

# Sistemas Operativos 1

## **Procesos** Estados e Hilos

Edwin Salvador

5 de mayo de 2017

Clase 3

## 1 Estados

- Modelos de estados de procesos
- Swapping

## 2 Procesos e hilos

- Funcionalidad de los Hilos
- Hilos de nivel de usuario y de nivel núcleo
- Programación Concurrente en Java

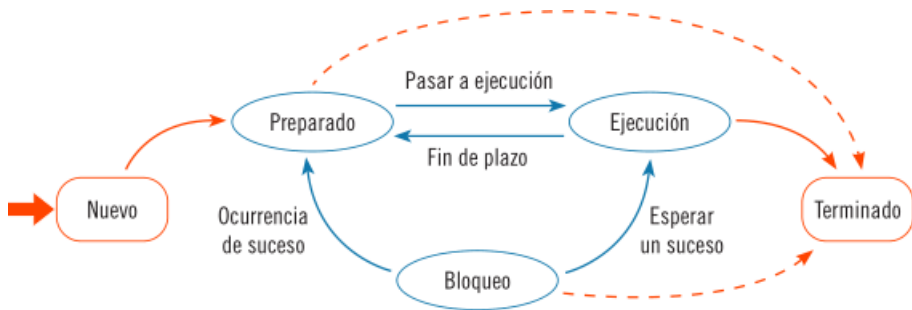
## 3 Procesos en Windows y Ubuntu

# Estados de un proceso

Los procesos evolucionan de manera cíclica entre diferentes estados. Estos estados pueden ser (los más comunes):

- **Nuevo** procesos que acaban de ser incluidos y el SO aún no los admite para ser ejecutados.
- **Listo** procesos que cuentan con todos los recursos para comenzar o seguir su ejecución.
- **Ejecución** proceso que tiene el control del procesador. En sistemas con un solo procesador solo un proceso puede estar en este estado.
- **Bloqueado** procesos en espera de algún recurso o evento para continuar su ejecución.
- **Terminado** procesos excluidos por el SO del grupo de procesos ejecutables. Un proceso pasa a terminado:
  - terminación normal
  - error irrecuperable
  - terminación por otro proceso autorizado.

# Estados de un proceso



# Contenido I

## 1 Estados

- Modelos de estados de procesos
- Swapping

## 2 Procesos e hilos

- Funcionalidad de los Hilos
- Hilos de nivel de usuario y de nivel núcleo
- Programación Concurrente en Java

## 3 Procesos en Windows y Ubuntu

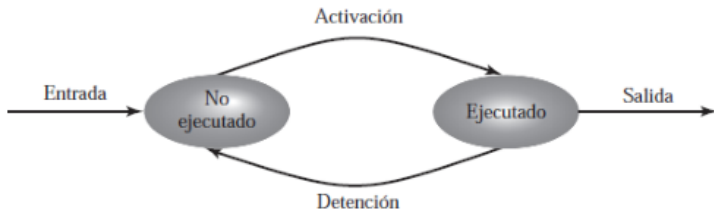
# Modelos de estados de procesos

- La responsabilidad principal del SO es controlar la ejecución de los procesos, por lo tanto debe determinar el patrón de entrelazado para la ejecución y asignación de recursos a estos procesos.
- El primer paso para el diseño de un SO es describir el comportamiento que se desea que tengan los procesos.
- A continuación veremos dos modelos de estados de procesos: un modelo de dos estados y uno de cinco estados.

# Modelo de procesos de dos estados

- Si tenemos en cuenta que en un instante dado un proceso está siendo **ejecutado o no**, entonces podemos decir que un proceso estará en dos posibles estados **Ejecutando** y **No Ejecutando**.
- Cuando el SO, crea un proceso, crea un BCP para este nuevo proceso e inserta el proceso en el sistema en estado *No Ejecutando*. Entonces, en este punto el proceso ya existe, es conocido por el SO pero está esperando su oportunidad de ser ejecutado.
- Cada cierto tiempo, el proceso actualmente en ejecución se interrumpirá y pasará al estado *No Ejecutando*.
- El **activador** seleccionará otro proceso para ejecutar y este pasará al estado *Ejecutando*.
- Los procesos en estado *No Ejecutando* deben ser almacenados en una especie de cola que contiene punteros a los BCP de cada proceso en este estado.

