Universidad de los Andes

Infraestructura computacional

Concurrencia

Universidad de los Andes

Problemática de la concurrencia

- Aspectos propios de la programación concurrente:
  - No-determinismo
  - Condición de carrera
  - Espera activa
  - Interbloqueos (deadlock)
  - Bloqueos activos (livelock)
  - Inanición

Universidad de los Andes

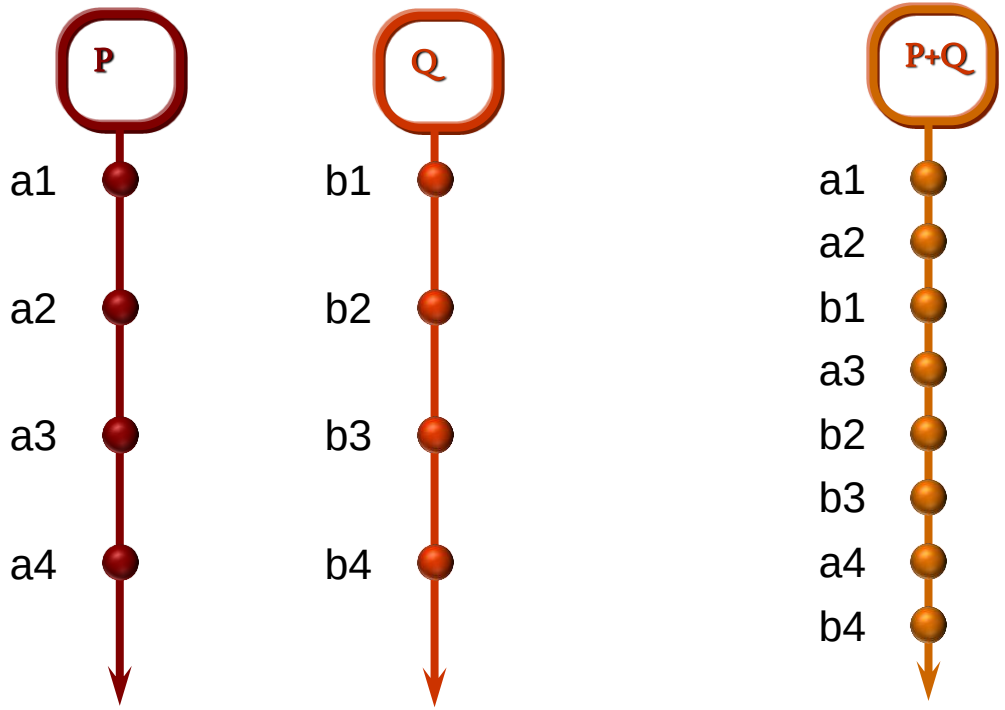
Problemática de la concurrencia

- No-determinismo
  - Al mismo tiempo característica y modelo
  - Diferentes ejecuciones pueden transcurrir de manera distinta
  - Todo camino de ejecución debe conducir a un resultado válido
  - Los resultados pueden ser diferentes pero deben satisfacer los requerimientos
  - Le proporciona reactividad al sistema
  - Dificulta la prueba y depuración de programas
  - **Ejemplo**: búsqueda del máximo de un vector cuando este se puede encontrar repetido
    - En diferentes ejecuciones, el máximo se puede encontrar en posiciones diferentes

