

# Procesos

**Stallings 5ta ed. capítulo 3.**  
**Silberschatz 7ma ed. capítulo 3.**

# Clase Anterior

## Introducción Hardware:

- Componente de la computadora.
- Registros del procesador.
- Ciclo de instrucción.
- Interrupciones.

# Clase Anterior

## **Introducción Hardware:**

- Componente de la computadora.
- Registros del procesador.
- Ciclo de instrucción.
- Interrupciones.

## **Introducción Sistema Operativo:**

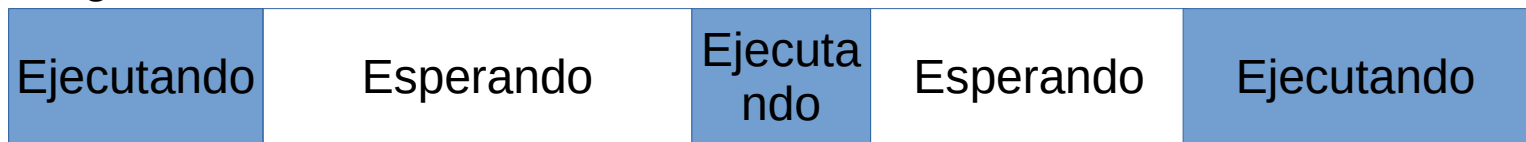
- Funciones del sistema operativo.
- Cambios de Modo.
- Llamadas al sistema (Syscall).

# Procesos

## Introducción

- Sistemas Monoprogramados

Programa 1:

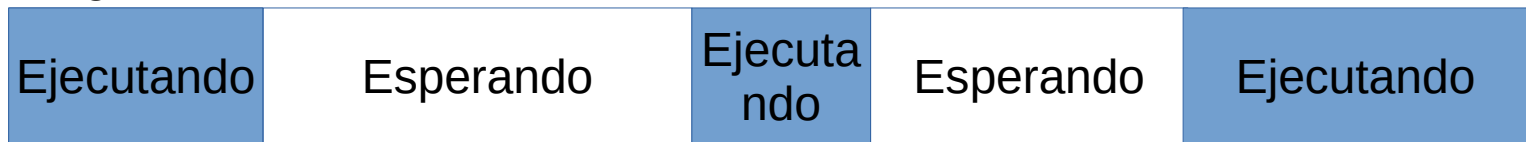


# Procesos

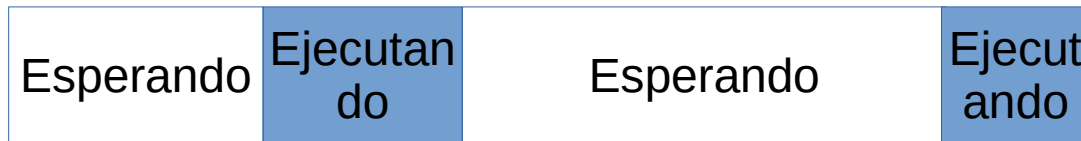
## Introducción

- Sistemas Multiprogramados

Programa 1:



Programa 2:



—————→  
Tiempo

# Procesos

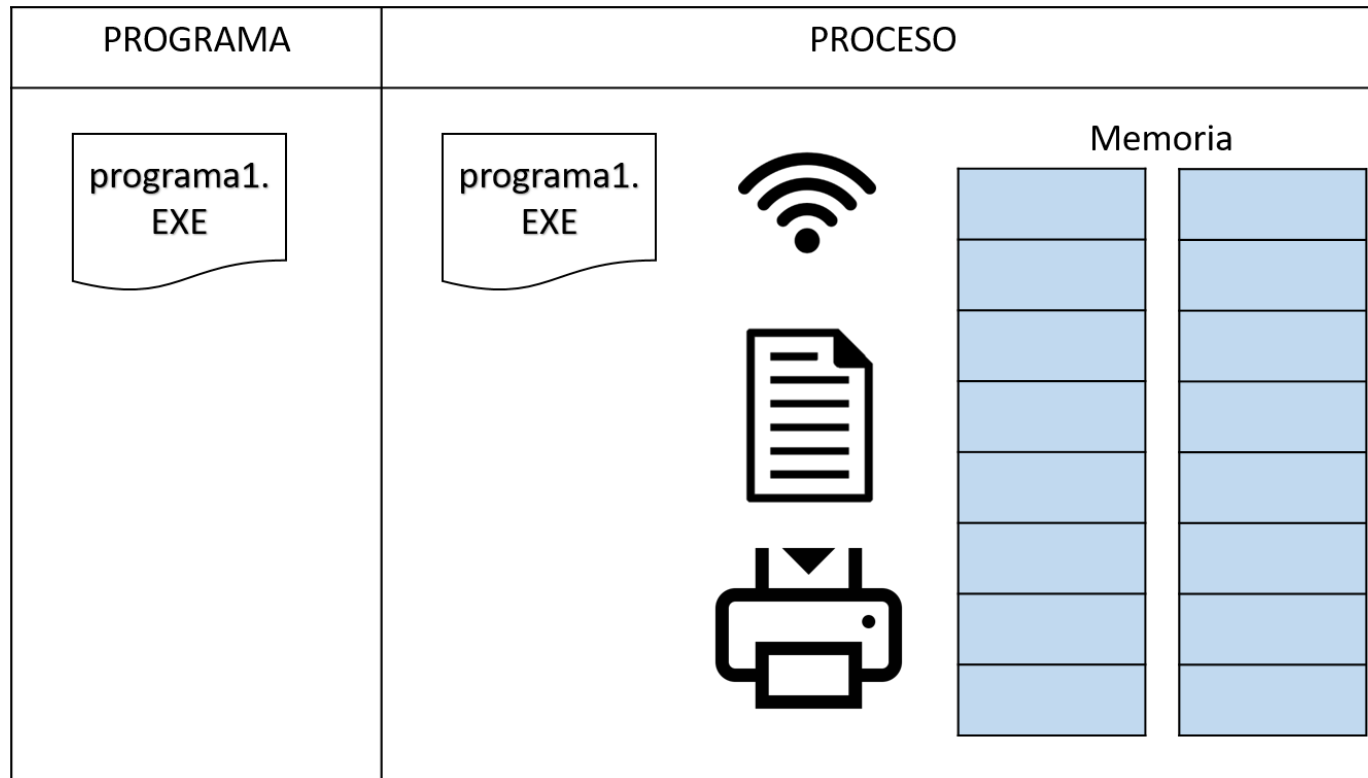
## Definiciones previas

- PROGRAMA: Secuencia de instrucciones compiladas a código máquina.
- EJECUCIÓN CONCURRENTES: Dos o más programas ejecutando en el mismo intervalo de tiempo.
- MULTIPROGRAMACIÓN: Dos o más procesos ejecutando de forma concurrente.
- MULTIPROCESAMIENTO: no es lo mismo que multiprogramación.

# Procesos

## Proceso

Secuencia de instrucciones (un programa) que están siendo ejecutadas y... muchas cosas más.



# Procesos

## Proceso

Estructura de un proceso:

- CÓDIGO
- DATOS
- PILA
- HEAP (CÚMULO)



# Procesos

## Proceso

### Estructura de un proceso:

- **CÓDIGO**: Espacio asignado para almacenar la secuencia de instrucciones del programa.

# Procesos

## Proceso

### Estructura de un proceso:

- **CÓDIGO**: Espacio asignado para almacenar la secuencia de instrucciones del programa.

```
int VARIABLEGLOBAL = 8;
int main () {

    int x = 2, y = 3, z;

    char *p;
    z = sumar(x ,y) ;
    printf ("Resultado:%d\n", z) ;

    p = malloc(5) ;
    free(p) ;

    Return 0;      }
```

```
push %rbp
mov  %rsp,%rbp
sub  $0x10,%rsp
movl $0x5,-0x4(%rbp)
movl $0x8,-0x8(%rbp)
movl $0x0,-0xc(%rbp)
mov  -0x4(%rbp),%edx
mov  -0x8(%rbp),%eax
add  %edx,%eax
mov  %eax,-0xc(%rbp)
mov  -0xc(%rbp),%eax
mov  %eax,%esi
mov  $0x4005d4,%edi
mov  $0x0,%eax
callq 4003e0 <printf@plt>
mov  $0x0,%eax
leaveq
retq
nopl 0x0(%rax)
```

# Procesos

## Proceso

### Estructura de un proceso:

- **DATOS**: Espacio asignado para almacenar variables globales.

```
int VARIABLEGLOBAL = 8;  
int main () {  
  
    int x = 2, y = 3, z;  
  
    char *p;  
    z = sumar(x ,y) ;  
    printf ("Resultado:%d\n", z) ;  
  
    p = malloc(5) ;  
    free(p) ;  
  
    Return 0;    }
```

0xAB00

8



# Procesos

## Proceso

### Estructura de un proceso:

- **PILA (STACK)**: Espacio usado para llamadas a función, parámetros y variables locales.

```
int VARIABLEGLOBAL = 8;
int main () {

    int x = 2, y = 3, z;

    char *p;
    z = sumar(x ,y) ;
    printf ("Resultado:%d\n", z) ;

    p = malloc(5) ;
    free(p) ;

    Return 0;      }
```

# Procesos

## Proceso

### Estructura de un proceso:

- **PILA (STACK)**: Espacio usado para llamadas a función, parámetros y variables locales.