# Modelo de procesos de dos estados



(a) Diagrama de transiciones de estados



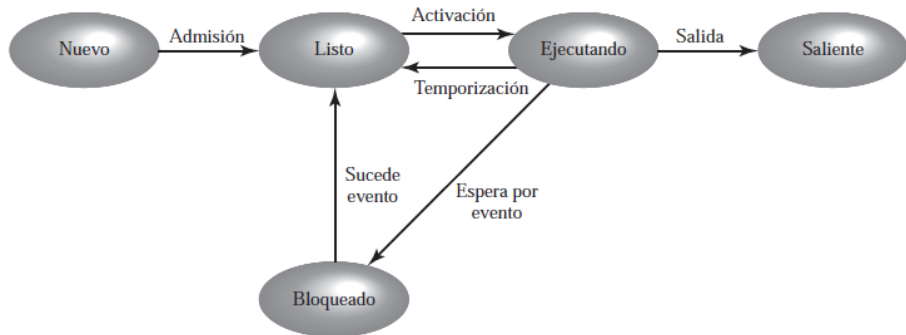
# Modelo de procesos de dos estados

- Este modelo de gestión de colas sería efectiva si todos los procesos estuviesen siempre preparados para ejecutar.
- Esta cola es de tipo FIFO por lo tanto no toma en cuenta que procesos llevan más tiempo en la cola o que procesos están bloqueados esperando alguna operación de E/S.
- Por esta razón este modelo de dos estados no es eficiente.

# Modelo de procesos de cinco estados

- Un modelo más eficiente podría dividir el estado *No Ejecutando* en dos: *Listo* y *Bloqueado*.
- Este modelo añade también un estado *Nuevo* y otro *Saliente*.
- Entonces los cinco estados son:
  - Ejecutando
  - Listo
  - Bloqueado (en espera de evento u operación E/S)
  - Nuevo (BCP creado pero aún no cargado en memoria principal).
  - Saliente (proceso detenido o abortado)

# Modelo de procesos de cinco estados



# Modelo de procesos de cinco estados

## Posibles transiciones entre estados

Las posibles transiciones entre los 5 estados son:

- Null  $\rightarrow$  Nuevo
- Nuevo  $\rightarrow$  Listo
- Listo  $\rightarrow$  Ejecutando
- Ejecutando  $\rightarrow$  Saliente
- Ejecutando  $\rightarrow$  Listo
- Ejecutando  $\rightarrow$  Bloqueado
- Bloqueado Listo  $\rightarrow$  Nuevo
- Listo  $\rightarrow$  Saliente
- Bloqueado  $\rightarrow$  Saliente

# Modelo de procesos de cinco estados

- En este modelo los estados *Listo* y *Bloqueado* son manejados por colas independientes.
- Cuando el SO debe seleccionar el siguiente proceso a ejecutar busca en la cola de *Listos*. Podría existir un esquema de prioridades o simplemente un esquema FIFO.
- Si se maneja un esquema de prioridades, sería conveniente tener varias colas de procesos listos, una por cada prioridad. Así el SO podría seleccionar el proceso listo de mayor prioridad más rápidamente.

# Modelo de procesos de cinco estados

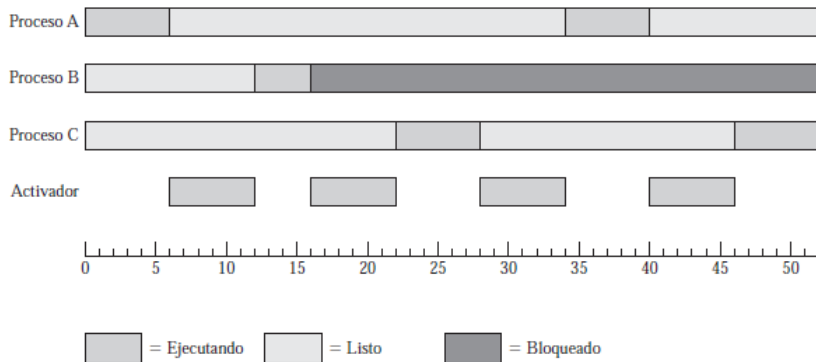
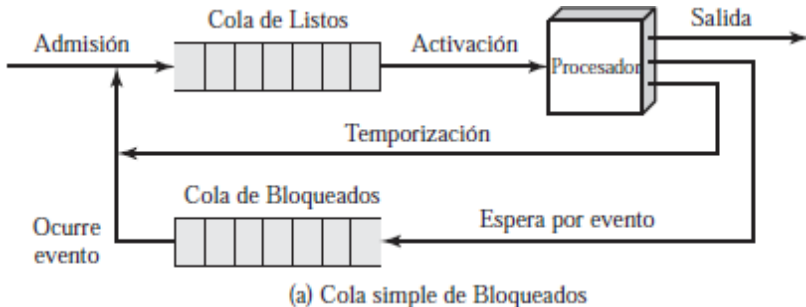


