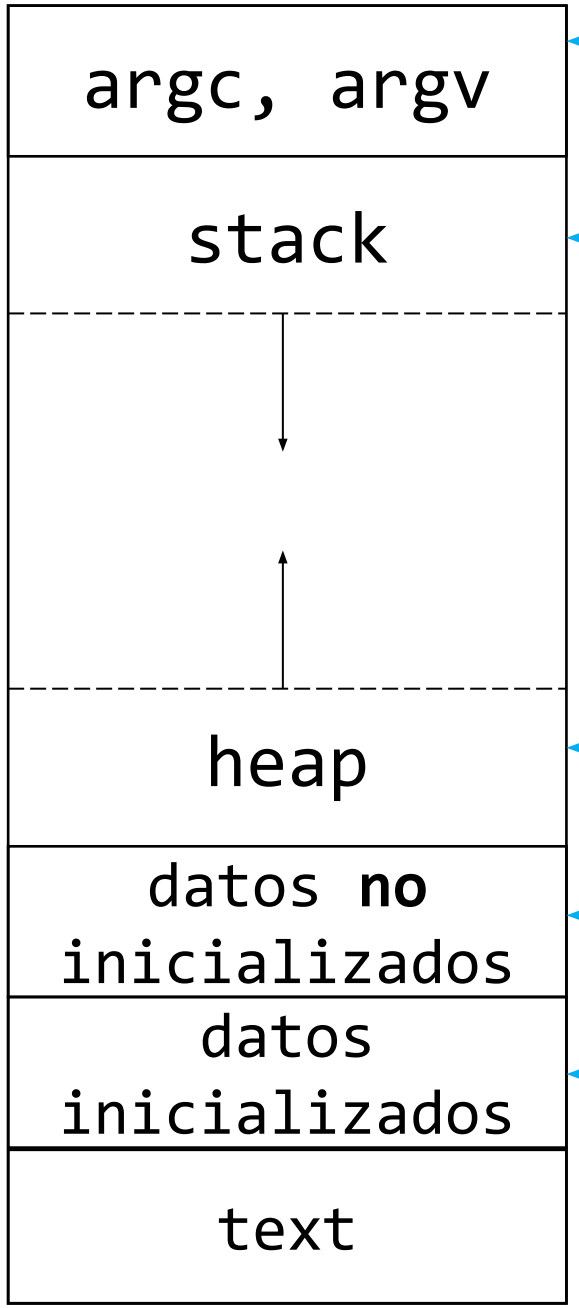
- Programa en ejecución
  - Programa en disco es una entidad estática
  - Programa cargado en memoria es una entidad dinámica
- Es la unidad de trabajo del sistema operativo
  - El S.O también tiene sus propios procesos
- Un sistema computacional moderno consiste de una colección de procesos
  - Ejecutando código de usuario (aplicaciones de usuario: ring 3)
  - Ejecutando código del núcleo del S.O (procesos del S.O: ring 0)
- Todos los procesos podrían ejecutarse concurrentemente en CPU
  - Tiempo compartido (multiplexión en el tiempo)

# Definición

- Un programa se vuelve proceso cuando:
  - El archivo ejecutable es cargado en memoria
  - Todos los programas deben estar cargados en memoria para su ejecución
- ¿Cómo se carga un archivo ejecutable en memoria?
  - Usuario da la orden: doble clic, enter, orden en línea de comandos
  - El S.O lo carga automáticamente: servicios, demonios, procesos propios
- Mapa de memoria de un proceso
  - Sección ***text***: código ejecutable
  - Sección ***data***: variables globales
  - Sección ***heap***: memoria dinámica (p. ej.: listas)
  - Sección ***stack***: datos temporales, parámetros funciones, variables locales

# Mapa de memoria de un proceso





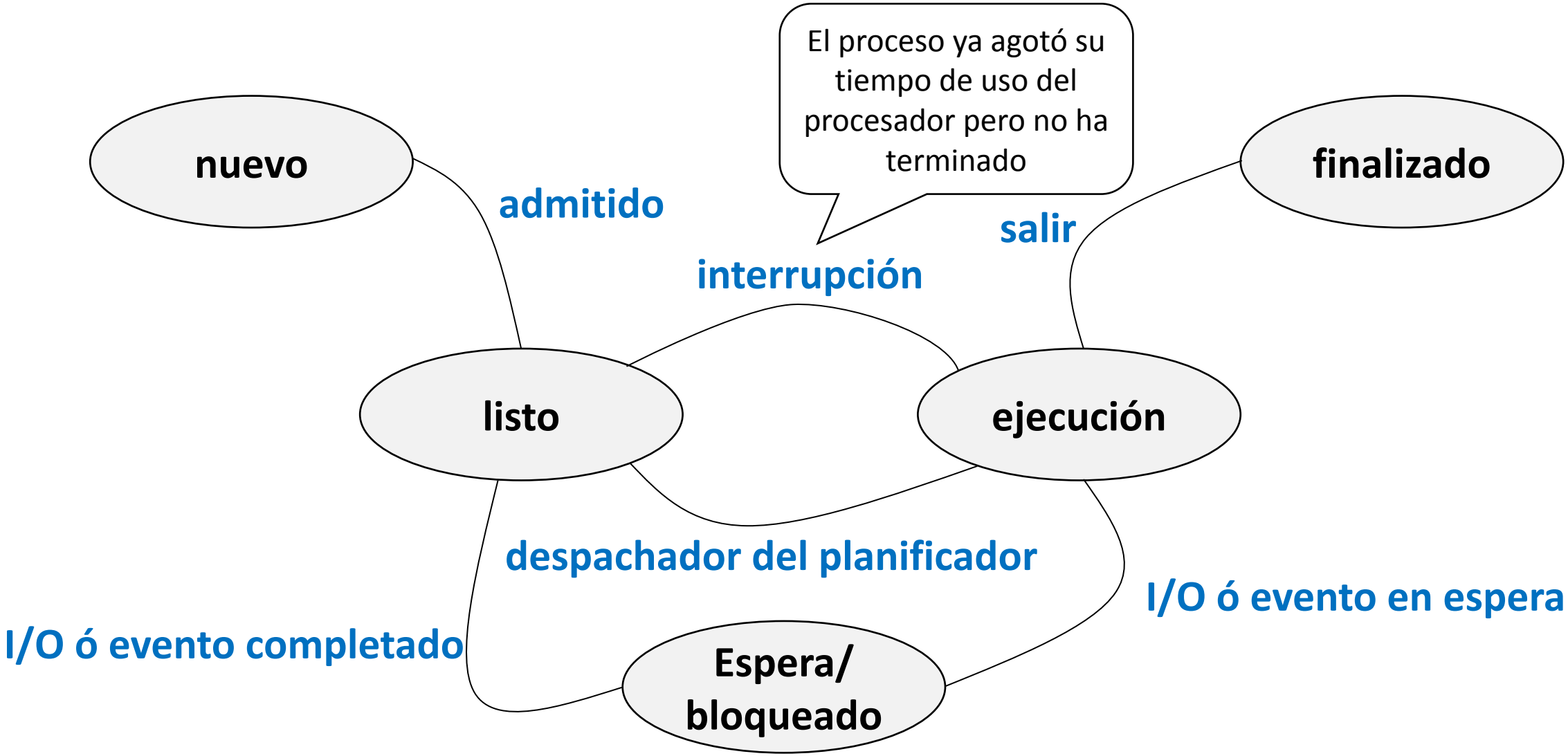

```c
#include <stdio.h>
#include <stdlib.h>

int x;
int y = 15;

int main(int argc, char *argv[])
{
    int *values;
    int i;

    values = (int *)malloc(sizeof(int)*5);

    for (i = 0; i < 5; i++)
        values[i] = i;

    return 0;
}
```

