ARCOS Group
Computer Science and Engineering Department
Universidad Carlos III de Madrid

Lesson 3b
process, devices, drivers, and extended services

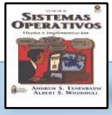
Operating System Design
Degree in Computer Science and Engineering, Double Degree CS&E + BA

Recommended readings

Base


1. Carretero 2007:
   1. Cap.7

Recommended



1. Tanenbaum 2006(en):
   1. Cap.3

1. Stallings 2005(en):
   1. Parte tres

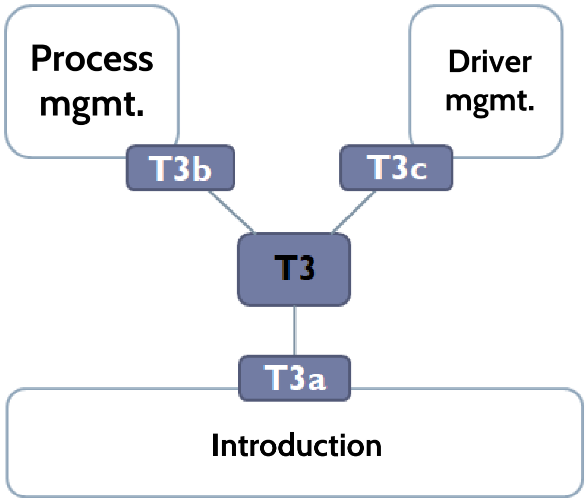
1. Silberschatz 2006:
   1. Cap. Sistemas Module

To remember…

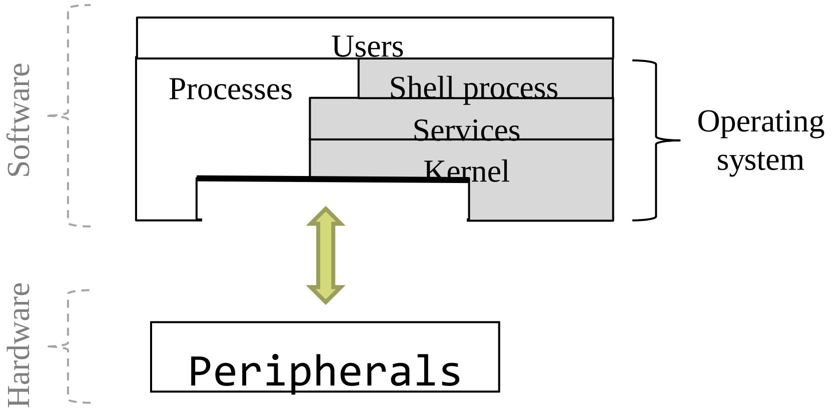
1. To prepare and review the class explanations.
   - Study the bibliography material: only slides are not enough.
   - Ask your doubts.

1. To exercise skills and abilities.
   - Solve as much exercises as possible.
   - Perform the guided laboratories progressively.
   - Build laboratories progressively.

General context…

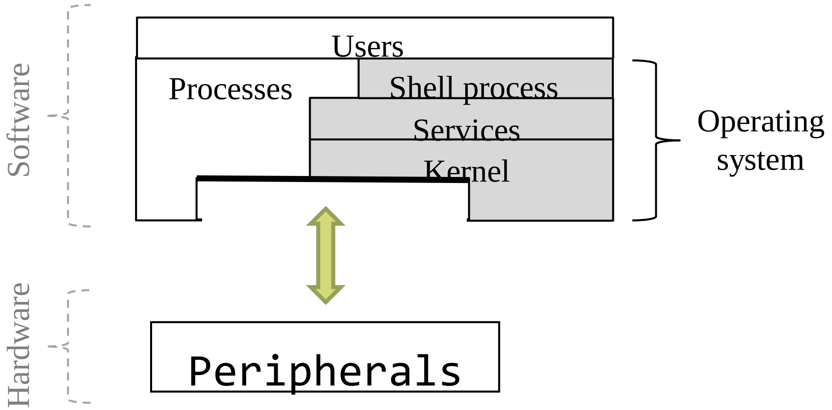


Overview



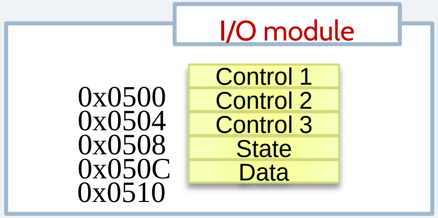
- Introduction
- V.C.S.
- Timing and I.C.S.
- Scheduling

Overview



- Introduction
- V.C.S.
- Timing and I.C.S.
- Scheduling

Impact in the operating system of the device handling



- Control information 1
  - 0: read, 1: write
- Control information 2
  - Memory address
- Control information 3
  - Number of elements
- State information
  - 0: device busy
  - 1: device (data) ready
- Data
  - Device data

- **Direct I/O (Programmed I/O)**
- **Interrupt I/O**
- **DMA I/O**

Example

Direct I/O, Interrupt I/O, and DMA I/O

```
request:
for (i=0; i<100;i++)
{
    // read next
    out(0x500,0) ;

    // wait loop
    do {
       in(0x508,&p.status) ;
    } while (0 == p.status) ;

    // read data
    in(0x50C,&(p.data[i])) ;
}
```

```
request:
    p.counter = 0;
    p.neltos = 100;
    out(0x500, 0) ;
    // V.C.S.
```

```
INT_05:
  in(0x508, &(p.status));
  in(0x50C, &(p.data[p.counter]));
  if ( (p.counter<p.neltos) &&
      (p.status== OK))
  {
      p.counter++ ;
      out(0x500,0) ;  // read
  }
  else { // petitioner process to ready state }
 ret_int # restore registers & return
```

```
request:
    out(0x500, 0) ;
    out(0x504,p.data) ;
    out(0x508,100) ;
    // V.C.S.
```

```
INT_05:
   // read state and data
   in(0x50C, &status) ;

   if (p.status...

   // petitioner process to ready state
  ret_int # restore registers & return
```

Example

Direct I/O, Interrupt I/O, and DMA I/O

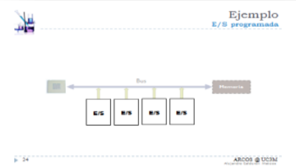
```
request:
for (i=0; i<100;i++)
{
    // read next
    out(0x500,0) ;

    // wait loop
    do {
       in(0x508,&p.status) ;
    } while (0 == p.status) ;

    // read data
    in(0x50C,&(p.data[i])) ;
}
```

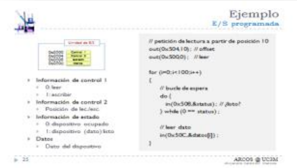
```
request:
    p.counter = 0;
    p.neltos = 100;
    out(0x500, 0) ;
    // V.C.S.

INT_05:
  in(0x508, &(p.status));
  in(0x50C, &(p.data[p.counter]));
  if ( (p.counter<p.neltos) &&
     (p.status== OK))
  {
     p.counter++ ;
     out(0x500,0) ;  // read
  }
  else { // petitioner process to ready state }
ret_int # restore registers & return
```

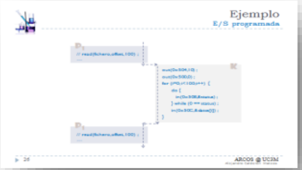
```
request:
    out(0x500, 0) ;
    out(0x504,p.data) ;
    out(0x508,100) ;
    // V.C.S.

INT_05:
  // read state and data
  in(0x50C, &status) ;

  if (p.status...

  // petitioner process to ready state
ret_int # restore registers & return
```

Make better use of waiting times

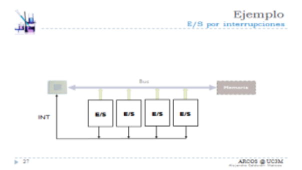
- Direct I/O (Programmed I/O)

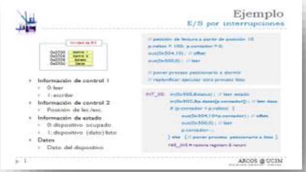


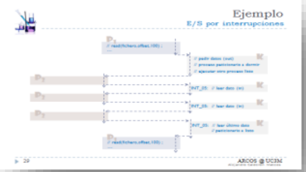




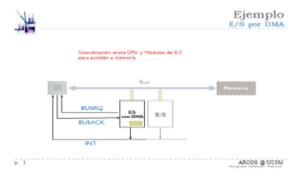
- Interrupt I/O

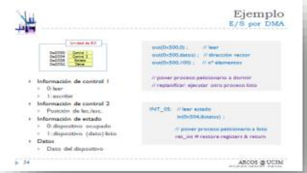


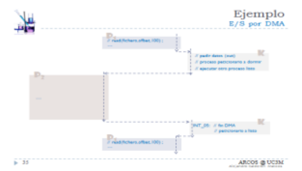




- DMA I/O

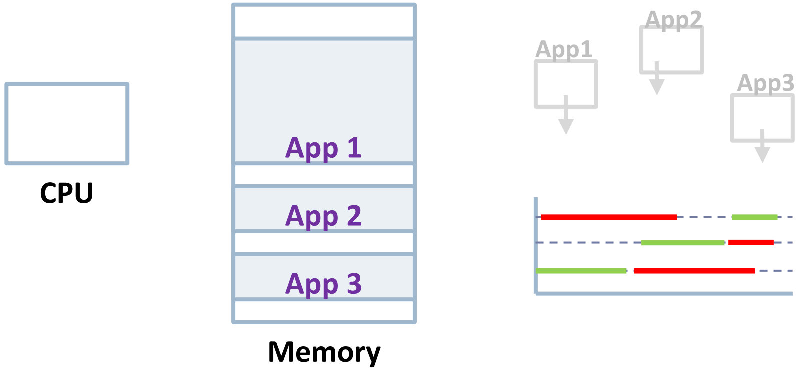






Proposed model

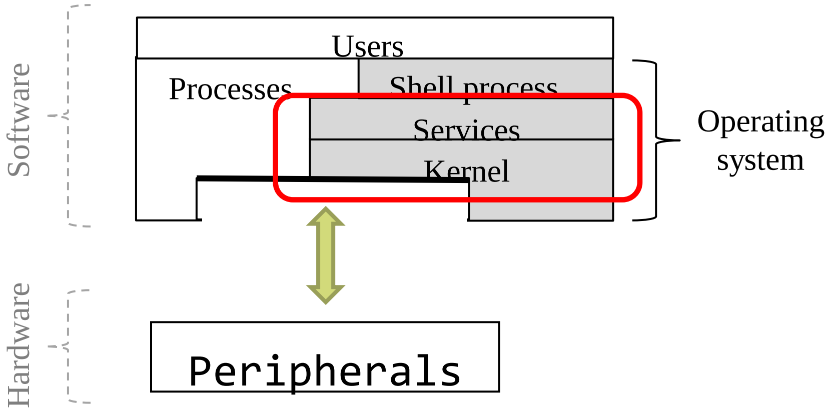
- resource
- **multiprogramming**
  - isolation/sharing
  - process hierarchy
- multitasking
- multiprocess



Multiprogramming

- Several applications loaded in main memory
- If one blocks because request some slow I/O then another is executed until this new one get blocket too
  - Voluntary Context Switching (V.C.S.)
- Efficiency in the use of the processor.
- Degree of multiprogramming = number of applications loaded in main memory

Overview

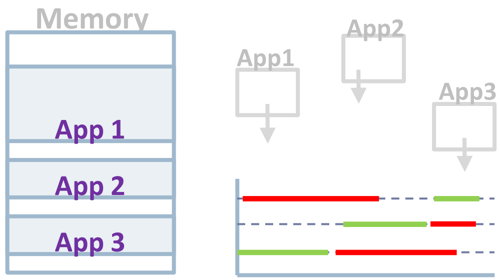


- Introduction
- V.C.S.
- Timing and I.C.S.
- Scheduling

Multiprogramming (data & functions)