Figura: Estados de los procesos A, B y C

# Modelos de procesos de cinco estados

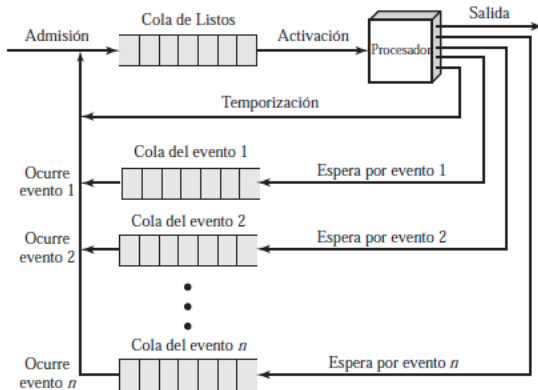
## Modelo de cola simple



Cuando sucede algún evento el SO debe recorrer la cola de *Bloqueados* en busca de los procesos que estén esperando por este evento.

# Modelos de procesos de cinco estados

## Modelo de múltiples colas de bloqueados



(b) Múltiples colas de Bloqueados

En algunos SO las colas de procesos *Bloqueados* podría contener miles de procesos por lo que sería más eficiente manejar una cola por cada evento.



## 1 Estados

- Modelos de estados de procesos
- Swapping

## 2 Procesos e hilos

- Funcionalidad de los Hilos
- Hilos de nivel de usuario y de nivel núcleo
- Programación Concurrente en Java

## 3 Procesos en Windows y Ubuntu

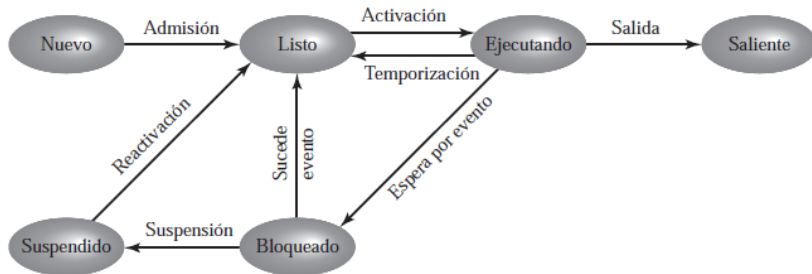
# Swapping (procesos suspendidos)

- El modelo de múltiples colas de procesos *Bloqueados* aumenta la eficiencia del sistema, sin embargo la diferencia entre la velocidad del procesador y los dispositivos E/S es tan grande que es muy probable que todos los procesos lleguen a estar en un estado *Bloqueado*.
- Si tenemos miles de procesos en estado *Bloqueado*, estos podrían estar ocupando todo el espacio en memoria principal y el procesador estaría ocioso debido a que no tendríamos capacidad para aceptar nuevos procesos.
- Es cierto que podríamos aumentar la memoria pero esto también aumentaría el costo del sistema. Y es importante notar que debido a las grandes memorias disponibles, la tendencia es a ejecutar **procesos más grandes y no más procesos**.

# Swapping (procesos suspendidos)

- Una solución para lograr aceptar más procesos y disminuir el tiempo ocioso del procesador es utilizar una técnica llamada *swapping* (memoria de intercambio).
- El swapping consiste en intercambiar parte o todo el proceso a de memoria principal a disco.
- Si no existen procesos en memoria principal en estado *Listo*, el SO mueve uno de estos procesos *Bloqueados* a disco (a la cola de procesos *Suspendidos*).
- El swapping añade el estado *Suspendido*
- Así el SO puede continuar con la creación de nuevos procesos o traer procesos de la cola de *Suspendidos* que puedan continuar siendo ejecutados.
- Esta técnica generalmente mejora el rendimiento del sistema.