Universidad de los Andes

Problemática de la concurrencia

- No-determinismo: modelo



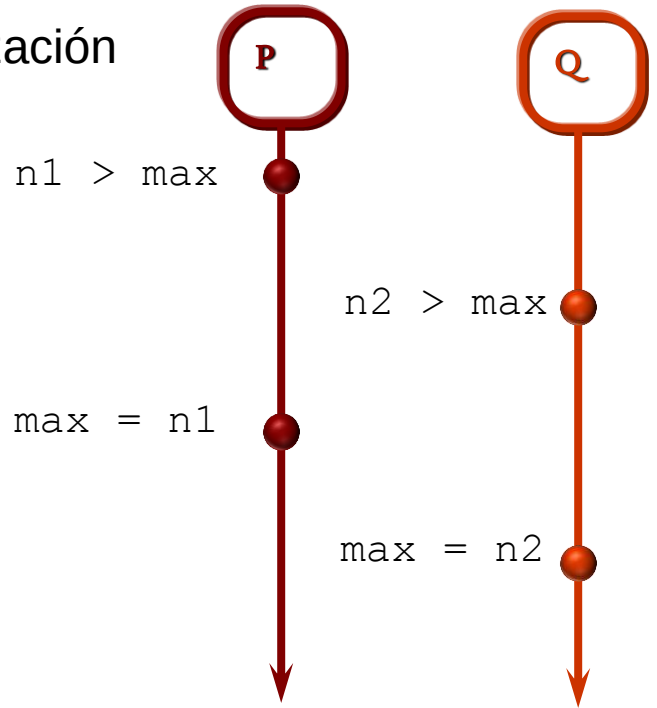
Universidad de los Andes

Problemática de la concurrencia

- Condición de carrera
  - El resultado depende de cuál proceso llegue primero a un punto
  - Los respectivos resultados son incoherentes
  - Se impide por medio de la sincronización

```
if ( n1 > max ) max = n1;
```

```
if ( n2 > max ) max = n2;
```



Universidad de los Andes

Problemática de la concurrencia

- Espera activa
  - Un proceso espera un evento mientras ejecuta
  - Consume procesador inútilmente
  - Puede bloquear a los procesos que están en capacidad de generar el evento
  - ... su complemento es la espera pasiva

```
while( !termino );
```

```
termino = true;
```

Universidad de los Andes

Problemática de la concurrencia

- Bloqueos activos (livelock)
  - El sistema no avanza porque los procesos continuamente intentan una acción que fracasa
  - … o su complemento la vivacidad: el sistema avanza continuamente hacia su objetivo
  - Típicamente surge cuando se intenta evitar un interbloqueo:

```
Reservar recurso 1
while ( Recurso 2 ocupado )
  Liberar recurso 1
  Esperar
  Reservar recuso 1
Reservar recurso 2
```

```
Reservar recurso 2
while ( Recurso 1 ocupado )
  Liberar recurso 2
  Esperar
  Reservar recuso 2
Reservar recurso 1
```

Universidad de los Andes

Problemática de la concurrencia

- Inanición
  - A un proceso se le impide continuamente (e indefinidamente) efectuar una acción
  - … o su complemento la ejecución equitativa: todo proceso que quiera efectuar una acción lo logra en un tiempo finito.
  - **Ejemplo**: lectores-redactores no equitativo

Universidad de los Andes

Problemática de la concurrencia

- Interbloqueos (deadlock)
  - Todo proceso está esperando por un evento que otro proceso debe producir (pero que no lo hace porque, a su vez, está esperando)
  - Requiere de cuatro condiciones necesarias:
    - Los recursos se usan en Exclusión Mutua
    - Los procesos solicitan recursos mientras tienen otros (*hold and wait*)
    - Hay apropiación (en realidad, no hay “expropiación”)
    - Hay un ciclo de dependencias

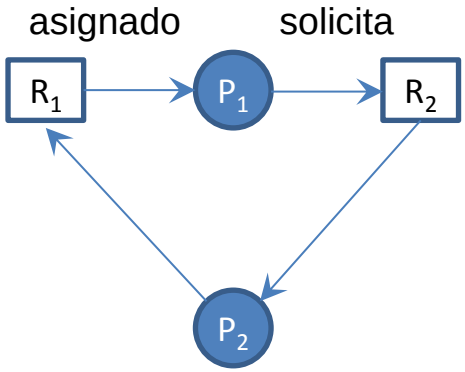
```
P( s1 )
P( s2 )
 … acciones
V( s2)
V( s1 )
```

```
P( s2 )
P( s1 )
 … acciones
V( s1 )
V( s2 )
```

Universidad de los Andes

Problemática de la concurrencia

- Interbloqueos: ciclos de dependencias
  - Grafos de asignación de recursos



Universidad de los Andes

Problemática de la concurrencia

- Interbloqueos
  - Tratamiento
    - Prevenir
    - Evitar
    - Detectar
    - Ignorar
  - Para impedir, una de las condiciones no se debe cumplir:
    - Tres primeras: muy difíciles de evitar
    - Ciclo de dependencias
      - Impedir que se forme
      - Detectar y destruir

Universidad de los Andes

Implementación de la concurrencia: procesos

- Unidad de trabajo del sistema operativo
- El S.O. es responsable de:
  - Creación
  - Despacho
  - Destrucción
  - Proveer servicios (comunicación, sincronización, E/S, etc.)
- S.O. administra y asigna los recursos:
  - Tiempo de CPU
  - Memoria
  - Archivos
  - Dispositivos de E/S, etc.

