# PROCESOS Y TAREAS

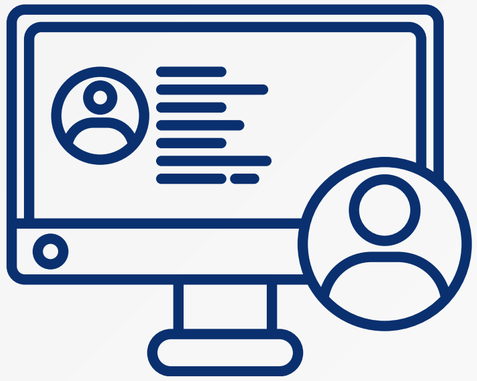
Presentado por: Carreño Sanchez Luis Angel
Valencia Perez Adamari Pamela

# Introducción

“Procesos” se define formalmente como "la entidad que puede ser asignada a un procesador y ejecutada por él". Básicamente un proceso es un programa en ejecución.

Se puede considerar ya como un programa en ejecución, que consta de un programa ejecutable y de los datos necesarios para el programa (variables, espacio de trabajo, etc.) y del contexto de ejecución del programa.

Y en esta presentación hablaremos mas a fondo sobre lo que esta acción lleva acabo en nuestro sistema operativo y sus variaciones y caracteristicas.