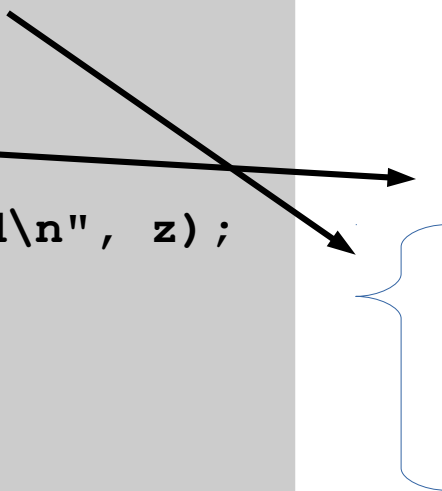
```
int VARIABLEGLOBAL = 8;
int main () {

    int x = 2, y = 3, z;

    char *p;
    z = sumar(x ,y);
    printf ("Resultado:%d\n", z);

    p = malloc(5);
    free(p);

    Return 0;    }
```



The diagram illustrates the stack structure. It shows a vertical stack of memory cells. The top four cells are empty. The fifth cell contains the address 0x0D00. The bottom three cells contain the values 5, 3, and 2. Arrows from the code point to these cells: one from 'z' to 5, one from 'p' to 3, and one from 'x' to 2. A blue bracket groups the bottom three cells.

0x00C4	0x0D00
0x00C3	5
0x00C2	3
0x00C1	2

# Procesos

## Proceso

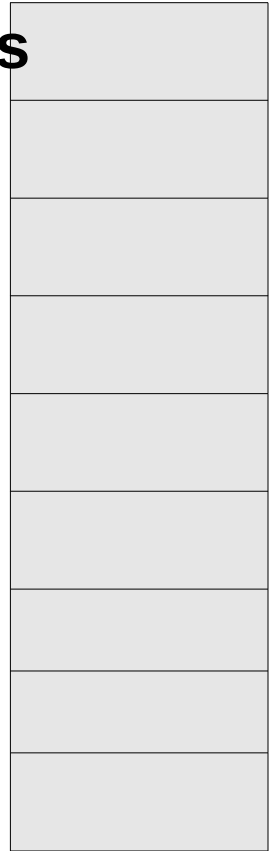
### Estructura de un proceso:

- **PILA (STACK)**: Espacio usado para llamadas a función, parámetros y variables locales.

```
int sumar (int x, int y)  {
    int z = 0;
    z = x + y;
    return z;              }

int main ()               {
    int a = 10, b = 20, c = 0;
    c = sumar (a, b);
    printf ("Resultado: %d\n", c);

    Return 0;    }
```



# Procesos

## Proceso

### Estructura de un proceso:

- **PILA (STACK)**: Espacio usado para llamadas a función, parámetros y variables locales.

```
int sumar (int x, int y)  {  
    int z = 0;  
    z = x + y;  
    return z;             }  
  
int main ()               {  
    → int a = 10, b = 20, c = 0;  
    c = sumar (a, b);  
    printf ("Resultado: %d\n", c);  
  
    Return 0;    }  

```

c	0
b	20
a	10

Retorno	0x123
c	0
b	20
a	10




# Procesos

## Proceso

### Estructura de un proceso:

- **PILA (STACK)**: Espacio usado para llamadas a función, parámetros y variables locales.

```
int sumar (int x, int y)  {
     int z = 5;
    z = x + y;
    return z;
}

int main ()
{
    int a = 10, b = 20, c = 0;
    c = sumar (a, b); // 0x123
    printf ("Resultado: %d\n", c);

    Return 0;
}
```


z	5
y	20
x	10
Retorno	0x123
c	0
b	20
a	10

# Procesos

## Proceso

### Estructura de un proceso:

- **PILA (STACK)**: Espacio usado para llamadas a función, parámetros y variables locales.

```
int sumar (int x, int y)  {
    int z = 5;
    z = x + y;
     return z;
}

int main ()      {
    int a = 10, b = 20, c = 0;
    c = sumar (a, b); // 0x123
    printf ("Resultado: %d\n", c);

    Return 0;    }
```

z	30
y	20
x	10
Retorno	0x123
c	0
b	20
a	10

# Procesos

## Proceso

### Estructura de un proceso:

- **PILA (STACK)**: Espacio usado para llamadas a función, parámetros y variables locales.

```
int sumar (int x, int y)  {
    int z = 5;
    z = x + y;
    return z;
}

int main ()
{
    int a = 10, b = 20, c = 0;
    → c = sumar (a, b); // 0x123
    printf ("Resultado: %d\n", c);

    Return 0;  }
```

z	30
y	20
x	10
Retorno	0x123
c	0
b	20
a	10

# Procesos

## Proceso

### Estructura de un proceso:

- **PILA (STACK)**: Espacio usado para llamadas a función, parámetros y variables locales.

```
int sumar (int x, int y)  {
    int z = 5;
    z = x + y;
    return z;              }

int main ()                {
    int a = 10, b = 20, c = 0;
    c = sumar (a, b); // 0x123
    → printf ("Resultado: %d\n", c);

    Return 0;    }
```

c	30
b	20
a	10

# Procesos

## Proceso

### Estructura de un proceso:

- HEAP: Espacio asignado para el uso de memoria dinámica.

```
int VARIABLEGLOBAL = 8;
int main () {

    int x = 2, y = 3, z;

    char *p;
    z = sumar(x ,y) ;
    printf ("Resultado:%d\n", z) ;

    p = malloc(5) ;
    free(p) ;

    Return 0;      }
```

# Procesos

## Proceso

### Estructura de un proceso:

- **HEAP**: Espacio asignado para el uso de memoria dinámica.

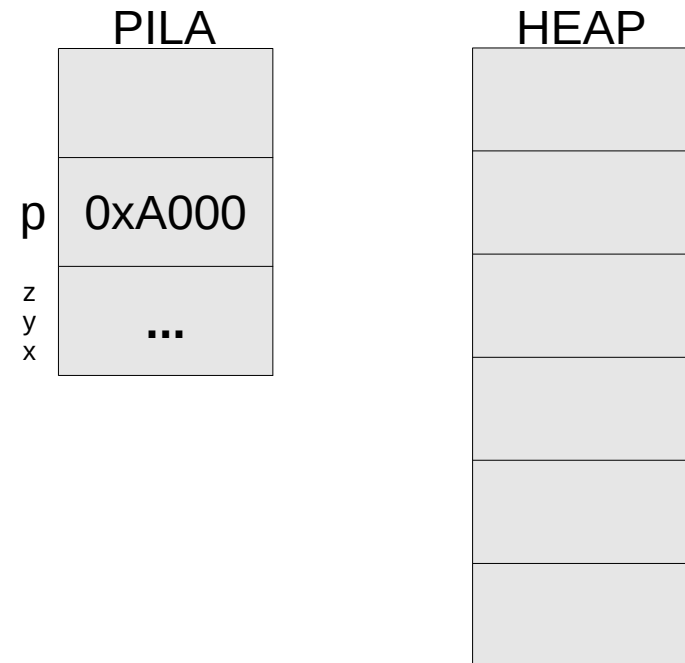
```
int VARIABLEGLOBAL = 8;
int main () {

    int x = 2, y = 3, z;

    char *p;
    z = sumar(x ,y);
    printf ("Resultado:%d\n", z);

    p = malloc(5);
    free(p);

    Return 0;    }
```



# Procesos

## Proceso

### Estructura de un proceso:

- **HEAP**: Espacio asignado para el uso de memoria dinámica.

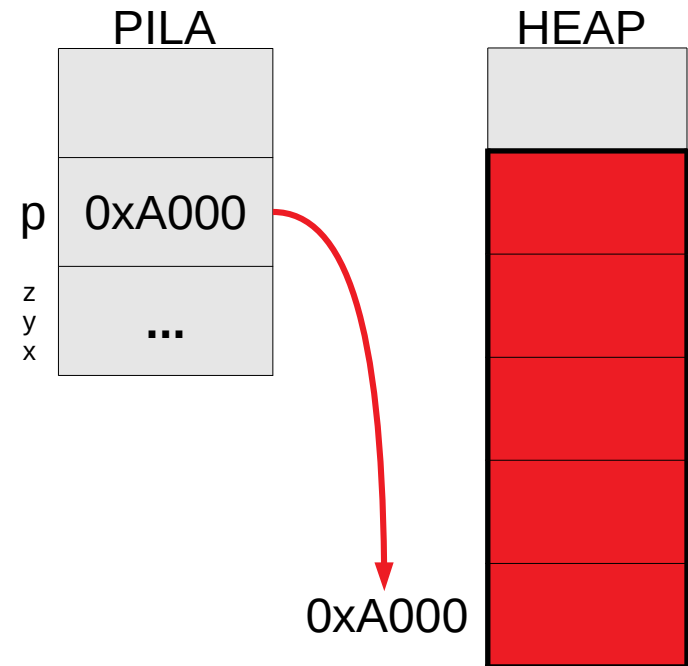
```
int VARIABLEGLOBAL = 8;
int main () {

    int x = 2, y = 3, z;

    char *p;
    z = sumar(x ,y);
    printf ("Resultado:%d\n", z);

    p = malloc(5); ←
    free(p);

    Return 0;    }
```



# Procesos

## Proceso

### Estructura de un proceso:

- HEAP: Espacio asignado para el uso de memoria dinámica.

