Procesos del Sistema Operativo

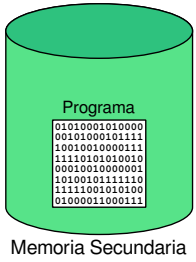
David Alejandro González Márquez

Clase disponible en: `https://github.com/fokerman/computingSystemsCourse`

Diferencias entre Programa y Proceso

Programa → entidad pasiva

Es el código fuente de la aplicación a ejecutar.
Compuesto por un conjunto de instrucciones y datos.
Se almacena en almacenamiento secundario (disco).



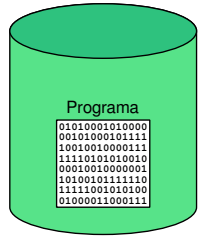
Diferencias entre Programa y Proceso

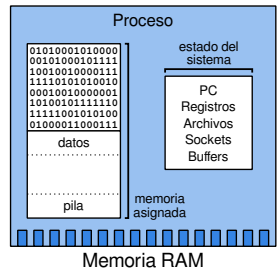
Programa → entidad pasiva

Es el código fuente de la aplicación a ejecutar.
Compuesto por un conjunto de instrucciones y datos.
Se almacena en almacenamiento secundario (disco).

Proceso → entidad activa

Es un programa en ejecución.
Compuesto por el programa y los recursos para su ejecución.
Se almacena en memoria RAM y en estructuras del sistema.





Diferencias entre Programa y Proceso

Programa → entidad pasiva

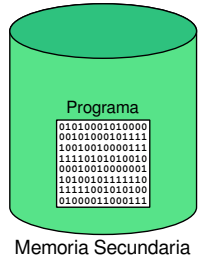
Es el código fuente de la aplicación a ejecutar.
Compuesto por un conjunto de instrucciones y datos.
Se almacena en almacenamiento secundario (disco).

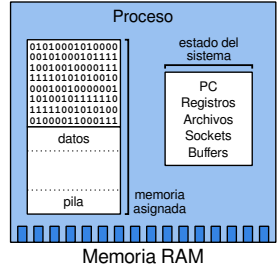
Proceso → entidad activa

Es un programa en ejecución.
Compuesto por el programa y los recursos para su ejecución.
Se almacena en memoria RAM y en estructuras del sistema.

Para cada **programa**, podemos instanciar múltiples **procesos**.
Cada proceso ejecutará utilizando **recursos diferentes** y potencialmente utilizando el mismo código.

El sistema operativo administra recursos y procesos.

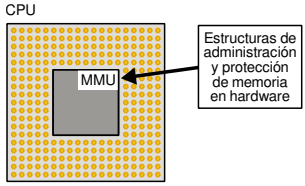


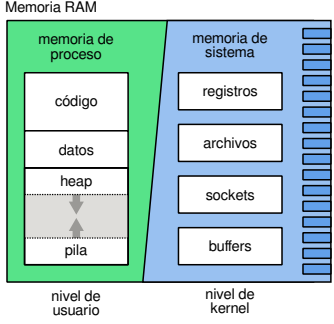


Memoria y recursos de un proceso

El sistema operativo administra los **recursos** que los procesos pueden acceder.

- Rangos de memoria que pueden leer y escribir.
- Prioridades para el uso del CPU y tiempo que lo puede utilizar.
- Archivos a los que tienen acceso.
- Interfaces de red que pueden escuchar.
- etc.



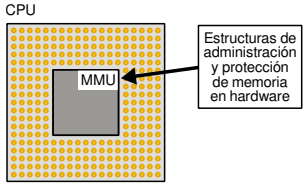


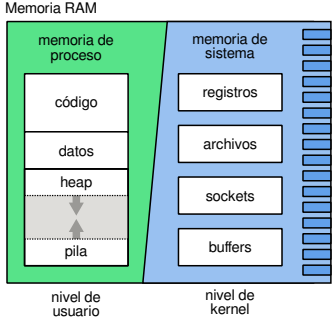
Memoria y recursos de un proceso

El sistema operativo administra los **recursos** que los procesos pueden acceder.

- Rangos de memoria que pueden leer y escribir.
- Prioridades para el uso del CPU y tiempo que lo puede utilizar.
- Archivos a los que tienen acceso.
- Interfaces de red que pueden escuchar.
- etc.

La asignación y protección de recursos se realiza tanto por hardware, como por software.





Ejecución de procesos

Los primeros sistemas eran **monotarea**.
Un **solo** proceso podia estar utilizando el procesador y acceder a los recursos del sistema.

Ejecución de procesos

Los primeros sistemas eran **monotarea**.
Un **solo** proceso podia estar utilizando el procesador y acceder a los recursos del sistema.

Los programas se ejecutaban uno a uno en orden de llegada, como *procesos batch*.
Cada programa resolvía una tarea específica sobre distintos conjuntos de datos de entrada.

Ejecución de procesos

Los primeros sistemas eran **monotarea**.
Un **solo** proceso podia estar utilizando el procesador y acceder a los recursos del sistema.

Los programas se ejecutaban uno a uno en orden de llegada, como *procesos batch*.
Cada programa resolvía una tarea específica sobre distintos conjuntos de datos de entrada.

Las primeras computadoras hogareñas seguían este modelo y permitían ejecutar sólo una tarea por vez.
No compartían recursos y sí los compartían los utilizaban cooperativamente entre procesos.
MS-DOS es un ejemplo de este sistema.

Los sistemas más avanzados tenían la posibilidad de ejecutar múltiples procesos concurrentemente a pesar de tener un solo procesador.

Ejecución de procesos

Los primeros sistemas eran **monotarea**.
Un **solo** proceso podia estar utilizando el procesador y acceder a los recursos del sistema.

Los programas se ejecutaban uno a uno en orden de llegada, como *procesos batch*.
Cada programa resolvía una tarea específica sobre distintos conjuntos de datos de entrada.

Las primeras computadoras hogareñas seguían este modelo y permitían ejecutar sólo una tarea por vez.
No compartían recursos y sí los compartían los utilizaban cooperativamente entre procesos.
MS-DOS es un ejemplo de este sistema.

Los sistemas más avanzados tenían la posibilidad de ejecutar múltiples procesos concurrentemente a pesar de tener un solo procesador.

Pero, ¿Cómo ejecutan múltiples procesos en un solo procesador?

Ejecución de múltiples procesos: *Multitasking*

Para simular que ejecutamos **simultáneamente** varios procesos,
vamos a ejecutar por una fracción de tiempo cada uno, una y otra vez.

Ejecución de múltiples procesos: *Multitasking*

Para simular que ejecutamos **simultáneamente** varios procesos,
vamos a ejecutar por una fracción de tiempo cada uno, una y otra vez.

Vamos a hacer esto tan rápido que el usuario **no va a percibir** que en realidad hay un solo proceso corriendo por unidad de tiempo. Ejecutamos los procesos **concurrentemente**.

Ejecución de múltiples procesos: *Multitasking*

Para simular que ejecutamos **simultáneamente** varios procesos,
vamos a ejecutar por una fracción de tiempo cada uno, una y otra vez.

Vamos a hacer esto tan rápido que el usuario **no va a percibir** que en realidad hay un solo proceso corriendo por unidad de tiempo. Ejecutamos los procesos **concurrentemente**.

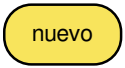
El tiempo del procesador es compartido por múltiples procesos en ejecución.