# Formación de un proceso

- Completar toda la información que lo constituye
  - Asignar espacio de memoria (virtual) constituido por varios segmentos
  - Seleccionar un PCB libre en tabla de procesos
  - Rellenar el BCP con la información del proceso
  - Cargar en el segmento de texto el código + rutinas del sistema
  - Cargar en el segmento de datos los datos inicializados
  - Crear el segmento de pila con los parámetros que se pasan al programa

# Estados y transiciones

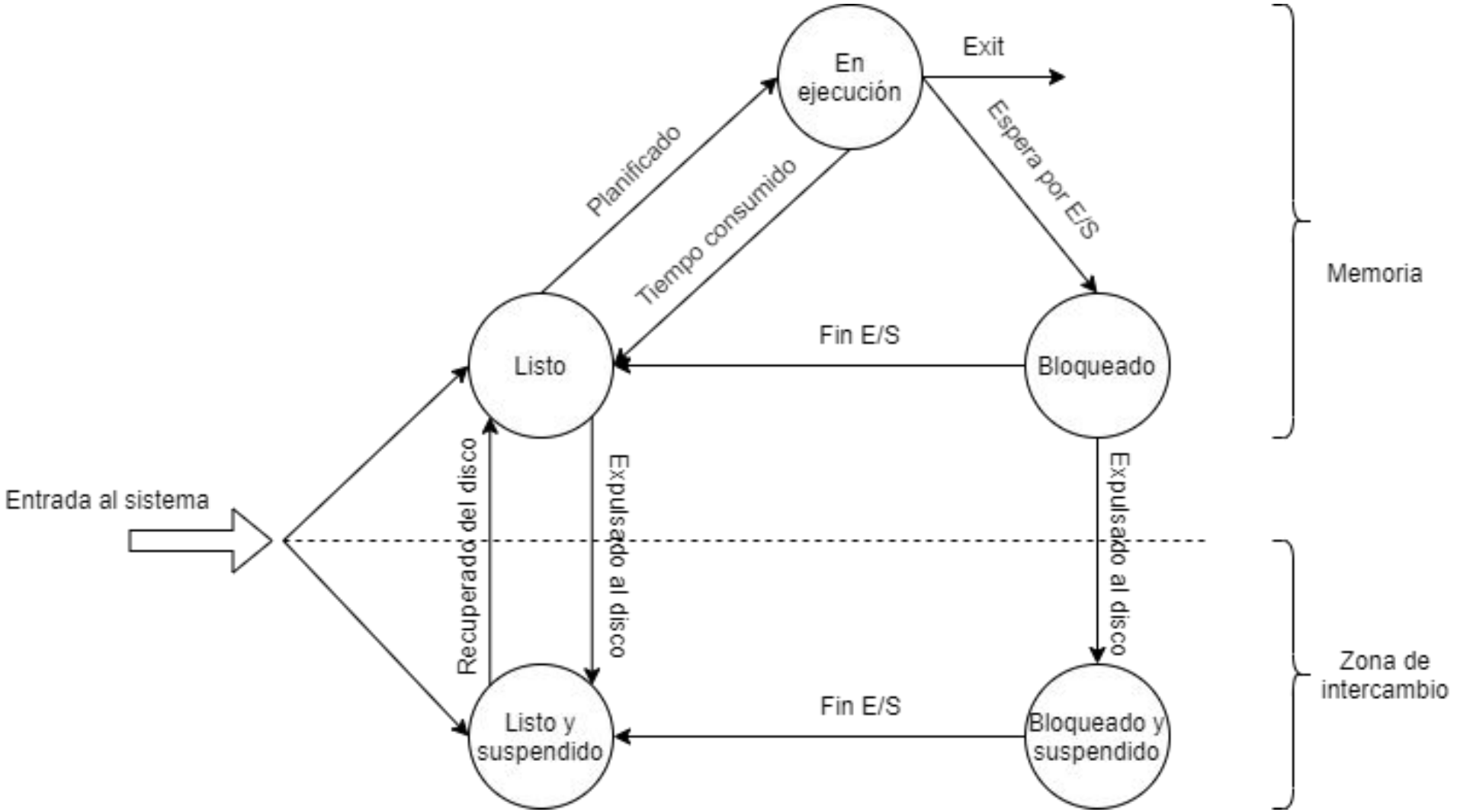
Solo un proceso puede estar en ejecución en cualquier núcleo de procesador a la vez.