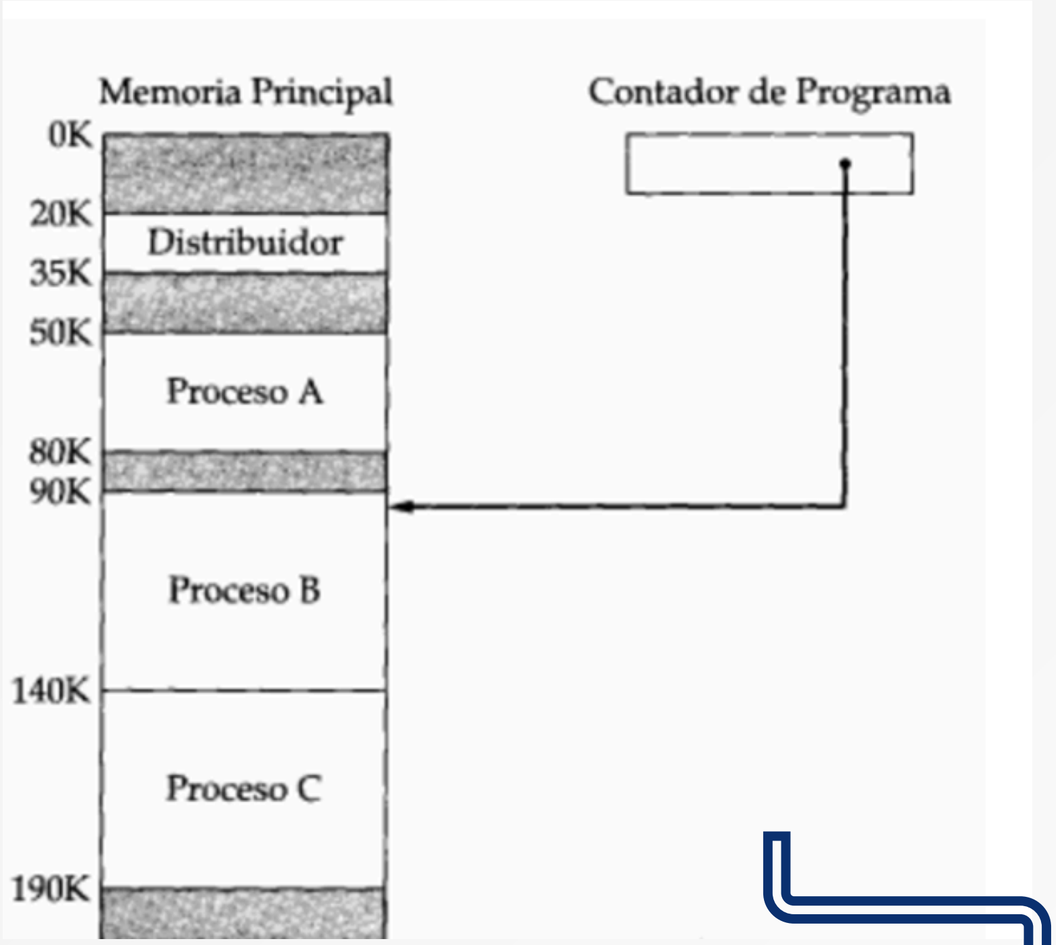
# Estados de un proceso

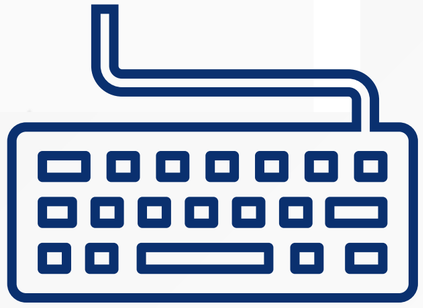
La misión principal del procesador es ejecutar instrucciones de la máquina desde la memoria principal, organizadas en programas con secuencias de instrucciones. Para mejorar la eficiencia y la programación, el procesador alterna la ejecución de varios programas en el tiempo. Sigue una secuencia dictada por el contador de programa (PC "*Program Counter*"), que puede apuntar a diferentes códigos de aplicaciones. Para un programa, su ejecución es una secuencia de instrucciones específicas, conocida como ***tarea o proceso.***

## TRAZA DE PROCESOS.

El comportamiento de un proceso se puede describir mediante la secuencia de instrucciones ejecutadas, conocida como *traza del proceso*. El comportamiento del procesador se puede describir mostrando cómo se intercalan las trazas de varios procesos.

# TRAZA DE PROCESOS.



| | | |
|---|---|---|
| 1 | α + 0 | |
| 2 | α + 1 | |
| 3 | α + 2 | |
| 4 | α + 3 | |
| 5 | α + 4 | |
| 6 | α + 5 | |
| ---------- | fin de plazo | |
| 7 | δ + 0 | |
| 8 | δ + 1 | |
| 9 | δ + 2 | |
| 10 | δ + 3 | |
| 11 | δ + 4 | |
| 12 | δ + 5 | |
| 13 | β + 0 | |
| 14 | β + 1 | |
| 15 | β + 2 | |
| 16 | β + 3 | |
| ---------- | Solicitud de E/S | |
| 17 | δ + 0 | |
| 18 | δ + 1 | |
| 19 | δ + 2 | |
| 20 | δ + 3 | |
| 21 | δ + 4 | |
| 22 | δ + 5 | |
| 23 | γ + 0 | |
| 24 | γ + 1 | |
| 25 | γ + 2 | |
| 26 | γ + 3 | |
| 27 | γ + 4 | |
| 28 | γ + 5 | |
| ---------- | fin de plazo | |
| 29 | δ + 0 | |
| 30 | δ + 1 | |
| 31 | δ + 2 | |
| 32 | δ + 3 | |
| 33 | δ + 4 | |
| 34 | δ + 5 | |
| 35 | α + 6 | |
| 36 | α + 7 | |
| 37 | α + 8 | |
| 38 | α + 9 | |
| 39 | α + 10 | |
| 40 | α + 11 | |
| ---------- | fin de plazo | |
| 41 | δ + 0 | |
| 42 | δ + 1 | |
| 43 | δ + 2 | |
| 44 | δ + 3 | |
| 45 | δ + 4 | |
| 46 | δ + 5 | |
| 47 | γ + 6 | |
| 48 | γ + 7 | |
| 49 | γ + 8 | |
| 50 | γ + 9 | |
| 51 | γ + 10 | |
| 52 | γ + 11 | |
| ---------- | fin de plazo | |

# TRAZA DE PROCESOS.

El modelo más sencillo de gestión de procesos considera que un proceso puede estar en uno de dos estados: ***Ejecución*** o ***No Ejecución***. Cuando el sistema operativo crea un nuevo proceso, este comienza en estado de No ***Ejecución***, esperando su oportunidad para ejecutarse. Periódicamente, el proceso en ***Ejecución*** es interrumpido y el sistema operativo selecciona otro proceso para ejecutarse, cambiando el estado del primero a ***No Ejecución*** y del nuevo a ***Ejecución.***

En este modelo, cada proceso debe ser representado de manera que el sistema operativo pueda encontrarlo, lo cual incluye información sobre su estado actual y posición en memoria. Los procesos que no están ejecutándose se guardan en una cola, esperando su turno para ejecutarse.

## RAZONES PARA LA CREACIÓN DE PROCESOS:

**Nuevo trabajo por lotes** :El sistema operativo tiene un flujo de control de trabajos por lotes, generalmente para cinta o disco.

Cuando el sistema operativo se prepara para tomar un nuevo trabajo, leerá a próxima secuencia de Ordenes de control de trabajos.