Universidad de los Andes

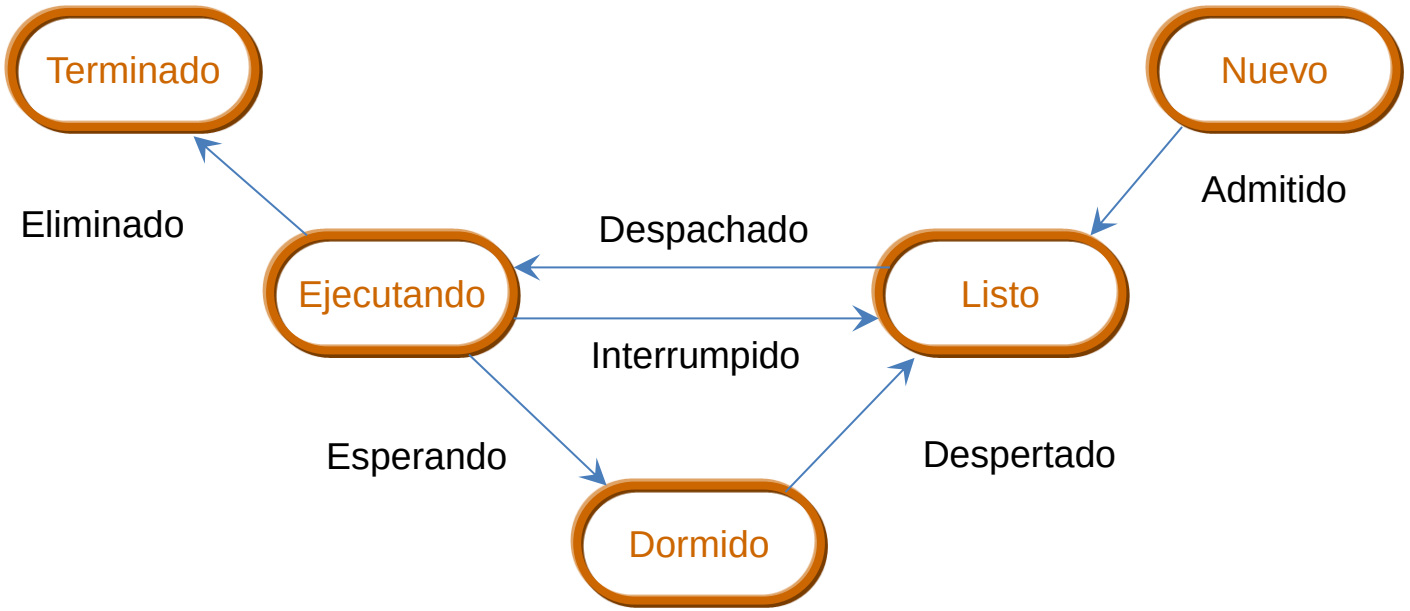
Implementación de la concurrencia: procesos

- Los procesos están representados por su Bloque de Control de Proceso (PCB - Process Control Block)
- El PCB guarda la información sobre el estado de ejecución del proceso:
  - Estado del proceso
  - Contador de programa (PC)
  - Registros
  - Información de despacho de la CPU (prioridad, etc.)
  - Información de administración de la memoria
  - Información contable (recursos asignados)
  - Información de E/S (lista de dispositivos, de archivos, etc.)