# Estados y transiciones

- Nuevo
  - Proceso creado (cargado en memoria)
- Ejecución
  - Instrucciones siendo ejecutadas
- Espera/bloqueado
  - El proceso está a la espera de que ocurra algo
  - Que se complete una solicitud de I/O
  - Que se reciba una señal
- Listo
  - El proceso está a la espera de que se le asigne tiempo de procesador
- Finalizado
  - El proceso completó su ejecución

# Estados y transiciones




- Estado de suspensión: retira los marcos de página del proceso y los envía al área de ***swapping***.
- Liberar memoria para los procesos no suspendidos

# Bloque de control de proceso (BCP/PCB)

- Se requiere una estructura de control y representación de la información de un proceso
  - Control de ejecución
  - Tiempos asignados
  - Recursos usados
  - Tiempos de espera
- El despachador del planificador requiere cierta información del proceso
  - Asignar nuevamente tiempo de procesador
  - Hacer transiciones hacia otros estados

# Bloque de control de proceso

- Registra la «vida» de un proceso en el sistema

| Estado del proceso |
| --- |
| Número de proceso (PID) |
| Contador de programa (IP: Instruction pointer) |
| Registros de CPU |
| Límites de memoria |
| Archivos abiertos |
| ... |

# Bloque de control de proceso

- Estado del proceso
  - Nuevo, listo, en ejecución, en espera, terminado, etc.
- Contador del programa
  - Dirección de la siguiente instrucción a ejecutarse
- Registros de CPU
  - Todos los registros que se requieran para interrumpir y ejecutar nuevamente el proceso.
- Información del planificador
  - Prioridad, apuntadores a las colas de planificación, entre otros

# Bloque de control de proceso

- Información de administración de memoria
  - Valores de límites de memoria, registros base, tablas de páginas, etc.
- Información de contabilidad
  - Cantidad de tiempo en CPU, límites de CPU, etc.
- Información de estado de I/O
  - Dispositivos de I/O asignados al proceso
  - Archivos abiertos
- Otra información
  - El PCB es dependiente de las consideraciones de diseño de cada S.O
  - No es una estructura estándar.

# PCB en xv6 [1/2]

```
// Registros del proceso
struct context {
    int eip;
    int esp;
    int ebx;
    int ecx;
    int edx;
    int esi;
    int edi;
    int ebp;
};
```

```
// Estado del proceso
enum proc_state {
    UNUSED,
    EMBRYO,
    SLEEPING,
    RUNNABLE,
    RUNNING,
    ZOMBIE
};
```

# PCB en xv6 [2/2]

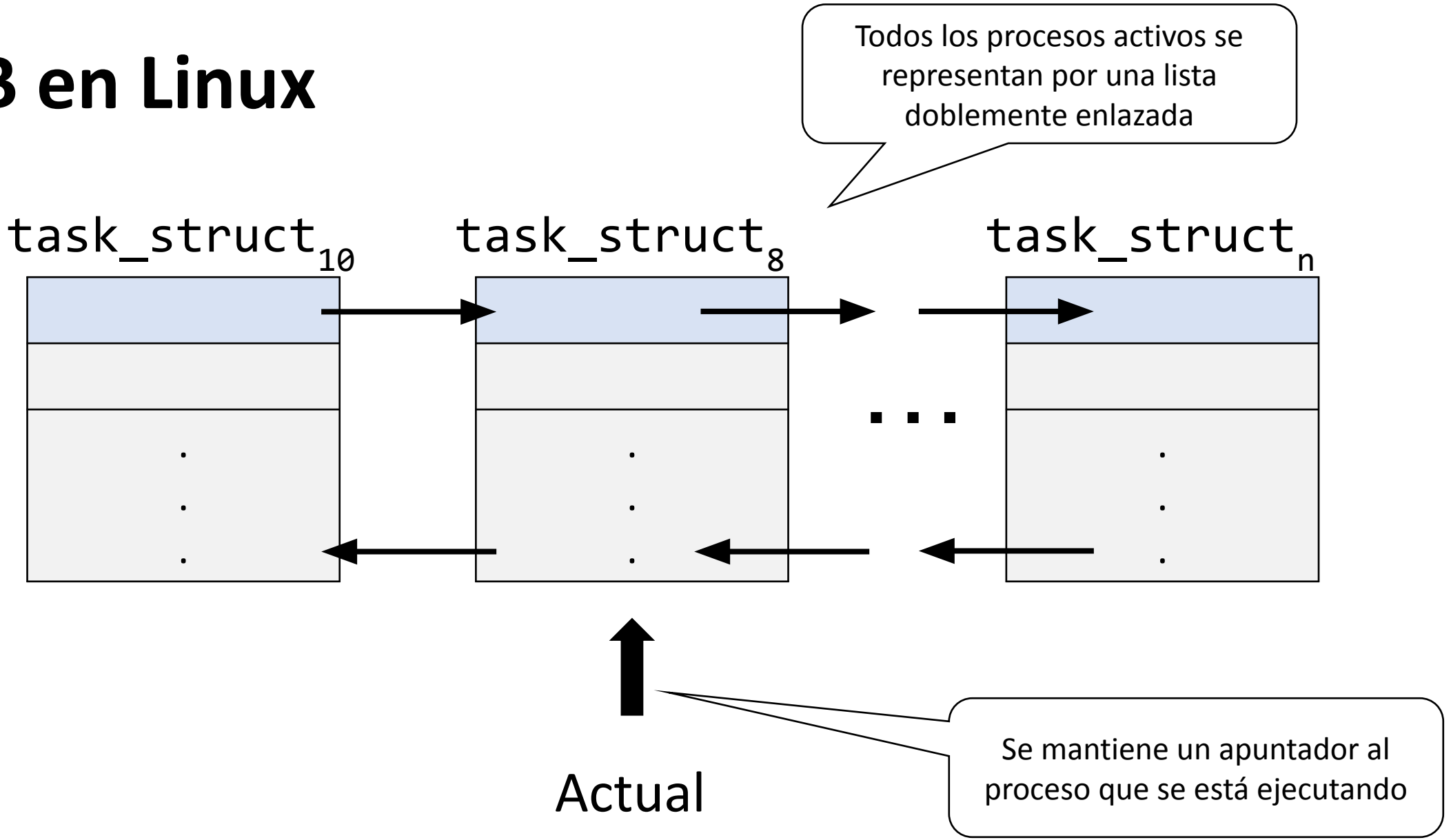
```
struct proc {
    char *mem;                  // Dónde inicial el proceso en memoria
    uint sz;                    // Tamaño del proceso en memoria
    char *kstack;               // Parte inferior pila Kernel para proceso
    enum proc_state state;      // Estado del proceso
    int pid;                    // ID del proceso
    struct proc *parent;        // Proceso padre
    void *chan;                 // Si !=0, proceso durmiendo
    int killed;                 // Si !=0 proceso killed
    struct file                 // Archivos abiertos
*ofile[NOFILE];                 // Directorio actual
    struct inode *cwd;          // Contexto del proceso
    struct context context;     // Para manejo de interrupciones
    struct trapframe *tf;
};
```