**Conexión interactiva**: Un usuario entra en el sistema desde un terminal.

**Creado por el SO para dar un servicio** : Puede crear un proceso para llevar a cabo una función de parte de un programa usuario, sin que el usuario tenga que esperar.

**Generado por un proceso existente** :Con afán de modular o aprovechar el paralelismo, un programa de usuario puede ordenar la creación de una serie de procesos.

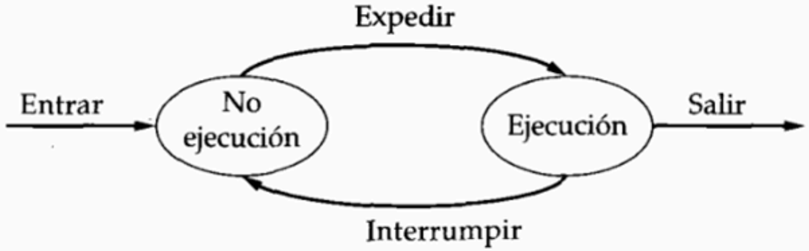
# CREACION DE PROCESOS.

Los sistemas operativos crean procesos de forma transparente para el usuario. Sin embargo, es útil permitir que un proceso cree otros procesos. Esta acción se llama **generación de procesos**.

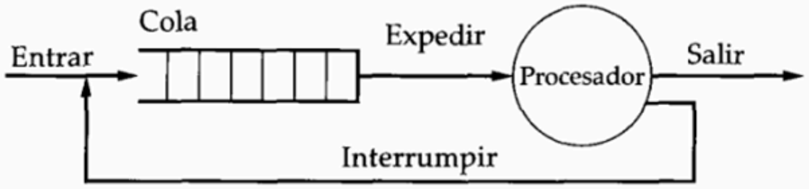
Cuando un proceso crea otro, se les llama **proceso padre y proceso hijo**, respectivamente. Estos procesos a menudo necesitan comunicarse y cooperar, lo cual es una tarea compleja para el programador.

# TERMINACIÓN DE PROCESOS.

En sistemas informáticos, un proceso debe indicar de alguna forma que ha terminado. En trabajos por lotes, se usa una instrucción de detención o una llamada a un servicio del sistema operativo. En aplicaciones interactivas, el usuario indica el fin del proceso.




(a) Diagrama de transición de estados



# RAZONES PARA LA TERMINACIÓN DE UN PROCESO

**Terminación normal**: El proceso ejecuta una llamada a un servicio del SO que indica que ha terminado de ejecutar.

**Tiempo limite excedido**: Tiempo total transcurrido, el tiempo que ha estado ejecutando y tiempo transcurrido desde que el usuario realizó su última entrada de datos.

**No hay memoria disponible:** El proceso necesita más memoria de la que el sistema le puede proporciona.

**Violación de límites**: El proceso trata de acceder a una posición de memoria a la que no le está permitido acceder.

**Error de protección**: El proceso intenta utilizar un recurso o un archivo que no le está permitido utilizar, o trata de utilizarlo de forma incorrecta.

**Intervención del operador o del SO**: Por alguna razón el operador o el SO termina con el proceso

**Terminación del padre**: Cuando un proceso padre finaliza, el sistema operativo puede diseñarse para terminar automáticamente con todos sus hijos.

**Error aritmético**: El proceso intenta hacer un cálculo prohibido.

**Tiempo máximo de espera rebasado:** El proceso ha esperado más allá del tiempo máximo especificado para que se produzca cierto suceso

**Fallo de E/S**: Se produce un error en la entrada o la salida, tal como la incapacidad de encontrar un archivo, un fallo u una operación ilegal

**Instrucción inválida**: El proceso intenta ejecutar una instrucción inexistente

**Instrucción privilegiada:** El proceso intenta usar una instrucción reservada para el sistema operativo.

**Mal uso de los datos**: Un elemento de dato es de un tipo equivocado o no está inicializado.

**Solicitud del padre**: Un proceso padre tiene normalmente la autoridad de terminar con cualquiera de sus descendientes.