# Swapping (procesos suspendidos)



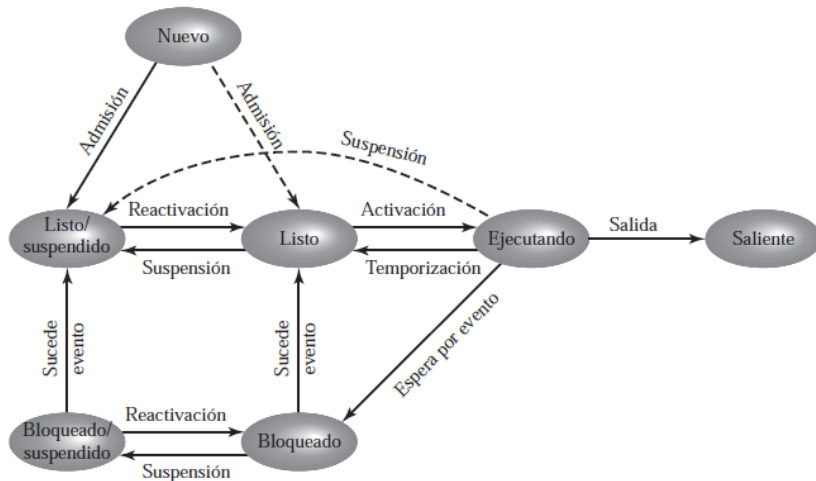
# Modelo de estados con swapping

Para que el SO pueda determinar más fácilmente que proceso debe traer de la cola de *Suspendidos* se necesitan los siguientes estados:

- Listo
- Bloqueado
- Bloqueado/Suspendido
- Listo/Suspendido

# Modelo de estados con swapping

Utilizando dos estados suspendido



# Transiciones con swapping (dos estados suspendidos)

La figura anterior nos muestra las posibles transiciones entre estados cuando se utilizan dos estados suspendidos. Estas transiciones son:

- **Bloqueado** → **Bloqueado/Suspendido**
- **Bloqueado/Suspendido** → **Listo/Suspendido**
- **Listo/Suspendido** → **Listo**
- **Listo** → **Listo/Suspendido** (liberar memoria o prioridad baja)
- **Nuevo** → **Listo/Suspendido** o **Listo** (depende espacio en memoria)
- **Bloqueado/Suspendido** → **Bloqueado** (prioridad alta o evento pronto a ocurrir)
- **Ejecutando** → **Listo/Suspendido** (liberar memoria)
- **Cualquier estado** → **Saliente**

## 1 Estados

- Modelos de estados de procesos
- Swapping

## 2 Procesos e hilos

- Funcionalidad de los Hilos
- Hilos de nivel de usuario y de nivel núcleo
- Programación Concurrente en Java

## 3 Procesos en Windows y Ubuntu



# Threads

- ¿Qué es un thread o hilo de un proceso?
- Los threads son procesos que comparten el mismo espacio de memoria (código y datos).
- Los threads de un proceso comparten el mismo espacio de memoria pero tienen su propio stack.

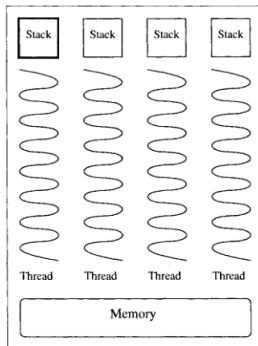
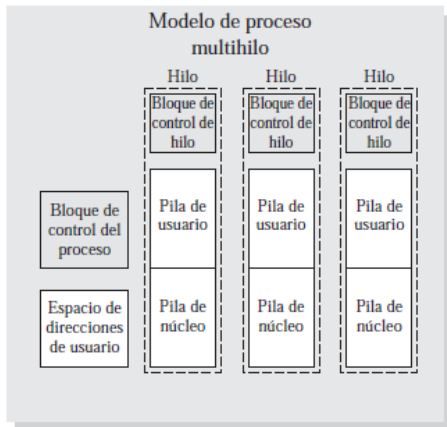
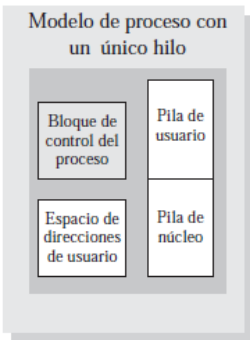


Figura: Un proceso con 4 hilos

- Multihilo es la capacidad de un SO de dar soporte a múltiples hilos de ejecución en un solo proceso.
- No todos los SO soportan multihilo. Ejemplo:
  - MS-DOS = 1 proceso de usuario y 1 único hilo.
  - Otros SO y algunas versiones de UNIX = múltiples procesos de usuario, pero 1 solo hilo.
- ¿Cuál es la diferencia entre varios procesos de un solo hilo y un proceso de varios hilos?
- Windows, Solaris, Mac OS X, Linux entre otros soportan múltiples procesos y cada uno con múltiples hilos.