El sistema operativo cuenta con una componente denominada *scheduler*
o **planificador**, que decide cual será el próximo proceso en ser ejecutado.

Estados de un proceso

Los procesos antes de comenzar a ser ejecutados o mientras son ejecutados pueden tomar multiples **estados**.

Estados de un proceso



Los procesos antes de comenzar a ser ejecutados o mientras son ejecutados pueden tomar multiples **estados**.

- **Nuevo**: Estado inicial, se está cargando poco a poco en memoria y solicitando recursos.

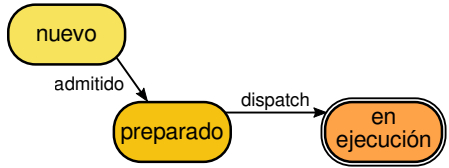
Estados de un proceso

Los procesos antes de comenzar a ser ejecutados o mientras son ejecutados pueden tomar multiples **estados**.

- **Nuevo**: Estado inicial, se está cargando poco a poco en memoria y solicitando recursos.
- **Preparado**: El proceso puede ser ejecutado, pero el procesador aun no esta disponible.

Estados de un proceso

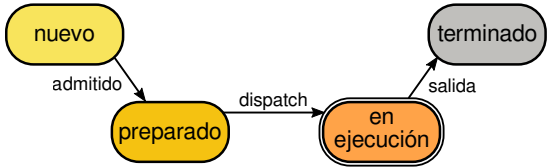
Los procesos antes de comenzar a ser ejecutados o mientras son ejecutados pueden tomar multiples **estados**.



- **Nuevo**: Estado inicial, se está cargando poco a poco en memoria y solicitando recursos.
- **Preparado**: El proceso puede ser ejecutado, pero el procesador aun no esta disponible.
- **En Ejecución**: El proceso está siendo ejecutado actualmente. Existe un solo proceso en este estado.

Estados de un proceso

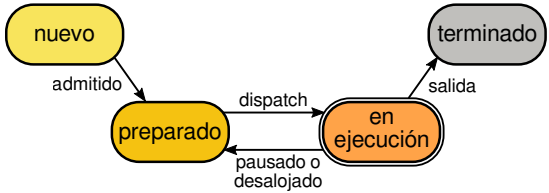
Los procesos antes de comenzar a ser ejecutados o mientras son ejecutados pueden tomar multiples **estados**.



- **Nuevo**: Estado inicial, se está cargando poco a poco en memoria y solicitando recursos.
- **Preparado**: El proceso puede ser ejecutado, pero el procesador aun no esta disponible.
- **En Ejecución**: El proceso está siendo ejecutado actualmente. Existe un solo proceso en este estado.
- **Terminado**: Proceso terminado, no puede volver a correr. Aun existe en la medida que desaloje todos los recursos.

Estados de un proceso

Los procesos antes de comenzar a ser ejecutados o mientras son ejecutados pueden tomar multiples **estados**.

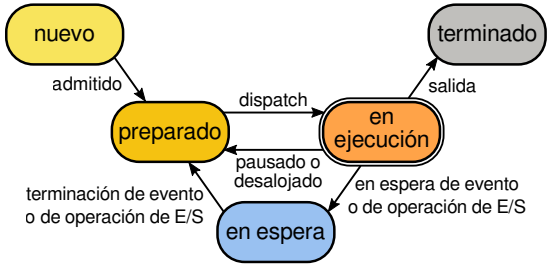


- **Nuevo**: Estado inicial, se está cargando poco a poco en memoria y solicitando recursos.
- **Preparado**: El proceso puede ser ejecutado, pero el procesador aun no esta disponible.
- **En Ejecución**: El proceso está siendo ejecutado actualmente. Existe un solo proceso en este estado.
- **Terminado**: Proceso terminado, no puede volver a correr. Aun existe en la medida que desaloje todos los recursos.

Pasar de **En Ejecución** a **Preparado** es posible cuando se *pausa* o *desalojo* un proceso.

Estados de un proceso

Los procesos antes de comenzar a ser ejecutados o mientras son ejecutados pueden tomar multiples **estados**.



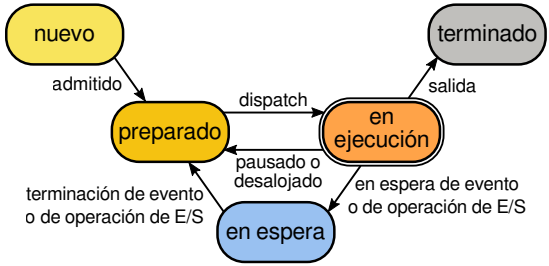
- **Nuevo**: Estado inicial, se está cargando poco a poco en memoria y solicitando recursos.
- **Preparado**: El proceso puede ser ejecutado, pero el procesador aun no esta disponible.
- **En Ejecución**: El proceso está siendo ejecutado actualmente. Existe un solo proceso en este estado.
- **Terminado**: Proceso terminado, no puede volver a correr. Aun existe en la medida que desaloje todos los recursos.

Pasar de **En Ejecución** a **Preparado** es posible cuando se *pausa* o *desalojo* un proceso.

- **En Espera**: El proceso queda en espera por un evento, será pasado a *Preparado* cuando suceda el evento por el cual espera suceda.

Estados de un proceso

Los procesos antes de comenzar a ser ejecutados o mientras son ejecutados pueden tomar multiples **estados**.



- **Nuevo**: Estado inicial, se está cargando poco a poco en memoria y solicitando recursos.
- **Preparado**: El proceso puede ser ejecutado, pero el procesador aun no esta disponible.
- **En Ejecución**: El proceso está siendo ejecutado actualmente. Existe un solo proceso en este estado.
- **Terminado**: Proceso terminado, no puede volver a correr. Aun existe en la medida que desaloje todos los recursos.

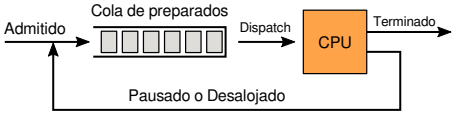
Pasar de **En Ejecución** a **Preparado** es posible cuando se *pausa* o *desalojo* un proceso.

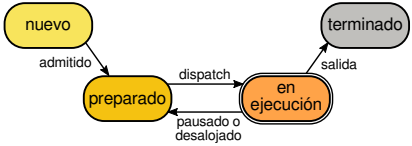
- **En Espera**: El proceso queda en espera por un evento, será pasado a *Preparado* cuando suceda el evento por el cual espera suceda.

¿Cómo espera un proceso?

Colas de espera

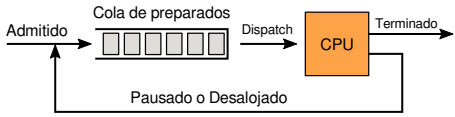
Los procesos aguardan por su turno en colas de procesos.

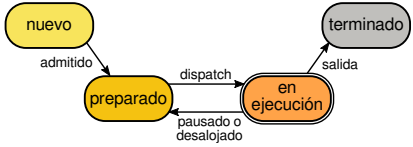




Colas de espera

Los procesos aguardan por su turno en colas de procesos.