# PCB en Linux

```
long state;                          // Estado del proceso
struct sched_entity se;              // Información del planificador
struct task struct *parent;          // Proceso padre de este proceso
struct list head children;           // Hijos de este proceso
struct files struct *files;          // Lista de archivos abiertos
struct mm_struct *mm;                // Espacio de direcciones en RAM
```

- El PCB en Linux está representador por la estructura **`task_struct`**.
- **Algunos** miembros de esta estructura son los indicados en el código anterior.
- La estructura **`task_struct`** está definida en **`<include/linux/sched.h>`**

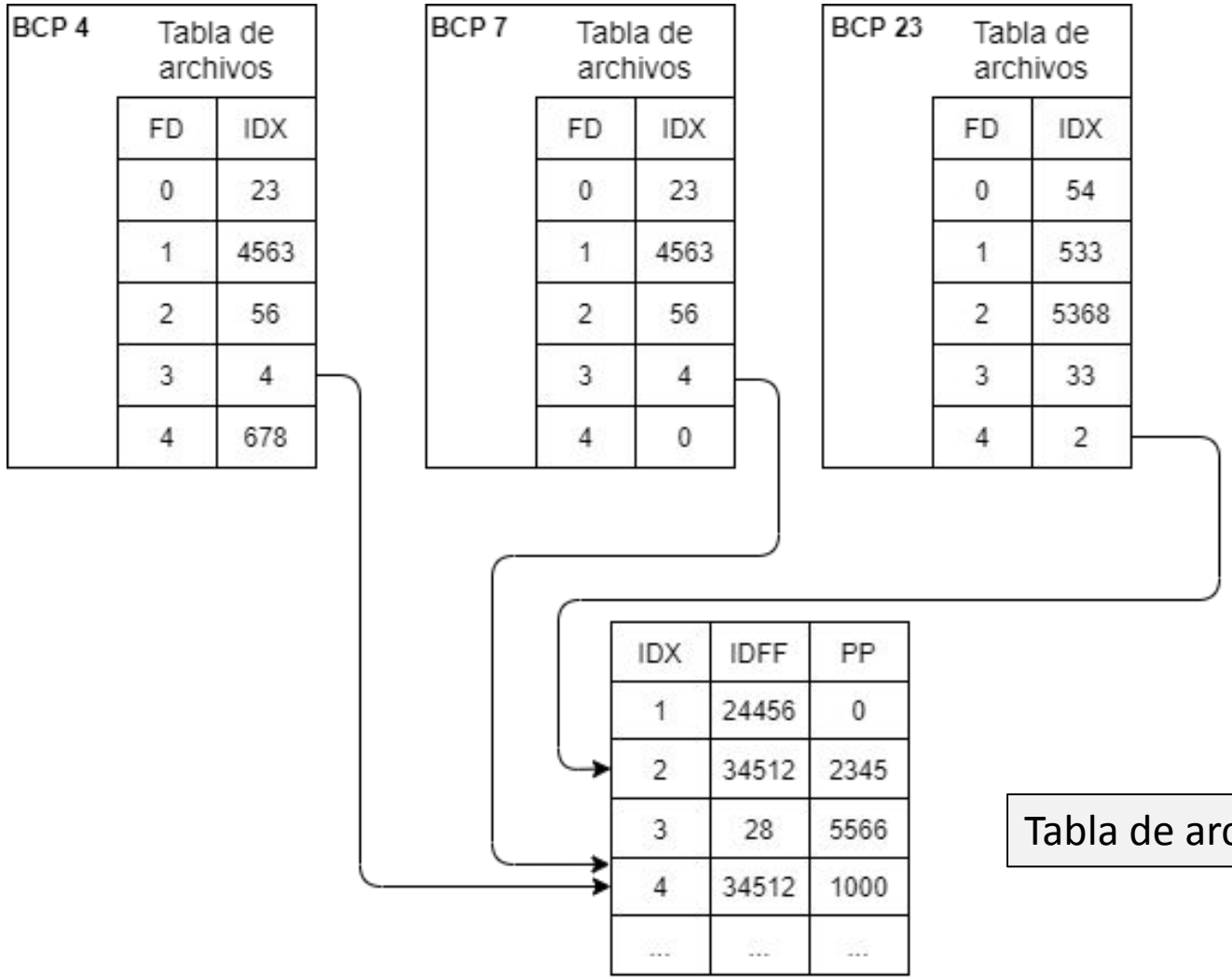
# PCB en Linux


Todos los procesos activos se representan por una lista doblemente enlazada
task_struct10
task_struct8
task_structn
Actual
Se mantiene un apuntador al proceso que se está ejecutando