- En un entorno multihilo, un proceso es una unidad de asignación de recursos y una unidad de protección.
- Dentro de un proceso puede haber uno o más hilos, cada uno con:
  - Un estado de ejecución,
  - Un contexto de hilo (que se almacena cuando el hilo no está en ejecución).
  - Una pila de ejecución.
  - Espacio de almacenamiento para variables locales.
  - Acceso a la memoria y recursos de su proceso, compartido con todos los hilos de su mismo proceso.

# Diferencia entre proceso e hilo



- Multihilo: existe un solo BCP y un espacio de direcciones de usuario para el proceso, pero hay pilas separadas para cada hilo, y un BCP para cada hilo (registros, prioridad, etc).

- Todos los hilos de un proceso:
  - comparten el estado y los recursos de ese proceso.
  - residen en el mismo espacio de direcciones
  - tienen acceso a los mismos datos.

# Beneficios de los hilos

Los mayores beneficios provienen de las consecuencias del rendimiento:

- Es más rápido crear un nuevo hilo en un proceso que crear un nuevo proceso (10-100 veces más rápido).
- Un hilo se finaliza más rápido que un proceso.
- Es más rápido cambiar entre dos hilos del **mismo** proceso.
- Mejoran la eficiencia de la comunicación entre diferentes programas.

# El problema de los Hilos

Aunque los hilos parecen ser un pequeño paso partiendo del cómputo secuencial, de hecho, son un paso inmenso. Descartan las propiedades más esenciales y atractivas del cómputo secuencial: Facilidad de comprensión, predictabilidad y determinismo. Los hilos, como un modelo de computación, son salvajemente no-determinísticos, y el trabajo del programador se convierte en /podar/ ese no-determinismo.

- Entonces, si se desea crear una aplicación como un conjunto de unidades de ejecución relacionadas, es mucho más eficiente hacerlo con un conjunto de hilos que con un conjunto de procesos independientes.
- Un ejemplo de aplicación que puede hacer uso de hilos es un servidor de archivos. Cada vez que se realiza una petición de archivos, el gestor de archivos puede ejecutar un nuevo hilo. Así en un ambiente **multiprocesador** se pueden ejecutar simultáneamente múltiples hilos del mismo proceso en diferentes procesadores. Y los hilos pueden compartir archivos de datos y coordinar acciones de manera más rápida.
- ¿Ejemplos? Hacer grupos y escribir la mayor cantidad de ejemplos sobre programas que utilizan hilos durante su ejecución.



# ¿Cómo se utilizan los hilos?

## Patrones de trabajo

### **Jefe / trabajador**

- Un hilo tiene una tarea distinta de todos los demás
- El hilo /jefe/ genera o recopila tareas
- Los hilos /trabajadores/ efectúan el trabajo.
- Modelo común para procesos servidor, GUIs que procesan eventos
- El jefe puede llevar la contabilidad de los trabajos realizados

# ¿Cómo se utilizan los hilos?

## Patrones de trabajo

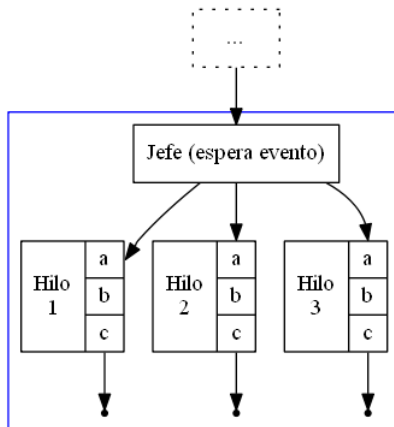


Figura: Jefe / trabajador

# ¿Cómo se utilizan los hilos?

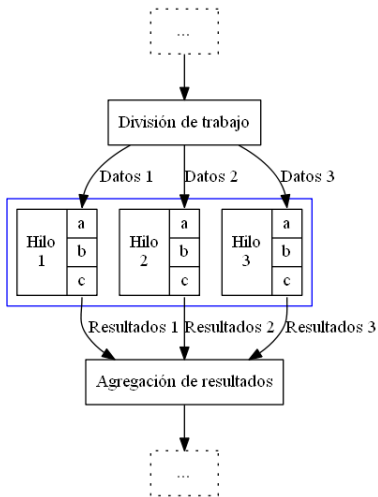
## Patrones de trabajo

### Equipo de trabajo

- A partir de hilos idénticos
- Realizarán las mismas tareas sobre diferentes datos
- Muy frecuentemente utilizado para cálculos matemáticos (p.ej. criptografía, render).
- Puede combinarse con jefe/trabajador para generar previsualizaciones (la tarea se realiza progresivamente)