Universidad de los Andes

Implementación de la concurrencia: procesos

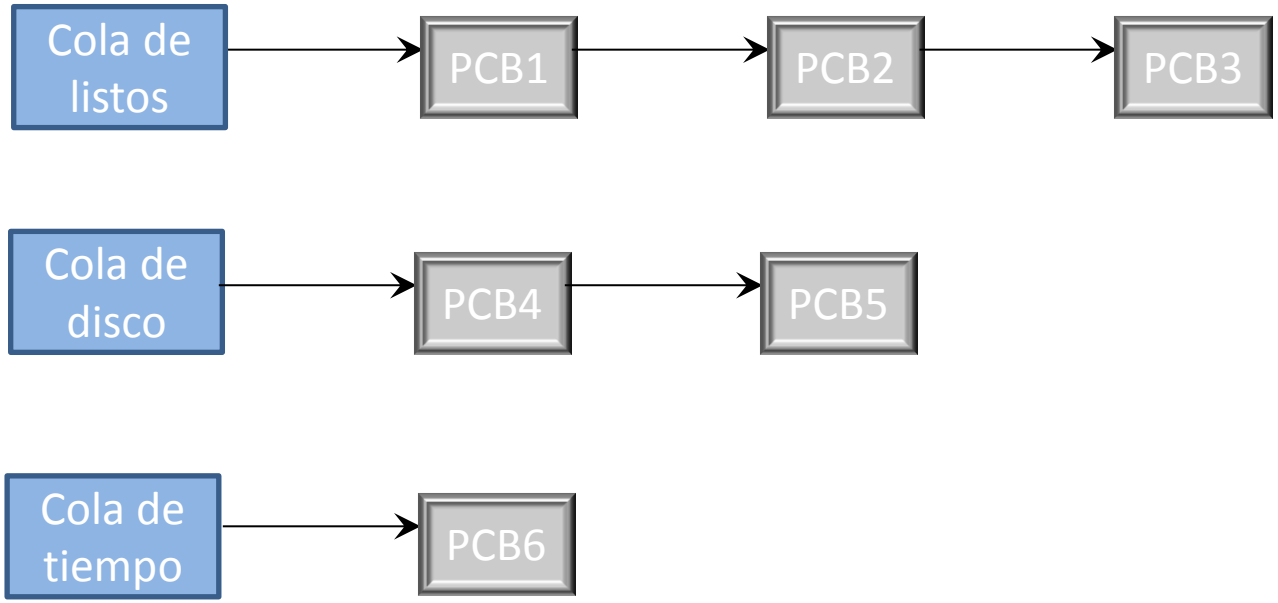
- Estados de los procesos



Universidad de los Andes

Implementación de la concurrencia: procesos

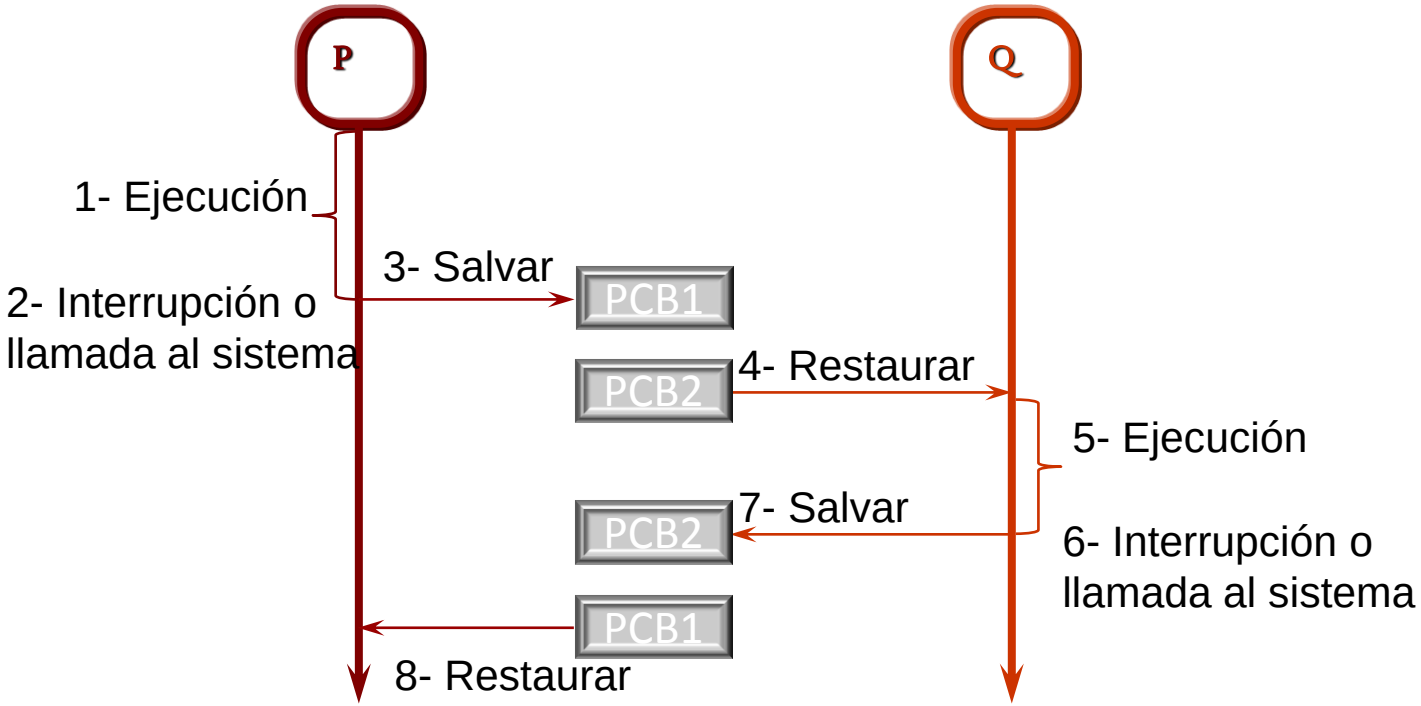
- Colas de despacho



Universidad de los Andes

Implementación de la concurrencia: procesos