# PCB e información compartida

- La información compartida por procesos no puede residir en el PCB
  - El PCB es único a cada proceso.
  - Información restringida de cada proceso.
- El PCB debe incluir un apuntador a la información compartida
- Considere el caso de dos procesos que abrieron el mismo archivo
  - Archivo se heredó de proceso padre.
  - Archivo se abrió de manera independiente por los dos procesos
  - Se debe compartir el puntero de posición
  - Puntero de posición no debe estar en el BCP

# PCB/BCP e información compartida




- Cada apertura del archivo genera una nueva entrada en la tabla de archivos externa.
- Cada proceso tiene su PP puntero de posición en el archivo.
- BCP 7 es hijo de BCP 4, están compartiendo el mismo PP.
- El PP no puede estar en el BCP: considere casos de BCP 4 y BCP 7

Tabla de archivos externa al PCB/BCP

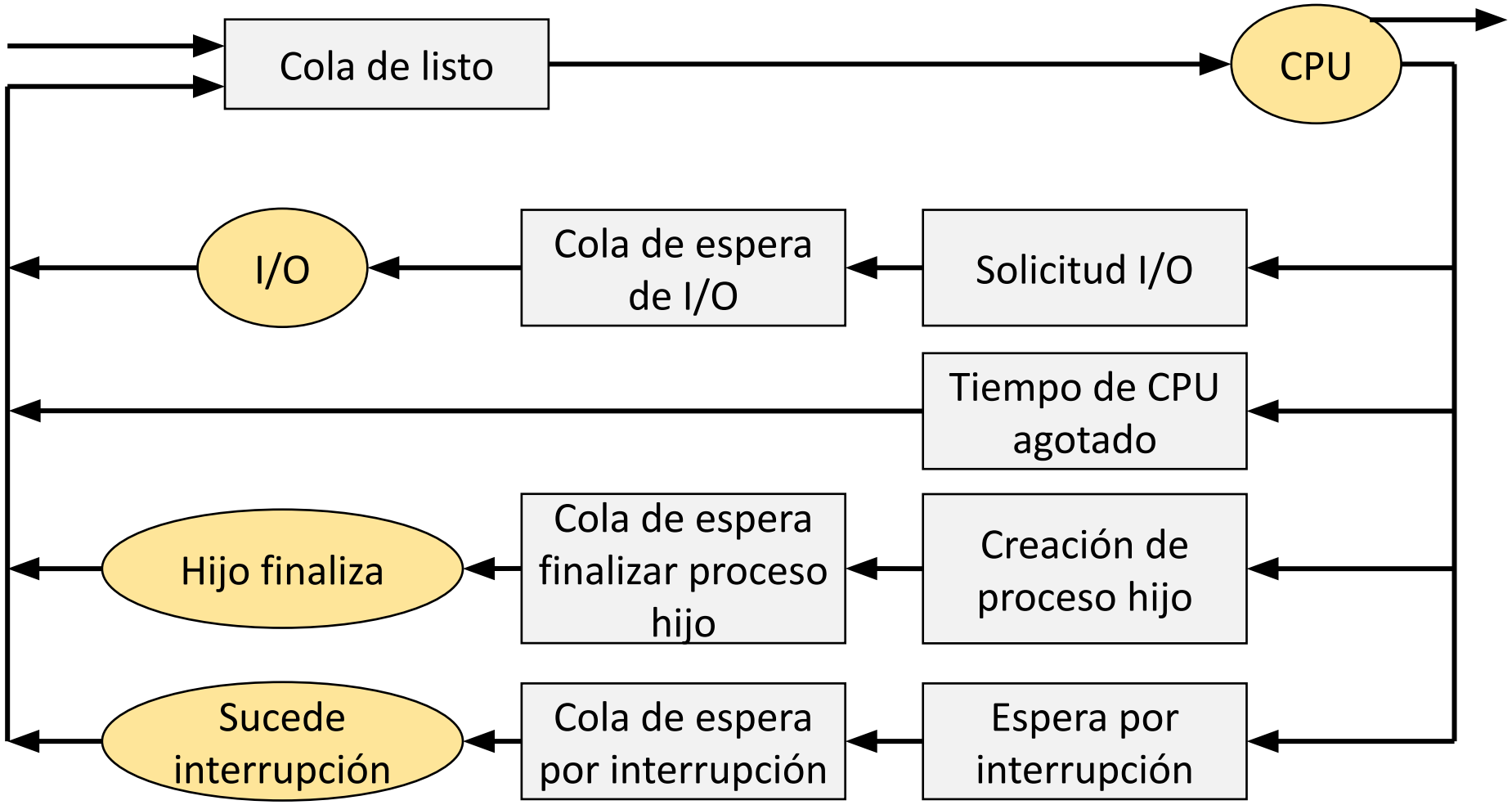
# Planificación de procesos

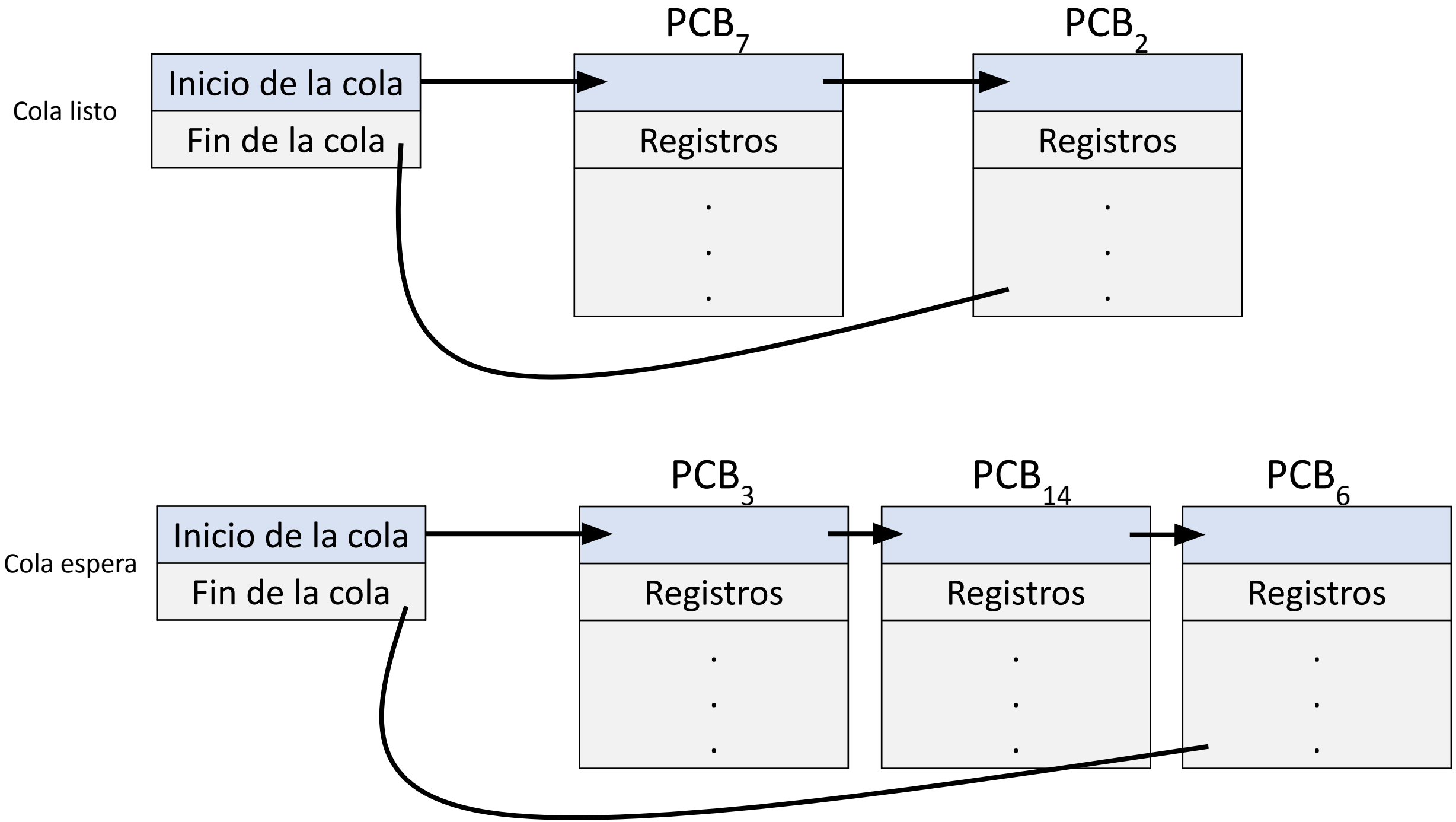
- Objetivos
  - Maximizar el uso del procesador
  - Compartir en el tiempo el uso del procesador por todos los procesos que lo requieran
    - De la mejor manera posible
    - Ningún proceso debe monopolizar el tiempo de procesador
- Cada núcleo de procesador puede ejecutar un proceso a la vez