# ¿Cómo se utilizan los hilos?

## Patrones de trabajo



**Figura: Equipo de trabajo**

# ¿Cómo se utilizan los hilos?

## Patrones de trabajo

### **Línea de ensamblado**

- Una tarea larga que puede dividirse en pasos
- Cada hilo se enfoca en una sola tarea
- Pasa los datos a otro hilo conforme va terminando
- Ayuda a mantener las rutinas simples de comprender
- Permite continuar procesando datos cuando hay hilos esperando E/S

# ¿Cómo se utilizan los hilos?

## Patrones de trabajo

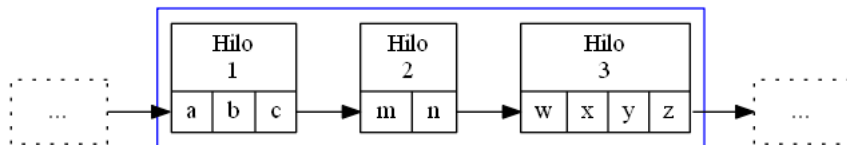


Figura: Línea de ensamblado

- Debido a que todos los hilos de un proceso comparten el mismo espacio de direcciones que su proceso, al suspender el proceso, todos los hilos se suspenden al mismo tiempo. De igual manera cuando se finaliza un proceso, se finalizan todos los hilos de ese proceso.

## 1 Estados

- Modelos de estados de procesos
- Swapping

## 2 Procesos e hilos

- Funcionalidad de los Hilos
- Hilos de nivel de usuario y de nivel núcleo
- Programación Concurrente en Java

## 3 Procesos en Windows y Ubuntu

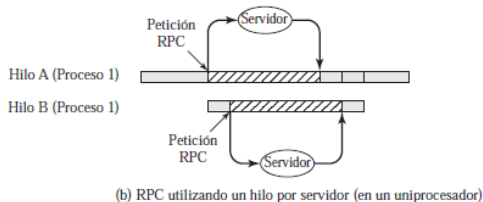
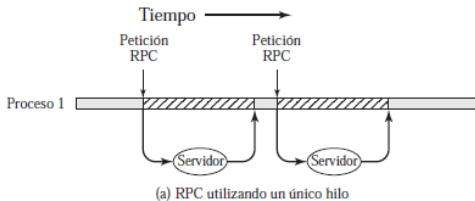


# Funcionalidad de los Hilos

## Estados de los hilos

- Los principales estados de los hilos son Ejecutando, Listo y Bloqueado.
- El estado Suspendido se aplica solo a nivel proceso ya que si se expulsa un proceso de memoria, todos sus hilos deben también ser expulsados. Porqué? porque comparten el mismo espacio de direcciones.

# Monohilo vs Multihilo



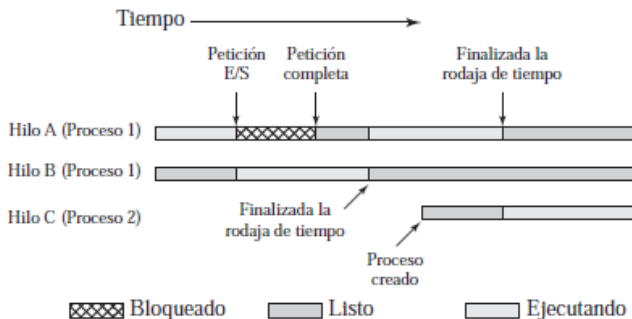
▨ Bloqueado, esperando respuesta RPC

□ Bloqueado, esperando el procesador, que está en uso por Hilo B

□ Ejecutando

- Programa que realiza 2 RPC a dos máquinas diferentes y poder combinar resultados.
- en *a* los resultados se obtienen en secuencia.
- en *b* se mejora la velocidad de ejecución del programa.