# UN MODELO DE CINCO ESTADOS.

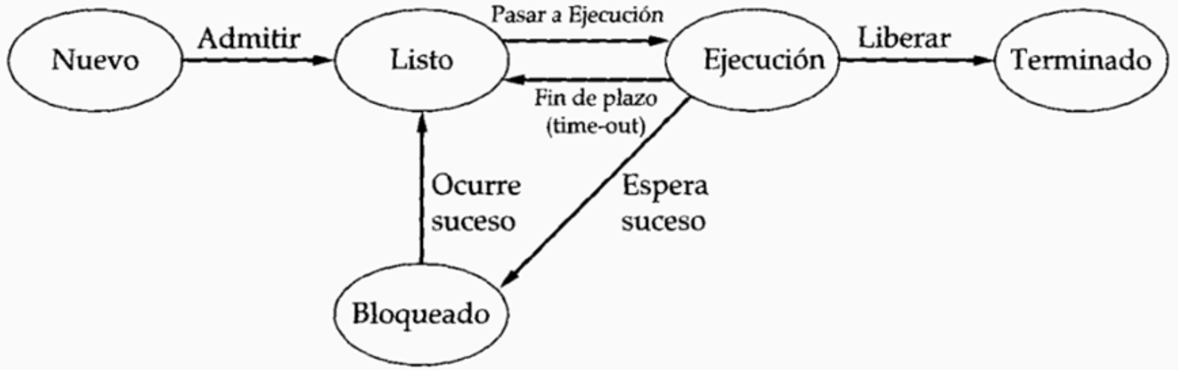
Ejecución: El proceso actualmente en ejecución. Solo un proceso puede estar en este estado a la vez en sistemas con un único procesador.

Listo: Procesos preparados para ejecutar cuando se les dé la oportunidad

Bloqueado: Procesos que no pueden ejecutar hasta que ocurra un evento específico, como la finalización de una operación de E/S.

Nuevo: Procesos recién creados que aún no han sido admitidos por el sistema operativo como ejecutables.

Terminado: Procesos que han sido excluidos del grupo de procesos ejecutables, ya sea porque se detuvieron o por alguna otra razón.



# EL PROCESO PASA POR VARIAS TRANSICIONES DE ESTADO:

1. ***Nuevo a Listo:*** Cuando el SO está preparado para aceptar más procesos.
2. ***Listo a Ejecución***: El SO elige un proceso del estado Listo para ejecutar.
3. ***Ejecución a Terminado***: El proceso finaliza su ejecución, ya sea porque completó su tarea o por algún error.
4. ***Ejecución a Listo***: Cuando el proceso ha utilizado el máximo tiempo permitido o si otro proceso de mayor prioridad necesita ejecutarse.
5. ***Ejecución a Bloqueado:*** Si el proceso necesita esperar por un recurso o una acción específica.
6. ***Bloqueado a Listo:*** Cuando el evento que el proceso estaba esperando ocurre.
7. ***Listo a Terminado y Bloqueado a Terminado:*** Puede ocurrir si el proceso es finalizado por otro proceso o si el proceso padre termina.

# DESCRIPCIÓN Y CONTROL DE PROCESOS

En un sistema operativo, es crucial construir tablas para administrar procesos y asignarles espacio de direcciones. Una estrategia es mantener una reserva de procesos no bloqueados, aunque esto puede limitar el espacio en memoria principal para nuevos procesos, justificando la transición *Nuevo → Listo y suspendido*.

# TRANSICIONES DE ESTADOS DE PROCESOS:

01. ***Bloqueado y suspendido → Bloqueado:*** Trae a la memoria un proceso que no está listo para ejecutarse, es justificable si dicho proceso tiene una prioridad mayor y se espera que pronto ocurra el suceso que lo desbloquee.

02. ***Ejecución → Listo y suspendido:*** Un proceso puede ser trasladado a Listo y suspendido si un proceso de mayor prioridad se desbloquea, liberando así espacio en memoria principal.

03. ***Varios → Terminado:*** Los procesos pueden finalizar en cualquier estado, no solo durante la ejecución, debido a condiciones externas o decisiones del proceso padre

# Control de procesos

La mayoría de los procesadores soportan al menos dos modos de ejecución: el ***modo privilegiado y el modo de usuario.*** En ***modo privilegiado***, el sistema puede ejecutar instrucciones críticas como la gestión de memoria y la E/S, y acceder a regiones de memoria restringidas. El ***modo de usuario*** es el menos privilegiado y es utilizado por los programas de usuario para proteger el sistema operativo y sus componentes críticos, como los bloques de control de procesos y de interferencias.