- Despacho

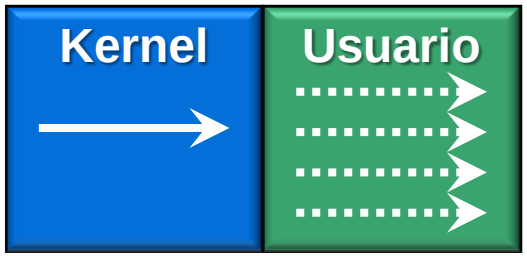


Universidad de los Andes

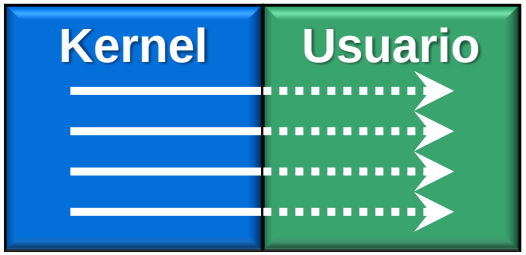
Implementación de la concurrencia: threads

- Manejo de threads

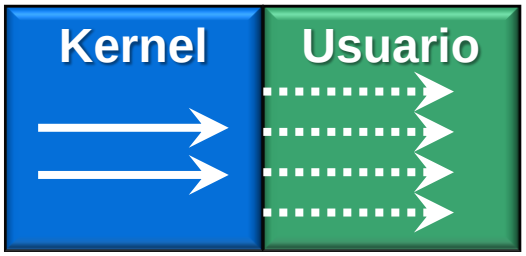
En el espacio del usuario



En el espacio del kernel



Mixto (thread ligero)