# Multihilo en uniprocador con multiprogramación



- Multiprogramación permite **intercalado** de hilos de varios procesos pero nunca se llegan a ejecutar en paralelo.

# Sincronización de hilos

- Debido a que todos los hilos de un proceso comparten recursos y espacio de direcciones, cualquier modificación de un recurso (archivo) por parte de un hilo, afecta el entorno de todos los hilos del mismo proceso.
- Por este motivo es necesario sincronizar las actividades de los hilos para que no interfieran entre ellos o corrompan estructuras de datos.

# Contenido I

## 1 Estados

- Modelos de estados de procesos
- Swapping

## 2 Procesos e hilos

- Funcionalidad de los Hilos
- Hilos de nivel de usuario y de nivel núcleo
- Programación Concurrente en Java

## 3 Procesos en Windows y Ubuntu

# Hilos de nivel de usuario y de nivel núcleo

- **Hilos de nivel usuario (ULT):** la aplicación gestiona todo el trabajo de los hilos y el núcleo no es consciente de la existencia de los mismos. Se utilizan bibliotecas de hilos para gestionarlos (creación, suspensión, destrucción).
- **Hilos de nivel núcleo (KLT):** El núcleo realiza todo el trabajo de gestión de hilos. No hay código de gestión de hilos en la aplicación, solamente una API para acceder a las utilidades de hilos del núcleo. Windows utiliza este enfoque.

# Contenido I

## 1 Estados

- Modelos de estados de procesos
- Swapping

## 2 Procesos e hilos

- Funcionalidad de los Hilos
- Hilos de nivel de usuario y de nivel núcleo
- Programación Concurrente en Java

## 3 Procesos en Windows y Ubuntu

# Clase Thread

- Java proporciona la clase predefinida Thread.
- Una clase puede extender la clase Thread, sobrescribir el método run y luego llamar a start() para lanzar el thread.

```
public class HelloWorldThread extends Thread {  
    public void run() {  
        System.out.println("Hello World");  
    }  
    public static void main( String[] args) {  
        HelloWorldThread t = new HelloWorldThread();  
        t.start();  
    }  
}
```



# Ejemplo del uso de Hilos

- ¿Cuál es el algoritmo para calcular el enésimo número de la serie de Fibonacci  $F_n$ ?
- Podemos utilizar la relación recursiva:

$$F_n = F_{n-1} + F_{n-2}$$

Para  $n \geq 2$ . Los casos bases son:

$$F_0 = 1$$

y

$$F_1 = 1$$

# Ejercicio

## Threads y join

- Podemos implementar el programa anterior utilizando threads y el constructor `join`.
- Para computar  $F_n$ , el método `run` se bifurca (`fork`) en dos threads que computan  $F_n - 1$  y  $F_n - 2$  recursivamente.
- El thread principal espera a que finalicen estos dos threads utilizando la función `join`.

## 1 Estados

- Modelos de estados de procesos
- Swapping

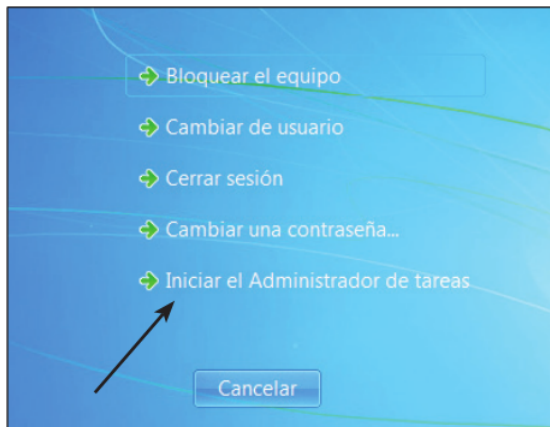
## 2 Procesos e hilos

- Funcionalidad de los Hilos
- Hilos de nivel de usuario y de nivel núcleo
- Programación Concurrente en Java

## 3 Procesos en Windows y Ubuntu

# Acceder a procesos en Windows 7

- ¿Cómo accedemos a ver los procesos ejecutándose en Windows 7?  
ctrl + alt + supr **Administrador de tareas**.
- ¿Otras opciones en Windows 7?
- ¿Opciones en Windows 8?