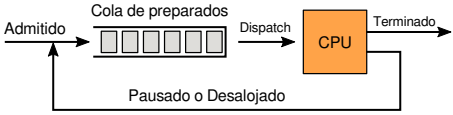


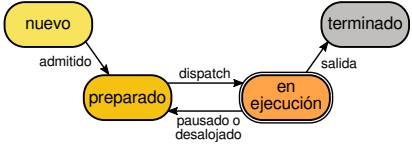


Una vez dentro de la CPU, los procesos pueden ser *pausados*, *desalojados* o *terminados*.

Colas de espera

Los procesos aguardan por su turno en colas de procesos.



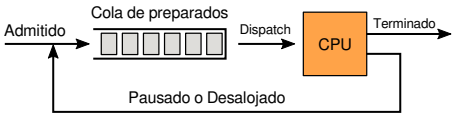


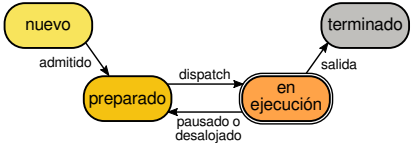
Una vez dentro de la CPU, los procesos pueden ser *pausados*, *desalojados* o *terminados*.

Si el proceso decide ser *pausado*, el sistema denomina **cooperativo**.

Colas de espera

Los procesos aguardan por su turno en colas de procesos.





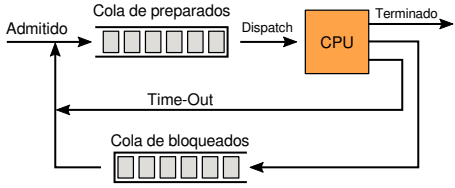
Una vez dentro de la CPU, los procesos pueden ser *pausados*, *desalojados* o *terminados*.

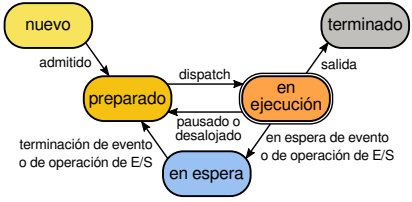
Si el proceso decide ser *pausado*, el sistema denomina **cooperativo**.

Si el proceso es *desalojado* en cualquier momento por el sistema, se dice que el sistema tiene **desalojo**.

Colas de espera

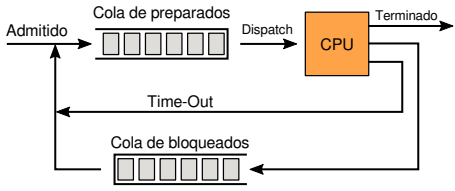
Un proceso bloqueado por entrada/salida espera en una cola diferente.

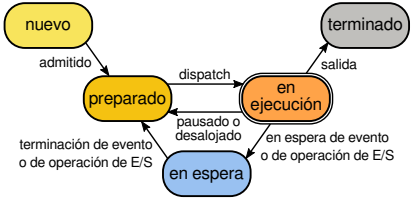




Colas de espera

Un proceso bloqueado por entrada/salida espera en una cola diferente.

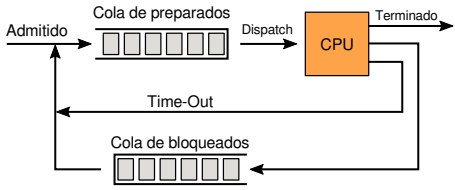


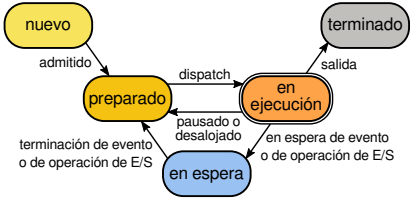


No todos los eventos de entrada salida demorarán el mismo tiempo.

Colas de espera

Un proceso bloqueado por entrada/salida espera en una cola diferente.



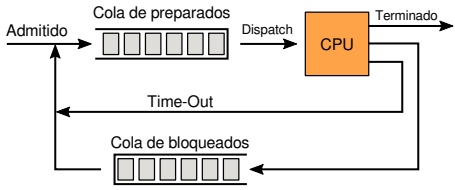


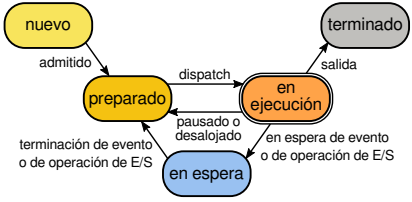
No todos los eventos de entrada salida demorarán el mismo tiempo.

Por lo tanto podemos crear **múltiples colas de espera** dependiendo de la razón del desalojo.

Colas de espera

Un proceso bloqueado por entrada/salida espera en una cola diferente.





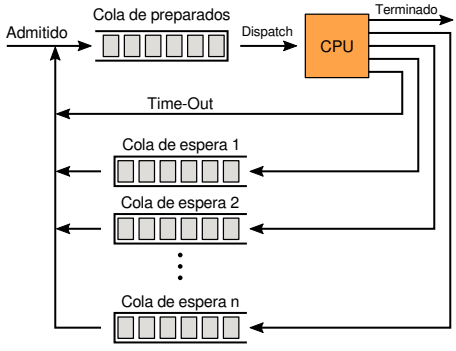
No todos los eventos de entrada salida demorarán el mismo tiempo.

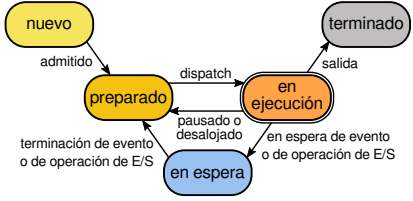
Por lo tanto podemos crear **múltiples colas de espera** dependiendo de la razón del desalojo.

Existirán colas de espera por eventos como escribir en pantalla, o escribir un archivo, como también por leer entradas del teclado.

Colas de espera

Un proceso bloqueado por entrada/salida espera en una cola diferente.





Las múltiples colas de espera, permiten **ordenar los tiempos de espera** entre los distintos procesos. Reduciendo el tiempo máximo que un proceso debe esperar en caso de ser encolado.

Bloque de Control de Proceso (PCB - *Process Control Block*)

El **PCB** es la estructura de datos dentro del Sistema Operativo que almacena toda la **información de un proceso**.

Bloque de Control de Proceso (PCB - *Process Control Block*)

El **PCB** es la estructura de datos dentro del Sistema Operativo que almacena toda la **información de un proceso**.

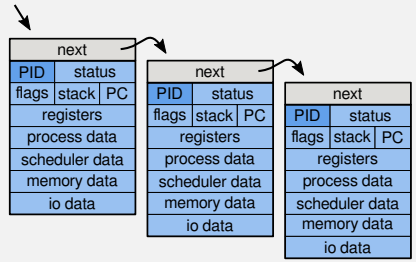
- Identificador del proceso (PID)
- Estado del proceso en el CPU
  - Registros (*Program Counter*, Flags, Pila, etc.)
- Información de planificación
  - Estadísticas, Prioridades, Tipos de procesos, etc.
- Información para la gestión de memoria del proceso
  - Memoria asignada, Memoria libre, etc.
- Información sobre la E/S del proceso
  - Archivos abiertos, *Sockets* asignados, *Buffers*, etc.

Bloque de Control de Proceso (PCB - *Process Control Block*)

El **PCB** es la estructura de datos dentro del Sistema Operativo que almacena toda la **información de un proceso**.