  - Sistemas multinúcleo pueden ejecutar más de un proceso a la vez
- Grado de multi programación
  - Número de procesos actualmente en memoria

# Colas de planificación de procesos

- Estado de **listo** es una cola
  - Proceso queda a la espera de que se le asigne tiempo de procesador
  - Usualmente se implementa como una lista enlazada (ligada)
  - Encabezado de la cola apunta al primer PCB en la lista
  - Cada PCB incluye un apuntador al siguiente PCB en la lista
- El S.O implementa varias colas para manejar los diferentes estados de un proceso

# Colas: una de listo y tres de espera




Cola listo
Inicio de la cola
Fin de la cola
PCB7
Registros
PCB2
Registros
Cola espera
Inicio de la cola
Fin de la cola
PCB3
Registros
PCB14
Registros
PCB6
Registros

# Colas: una de listo y tres de espera

- Los círculos/óvalos indican los recursos que sirven a las colas
- Flechas indican el flujo de un proceso en el sistema
- Eventos que pueden suceder cuando se ejecuta un proceso
  - Hace una solicitud de I/O. P. Ej.: la lectura de un archivo  Cola de I/O.
  - Crea un proceso hijo.  Cola hasta que hijo termine.
  - Sale de CPU por tiempo agotado o por alguna interrupción del proceso.
- Tres colas de espera para representar el estado de espera o bloqueado de un proceso