# Procesos en Windows

Administrador de tareas de Windows

Archivo Opciones Ver Ayuda

Aplicaciones **Procesos** Servicios Rendimiento Funciones de red Usuarios

Nombre de imagen	Nombre de usuario	CPU	Memoria ...	Descripción
firefox.exe	yo mismo	02	276.556 KB	Firefox
AcroRd32.exe	yo mismo	00	93.956 KB	Adobe Reader
chrome.exe	yo mismo	00	38.688 KB	Google Chrome
chrome.exe	yo mismo	00	36.980 KB	Google Chrome
wlmail.exe	yo mismo	00	34.076 KB	Windows Live Mail
explorer.exe	yo mismo	00	32.392 KB	Explorador de Windows
plugin-container.exe	yo mismo	00	29.764 KB	Plugin Container for Firefox
WINWORD.EXE	yo mismo	05	18.732 KB	Microsoft Office Word
wlcomm.exe	yo mismo	00	14.684 KB	Windows Live Communications Platform
soffice.bin	yo mismo	00	7.228 KB	OpenOffice.org 3.3
mspaint.exe	yo mismo	00	5.212 KB	Paint
sidebar.exe	yo mismo	00	4.836 KB	Gadgets de escritorio de Windows
AcroRd32.exe	yo mismo	00	2.732 KB	Adobe Reader
msnmsgr.exe	yo mismo	00	1.604 KB	Windows Live Messenger
taskmgr.exe	yo mismo	16	1.560 KB	Administrador de tareas de Windows
taskhost.exe	yo mismo	00	1.236 KB	Proceso de host para tareas de Windows
SweetIM.exe	yo mismo	00	1.232 KB	SweetIM Instant Messenger Enhancer
jucheck.exe	yo mismo	00	1.136 KB	Java(TM) Update Checker
mssecos.exe	yo mismo	00	1.052 KB	Microsoft Security Client User Interface
csrss.exe	SYSTEM	06	992 KB	Proceso en tiempo de ejecución del cliente-servidor
wuauclt.exe	yo mismo	00	620 KB	Windows Update
winlogon.exe	SYSTEM	00	620 KB	Aplicación de inicio de sesión de Windows
jucheck.exe	yo mismo	00	604 KB	Java(TM) Update Scheduler

☐ Mostrar procesos de todos los usuarios

Finalizar proceso

Procesos: 64    Uso de CPU: 30%    Memoria física: 58%

- La pestaña de procesos nos mostrará:
  - los procesos en ejecución.
  - el usuario propietario
  - el trabajo del procesador
  - la cantidad de memoria
  - descripción
  - clic derecho en la cabecera permite seleccionar que aspectos mostrar.
  - Total de procesos activos (inferior izquierda)
  - clic derecho sobre proceso mostrará:
    - Terminar proceso
    - Establecer prioridad
    - Buscar en línea
    - Propiedades

# Procesos en Ubuntu

- ¿Cómo abrimos el administrador de procesos en Ubuntu? Varias opciones: Información similar que en Win.
  - Buscar: Monitor del sistema
  - Buscar/Icono de aplicaciones/monitor del sistema
  - Instalar “indicador de carga del sistema”



Monitor del sistema

Procesos Recursos Sistemas de archivos

Carga media para los últimos 1, 5 y 15 minutos: 0,63, 0,69, 0,62

Actualizar Ver ▾

Nombre del proceso	Usuario	% CPU ▲	ID	Memoria	Prioridad
compiz	chalo	22	2006	170,3 MiB	Normal
gnome-system-monitor	chalo	10	3706	13,2 MiB	Normal
indicator-multiloader	chalo	1	3677	6,1 MiB	Normal
unity-scope-loader	chalo	0	3648	4,9 MiB	Normal
unity_picasa_daemon.py	chalo	0	3594	7,5 MiB	Normal
unity_shotwell_daemon.py	chalo	0	3593	5,2 MiB	Normal
unity_facebook_daemon.py	chalo	0	3590	7,1 MiB	Normal
unity_flickr_daemon.py	chalo	0	3589	7,4 MiB	Normal
unity-video-lens-daemon	chalo	0	3575	2,8 MiB	Normal
bash	chalo	0	3392	1,6 MiB	Normal
ubuntu-geoip-provider	chalo	0	2621	1,1 MiB	Normal
geoclue-master	chalo	0	2617	920,0 KiB	Normal
gvfsd-metadata	chalo	0	2570	392,0 KiB	Normal
deia-dup-monitor	chalo	0	2430	3,1 MiB	Normal

Finalizar proceso