- Identificador del proceso (PID)
- Estado del proceso en el CPU
  Registros (*Program Counter*, Flags, Pila, etc.)
- Información de planificación
  Estadísticas, Prioridades, Tipos de procesos, etc.
- Información para la gestión de memoria del proceso
  Memoria asignada, Memoria libre, etc.
- Información sobre la E/S del proceso
  Archivos abiertos, *Sockets* asignados, *Buffers*, etc.

Los PCB se suelen administrar como **listas enlazadas**, modelando así la funcionalidad de una cola de procesos.



Cambio de contexto (*context switch*)

Cuando se debe intercambiar al proceso que se encuentra corriendo por otro, se da un cambio de contexto.

Este procedimiento se puede dar en bajo tres condiciones:

- Se produce una **excepción** de protección.
- Se produce una **interrupción** de E/S.
- Se **termina** el tiempo (*quantum*) que el proceso tenia asignado.

Cambio de contexto (*context switch*)

Cuando se debe intercambiar al proceso que se encuentra corriendo por otro, se da un cambio de contexto.

Este procedimiento se puede dar en bajo tres condiciones:

- Se produce una **excepción** de protección.
- Se produce una **interrupción** de E/S.
- Se **termina** el tiempo (*quantum*) que el proceso tenia asignado.

Un cambio de contexto implica una operatoria compleja, en la cual se captura una **foto** del contexto de ejecución actual y se lo intercambia por otro contexto de ejecución diferente.

¿Qué tareas implica un cambio de contexto?

Cambio de contexto (*context switch*)

Procedimiento para el cambio de contexto entre PCB_0 y PCB_1:

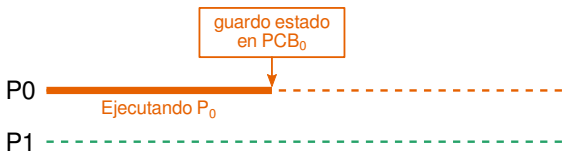
Cambio de contexto (*context switch*)

Procedimiento para el cambio de contexto entre PCB_0 y PCB_1:

Cambio de contexto (*context switch*)

Procedimiento para el cambio de contexto entre PCB_0 y PCB_1:

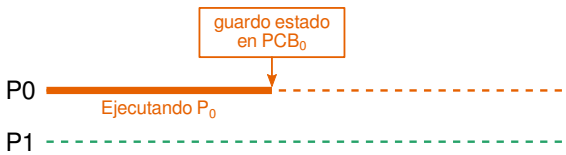
1. Guardar el contexto de ejecución (el estado del procesador) en el PCB_0.



Cambio de contexto (*context switch*)

Procedimiento para el cambio de contexto entre PCB_0 y PCB_1:

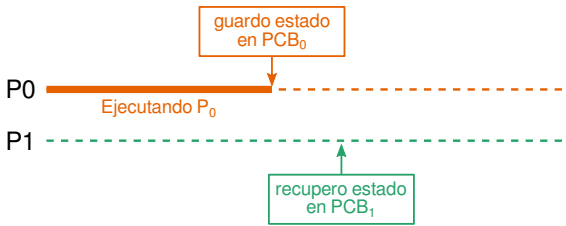
1. Guardar el contexto de ejecución (el estado del procesador) en el PCB_0.
2. Actualizar el estado del PCB_0 para indicar que no está más en ejecución.



Cambio de contexto (*context switch*)

Procedimiento para el cambio de contexto entre PCB_0 y PCB_1:

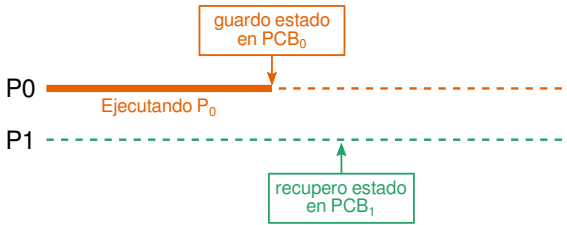
1. Guardar el contexto de ejecución (el estado del procesador) en el PCB_0.
2. Actualizar el estado del PCB_0 para indicar que no está más en ejecución.
3. Restaurar el contexto de ejecución del PCB_1.



Cambio de contexto (*context switch*)

Procedimiento para el cambio de contexto entre PCB_0 y PCB_1:

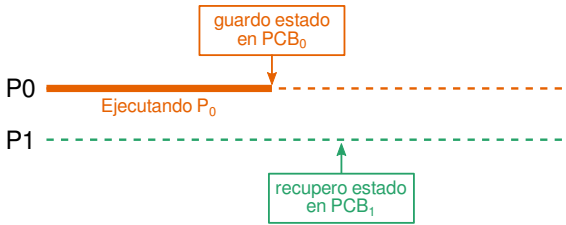
1. Guardar el contexto de ejecución (el estado del procesador) en el PCB_0.
2. Actualizar el estado del PCB_0 para indicar que no está más en ejecución.
3. Restaurar el contexto de ejecución del PCB_1.
4. Actualizar las estructuras de datos para la administración de memoria del PCB_1.



Cambio de contexto (*context switch*)

Procedimiento para el cambio de contexto entre PCB_0 y PCB_1:

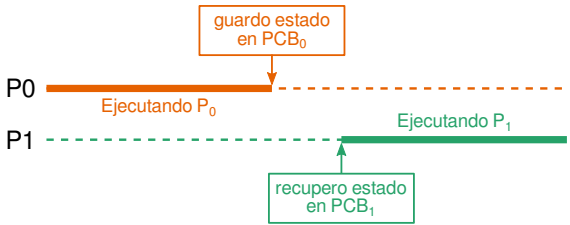
1. Guardar el contexto de ejecución (el estado del procesador) en el PCB_0.
2. Actualizar el estado del PCB_0 para indicar que no está más en ejecución.
3. Restaurar el contexto de ejecución del PCB_1.
4. Actualizar las estructuras de datos para la administración de memoria del PCB_1.
5. Actualizar el estado del PCB_1 para indicar que esta siendo ejecutado.



Cambio de contexto (*context switch*)

Procedimiento para el cambio de contexto entre PCB_0 y PCB_1:

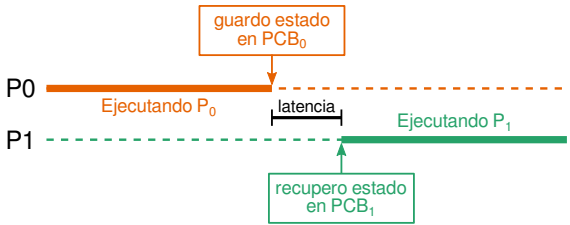
1. Guardar el contexto de ejecución (el estado del procesador) en el PCB_0.
2. Actualizar el estado del PCB_0 para indicar que no está más en ejecución.
3. Restaurar el contexto de ejecución del PCB_1.
4. Actualizar las estructuras de datos para la administración de memoria del PCB_1.
5. Actualizar el estado del PCB_1 para indicar que esta siendo ejecutado.
6. Retorna el control al proceso.



Cambio de contexto (*context switch*)

Procedimiento para el cambio de contexto entre PCB_0 y PCB_1:

1. Guardar el contexto de ejecución (el estado del procesador) en el PCB_0.
2. Actualizar el estado del PCB_0 para indicar que no está más en ejecución.
3. Restaurar el contexto de ejecución del PCB_1.
4. Actualizar las estructuras de datos para la administración de memoria del PCB_1.
5. Actualizar el estado del PCB_1 para indicar que esta siendo ejecutado.
6. Retorna el control al proceso.