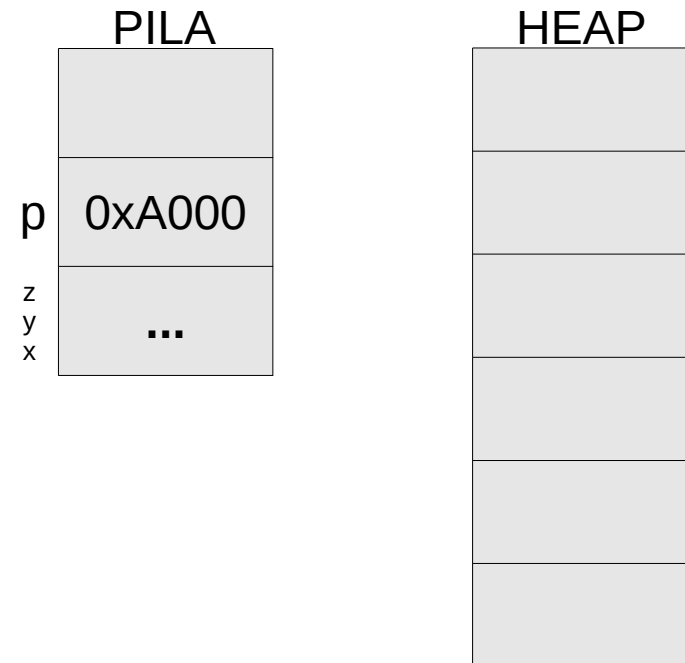
```
int VARIABLEGLOBAL = 8;
int main () {

    int x = 2, y = 3, z;

    char *p;
    z = sumar(x ,y);
    printf ("Resultado:%d\n", z);

    p = malloc(5);
    free(p); ←

    Return 0;    }
```





# Procesos

## Proceso

### Multiprogramación:

PROCESO 1	PROCESO 2
1000	2200
1001	2201
1002	2202
1003	2203
1004	
1005	
1006	
1007	

CPU	
	1000
	1001
	1002
	1003
	1004
SO	1005
	2200
	2201
	2202
SO	2203
	1006
	1007

# Procesos

## Proceso

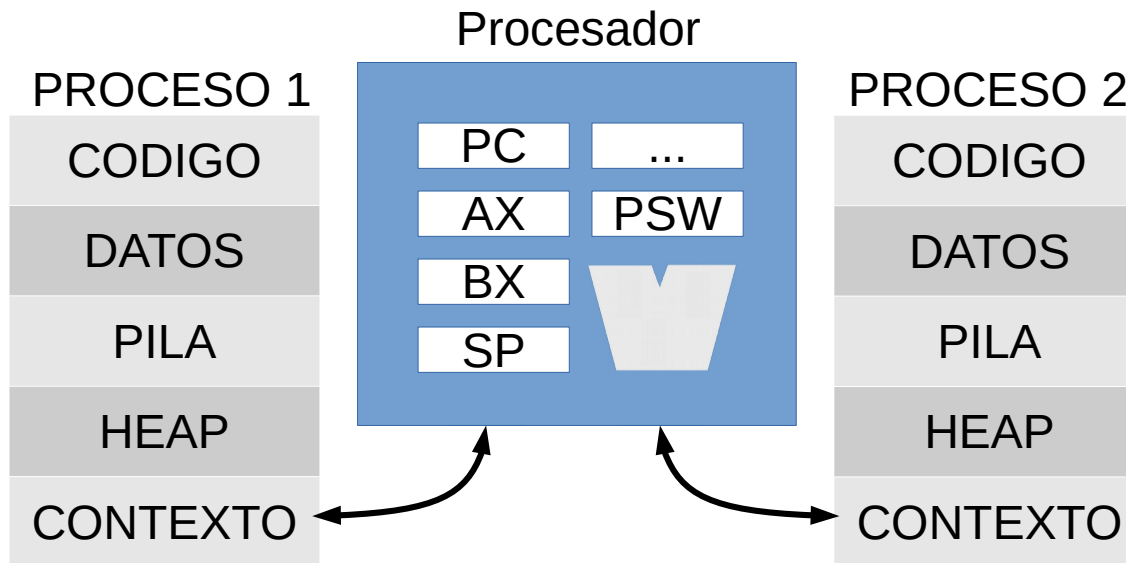
### Contexto de ejecución:

- **Registros del Procesador.**
- **PSW.**

# Procesos

## Proceso

## Contexto de ejecución:



	1000
	1001
	1002
	1003
	1004
SO	1005
<hr/>	
	2200
	2201
	2202
SO	2203
<hr/>	
	1006
	1007

# Procesos

## Proceso

### **Atributos:**

- Identificador: PID / PPID / UID.
- Información de gestión de memoria.
- Información de Planificación.
- Información de E/S.
- Información contable.

# Procesos

## Proceso

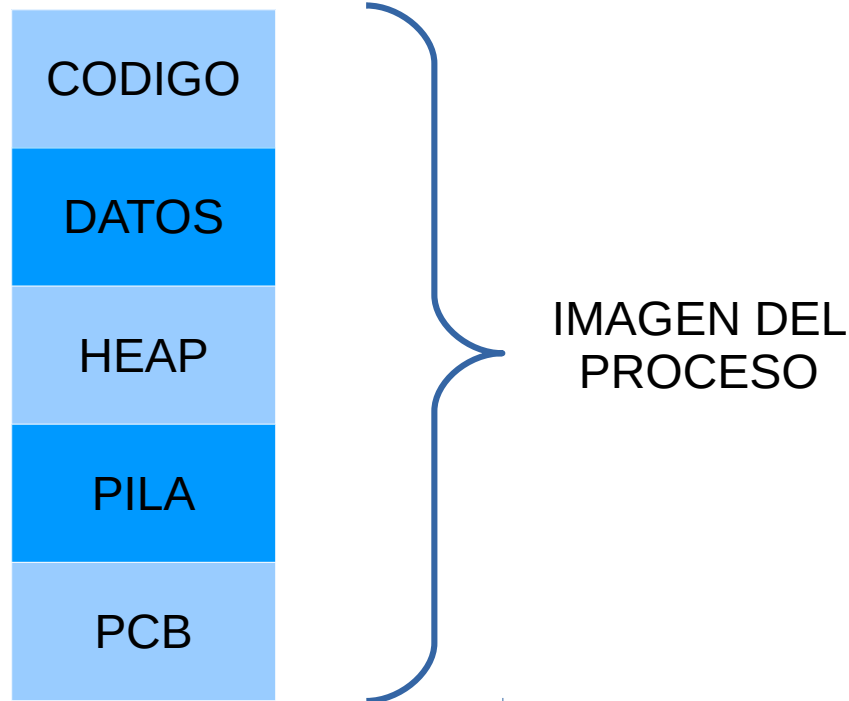
**Bloque de Control de Proceso (PCB): Contiene toda la información relacionada al proceso.**

- Hay uno por cada proceso en el sistema.
- Contiene la dirección de las estructuras del proceso.
- Guarda el Contexto de ejecución cuando el proceso no está ejecutando.
- Atributos.