Requirements	Information (in data structures)	Functions (Internals, services, and API)
Multiprogramming	• Execution state • Context: CPU registers… • Process list	• Hw./Sw. int. from devices • Scheduler • Create/Destroy/Schedule process

Multiprogramming

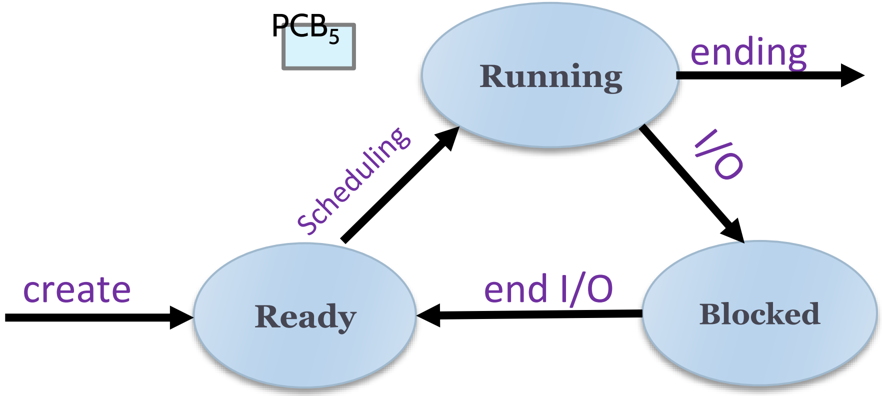


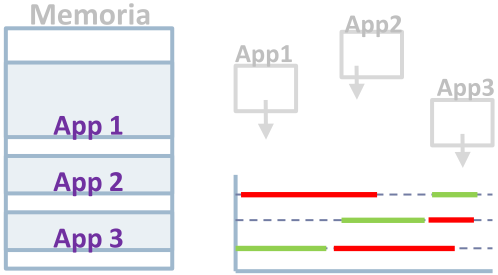
- There will be several applications loaded in memory.
- If an application is blocked by I/O then another will be executed (until it is blocked)
  - Voluntary context switching (V.C.S.)

Multiprogramming (data)

Process states (V.C.S.)

- **State**
- List/Queue
- Context



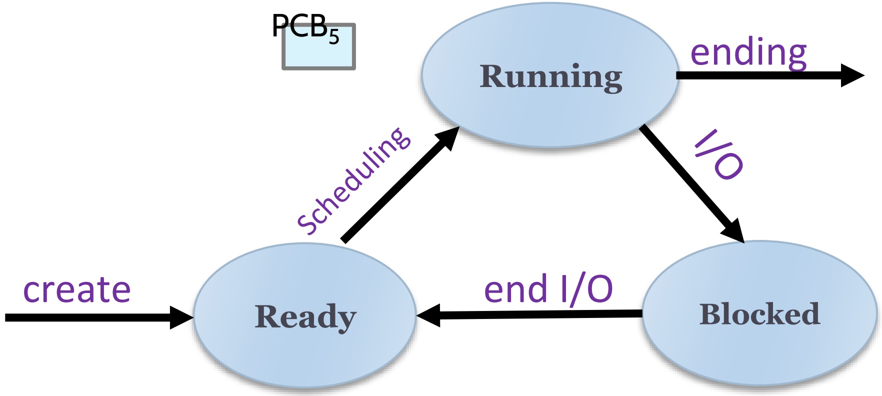


- There will be several applications loaded in memory.
- If an application is blocked by I/O then another will be executed (until it is blocked)
  - Voluntary context switching (V.C.S.)

Multiprogramming (data)

Process states (V.C.S.)

- **State**
- List/Queue
- Context

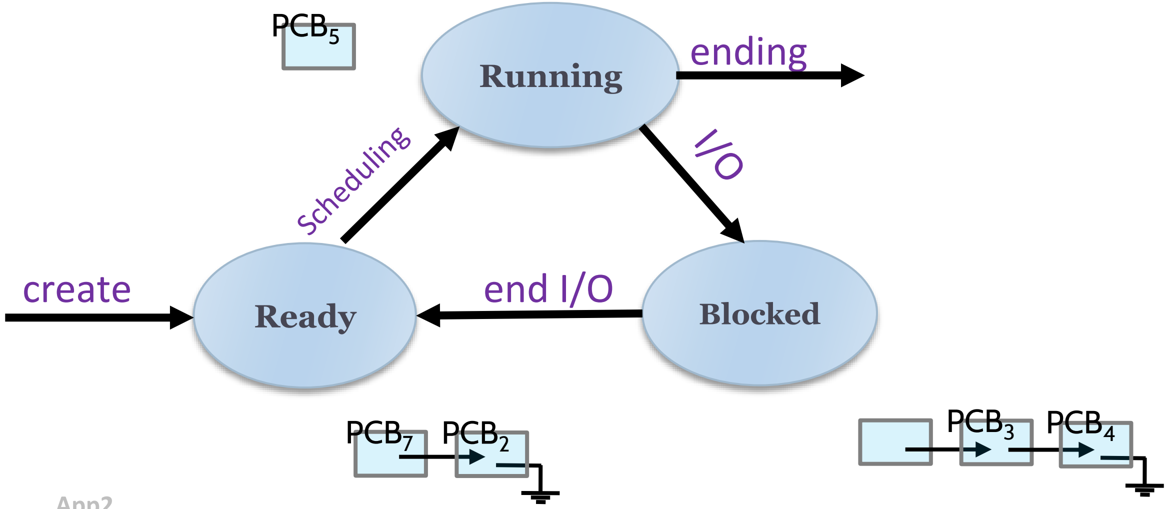


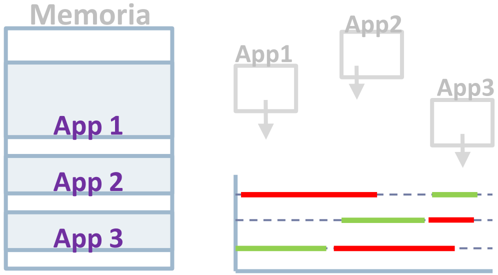
- Running: running in an assigned CPU
- Ready to run: no processor available for the process
- Blocked: waiting for an event

- Suspended and ready: preemption but ready to run
- Suspended and blocked: preemption and waiting for event

Multiprogramming (data)

List/Queues de Processes (V.C.S.)



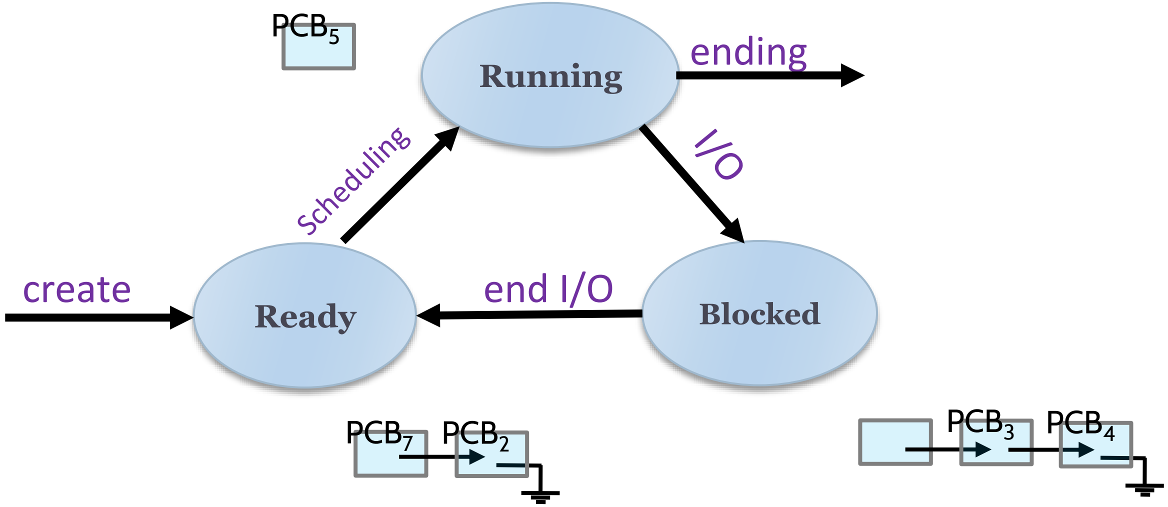


- There will be several applications loaded in memory.
- If an application is blocked by I/O then another will be executed (until it is blocked)
  - Voluntary context switching (V.C.S.)

Multiprogramming (data)

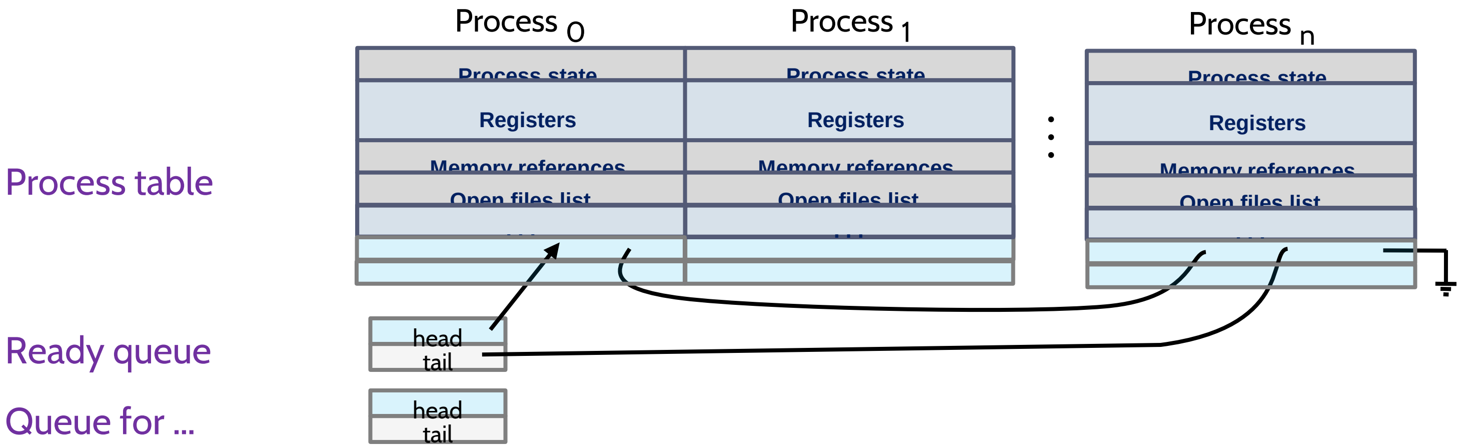
List/Queues de Processes (V.C.S.)