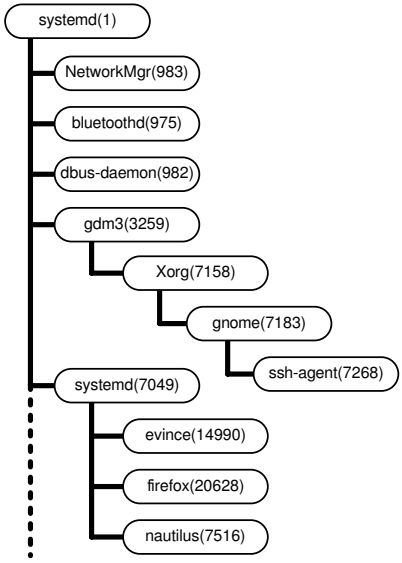


Cada cambio de contexto demora tiempo (*latencia*).
El procesador dispone de **mecanismos en hardware** para reducir este tiempo.
Aun así, actualizar las estructuras de administración de memoria es muy costo.

Creación de un nuevo proceso

Para crear un nuevo proceso se debe:

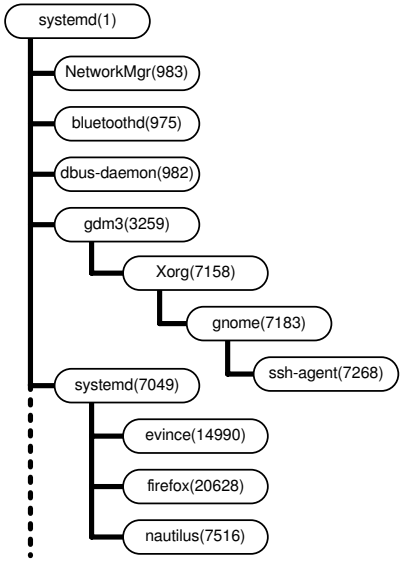
1. Asignar un identificador único al nuevo proceso (`PID`).



Creación de un nuevo proceso

Para crear un nuevo proceso se debe:

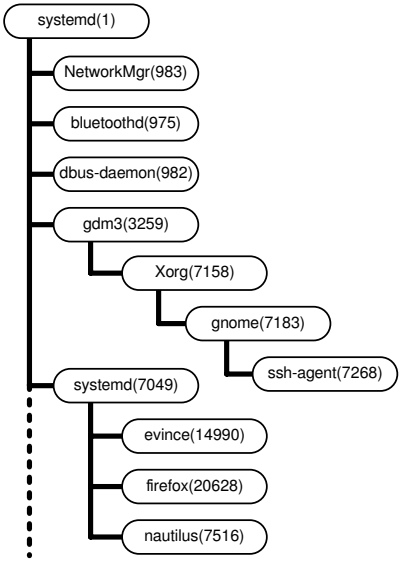
1. Asignar un identificador único al nuevo proceso (`PID`).
2. Reservar un espacio en memoria para el nuevo proceso.



Creación de un nuevo proceso

Para crear un nuevo proceso se debe:

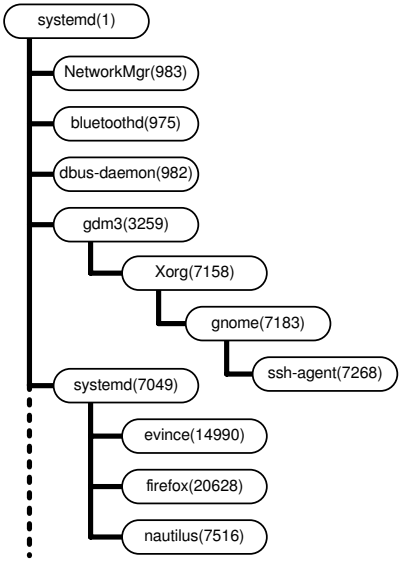
1. Asignar un identificador único al nuevo proceso (`PID`).
2. Reservar un espacio en memoria para el nuevo proceso.
3. Inicializar un nuevo PCB.



Creación de un nuevo proceso

Para crear un nuevo proceso se debe:

1. Asignar un identificador único al nuevo proceso (`PID`).
2. Reservar un espacio en memoria para el nuevo proceso.
3. Inicializar un nuevo PCB.
4. Enlazar el nuevo PCB en las estructuras de datos del sistema.



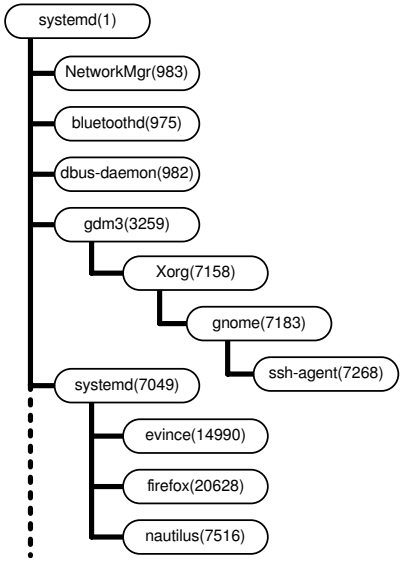
Creación de un nuevo proceso

Para crear un nuevo proceso se debe:

1. Asignar un identificador único al nuevo proceso (`PID`).
2. Reservar un espacio en memoria para el nuevo proceso.
3. Inicializar un nuevo PCB.
4. Enlazar el nuevo PCB en las estructuras de datos del sistema.

Los nuevos procesos son creados por otros procesos, utilizando la *syscall* `fork`.

Al proceso creador se lo llama `padre` (*parent*), mientras que al proceso `hijo` (*child*).



Creación de un nuevo proceso

Para crear un nuevo proceso se debe:

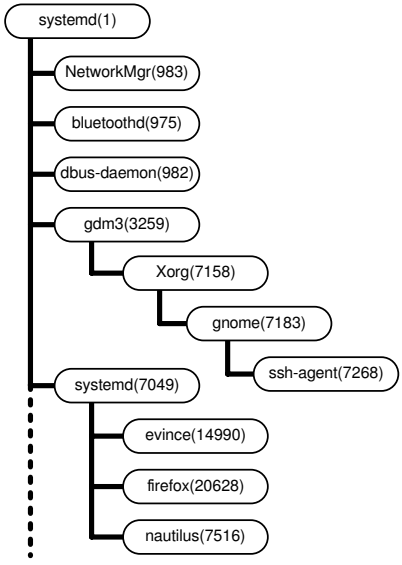
1. Asignar un identificador único al nuevo proceso (`PID`).
2. Reservar un espacio en memoria para el nuevo proceso.
3. Inicializar un nuevo PCB.
4. Enlazar el nuevo PCB en las estructuras de datos del sistema.

Los nuevos procesos son creados por otros procesos, utilizando la *syscall* `fork`.

Al proceso creador se lo llama `padre` (*parent*), mientras que al proceso `hijo` (*child*).

Un proceso puede tener múltiples procesos hijos, pero todos los procesos tienen un sólo proceso padre.

Este comportamiento configura un **árbol de procesos**.