# Procesos

## Proceso

Imagen del proceso: Representación del proceso en el sistema



# Procesos

## Proceso

### Ciclo de vida de un proceso

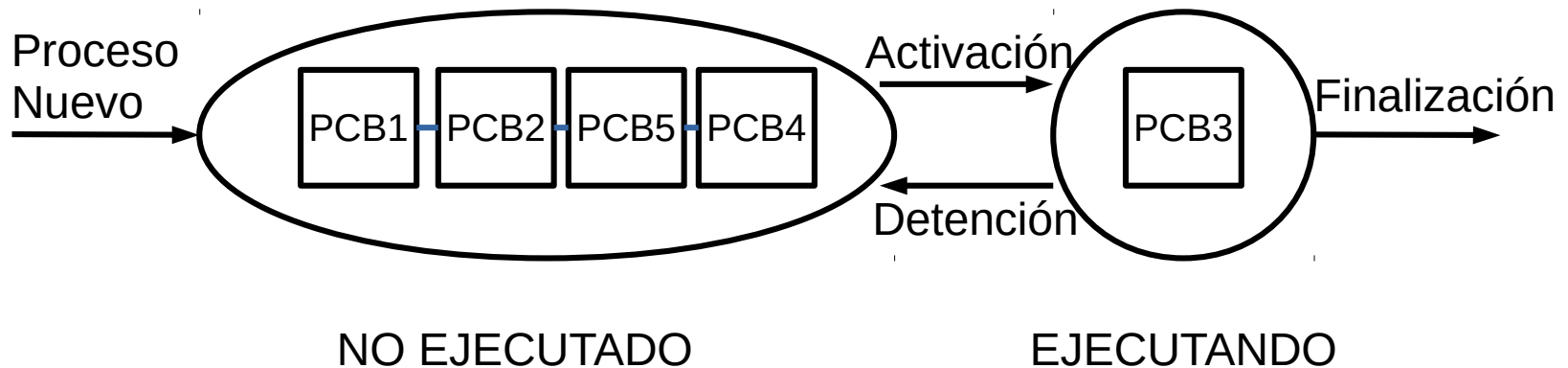
- Tiempo que transcurre entre su creación y su finalización.
- Durante el ciclo de vida pasa por varios estados.

# Procesos

## Proceso

### Ciclo de vida de un proceso

### Estado de los procesos: Diagrama de 2 estados





# Procesos

## Proceso

### Creación de un proceso

- Puede ser creado por el sistema operativo para dar algún servicio.
- Puede ser creado a pedido de otro proceso.
  - El proceso que solicita crearlo se llama proceso padre y al proceso creado se lo llama proceso hijo.
  - Proceso Padre e Hijo pueden ejecutar de forma concurrente o el Padre puede esperar que los hijos finalicen.

# Procesos

## Proceso

### **Creación de un proceso: Pasos**

- Asignación del PID.
- Reservar espacio para estructuras (código, datos, pila y heap).
- Inicializar PCB.
- Ubicar PCB en listas de planificación.

# Procesos

## Proceso

### Creación de un proceso: fork()

```
int id;
id = fork();

if (id == 0) {
/* código proceso hijo */
.....
}

if (id > 0 ) {
/* código proceso padre */
.....
}
```

# Procesos

## Proceso

### Creación de un proceso: fork()

```
int id;  
→ id = fork();  
  
if (id == 0) {  
    /* código proceso hijo */  
    .....  
}  
  
if (id > 0 ) {  
    /* código proceso padre */  
    .....  
}
```

Imagen  
Proceso

CODIGO

DATOS

PILA

HEAP

PCB

PID = 10

PPID = 5

id = ?

# Procesos

## Proceso

### Creación de un proceso: fork()

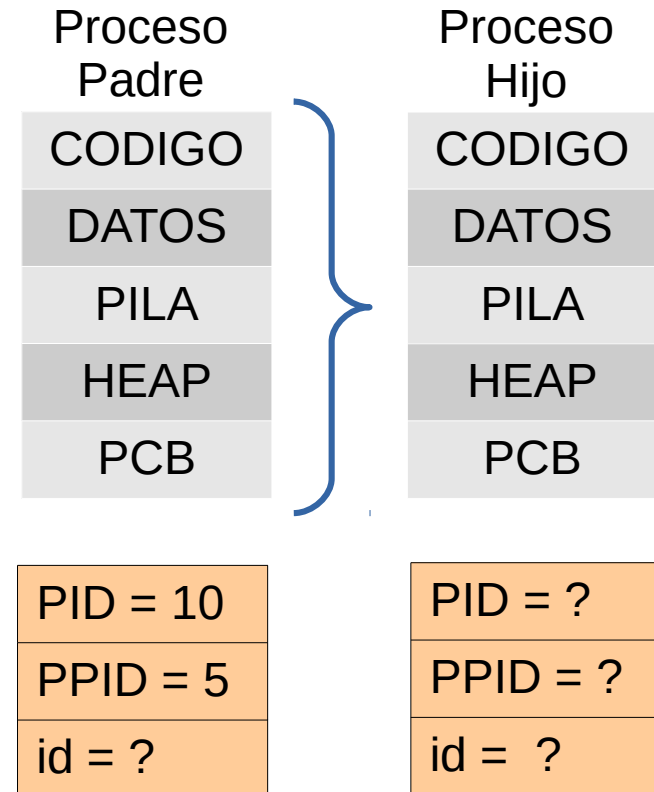
```

int id;
→ id = fork();

if (id == 0) {
    /* código proceso hijo */
    .....
}

if (id > 0 ) {
    /* código proceso padre */
    .....
}

```



# Procesos

## Proceso

### Creación de un proceso: fork()

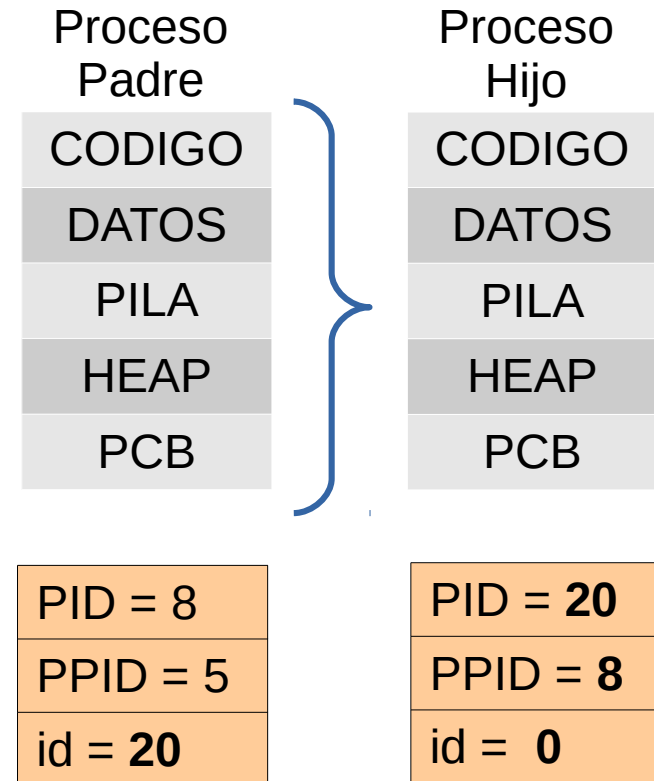
```

int id;
→ id = fork();

if (id == 0) {
    /* código proceso hijo */
    .....
}

if (id > 0 ) {
    /* código proceso padre */
    .....
}

```



# Procesos

## Proceso

### Creación de un proceso: fork()

```
int pid;  
char *programa[] = {"ps", "f", NULL};  
pid = fork();  
  
if (pid == 0) { /* hijo */  
    execv ("/bin/ps", programa);  
}  
  
if (pid > 0 ) { /* padre */  
    wait (NULL);  
    printf("hijo finalizado\n");  
}
```

# Procesos

## Proceso

### Terminación de un proceso

- Terminación normal.
  - `exit(int exit_status) / wait (int *status)`
- Terminado por otro proceso.
  - `kill (pid, sig)`
- Terminado por falla o condición de error.