## FUNCIONES BÁSICAS DEL NÚCLEO DE UN SISTEMA OPERATIVO

**Gestión de Procesos**

- Creación y terminación de los procesos
- Planificación y expedición de los procesos
- Cambio de procesos
- Sincronización de procesos y soporte para la comunicación entre procesos
- Gestión de los bloques de control de procesos

**Gestión de memoria**

- Asignación de espacios de direcciones a los procesos
- Intercambio
- Gestión de páginas y segmentos

**Gestión de E/S**

- Gestión de buffers
- Asignación de canales de E/S y dispositivos a los procesos

**Funciones de Soporte**

- Tratamiento de interrupciones
- Contabilidad
- Supervisión

# CAMBIO DE PROCESOS

***Cambio de proceso***: El cambio de proceso parece sencillo: un proceso en ejecución se interrumpe y el sistema operativo pone otro proceso en ejecución. Sin embargo, surgen diversas cuestiones de diseño como los sucesos que provocan el cambio, la distinción entre cambio de contexto y cambio de proceso, y la gestión de las estructuras de datos necesarias para llevar a cabo el cambio.

***Cuándo cambiar de proceso:*** El cambio de proceso puede ocurrir cada vez que el sistema operativo toma el control del proceso en ejecución. Las posibles causas de esta intervención incluyen dos tipos de interrupciones:

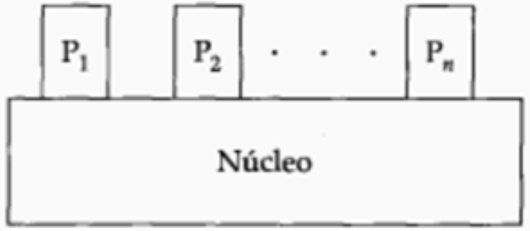
***Interrupción ordinaria:*** Provocada por eventos externos e independientes del proceso actual, como la culminación de una operación de E/S.

***Cepo***: Originada por una condición de error o excepción dentro del proceso actual, como un intento ilegal de acceso a un archivo.

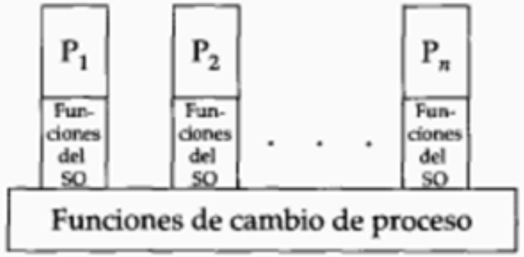
# EJECUCIÓN DEL SISTEMA OPERATIVO

**Núcleo fuera de todo proceso:** En este enfoque, el núcleo del sistema operativo opera como una entidad separada fuera de cualquier proceso de usuario. El SO puede ejecutar cualquier función y luego restaurar el contexto del proceso interrumpido o planificar otro proceso, dependiendo de la causa de la interrupción.

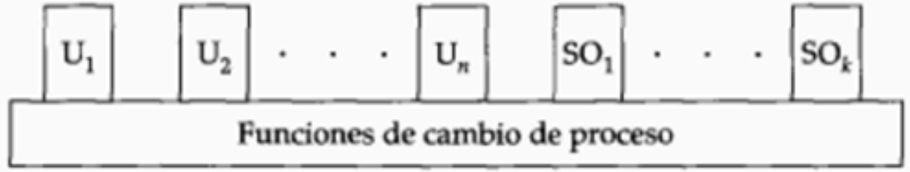
**Ejecución dentro de los procesos de usuario:** Común en SO's para máquinas pequeñas, este enfoque ejecuta la mayoría del software del SO dentro del contexto de un proceso de usuario. Aquí, el SO actúa como una colección de rutinas llamadas por el usuario, ejecutándose en el entorno del proceso de usuario.




(a) Núcleo separado




(b) Las funciones del SO se ejecutan dentro de los procesos de usuario

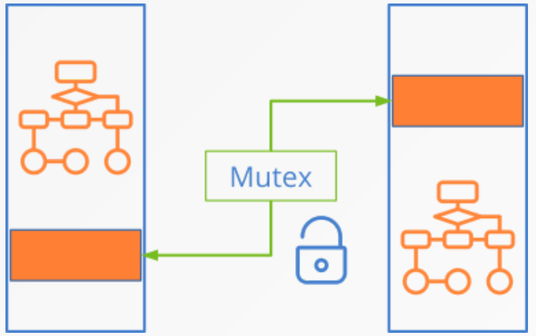



(c) Las funciones del SO que ejecutan como procesos separados

**Micronúcleos**: El micronúcleo (microkernel) se refiere a un pequeño núcleo que proporciona las bases para ampliaciones modulares. Este enfoque busca mantener en el núcleo solo las funciones esenciales del sistema operativo, mientras que otros servicios se ejecutan como subsistemas externos que interactúan con el núcleo

# Exclusión Mutua

El método más sencillo de comunicación entre procesos de un programa concurrente es el uso compartido de variables de datos. Sin embargo, esta simplicidad puede llevar a errores, ya que el acceso concurrente puede causar interferencias entre procesos. Lo mismo aplica a cualquier recurso del sistema que solo pueda ser usado por un proceso a la vez.