Creación de un nuevo proceso: En UNIX

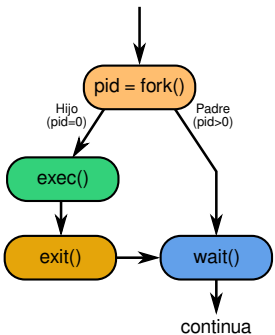
Para crear un nuevo proceso se utiliza dos llamados al sistema:

- `fork()`: Crea un nuevo proceso hijo con la misma memoria del proceso padre.
- `exec()`: Remplaza la memoria del proceso por la de un nuevo programa.

Creación de un nuevo proceso: En UNIX

Para crear un nuevo proceso se utiliza dos llamados al sistema:

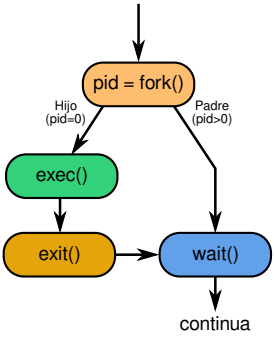
- `fork()`: Crea un nuevo proceso hijo con la misma memoria del proceso padre.
- `exec()`: Remplaza la memoria del proceso por la de un nuevo programa.



Creación de un nuevo proceso: En UNIX

Para crear un nuevo proceso se utiliza dos llamados al sistema:

- `fork()`: Crea un nuevo proceso hijo con la misma memoria del proceso padre.
- `exec()`: Remplaza la memoria del proceso por la de un nuevo programa.



Al ser creado un nuevo proceso, este **copia todo el estado del proceso padre**. Se diferenciarán en el valor de retorno del `fork`.

Esto permite modificar el comportamiento del proceso y llamar al servicio exec, que **remplaza toda la memoria del proceso original** por la de un nuevo programa.

El nuevo proceso, **con un código diferente**, es ejecutado hasta que decide terminar llamando a `exit`.

El proceso padre puede llamar a `wait` para esperar la finalización del proceso hijo.

Terminación de un proceso

La terminación de un proceso se puede dar en las siguientes situaciones:

- El proceso **decide terminar** llamando a la *syscal* `exit`.
- El proceso **genera un error** y el Sistema Operativo lo termina.
- El padre decide **terminar al proceso hijo** (`abort`).
- **El padre termina** y por lo tanto los hijos también (terminación en cascada).

Terminación de un proceso

La terminación de un proceso se puede dar en las siguientes situaciones:

- El proceso **decide terminar** llamando a la *syscal* `exit`.
- El proceso **genera un error** y el Sistema Operativo lo termina.
- El padre decide **terminar al proceso hijo** (`abort`).
- **El padre termina** y por lo tanto los hijos también (terminación en cascada).

Cuando un proceso termina el sistema debe **liberar sus recursos**:

- Liberar la memoria asignada.
- Liberar descriptores de archivos abiertos.
- Liberar los *buffers* de entrada/salida en uso.

Terminación de un proceso

La terminación de un proceso se puede dar en las siguientes situaciones:

- El proceso **decide terminar** llamando a la *syscal* `exit`.
- El proceso **genera un error** y el Sistema Operativo lo termina.
- El padre decide **terminar al proceso hijo** (`abort`).
- **El padre termina** y por lo tanto los hijos también (terminación en cascada).

Cuando un proceso termina el sistema debe **liberar sus recursos**:

- Liberar la memoria asignada.
- Liberar descriptores de archivos abiertos.
- Liberar los *buffers* de entrada/salida en uso.

El proceso padre, puede **esperar por la terminación** del proceso hijo llamando a `wait`.

El servicio *wait* retorna al padre el **estado de terminación** del proceso hijo.

Threads (Hilos)

Crear nuevos procesos, implica la costosa tarea de administrar la memoria, para estos nuevos procesos. Para reducir el **costo de crear nuevos contextos de ejecución**, existen los *threads* o hilos de ejecución.

Threads (Hilos)

Crear nuevos procesos, implica la costosa tarea de administrar la memoria, para estos nuevos procesos. Para reducir el **costo de crear nuevos contextos de ejecución**, existen los *threads* o hilos de ejecución.

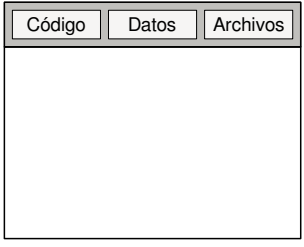
Los *threads* son **nuevos contexto de ejecución**, similares a un nuevo proceso, pero que comparten recursos con el proceso padre. Tienen el **mismo mapa de memoria** para código y datos que el padre.

Threads (Hilos)

Crear nuevos procesos, implica la costosa tarea de administrar la memoria, para estos nuevos procesos. Para reducir el **costo de crear nuevos contextos de ejecución**, existen los *threads* o hilos de ejecución.

Los *threads* son **nuevos contexto de ejecución**, similares a un nuevo proceso, pero que comparten recursos con el proceso padre. Tienen el **mismo mapa de memoria** para código y datos que el padre.

Single-Thread

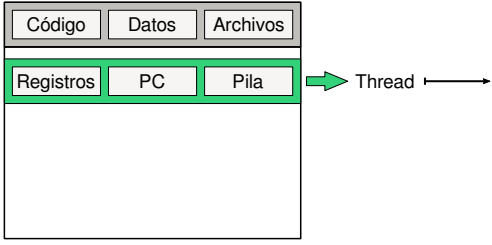


Threads (Hilos)

Crear nuevos procesos, implica la costosa tarea de administrar la memoria, para estos nuevos procesos. Para reducir el **costo de crear nuevos contextos de ejecución**, existen los *threads* o hilos de ejecución.

Los *threads* son **nuevos contexto de ejecución**, similares a un nuevo proceso, pero que comparten recursos con el proceso padre. Tienen el **mismo mapa de memoria** para código y datos que el padre.

Single-Thread

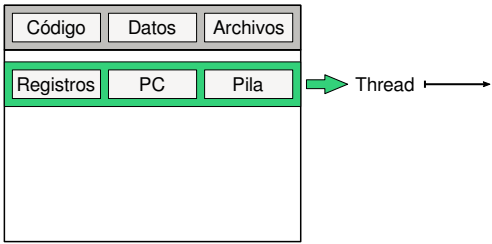


Threads (Hilos)

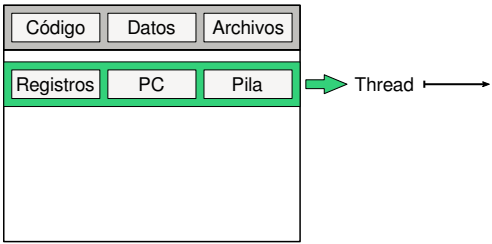
Crear nuevos procesos, implica la costosa tarea de administrar la memoria, para estos nuevos procesos. Para reducir el **costo de crear nuevos contextos de ejecución**, existen los *threads* o hilos de ejecución.

Los *threads* son **nuevos contexto de ejecución**, similares a un nuevo proceso, pero que comparten recursos con el proceso padre. Tienen el **mismo mapa de memoria** para código y datos que el padre.