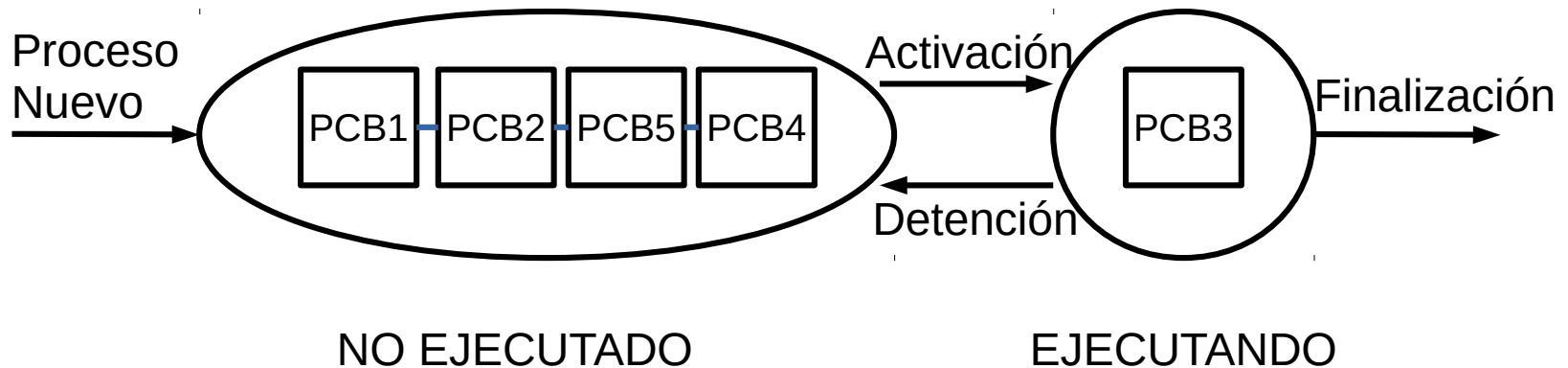


# Procesos

## Proceso

### Ciclo de vida de un proceso

### Estado de los procesos: Diagrama de 2 estados

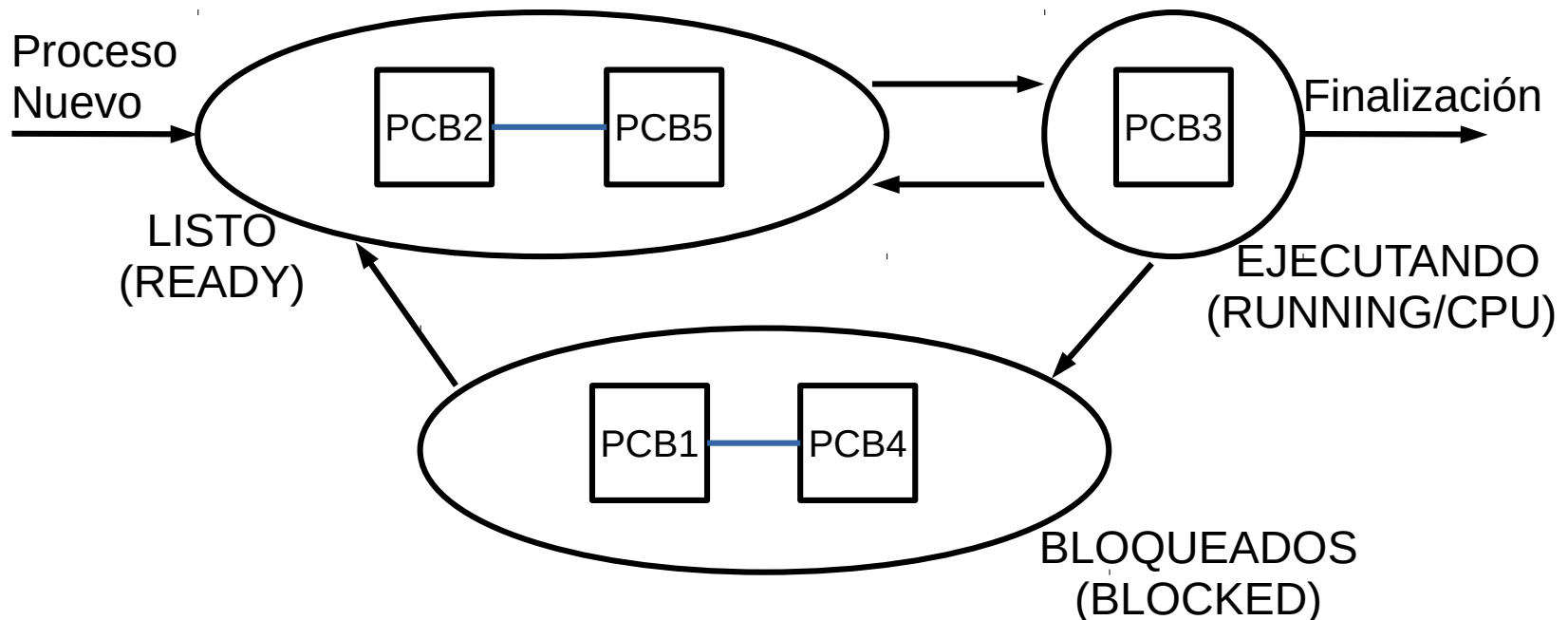


# Procesos

## Proceso

### Ciclo de vida de un proceso

### Estado de los procesos: Diagrama de 3 estados



# Procesos

## Proceso

### E/S Bloqueante - E/S no bloqueante

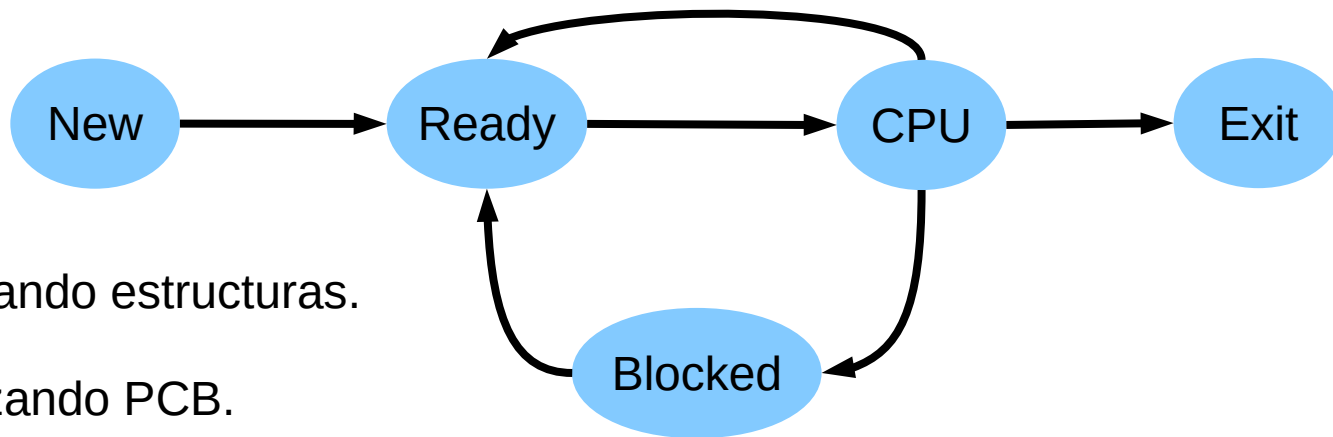
	Bloqueante	No Bloqueante
Respuesta inmediata	Realiza la operación	Realiza la operación
Respuesta en mucho tiempo o indefinida	Bloquea el proceso	<ul style="list-style-type: none"><li>• No realiza la operación</li><li>• Continúa ejecutando</li></ul>
Valores de retorno	OK / Error	OK / Error / Reintentar

# Procesos

## Proceso

### Ciclo de vida de un proceso

### Estado de los procesos: Diagrama de 5 estados



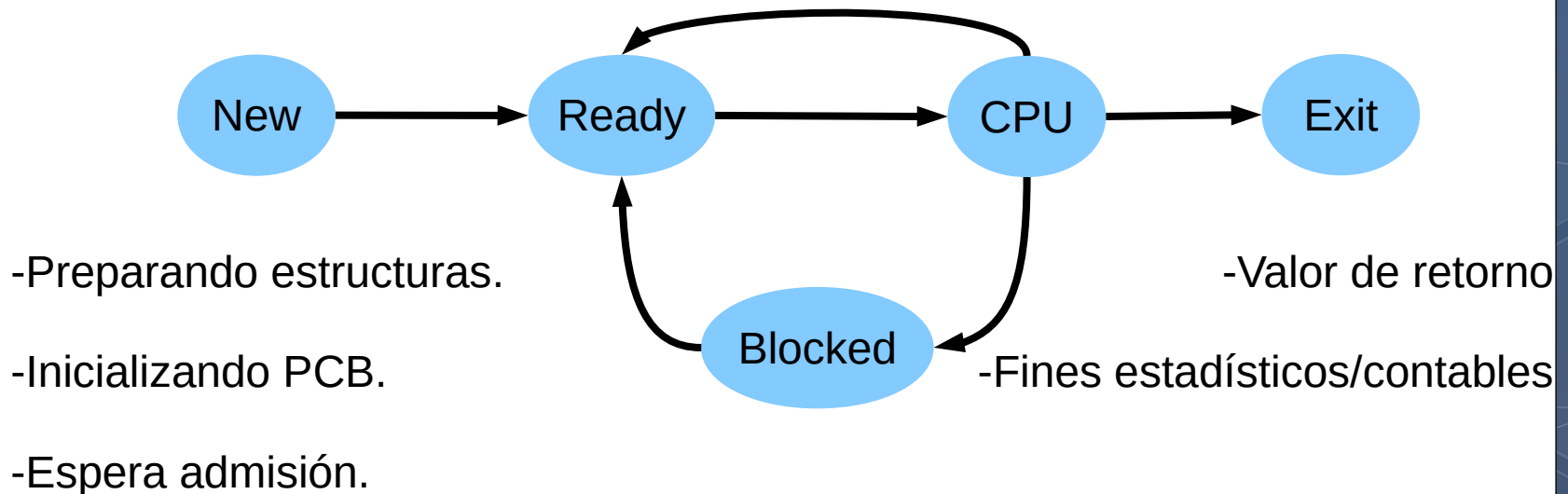
- Preparando estructuras.
- Inicializando PCB.
- Espera admisión.