Universidad de los Andes

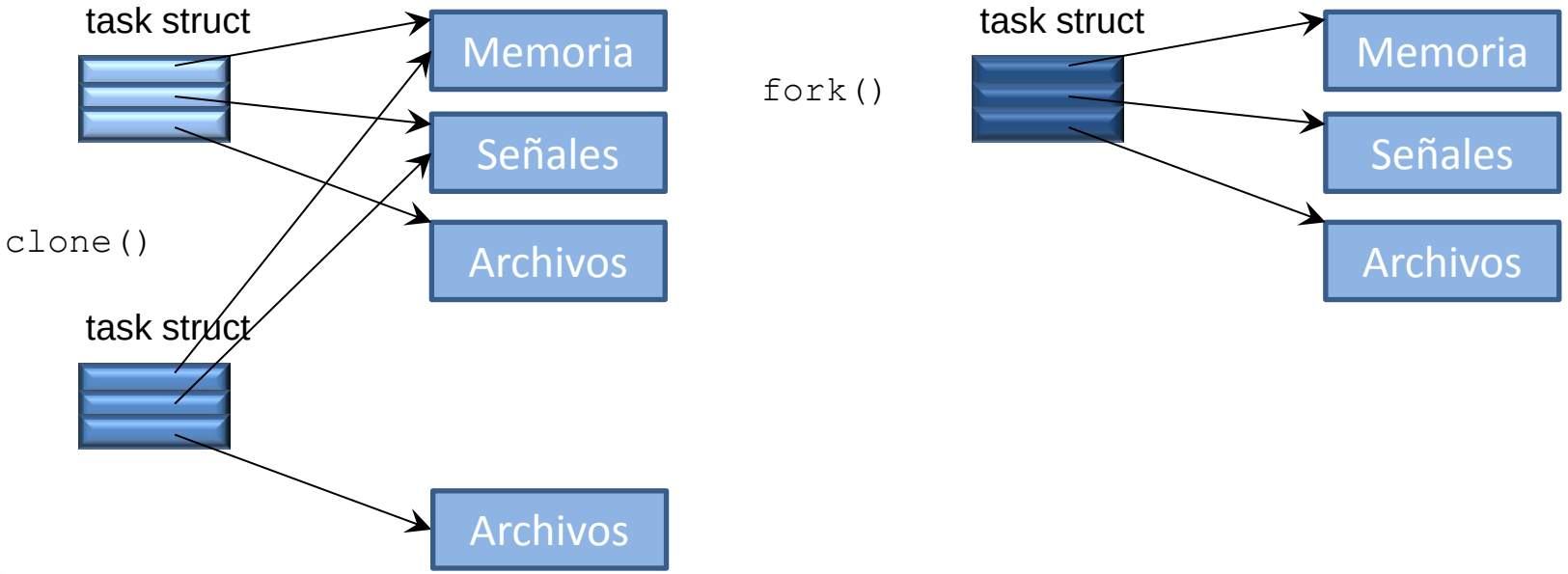
Implementación de la concurrencia: procesos y threads en Linux

- Ni procesos ni threads: tasks
- Primitiva `clone()`
- Permite elegir qué se comparte:
  - Información del sistema de archivos (directorio de trabajo, etc.)
  - Memoria
  - Manejadores de señales
  - Archivos abiertos
- Procesos (`fork()`): no comparten nada

Universidad de los Andes

Implementación de la concurrencia: procesos y threads en Linux

- Representación de tasks



Universidad de los Andes

Implementación de la concurrencia: exclusión mutua

- Deshabilitar interrupciones:
  - Flag de la máquina
  - Solo para monoprocesadores
- Instrucciones atómicas:
  - No interrumpibles
  - g*et-and-set* - algoritmo:

```
int getAndSet ( int * p, int v )
  int t = * p;
  *p = v
  return t
```

Ejemplo IA: `xchg eax, val`

Universidad de los Andes

Implementación de la concurrencia: exclusión mutua

- Ejemplo: exclusión mutua con espera activa

```
get( int * lock )
  while ( getAndSet( lock, true ) )
    liberar procesador
```

```
int lock = false

...
get( &lock )
... acciones
lock = false
```

Universidad de los Andes

Implementación de la concurrencia: exclusión mutua

- Ejemplo: exclusión mutua con espera semi-pasiva
  - Lock con tres elementos:
    - `boolean atom`: espera activa para garantizar atomicidad
    - `boolean ocupado`: el lock propiamente dicho (pasivo)
    - cola de procesos

```
acquire ( lock )
  get ( &lock.atom )
  if ( lock.ocupado )
    lock.cola ← proceso
    lock.atom = false
    dormir proceso
  else lock.ocupado = true
    lock.atom = false
```

```
release( lock )
  get ( &lock.atom )
  if (lock.cola != vacía )
    q ← lock.cola
    lock.atom = false
    despertar q
  else lock.ocupado = false
    lock.atom = false
```