Single-Thread



Multi-Thread

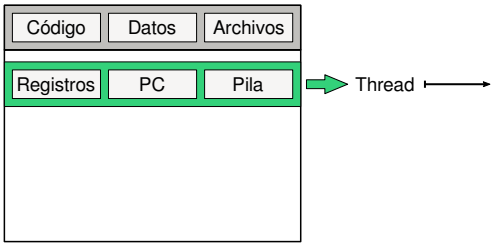


Threads (Hilos)

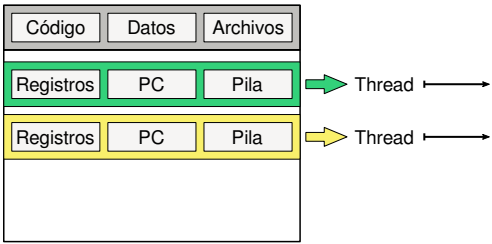
Crear nuevos procesos, implica la costosa tarea de administrar la memoria, para estos nuevos procesos. Para reducir el **costo de crear nuevos contextos de ejecución**, existen los *threads* o hilos de ejecución.

Los *threads* son **nuevos contexto de ejecución**, similares a un nuevo proceso, pero que comparten recursos con el proceso padre. Tienen el **mismo mapa de memoria** para código y datos que el padre.

Single-Thread



Multi-Thread

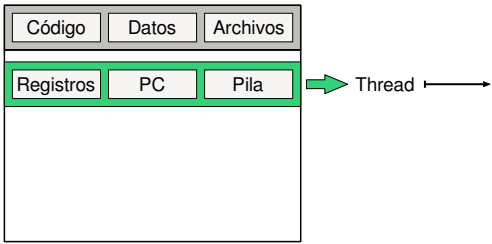


Threads (Hilos)

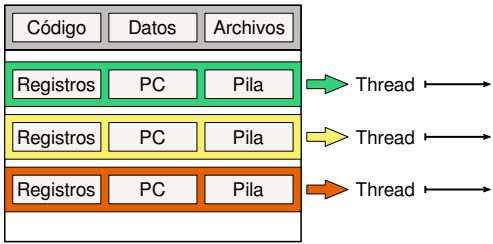
Crear nuevos procesos, implica la costosa tarea de administrar la memoria, para estos nuevos procesos. Para reducir el **costo de crear nuevos contextos de ejecución**, existen los *threads* o hilos de ejecución.

Los *threads* son **nuevos contexto de ejecución**, similares a un nuevo proceso, pero que comparten recursos con el proceso padre. Tienen el **mismo mapa de memoria** para código y datos que el padre.

Single-Thread

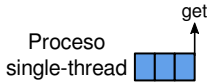


Multi-Thread



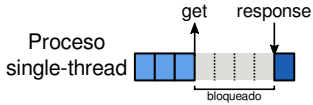
Threads (Hilos): Ejemplo

Un proceso hace tres *requests* a un servicio externo, quedando bloqueado en cada llamado.



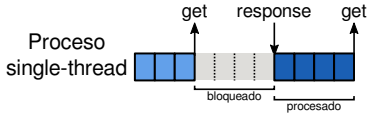
Threads (Hilos): Ejemplo

Un proceso hace tres *requests* a un servicio externo, quedando bloqueado en cada llamado.



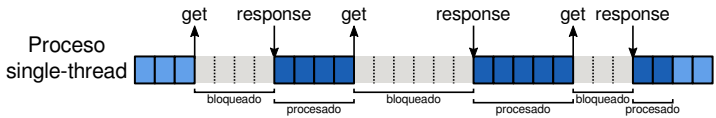
Threads (Hilos): Ejemplo

Un proceso hace tres *requests* a un servicio externo, quedando bloqueado en cada llamado.



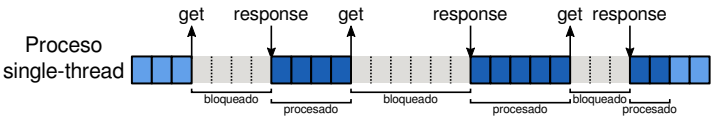
Threads (Hilos): Ejemplo

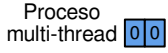
Un proceso hace tres *requests* a un servicio externo, quedando bloqueado en cada llamado.



Threads (Hilos): Ejemplo

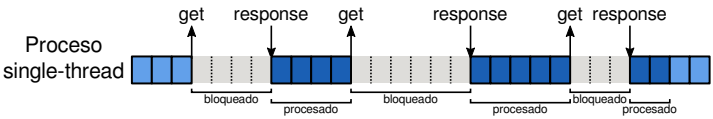
Un proceso hace tres *requests* a un servicio externo, quedando bloqueado en cada llamado.

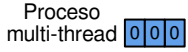


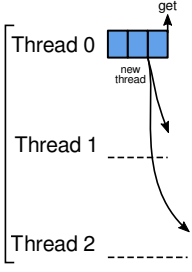


Threads (Hilos): Ejemplo

Un proceso hace tres *requests* a un servicio externo, quedando bloqueado en cada llamado.

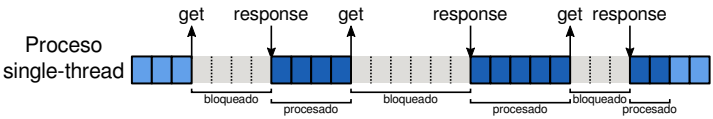


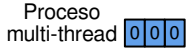


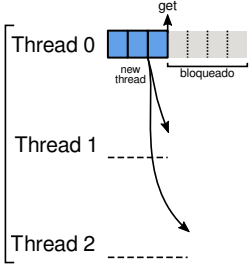


Threads (Hilos): Ejemplo

Un proceso hace tres *requests* a un servicio externo, quedando bloqueado en cada llamado.

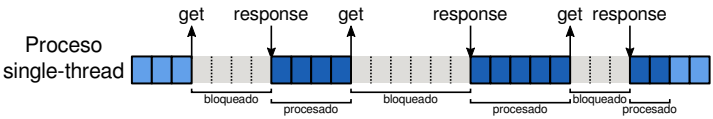






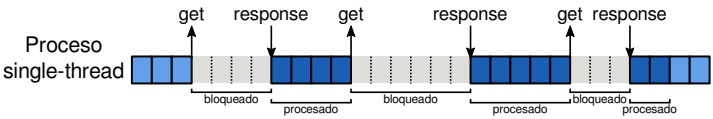
Threads (Hilos): Ejemplo

Un proceso hace tres *requests* a un servicio externo, quedando bloqueado en cada llamado.



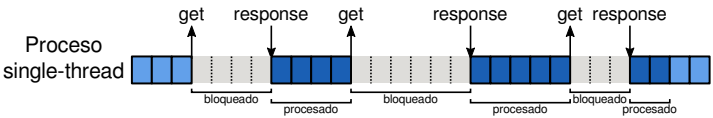
Threads (Hilos): Ejemplo

Un proceso hace tres *requests* a un servicio externo, quedando bloqueado en cada llamado.



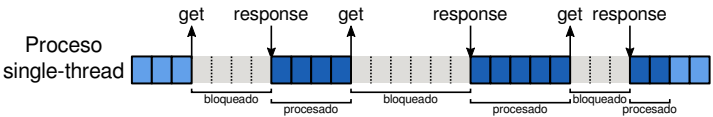
Threads (Hilos): Ejemplo

Un proceso hace tres *requests* a un servicio externo, quedando bloqueado en cada llamado.



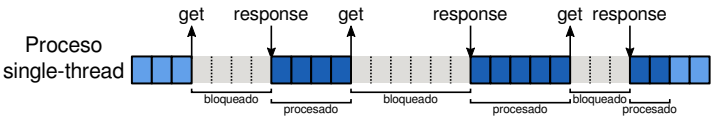
Threads (Hilos): Ejemplo

Un proceso hace tres *requests* a un servicio externo, quedando bloqueado en cada llamado.