# Planificación de CPU

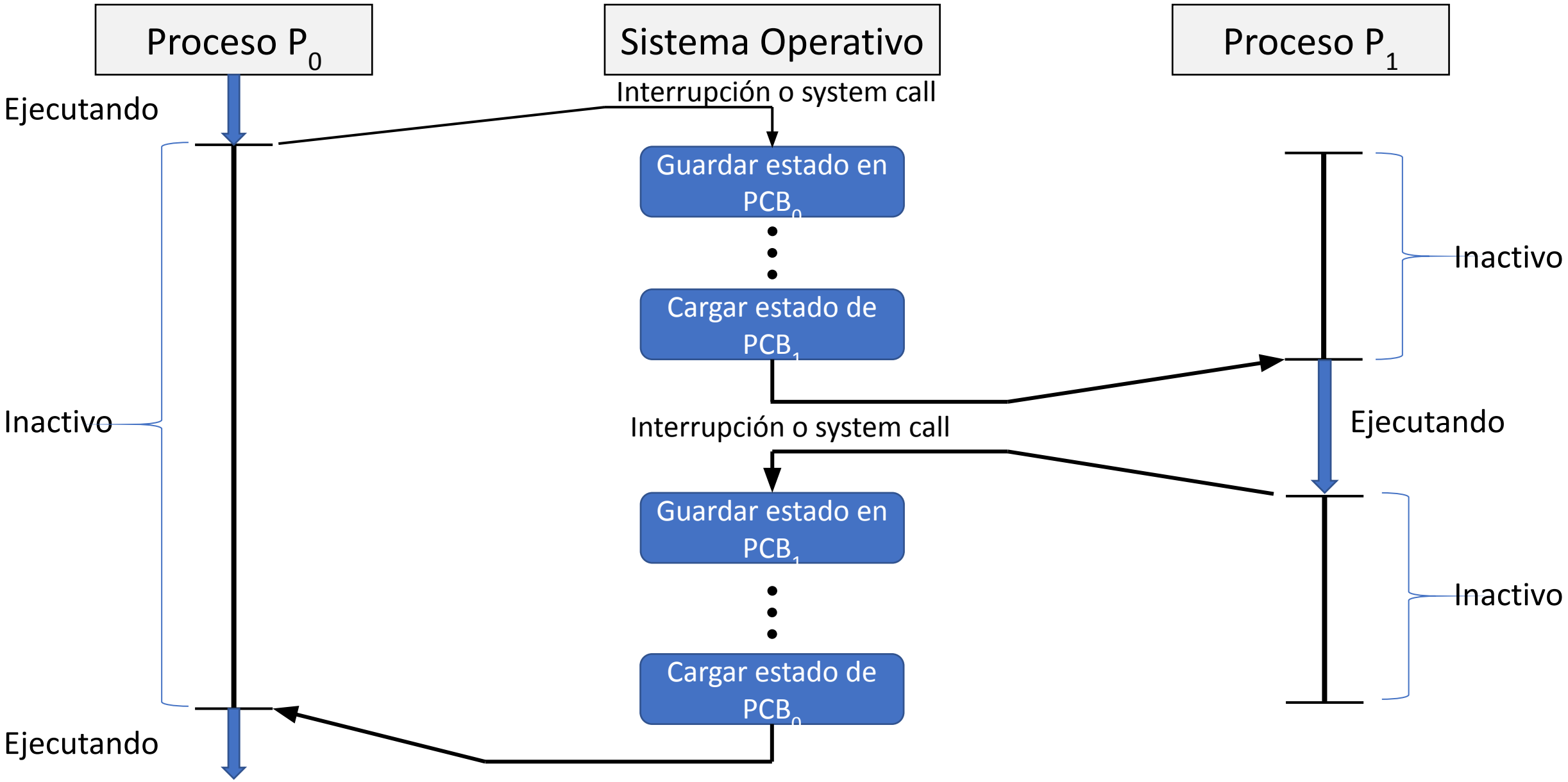
- Objetivos del planificador
  - Seleccionar un proceso (entre varios) para ejecutarse
  - De la mejor manera posible
- Procesos intensivos de I/O
  - Se ejecutan unos milisegundos antes de quedar en espera por I/O
- Procesos intensivos de CPU
  - Requieren más tiempo de uso de CPU
- Ningún proceso puede monopolizar el tiempo de CPU

# Cambios de contexto

- Cuando ocurre una interrupción (p. ej.: I/O) el S.O debe sacar de CPU al proceso en ejecución
  - Para atender la interrupción
  - El S.O ejecuta código que atiende la interrupción
  - Debe ejecutar otro proceso distinto en la CPU: Un proceso propio del S.O
- Se debe guardar el contexto del proceso interrumpido
  - Proceso que se estaba ejecutando antes de la interrupción
  - Para seguir ejecutándolo cuando se le asigne nuevamente tiempo de CPU

# Cambios de contexto

- El contexto de un proceso está representado en el PCB
- El cambio de contexto implica al núcleo del S.O
  - Guardar el contexto del proceso interrumpido en el PCB
  - Cargar el contexto del proceso que será ejecutado
  - Recordar que cada vez que se crea un proceso se crea el PCB del proceso
- El cambio de contexto es un gasto necesario (*overhead*).
  - El sistema no está haciendo nada útil mientras realiza el cambio de contexto
    - CPU no se usa en este cambio de contexto
  - Unos cuantos nano/**micro**/mili/segundos
    - Velocidad de la memoria RAM, número de registros, instrucciones especiales


Proceso $P_0$
Sistema Operativo
Proceso $P_1$
Ejecutando
Interrupción o system call
Guardar estado en $PCB_0$
Cargar estado de $PCB_1$
Inactivo
Inactivo
Interrupción o system call
Ejecutando
Guardar estado en $PCB_1$
Inactivo
Cargar estado de $PCB_0$
Ejecutando