- State
- **List/Queue**
- Context



- Ready queue: Processes waiting for CPU available
- Block queue per "resource": Processes waiting to the completion of a former request done to the associated "resource" (device, lock, etc.)
- One process MUST be, at most, in only one queue

Processes queues (traditional implementation)

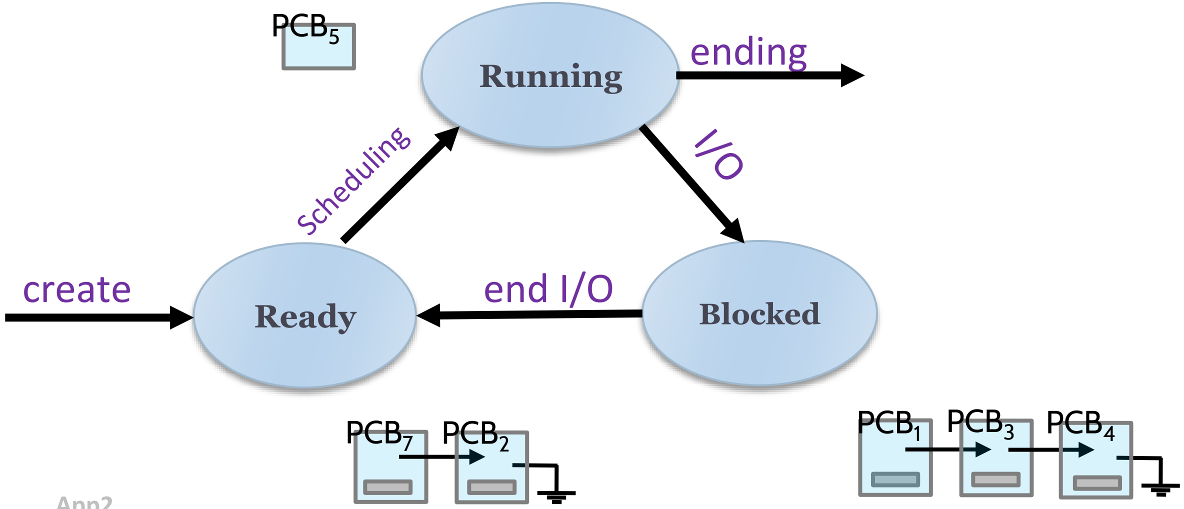


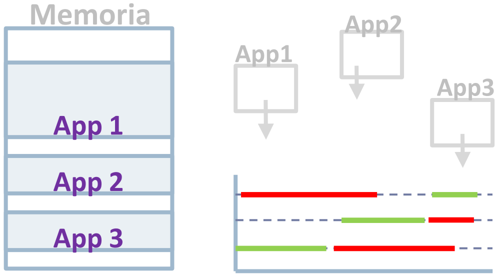
- Ready queue: Processes waiting for CPU available
- Block queue per “resource”: Processes waiting to the completion of a former request done to the associated “resource” (device, lock, etc.)
- One process MUST be, at most, in only one queue

Multiprogramming (data)

Context of a process

- State
- List/Queue
- **Context**

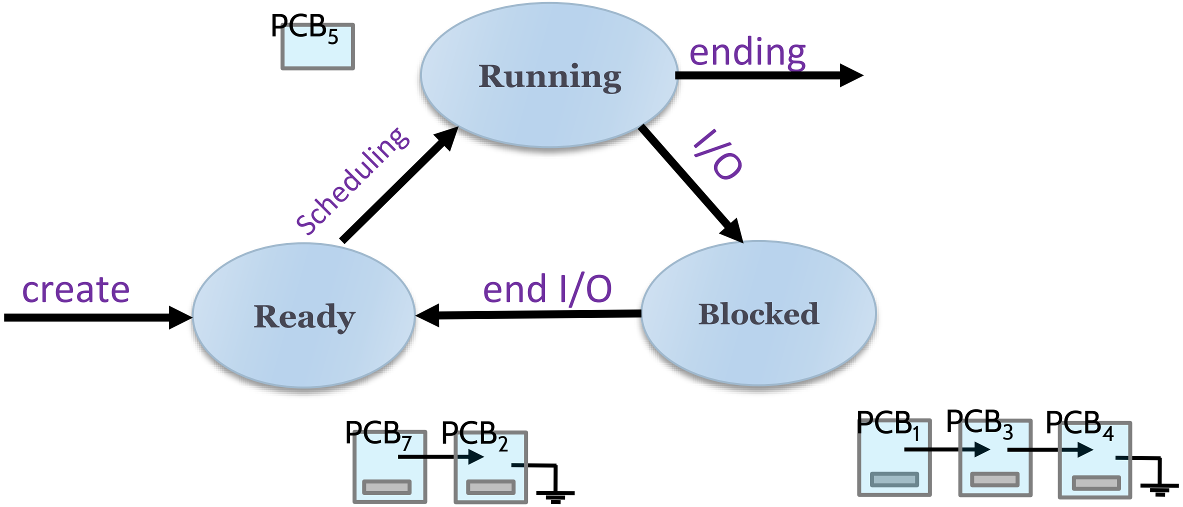




- There will be several applications loaded in memory
- Si una aplicación se bloquea por I/O, entonces se ejecuta otra (hasta que quede bloqueada)
  - Voluntary Context Switching (V.C.S.)

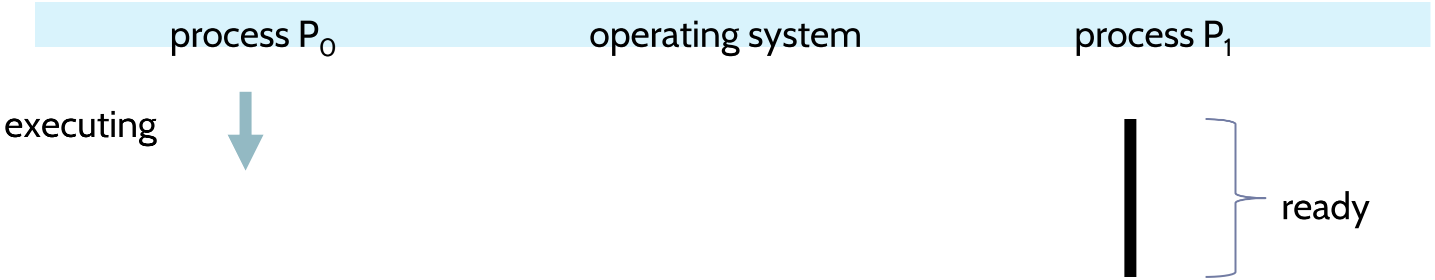
Multiprogramming (data)

Context of a process

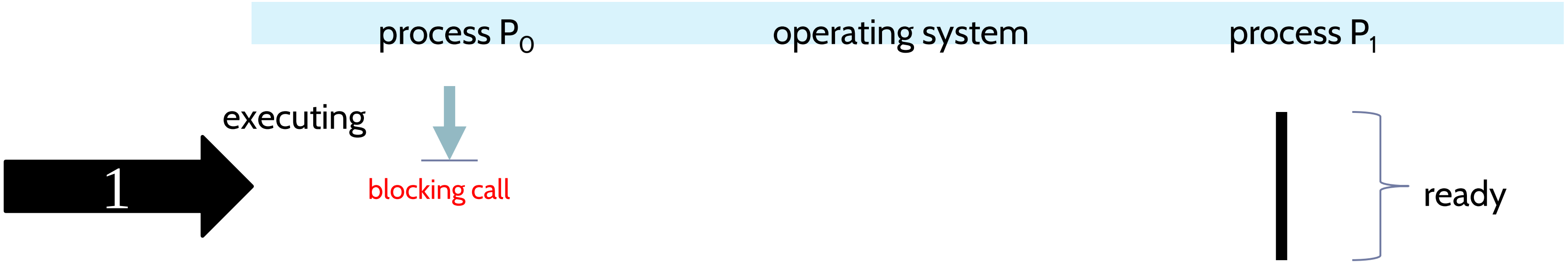


- General purpose registers: PC, SR, etc.
- Specific registers: Floating point, etc.
- Resource references: code pointer, data pointer, etc.