# Procesos

## Proceso

### Ciclo de vida de un proceso

### Estado de los procesos: Diagrama de 5 estados

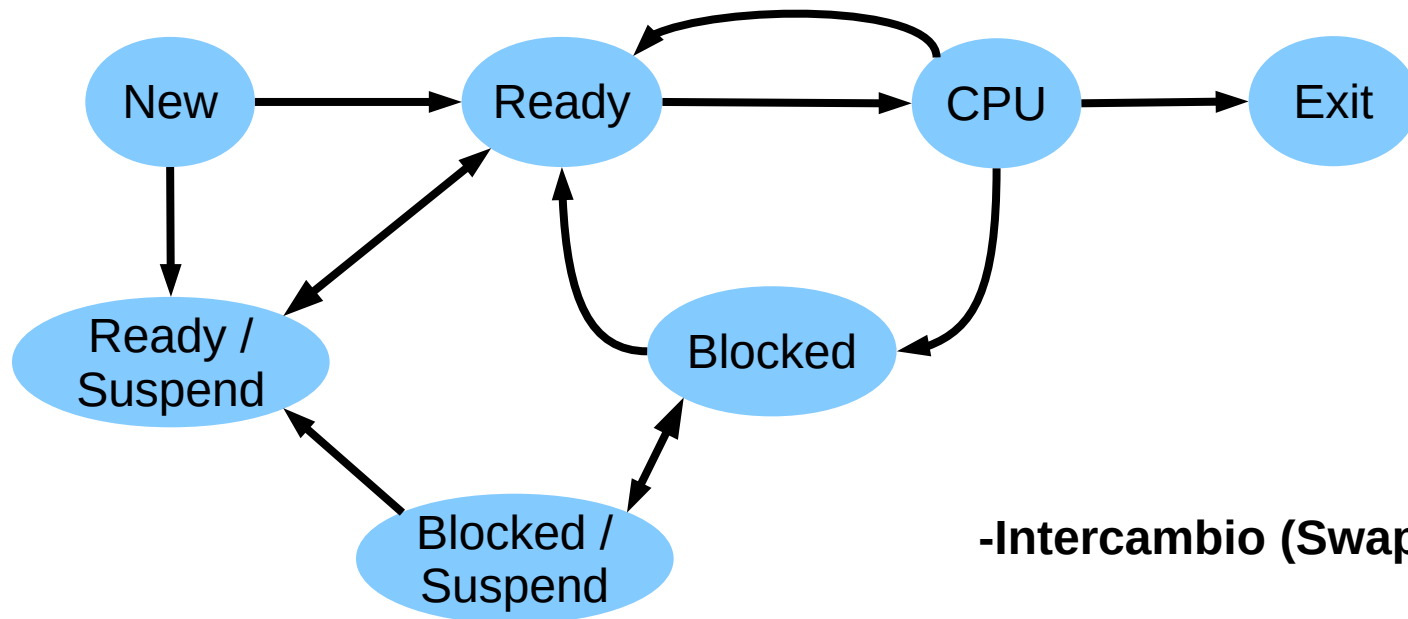


# Procesos

## Proceso

### Ciclo de vida de un proceso

### Estado de los procesos: Diagrama de 7 estados



# Procesos

## Proceso

**Ciclo de vida de un proceso**

**Estado de los procesos: Diagrama de 7 estados**

- Depuración (debugging)

# Procesos

## Proceso

### Ciclo de vida de un proceso

### Estado de los procesos: Diagrama de 7 estados

```
int main() {  
  
    long int a = 0;  
    long int b = 0;  
    int c = 0;  
    int d = 0;  
  
    a = numero_entre (1,10);  
    b = numero_entre (1,10);  
    c = a + b;  
    d = c * 2;  
  
    printf("Resultado: %d\n",d);  
}
```

(x)= a	4
(x)= b	7
(x)= c	11
(x)= d	0
+ Add new	



# Procesos

## Proceso

### Ciclo de vida de un proceso

### Estado de los procesos: Diagrama de 7 estados

```
int main() {  
    long int a = 0;  
    long int b = 0;  
    int c = 0;  
    int d = 0;  
  
    a = numero_entre (1,10);  
    b = numero_entre (1,10);  
    c = a + b;  
    d = c * 2;  
  
    printf("Resultado: %d\n",d);  
}
```