# Llamadas al sistema y cambios de contexto

- Consideremos la siguiente llamada al sistema

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

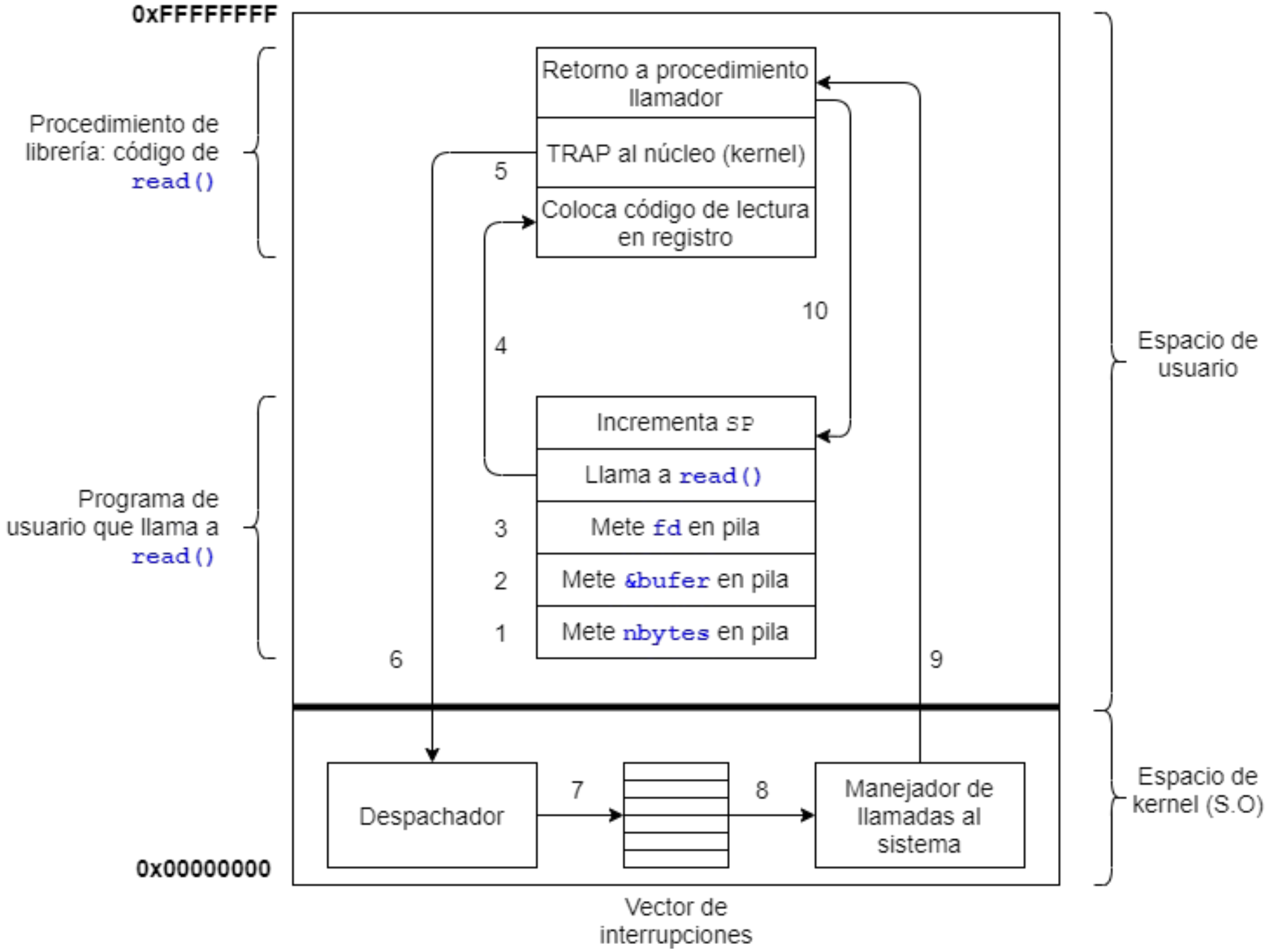
- Donde,
  - `<unistd.h>` archivo de cabecera que debe incluirse para hacer la llamada
  - `ssize_t` tipo de dato que retorna: entero con signo (POSIX.1.)
  - `read` nombre de la llamada
  - `int fd` manejador (*handle*) de un archivo abierto
  - `void *buf` apuntador al búfer para recibir los datos leídos
  - `size_t count` número de bytes a leer.
- La llamada `read()` retorna el número de bytes leídos

# Llamadas al sistema y cambios de contexto

- Programa llamador mete parámetros en pila en orden inverso.
  - Primer y tercer parámetro por valor
  - Segundo parámetro por referencia
- Se hace llamada a `read()`
- Código de `read()` coloca en registro código de la llamada al sistema
  - El S.O espera que el código esté en ese registro
- Código de `read()` ejecuta instrucción TRAP
  - Interrupción por software
- Se ejecuta código del núcleo del S.O que examina número de llamada

# Llamadas al sistema y cambios de contexto

- Pasa control a manejador de llamadas al sistema
  - En el núcleo del S.O.
  - Atiende llamada
- Se devuelve control al código de `read()` posterior a la instrucción TRAP
- Se devuelve control al programa llamador a la instrucción posterior a `read()`
- Programa llamador limpia la pila
  - Incrementa el apuntador de pila


0xFFFFFFFF
Procedimiento de librería: código de read()
Programa de usuario que llama a read()
Retorno a procedimiento llamador
TRAP al núcleo (kernel)
Coloca código de lectura en registro
Incrementa SP
Llama a read()
Mete fd en pila
Mete &bufer en pila
Mete nbytes en pila
1
2
3
4
5
6
7
8
9
10
Despachador
Manejador de llamadas al sistema
Espacio de usuario
Espacio de kernel (S.O)
0x00000000
Vector de interrupciones

# Llamadas al sistema y cambios de contexto

- ¿Cuántos cambios de contexto hubo en el caso anterior?
  - Una llamada al sistema no siempre produce un cambio de contexto a menos que la llamada sea bloqueante.
  - Si la llamada es bloqueante se aprovecha el tiempo para darle tiempo de procesador a otro proceso.

# Vector de interrupciones

- Cada entrada del vector contiene
  - La dirección de la rutina de tratamiento de esa interrupción
  - El vector está indexado por el número de interrupción
- Se transfiere el control (se ejecuta) las instrucciones a las que apunta la entrada en el vector

# Sistemas operativos monotarea

- **Monotarea o monoproceso**
  - Solo existe un proceso a cada instante en el procesador
  - El segundo proceso no puede ejecutarse hasta que el primero halla finalizado completamente
  - La memoria RAM la ocupa el S.O y el proceso en ejecución.
  - MS-DOS
- **Multitarea o multiproceso**
  - Permite la coexistencia de varios procesos activos a la vez
  - El S.O (planificador y despachador) selecciona un proceso para ejecutarse
  - Se requiere **cambio de contexto** entre proceso y proceso.

# Referencias


- Carretero Pérez, J., García Carballeira, F., de Miguel Anasagasti, P., & Pérez Costoya, F. (2001). Procesos. In *Sistemas operativos. Una Visión Aplicada* (pp. 77–160). McGraw Hill.
- Silberschatz, A., Baer Galvin, P., & Gagne, G. (2018). Process Management. In *Operating Systems Concepts* (10th ed., pp. 105–115). John Wiley & Sons, Inc.
- Tanenbaum, A. S. (2009). Introducción. In *Sistemas Operativos Modernos* (3rd ed., pp. 3–75). Pearson Educación.