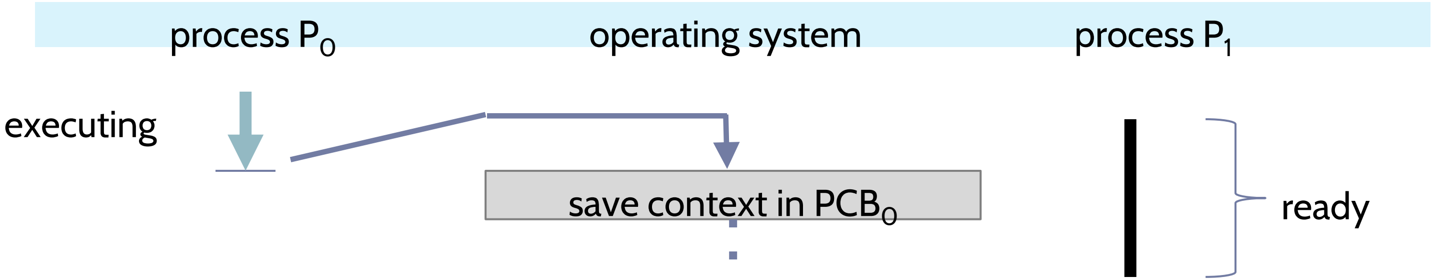
Multiprogramming: example of execution



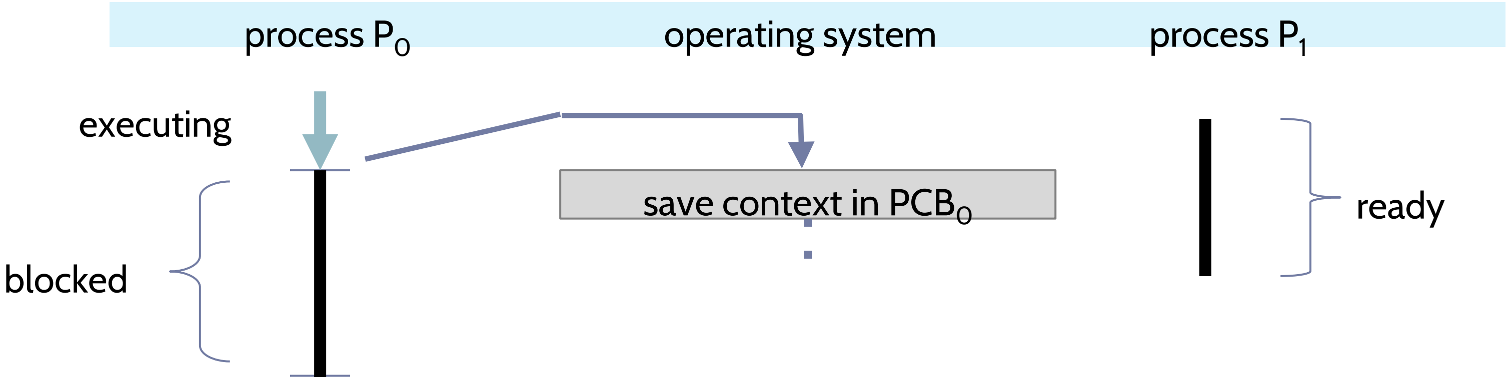
Multiprogramming: example of execution



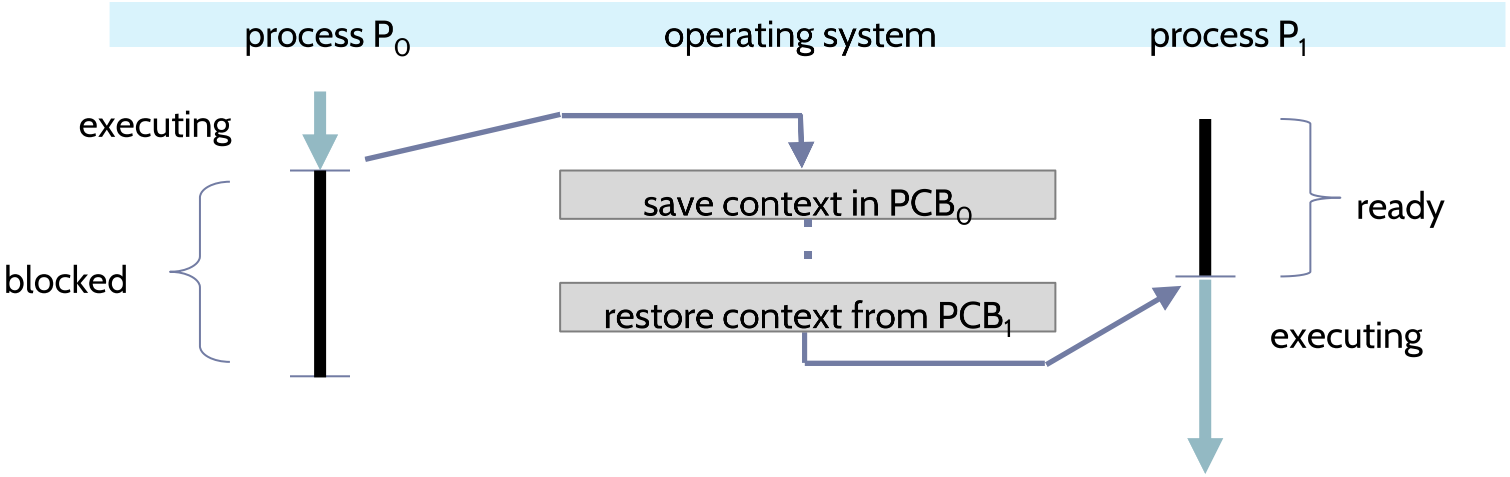
Multiprogramming: example of execution



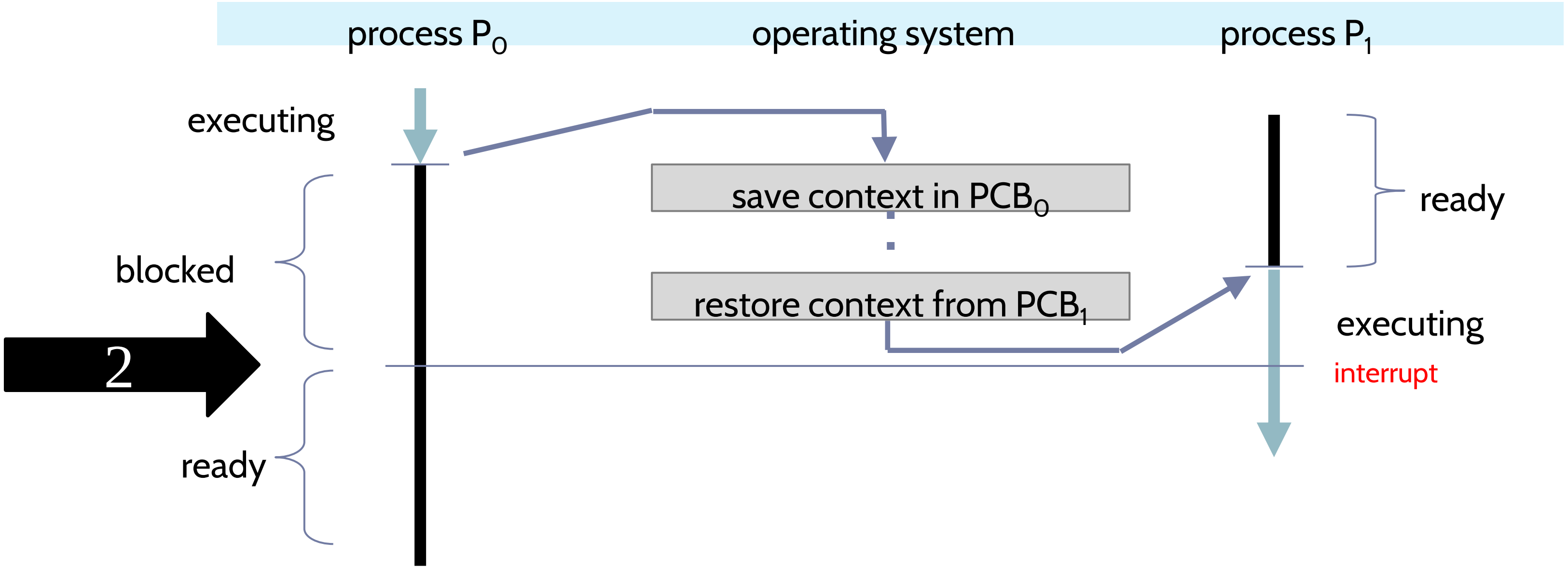
Multiprogramming: example of execution



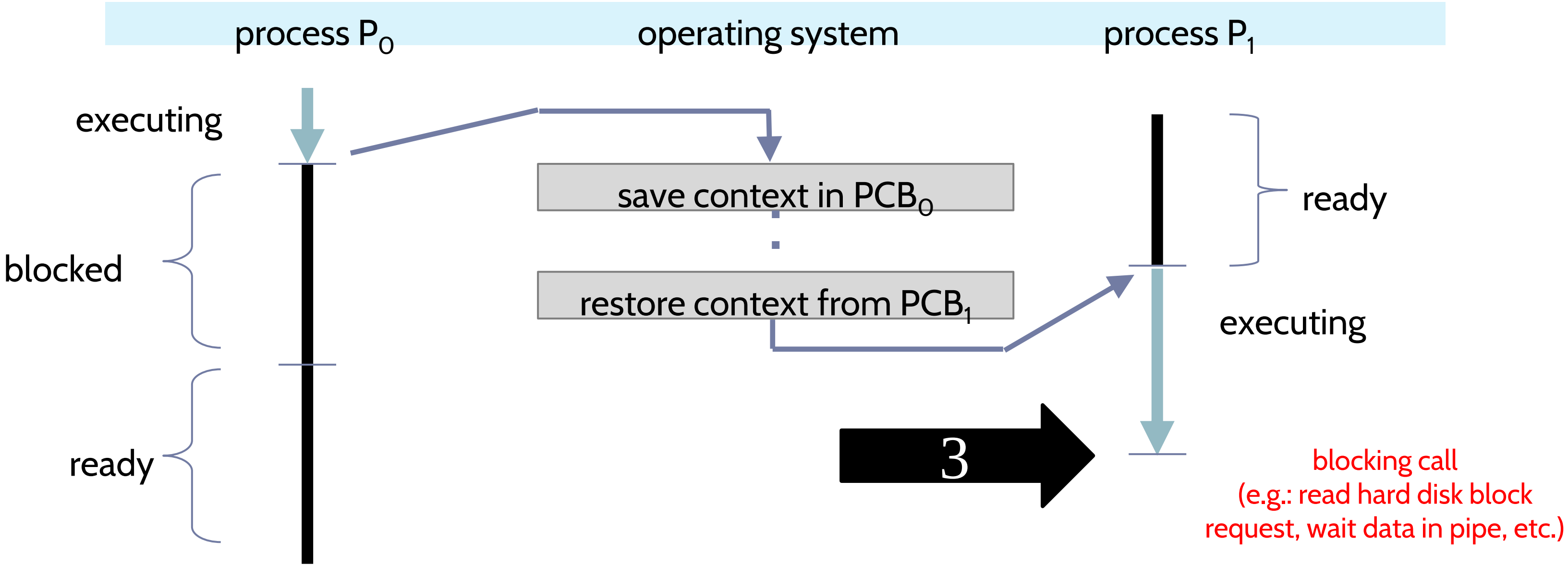
Multiprogramming: example of execution



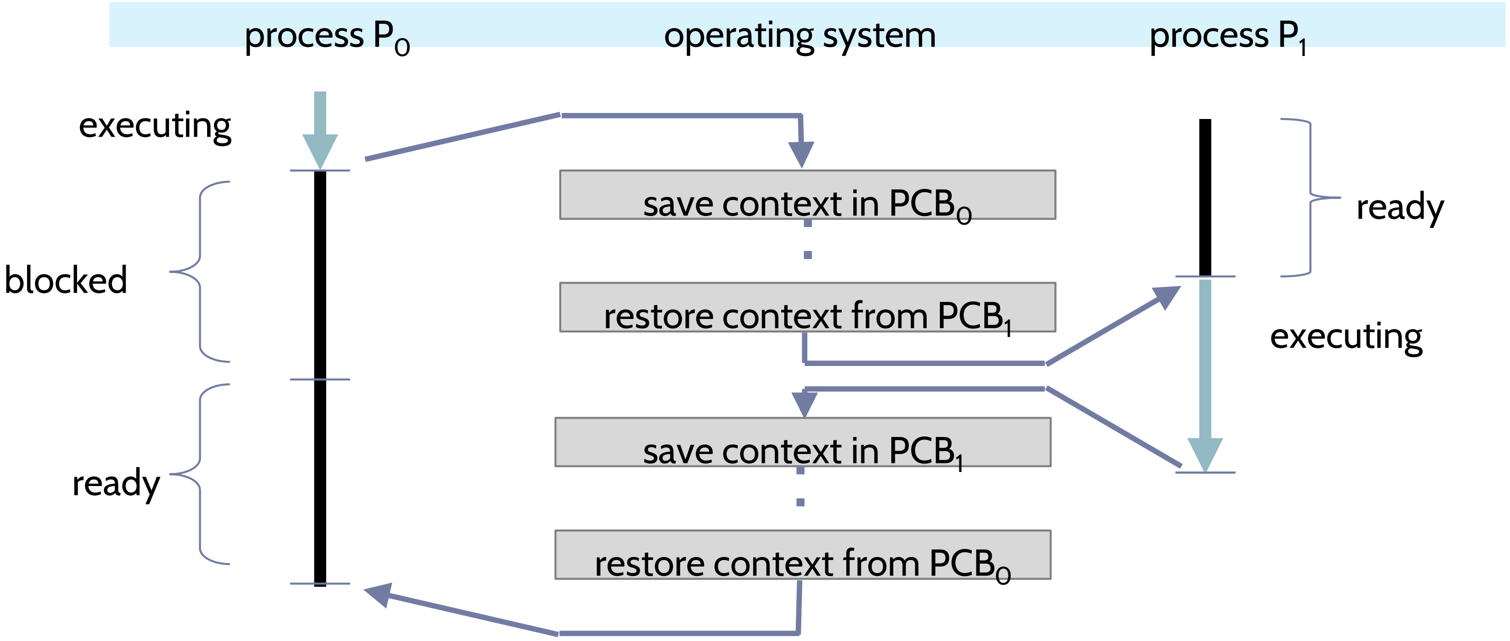
Multiprogramming: example of execution



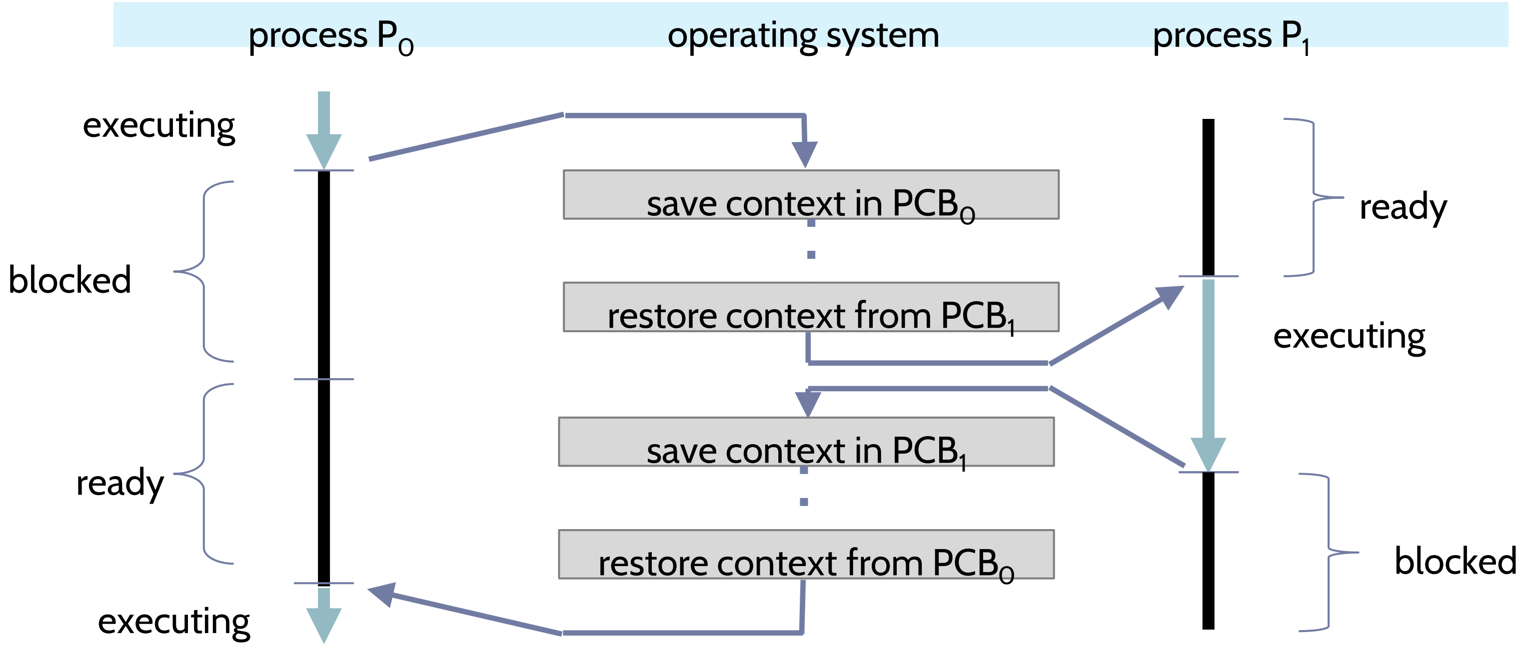
Multiprogramming: example of execution



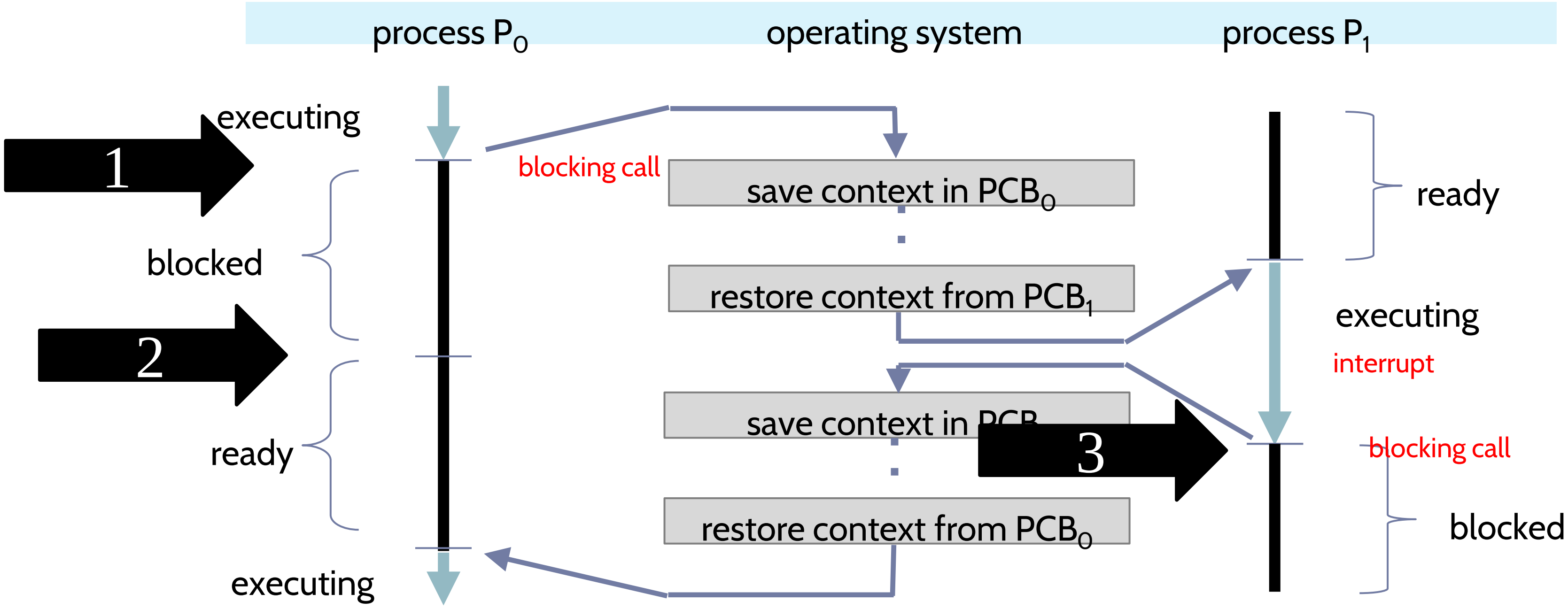
Multiprogramming: example of execution



Multiprogramming: example of execution



Multiprogramming: example of execution

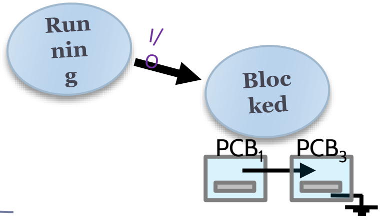


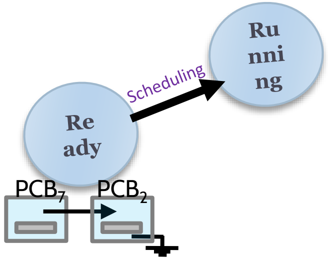
Example pseudocode (P0)

1

```
scheduler()
 • return extract(CPU_ready);
```

```
KEYBOARD_ReadKey()
 • Si (isEmpty(KBD_keys))
       • currentP->state = BLOCKED;
       • Insert(KBD_blocked, currentP);
       • process = currentP;

       • currentP = scheduler();
       • currentP->state = EXECUTION;

       • context_switch( &(process->context),
                               &(currentP->context));

 • return extract(KBD_keys) ;
```





save state in PCB_0

⋮

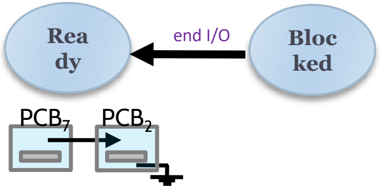
load state to PCB_1

Example pseudocode (P1)

2

Keyboard_Interrupt_Hardware ()
- T = in (KEYBOARD_HW_ID);
- process = insert (T, KBD_keys);
- Insert (Keyboard_Interrupt_Software);
- Activate_Software_Interrupt();

Keyboard_Interrupt_Software ()
- process = first (KBD_blocked);
- IF (process != NULL)
  - remove (KBD_blocked, process);
  - process->state = READY;
  - insert (CPU_ready, process);
- return ok;

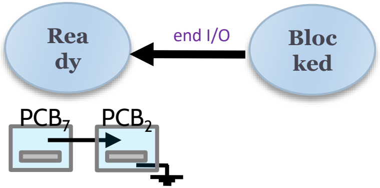


Example pseudocode (P1)

2

```
Keyboard_Interrupt_Hardware ()
  • T = in (KEYBOARD_HW_ID);