## BLOQUEO MEDIANTE USO DE VARIABLES COMPARTIDAS

Se presenta una forma de bloquear una sección crítica usando una variable compartida de tipo booleano llamada indicador (flag). El algoritmo se va perfeccionando para mostrar problemas comunes en la concurrencia entre procesos. Cada recurso compartido tendrá un flag. Antes de acceder al recurso, un proceso debe verificar el flag, que puede ser true (recurso en uso) o false (recurso disponible).

```
/*   Exclusión Mutua: Uso de un indicador*/

program/module Exclusión_Mutua_1;

var flag: boolean;

process P1
begin
    loop
        while flag = true do
            /* Espera a que el dispositivo se libere */
        end;
        flag := true;
        /* Uso del recurso Sección Crítica */
        flag := false;
        /* resto del proceso */
    end
end P1;

process P2
begin
    loop
        while flag = true do
            /* Espera a que el dispositivo se libere */
        end;
        flag := true;
        /* Uso del recurso Sección Crítica */
        flag := false;
        /* resto del proceso */
    end
end P2;

begin /* Exclusión_Mutua_1*/
    flag := false;
    cobegin
        P1;
        P2;
    coend
end Exclusión_Mutua_1;
```

# ALGORITMO DE PETERSON

La estrategia propuesta por Peterson (1981) para resolver problemas de acceso simultáneo a la sección crítica implica la introducción de una variable adicional llamada "turno". Esta variable solo se utiliza cuando ocurre un conflicto de peticiones concurrentes de acceso a la sección crítica.

```
/*  Exclusión Mutua: Solución de Peterson*/

program/module Exclusión_Mutua_P;

var flag1, flag2: boolean;
    turno: integer;

procedure bloqueo (var mi_flag, su_flag: boolean; su_turno: integer);
begin
    mi_flag := true;
    turno := su_turno;
    while su_flag and (turno = su_turno) do ; end
end bloqueo;

procedure desbloqueo (var mi_flag: boolean);
begin
    mi_flag := false;
end desbloqueo;

process P1
begin
    loop
        bloqueo (flag1, flag2, 2);
        /* Uso del recurso Sección Crítica */
        desbloqueo (flag1);
        /* resto del proceso */
    end
end P1;
```

```
process P2
begin
    loop
        bloqueo (flag2, flag1, 1);
        /* Uso del recurso Sección Crítica */
        desbloqueo (flag2);
        /* resto del proceso */
    end
end P2;

begin /* Exclusión_Mutua_P*/
    flag1 := false;
    flag2 := false;
    cobegin
```

```
/*  Exclusión Mutua: Solución de Dekker*/

program/module Exclusión_Mutua_D;

var flag1, flag2: boolean;
    turno: integer;

procedure bloqueo (var mi_flag, su_flag: boolean; su_turno: integer);
begin
    mi_flag := true;
    while su_flag do /* otro proceso en la sección crítica */
          if turno = su_turno then
              mi_flag := false;
              while turno = su_turno do
              ; /* espera a que el otro acabe */
              end;
               mi_flag := true;
          end;
    end;
end bloqueo;
procedure desbloqueo (var mi_flag: boolean; su_turno: integer);
```

```
begin
    turno := su_turno;
    mi_flag := false;
end desbloqueo;

process P1
begin
    loop
        bloqueo (flag1, flag2, 2);
        /* Uso del recurso Sección Crítica */
        desbloqueo (flag1, 2);
        /* resto del proceso */
    end
end P1;

process P2
begin
    loop
        bloqueo (flag2, flag1, 1);
        /* Uso del recurso Sección Crítica */
        desbloqueo (flag2, 1);
        /* resto del proceso */
    end
end P2;

begin /* Exclusión_Mutua_P*/
    flag1 := false;
    flag2 := false;
    turno := 1;
    cobegin
        P1;
        P2;
    coend
end Exclusión_Mutua_D;
```

# ALGORITMO DE DEKKER

La solución al problema de la exclusión mutua atribuida al matemático holandés Dekker y presentada por Dijkstra en 1968, utiliza una variable de turno, similar a la solución de Peterson. Esta variable establece la prioridad de los dos procesos y se actualiza en la sección crítica, evitando así interferencias entre los procesos.