Expression	Value
(x)= a	4
(x)= b	7
(x)= c	11
(x)= d	22
+ Add new	

# Procesos

## Proceso

## Cambio de proceso

Proceso 1

SO

Proceso 2



# Procesos

## Proceso

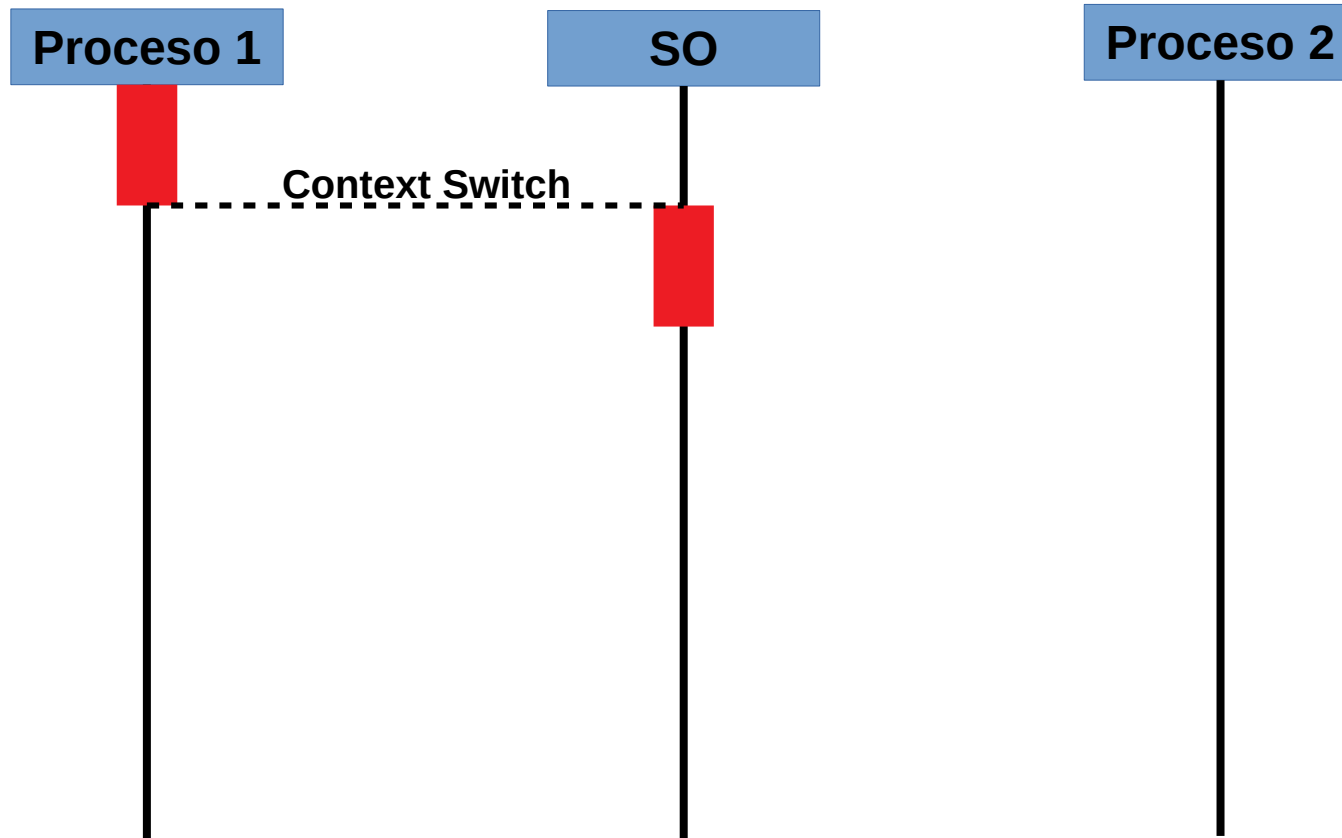
## Cambio de proceso



# Procesos

## Proceso

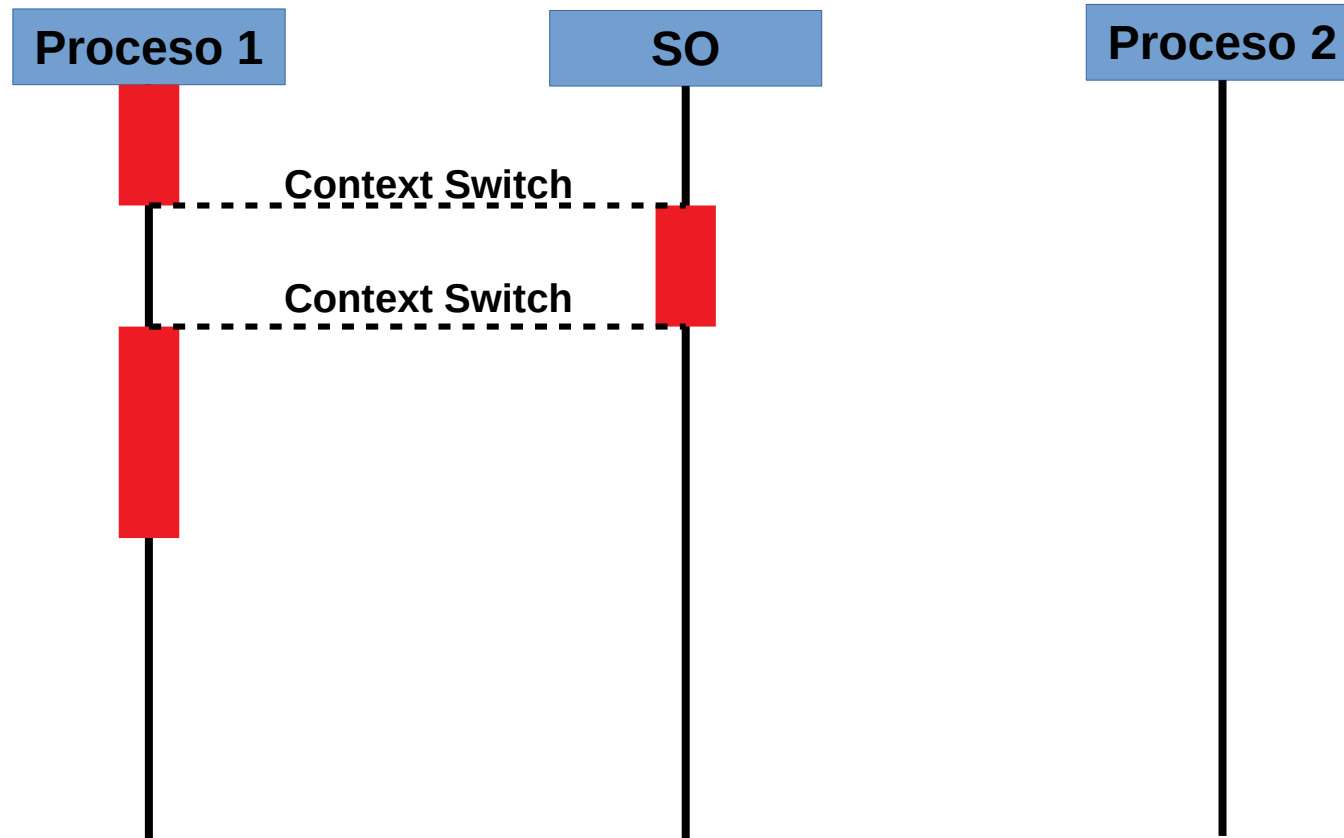
## Cambio de proceso



# Procesos

## Proceso

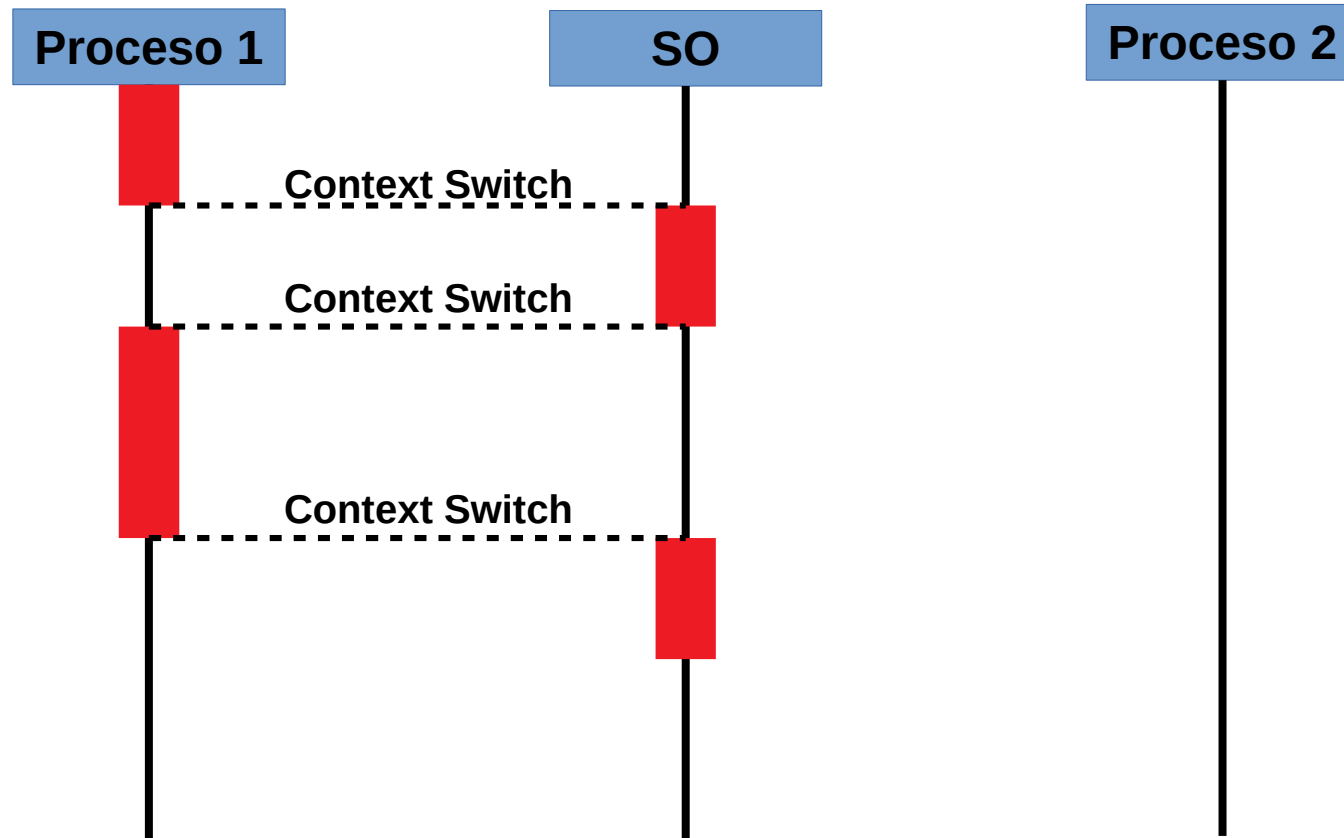
## Cambio de proceso



# Procesos

## Proceso

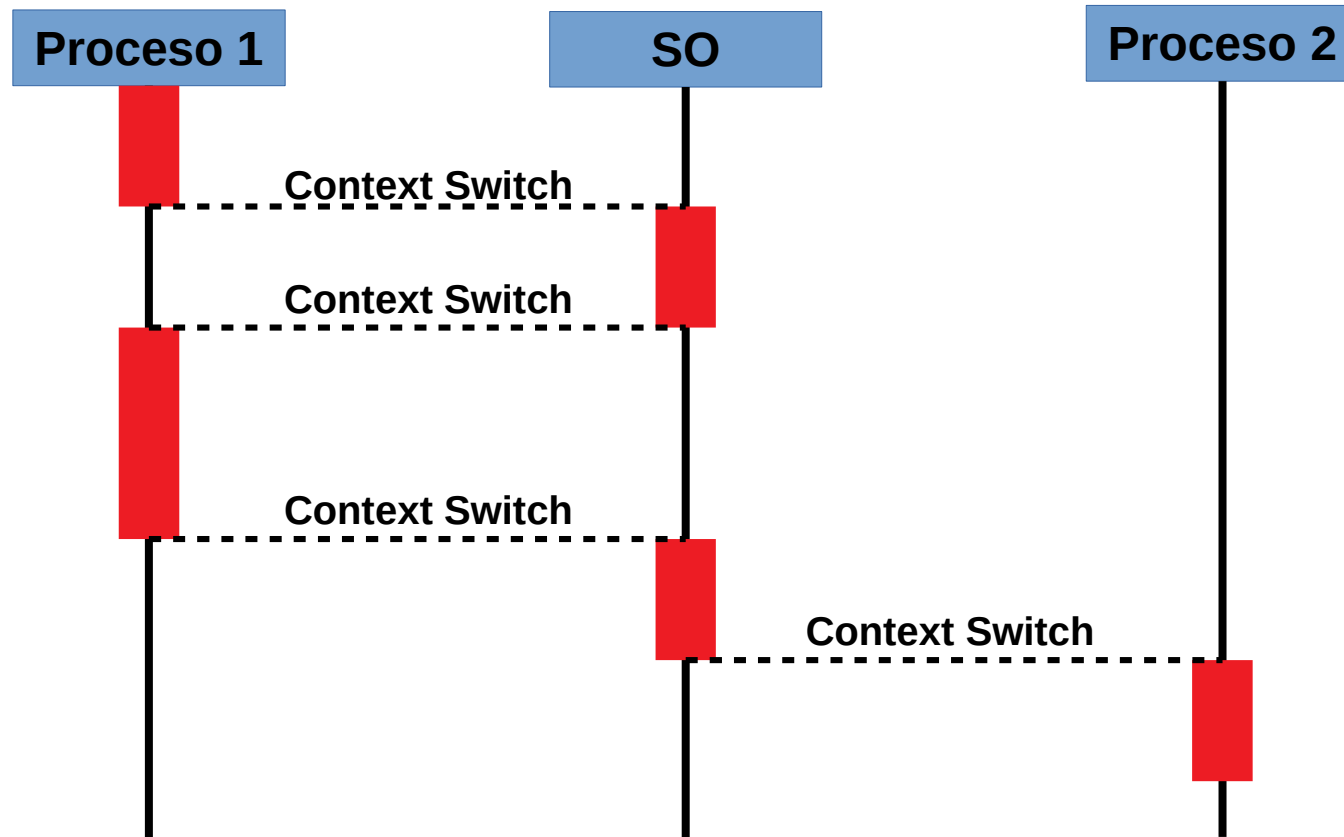
## Cambio de proceso



# Procesos

## Proceso

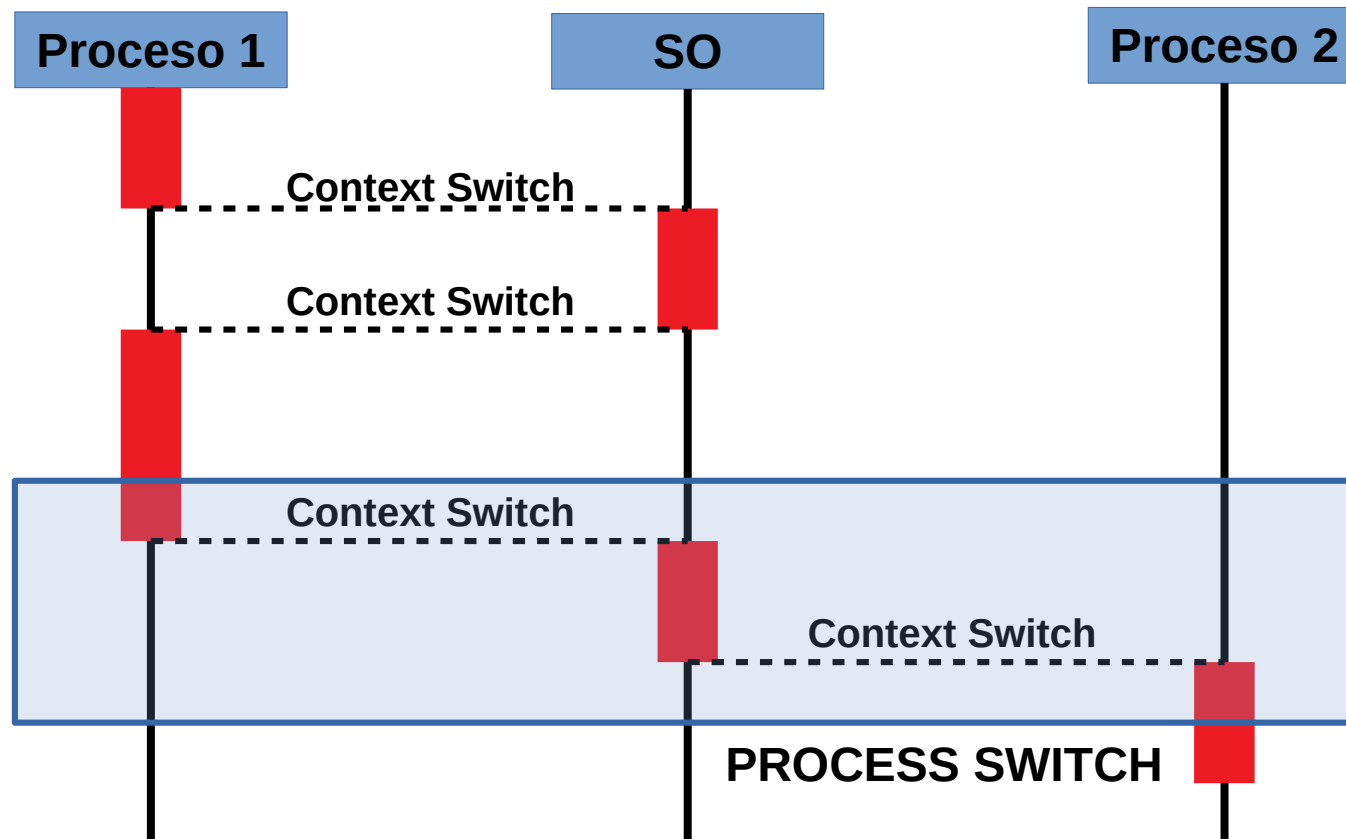
## Cambio de proceso



# Procesos

## Proceso

## Cambio de proceso

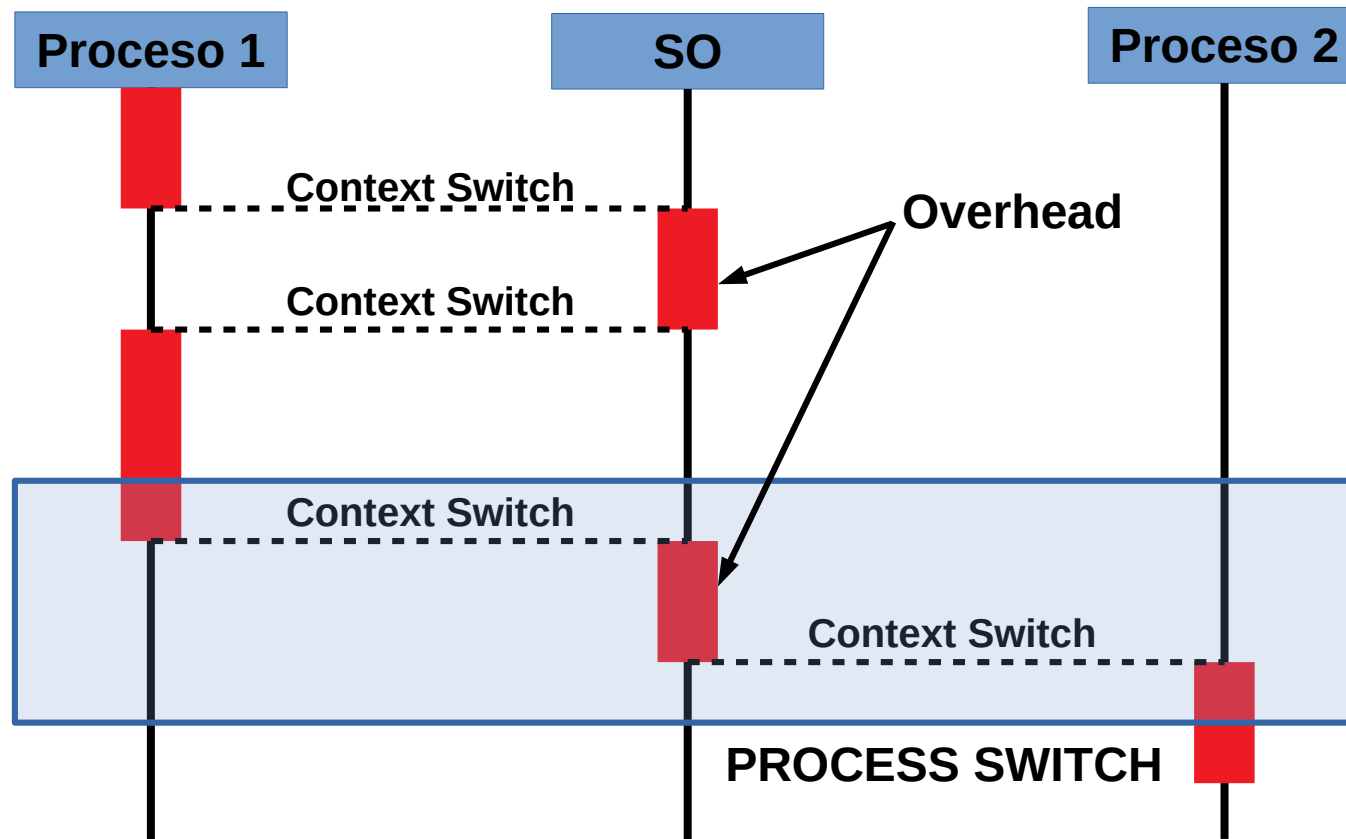




# Procesos

## Proceso

## Cambio de proceso



# Procesos

## Proceso

### Cambio de proceso

1. Ejecutando Proceso A, se produce una INTERRUPCIÓN.
2. Se guarda el PC y PSW del proceso A.
3. Se actualiza PC para atender la interrupción y se cambia de modo de ejecución (modo usuario a kernel).
4. Se guarda el resto del contexto del proceso A.
5. Se decide cambiar el proceso.
6. Se guarda el contexto del proceso A en su PCB.
7. Se cambia el estado del proceso A de Ejecutando a Listo (u otro).
8. El PCB del proceso A se ubica en la lista de procesos Listos (u otra).
9. Se selecciona otro proceso para ser ejecutado (el proceso B).
10. Se cambia el estado del proceso B a Ejecutando.
11. Se actualizan los registros de administración de memoria del procesador con los del proceso B.
12. Se actualizan los registros del procesador con los del PCB de B y se cambia el modo de ejecución (modo kernel a usuario).