  • process = insert (T, KBD_keys);
  • Insert (Keyboard_Interrupt_Software);
  • Activate_Software_Interrupt();
```

```
Keyboard_Interrupt_Software ()
  • process = first (KBD_blocked);
  • IF  (process != NULL)
          • remove (KBD_blocked);
          • process->state = READY;
          • insert (CPU_ready, process);
  • return ok;
```



- One process MUST be, at most, only in one queue:
  - [correct] remove + insert
  - [incorrect] insert + remove

Example pseudocode (P1)

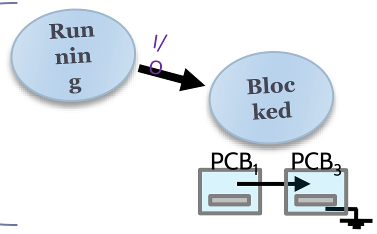
3

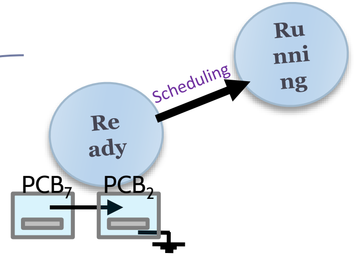
scheduler()
- return extract(CPU_ready);

```
DISK_ReadBlock()
 • IF (disk_block is not in cache)
      • currentP->state = BLOCKED;
      • Insert(DISK_blocked, currentP);
      • process = currentP;

      • currentP = scheduler();
      • currentP->state = EXECUTION;

      • context_switch( &(process->context),
                        &(currentP->context));

 • return extract(DISK_cache, bloque) ;
```

save state in PCB_1

load state to PCB_0





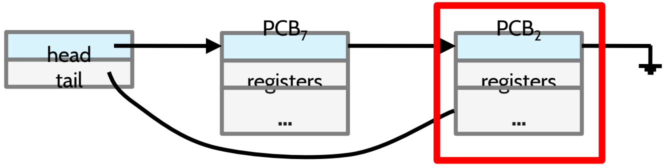
Example pseudocode (P0)

3

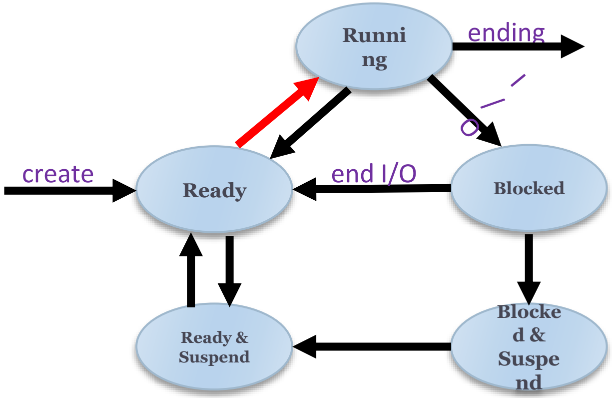
```
KEYBOARD_ReadKey()
 • IF (isEmpty(KBD_keys))
        • currentP->state = BLOCKED;
        • Insert(KBD_blocked, currentP);
        • process = currentP;

        • currentP = scheduler();
        • currentP->state = EXECUTION;

        • context_switch( &(process->context),
                          &(currentP->context));

 • return extract(KBD_keys) ;
```

Scheduler and activator

- **Scheduler**:
  Select the process to be executed among those who are ready to be executed

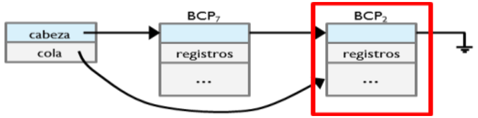


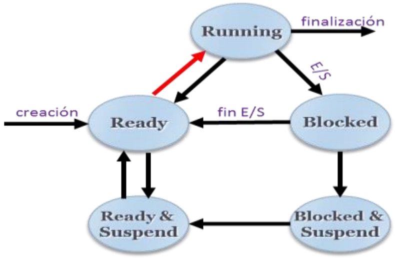
- **Activador**:
  Give control to the process that the scheduler has selected (context switch)



Scheduler and activator

scheduler()
- return extract(CPU_ready);