# Monitores

Un monitor es un conjunto de procedimientos que aseguran acceso exclusivo a recursos compartidos entre varios procesos. Estos procedimientos están dentro de un módulo que permite que solo un proceso esté activo a la vez. El monitor regula el acceso al recurso, asegurando que, si muchos procesos intentan acceder, solo uno pueda hacerlo a la vez mientras los demás esperan su turno.

El monitor asegura que solo un procedimiento esté activo a la vez. Si un proceso quiere escribir datos y el monitor está libre, se le permite la entrada. Si el monitor ya está en uso por otro proceso, el nuevo proceso debe esperar hasta que el monitor esté disponible.

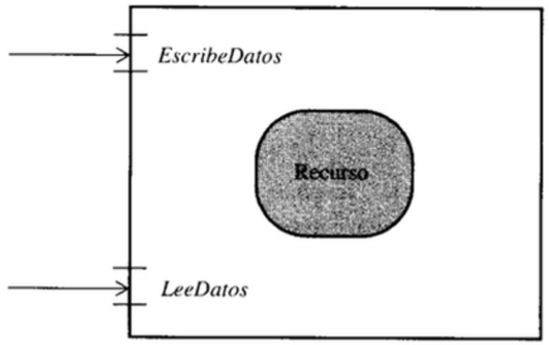



**Figura 3.6:** Monitor sencillo

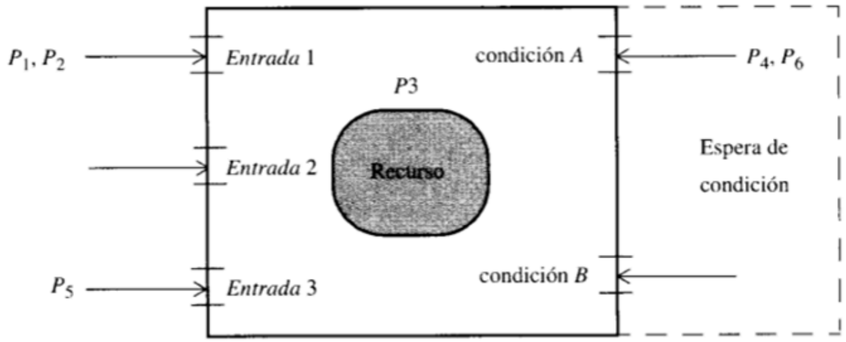



**Figura 3.7:** Monitor general

# Interbloqueo

El interbloqueo es un problema grave en sistemas concurrentes donde múltiples procesos compiten por recursos. Se produce cuando los procesos quedan atrapados, ya que cada uno posee un recurso y espera la liberación de otro retenido por otro proceso, impidiendo el progreso de todos los procesos involucrados.