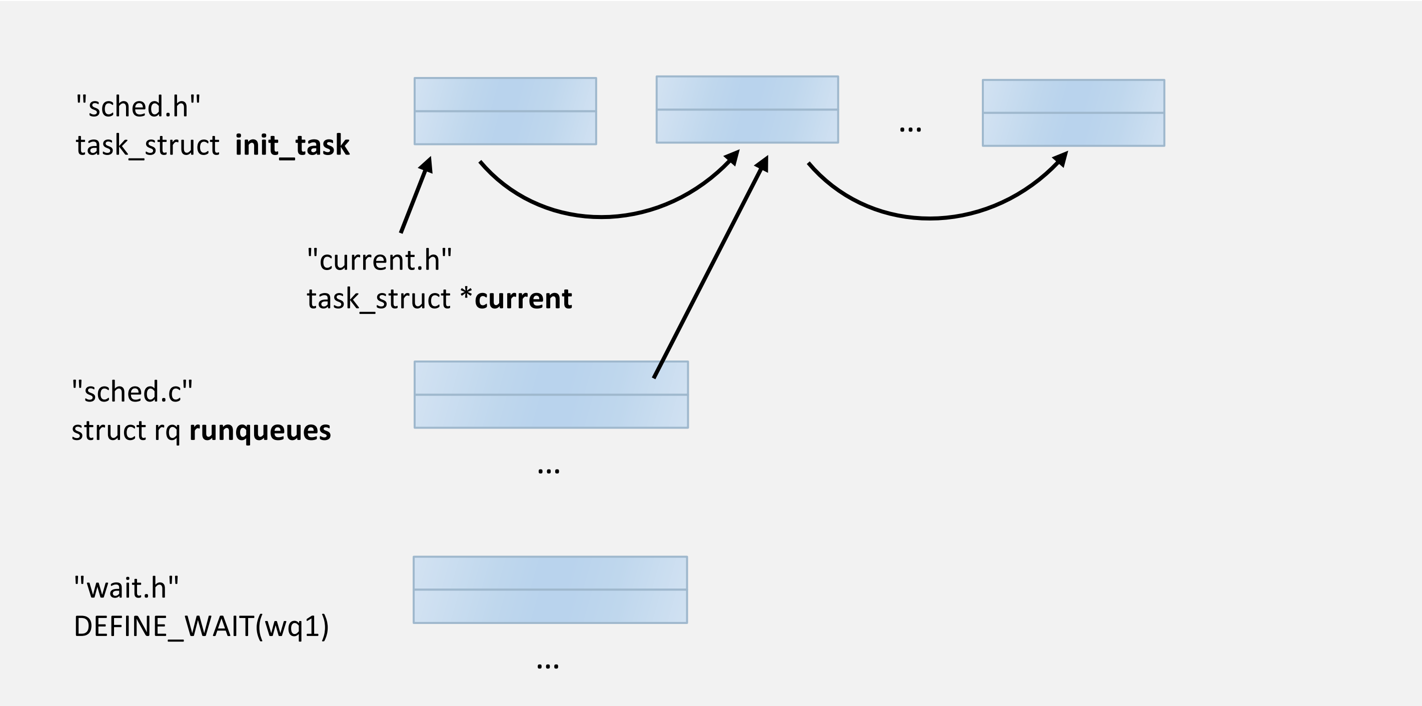




KEYBOARD_ReadKey()

- IF (isEmpty(KBD_keys))
  - currentP->state = BLOCKED;
  - Insert(KBD_blocked, currentP);
  - process = currentP;
  - currentP = **scheduler**();
  - currentP->state = EXECUTION;
  - **activator** (&(process->context), &(currentP->context));
- return extract(KBD_keys) ;

Queues/Lists of processes
Linux



Queues/Lists of processes

Linux

a.
```
atomic_t is_blocking_mode = ATOMIC_INIT(0);
DECLARE_WAIT_QUEUE_HEAD(dso_wq1);
```

b.
```
atomic_set(&is_blocking_mode, 0);
wait_event_interruptible(dso_wq1,
                 (atomic_read(&is_blocking_mode) == 1));
```

c.
```
atomic_set(&is_blocking_mode, 1);
wake_up_interruptible(&dso_wq1);
```

"wait.h"
DEFINE_WAIT(wq1)

…

Queues/Lists of

Linux

- DEFINE_WAIT, DECLARE_WAIT_QUEUE_HEAD(wq)
- wq->flags &= ~WQ_FLAG_EXCLUIFVE
  wq->flags |= WQ_FLAG_EXCLUIFVE

a. atomic_t is_blocking_mode = ATOMIC_INIT(0);
   DECLARE_WAIT_QUEUE_HEAD(dso_wq1);

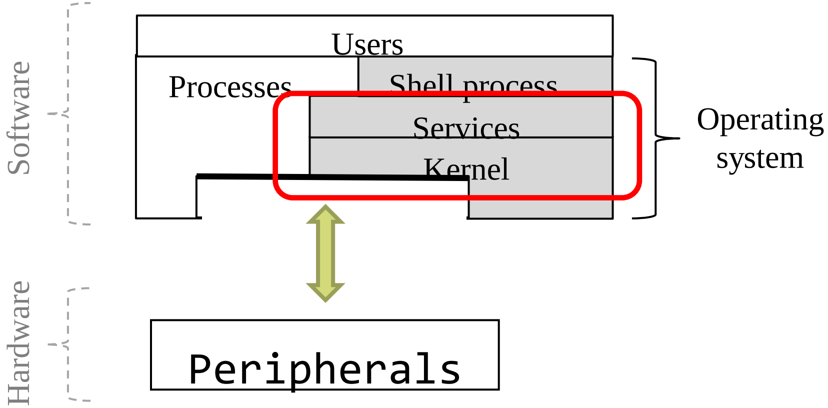
b. atomic_set(&is_blocking_mode, 0);
   wait_event_interruptible(dso_wq1,
   (atomic_read(&is_blocking_mode) == 1));

c. atomic_set(&is_blocking_mode, 1);
   wake_up_interruptible(&dso_wq1);

wait_event, wait_event_interruptible (wq, condition)
wait_event_timeout,
wait_event_interruptible_timeout (wq, condition, timeout)

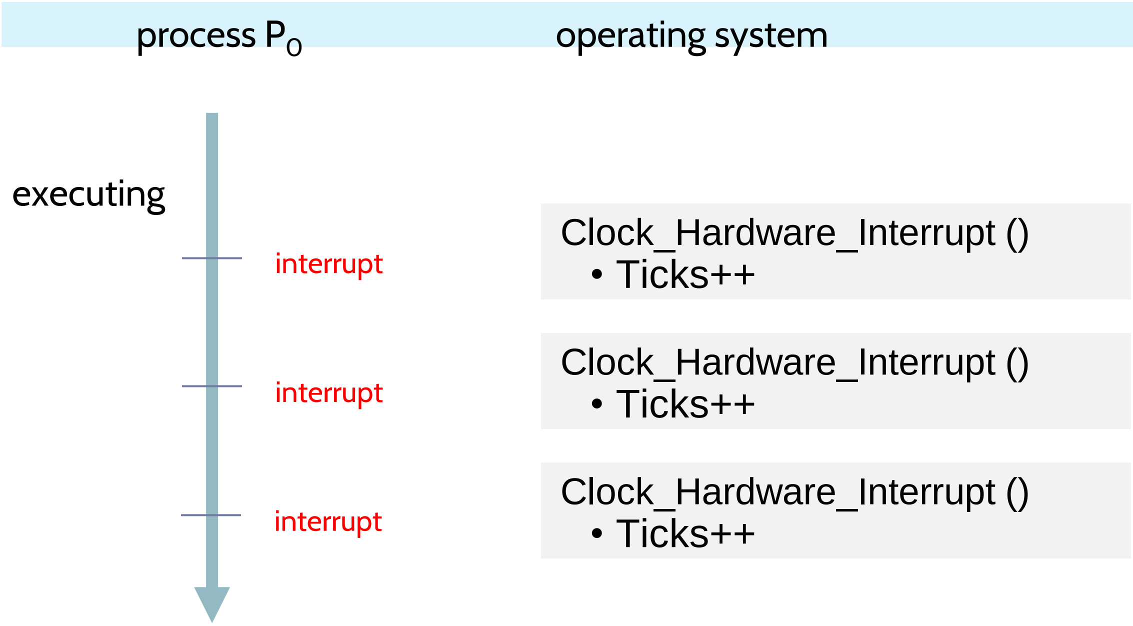
wake_up, wake_up_nr, wake_up_all, wake_up_interruptible,
wake_up_interruptible_nr, wake_up_interruptible_all,
wake_up_interruptible_sync, wake_up_locked(queue)

Overview



- Introduction
- V.C.S.
- **Timing and I.C.S.**
- Scheduling

Clock handler: basic behavior



Timing
Linux

- void process_timeout (unsigned long __data) {
      wake_up_process((task_t *)__data);
  }

- timespec t;
  unsigned long expire;
  struct timer_list timer;

Timing
Linux

```
expire = timespec_to_jiffies(&t) + 1 + jiffies;
init_timer(&timer);
timer.expires = expire;
timer.data = (unsigned long) current;
timer.function = process_timeout;
add_timer(&timer);
current->state = TASK_INTERRUPTIBLE;
schedule(); /* ejecutar mientras otro process */
del_singleshot_timer_sync(&timer);
```

Multitasks (data & functions)

Requirements	Information (in data structures)	Functions (Internals, services, and API)
Resources	• Areas of memory (code, data and stack) • Open files • Activated signals	• Several internal functions • Several service function for memory, files, etc.
Multiprogramming	• Execution state • Context: CPU registers… • Process list	• Hw./Sw. int. from devices • Scheduler • Create/Destroy/Schedule process

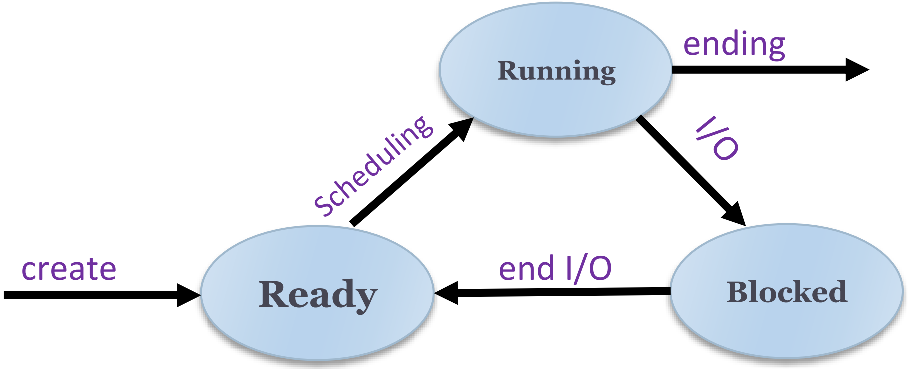
○ Insolation / Sharing	• Message passing
	  • Cola de mensajes de recepción
	• Memory compartida
	  • Zones, locks and conditions
		• Send/Receive message and management of the message queue
		• API for concurrency control (access to data structures)

○ Hierarchy of processes	• Family relationship • Related sets of processes • Processes from the same session	• Clonar/Cambiar imagen de proceso • Associate process and leader selection
Multitasking	• Quantum restante • Priority	• Hw./Sw. int. from clock device • Scheduler • Create/Destroy/Schedule process
Multiprocess	• Affinity	• Hw./Sw. int. from clock device • Scheduler • Create/Destroy/Schedule process