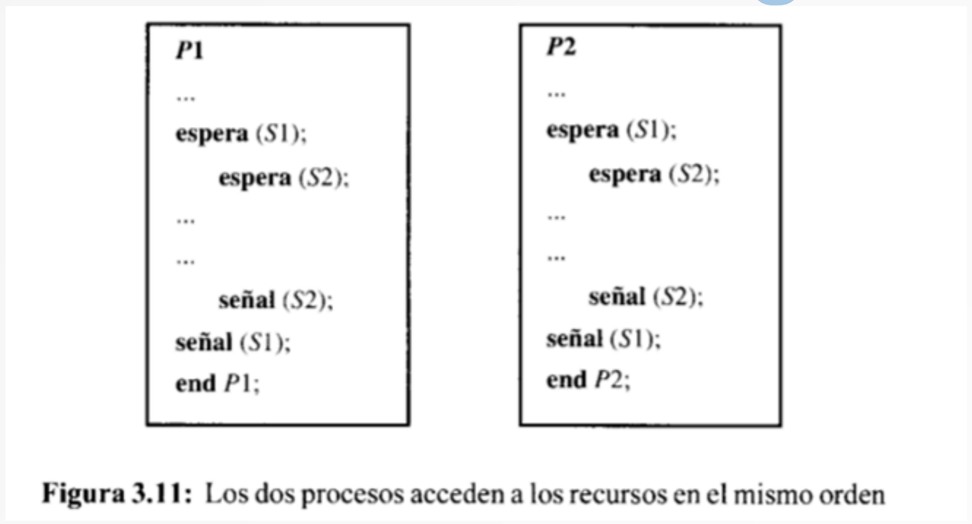



**Figura 3.11:** Los dos procesos acceden a los recursos en el mismo orden

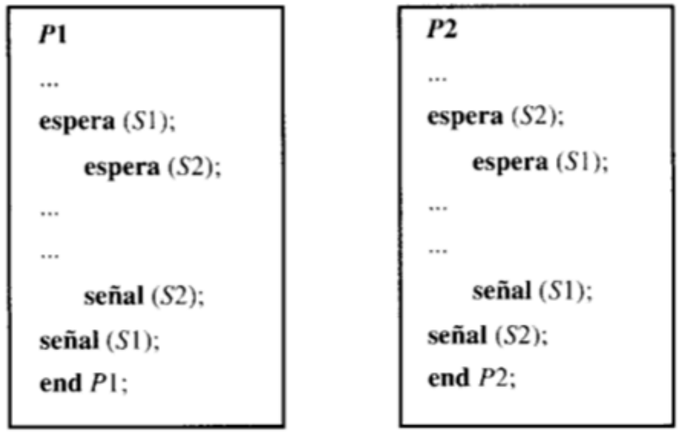



**Figura 3.12:** Los dos procesos acceden a los recursos en orden inverso

El interbloqueo ocurre cuando dos procesos compiten por recursos y se bloquean mutuamente, esperando indefinidamente. Es crucial detectar y prevenir el interbloqueo, aunque es poco frecuente, para evitar sus graves consecuencias. Entender las secuencias de adquisición y liberación de recursos es clave para evitarlo.

# CARACTERIZACIÓN DE INTERBLOQUEO

***Retención y Espera**: Este método se basa en la premisa de "todos o ninguno": un proceso solo puede solicitar recursos si están todos disponibles; si no, se suspende hasta que lo estén.*

**Espera Circular:** Para evitar la espera circular, se asignan números a los recursos y se solicitan en orden ascendente. Esto previene que un proceso espere a otro que esté esperando un recurso del mismo tipo o de orden inferior.

**Grafos de Asignación de Recursos:** Se emplean para detectar interbloqueos, representando asignaciones de recursos y peticiones de procesos. Los nodos son procesos y recursos, los arcos indican asignaciones y solicitudes.

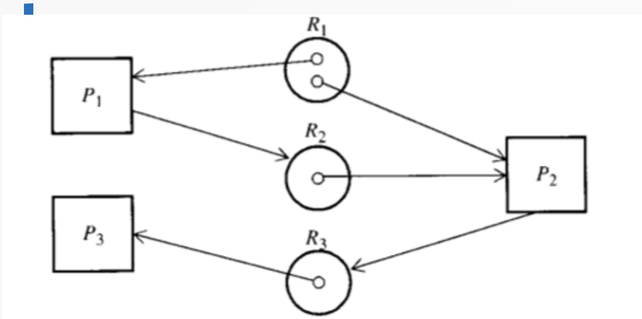



**Figura 3.13:** Grafo de asignación de los recursos del sistema

**No Expropiación:** Si un proceso solicita recursos adicionales que están en uso, puede liberar sus recursos actuales y esperar. Así, los recursos quedan disponibles para otros procesos, y el proceso original se reanuda cuando todos los recursos necesarios están disponibles.

## Recuperación de Interbloqueos:

Cuando se detecta un interbloqueo, es necesario romperlo para que los procesos puedan finalizar su ejecución y liberar los recursos. Se pueden considerar varias opciones, siendo la ideal la suspensión de algunos procesos bloqueados para tomar sus recursos y luego reanudar su ejecución.
Otras opciones incluyen reiniciar uno o más procesos bloqueados y expropiar recursos de algunos procesos.

## Clasificación de Recursos y Estrategias:

Una estrategia efectiva para evitar el interbloqueo es agrupar los recursos del sistema en clases disjuntas y ordenarlas para evitar la espera circular. Para cada clase de recurso, se elige la estrategia más adecuada:

# CLASIFICACIÓN DE RECURSOS Y ESTRATEGIAS:

## Espacio de Intercambio

**Se puede prevenir el interbloqueo solicitando toda la memoria necesaria de antemano o mediante la prevención del interbloqueo.**

## Memoria Principal

**La prevención mediante expropiación es la mejor opción, ya que los procesos expropiados se pueden mover a la memoria secundaria.**

## Recursos de los Procesos

**La prevención del interbloqueo o la ordenación de los recursos son opciones adecuadas, ya que los procesos suelen declarar los recursos necesarios antes de la ejecución.**

## Recursos Internos

**La prevención por ordenación de los recursos suele ser suficiente, ya que no suelen requerir elecciones entre las solicitudes pendientes.**

Al combinar estas estrategias, se puede lograr una mayor eficacia en la gestión de interbloqueos, aprovechando las ventajas de cada método.

# MUCHAS GRACIAS
# POR SU ATENCION