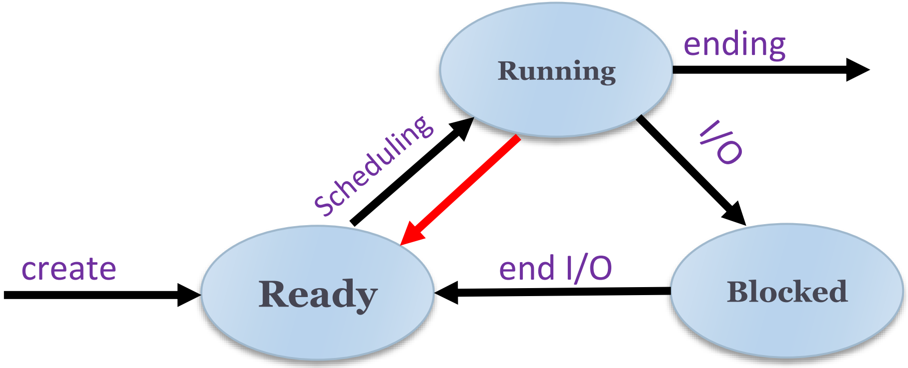
States of a process

- **State**
- List/Queue
- Context

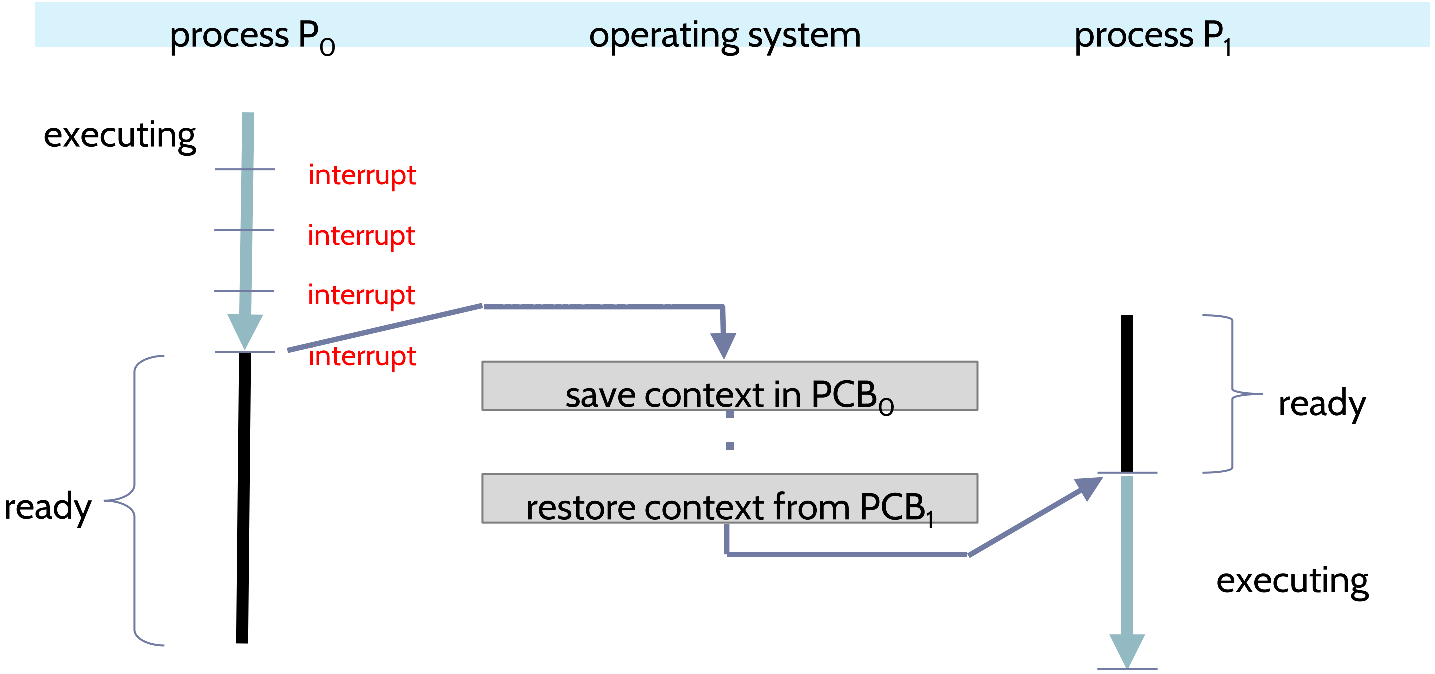
V.C.S.



V.C.S. + I.C.S.



Clock handler: with V.C.S. + I.C.S.



Example pseudocode (P0)

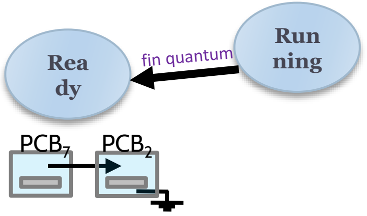
```
Clock_Hardware_Interrupt ()
  • Ticks++;
  • Insert (Clock_Schedule_Quantum);
  • Activate_Software_Interrupt();
```

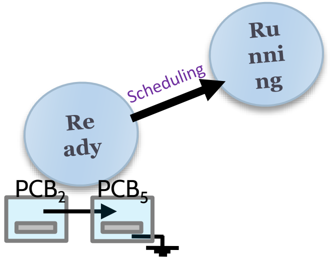
```
Clock_Schedule_Quantum ()
  • currentP->quantum = currentP->quantum - 1;
  • IF  (currentP->quantum == 0)
        • currentP->state = READY;
        • currentP->quantum  = QUANTUM;
        • insert (CPU_ready, currentP);
        • process = currentP;

        • currentP = scheduler();
        • currentP->state = EXECUTION;
        • context_switch(
              &(process->context),
              &(currentP->context));
  • return ok;
```

```
scheduler()
  • return extract(CPU_ready);
```

save state in PCB_0

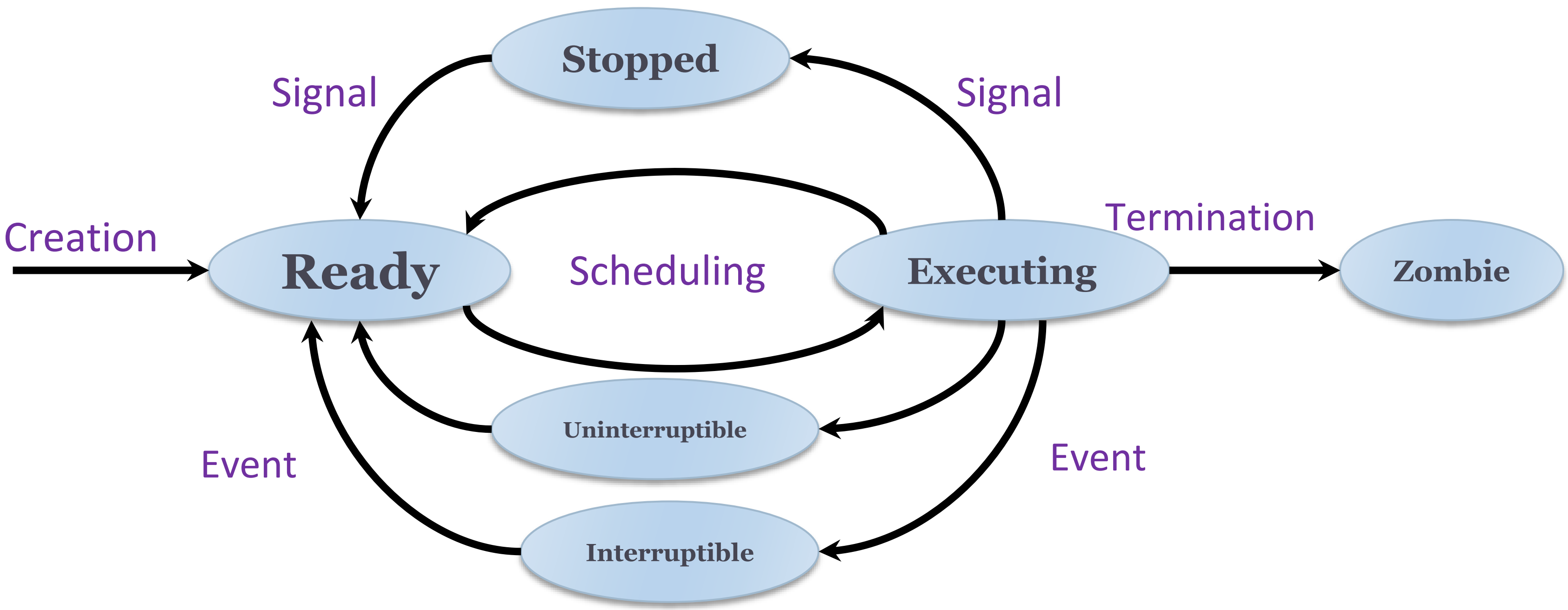
load state in PCB_1





Process states

Linux



Multiprocess

Requirements	Information (in data structures)	Functions (Internals, services, and API)
Resources	• Areas of memory (code, data and stack) • Open files • Activated signals	• Several internal functions • Several service function for memory, files, etc.
Multiprogramming	• Execution state • Context: CPU registers… • Process list	• Hw./Sw. int. from devices • Scheduler • Create/Destroy/Schedule process

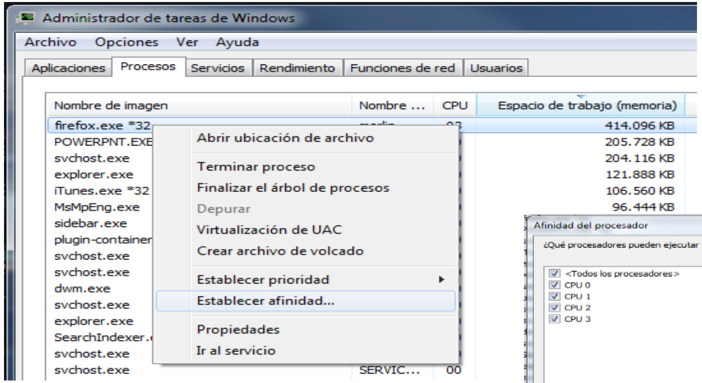
o Insolation / Sharing	• Message passing
	  • Cola de mensajes de recepción
	• Memory compartida
	  • Zones, locks and conditions
		• Send/Receive message and management of the message queue
		• API for concurrency control (access to data structures)

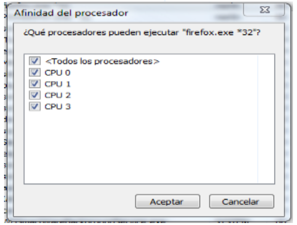
o Hierarchy of processes	• Family relationship • Related sets of processes • Processes from the same session	• Clonar/Cambiar imagen de proceso • Associate process and leader selection
Multitasking	• Quantum restante • Priority	• Hw./Sw. int. from clock device • Scheduler • Create/Destroy/Schedule process
Multiprocess	• Affinity	• Hw./Sw. int. from clock device • Scheduler • Create/Destroy/Schedule process

Alejandro Calderón Mateos

Multiprocess

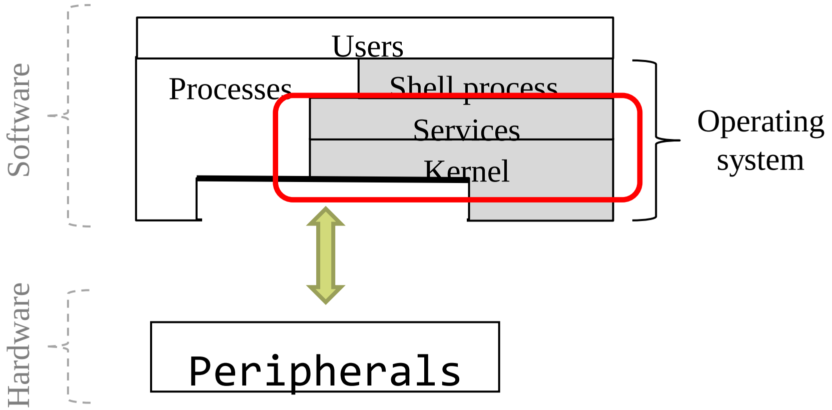
- **Afinity**:
  - Processes have affinity to a CPU: «better to come back to the same CPU»





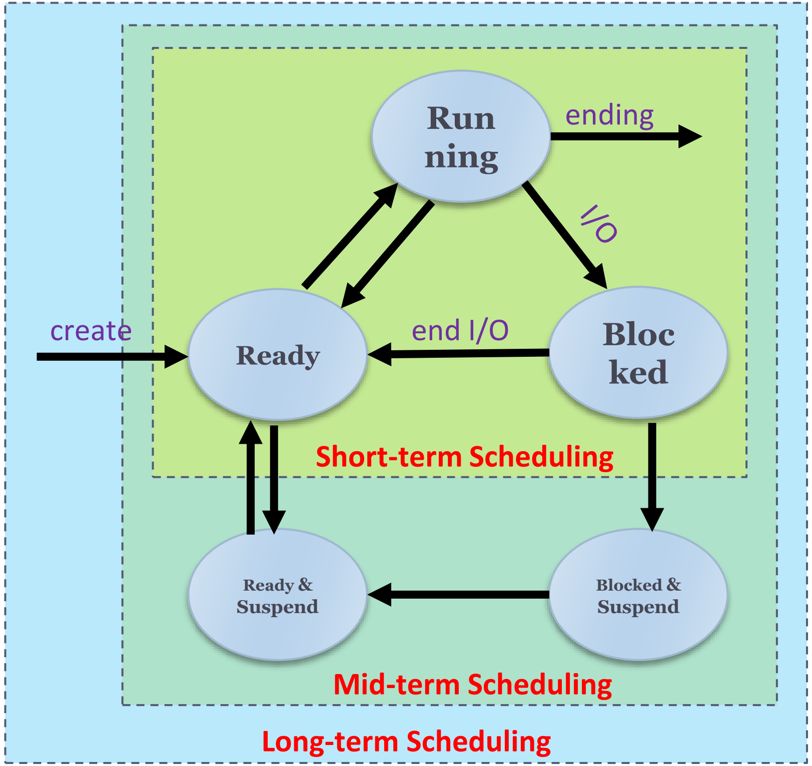
- **Symmetry**:
  - Some processes need to be executed in a particular CPU with specific capabilities.

Overview



- Introduction
- V.C.S.
- Timing and I.C.S.
- **Scheduling**

Process Scheduling
Scheduling **levels**



- Long-term
  - Add more processes to be executed
  - Low frequently invoked
    - Slower task
- Mid-term
  - Load more processes to RAM
- Short-term
  - What process in in CPU
  - High frequently invoked
    - Fast

Process Scheduling

goals of scheduling algorithms (by system)

- All systems:
  - Equitable – offers each process an equal part of the CPU
  - Expeditive – compliance with the policy of distribution
  - Balanced – keep all parts of the system occupied
- Batch systems:
  - Productivity – maximize the number of jobs per hour
  - Waiting time – minimize the time between issuance and termination of work
  - CPU usage – keep the CPU busy all the time
- Interactive systems:
  - Response time – respond to requests as quickly as possible
  - Adjusted – meet the expectations of the users
- Real-time systems:
  - Compliance with deadlines – avoid loss of data (when it is needed)
  - Predictable – avoid degradation of quality in multimedia systems

Process Scheduling

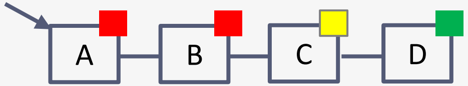
characteristics of scheduling algorithms (1/2)

- *Preemption*:
  - Without:
    - One process keeps CPU while it wants.
    - Volunteer Context Switching (V.C.S.)
    - [a/d] One process can block the full system but it easy to share resources
    - Windows 3.1, Windows 95 (16 bits), NetWare, MacOS 9.x.
  - With:
    - Some clock periodically interrupt:
      - when the assigned quantum expires, another process is executed
    - (It adds) Involuntary Context Switching (I.C.S.)
    - [a/d] Better interactivity but it needs concurrency control mechanisms
    - AmigaOS (1985), Windows NT-XP-Vista-7, Linux, BSD, MacOS X

Process Scheduling

characteristics of scheduling algorithms (2/2)

- Classification of elements in queues:
  - By priority
  - By type
    - CPU-bound (more 'burst' of time using CPU)
    - IO-bound (more 'burst' of time waiting I/O)



- *CPU-aware*:
  - Affinity:
    - Processes have *affinity* to one CPU: «better come back to the same CPU»
  - Symetry:
    - Processes are executed in some CPU with specific capabilities

Process Scheduling
Main scheduling algorithms (1/3)

- **Round Robin**:
  - Rotary assignation of the processor
    - A maximum processor time is assigned (quantum or quantum)

•Larga:
- low context switchs
- Worst interactivity

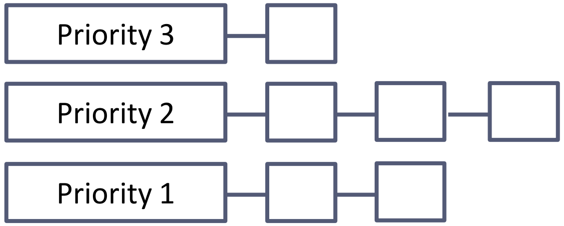
  - Equitable but interactive:
    - Better by UID than by process
    - Linux:
      - Introduced in 11/2010 one kernel patch that creates a task group by TTY in order to improve the interactivity in high loaded systems.
      - 224 lines of codes that modifie the kernel scheduler and first tests shows that the average latency drops to 60 times (1/60).
  - Used in timeshare systems

http://www.phoronix.com/scan.php?page=article&item=linux_2637_video&num=1

Process Scheduling

Main scheduling algorithms (2/3)

- Priority:
  - CPU assigned to the highest priority process
    - It can be combined with Round-Robin. Example with three priority classes.



  - Characteristics:
    - fixed priorities: problem of starvation
    - Not fixed: use of some aging algorithm
  - Use in timeshare systems with real-time aspects

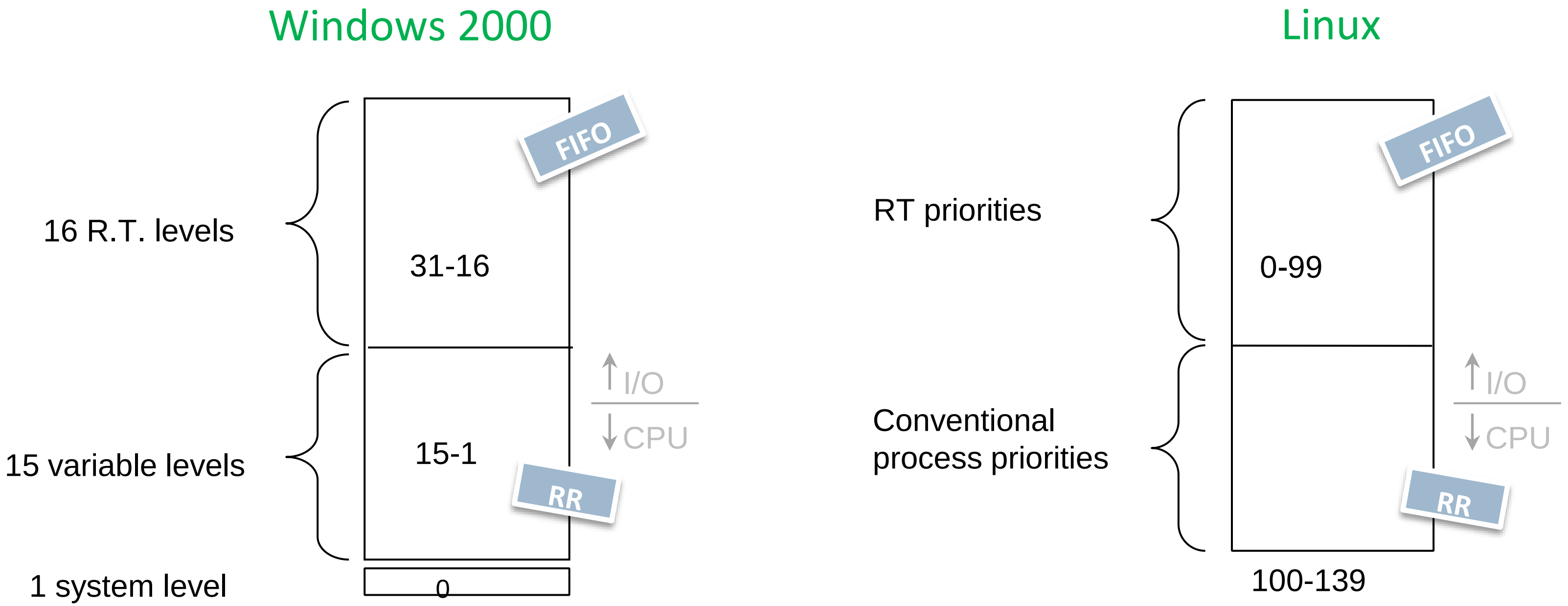
Main scheduling algorithms (3/3)

- First the shortest work:
  - Given a set of tasks that is known its total execution time, they are ordered from the lowest to the longest.
  - Features:
    - [a] Produces the shortest response time (in average)
    - [d] Penalize long works.
  - Used in batch systems.
- FIFO:
  - Execution by the strict order of arrival.
  - Features:
    - [a] Simple to be implemented.
    - [d] Penalizes priority tasks.
  - Used in batch systems.

Policy vs mechanism

- Divide what can be done from how it can be done
  - Usually one process knows which one is the high priority thread, the one with more I/O requests, etc.
- To use parametrize scheduling algorithm
  - Mechanism is in the kernel
- Parameters given by users processes
  - Policy set by user processes

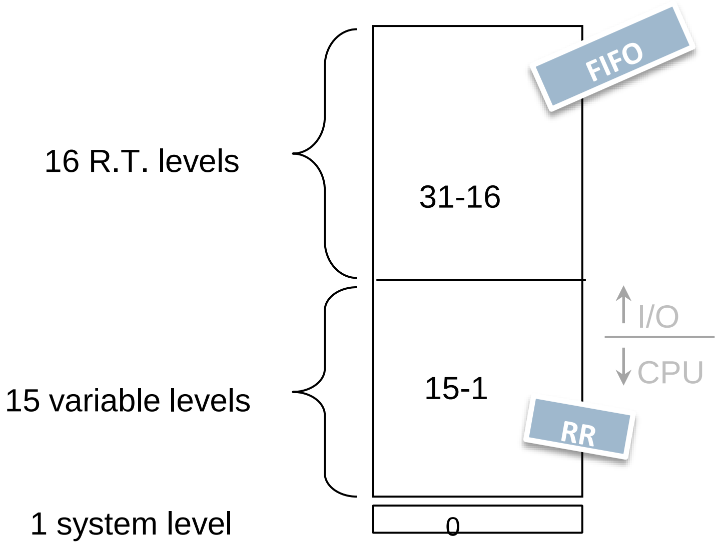
Multipolicy scheduling
Windows 2000 y Linux

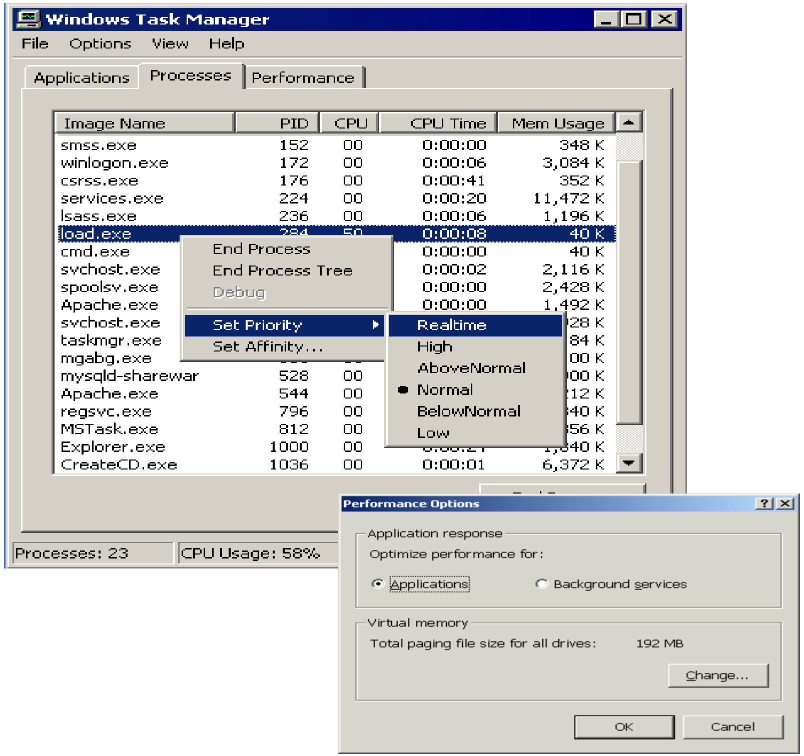


Multipolicy scheduling

Windows 2000

Windows 2000





ARCOS Group
Computer Science and Engineering Department
Universidad Carlos III de Madrid

Lesson 3b
process, devices, drivers, and extended services

Operating System Design
Degree in Computer Science and Engineering, Double Degree CS&E + BA