Threads (Hilos): Ejemplo

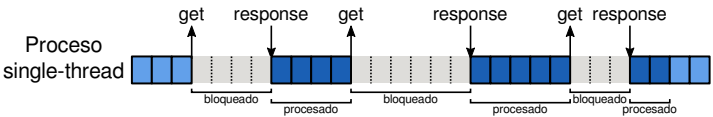
Un proceso hace tres *requests* a un servicio externo, quedando bloqueado en cada llamado.



Threads (Hilos): Ejemplo

Un proceso hace tres *requests* a un servicio externo, quedando bloqueado en cada llamado.

Si utilizamos *threads* en cada llamado, el tiempo bloqueado es aprovechado otro *thread*.

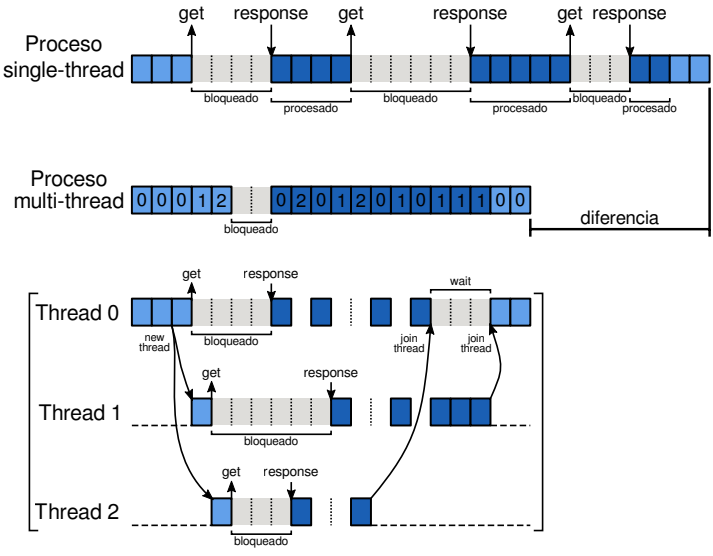


Threads (Hilos): Ejemplo

Un proceso hace tres *requests* a un servicio externo, quedando bloqueado en cada llamado.

Si utilizamos *threads* en cada llamado, el tiempo bloqueado es aprovechado otro *thread*.

Si bien se utiliza un solo procesador, esté queda bloqueado durante menos tiempo.



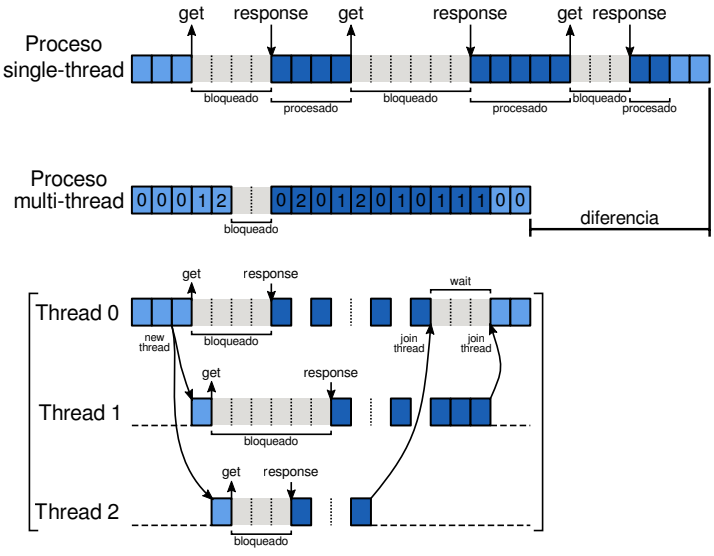
Threads (Hilos): Ejemplo

Un proceso hace tres *requests* a un servicio externo, quedando bloqueado en cada llamado.

Si utilizamos *threads* en cada llamado, el tiempo bloqueado es aprovechado otro *thread*.

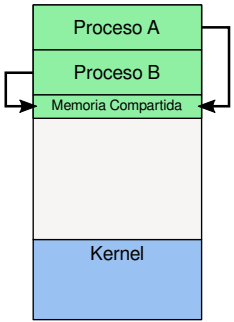
Si bien se utiliza un solo procesador, esté queda bloqueado durante menos tiempo.

Utilizar *threads* mejora la capacidad de respuesta de los procesos interactivos, mejorado el uso de los recursos del procesador y reduciendo la cantidad de memoria utilizada.



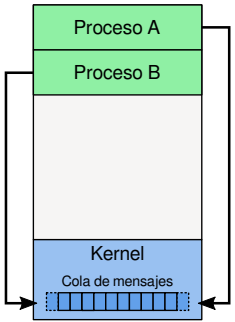
Comunicación entre procesos

Los procesos cooperativos necesitan comunicarse entre sí.
Dos formas de IPC (*Inter-process communication*)



Memoria compartida

Un área de memoria compartida entre dos o más procesos a nivel de usuario.



Pasaje de Mensajes

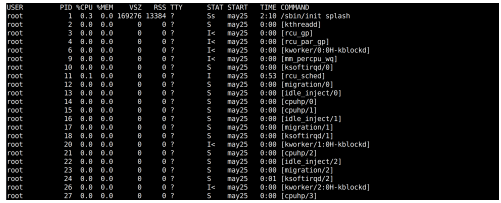
El Sistema Operativo proporciona un mecanismo para enviar y recibir mensajes entre procesos.

Comandos útiles en Linux

ps

Muestra el estado de todos los procesos del sistema.

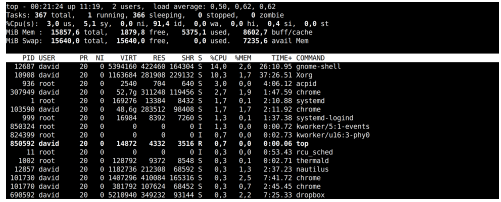
Ejemplo: `$ ps -aux` ⏎



top o htop

Muestra el consumo de CPU y memoria de cada proceso.

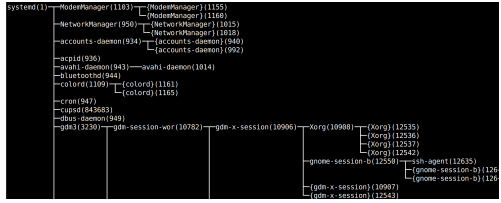
Ejemplo: `$ top` ⏎



pstree

Muestra el árbol de procesos de todo el sistema.

Ejemplo: `$ pstree -p` ⏎



kill

Envia un señal al proceso indicado por su PID.

Por defecto, envía la señal de terminación.

Ejemplo: `$ kill 1245` ⏎

Para más información consultar los manuales con el comando man

Bibliografía

- Silberschatz, “Fundamentos de Sistemas Operativos”, 7ma Edición, 2006.
  - **Capítulo 3 - Procesos**, páginas 73-86
  - **Capítulo 4 - Hebras**, páginas 113-115

- Stallings, “Operating Systems: Internals and Design Principles”, 9th Edition, 2018.
  - **Chapter 3 - Process Description and Control**, páginas 130-163
  - **Chapter 4 - Threads**, páginas 177-183

Ejercicios

Con lo visto, ya pueden resolver todos los ejercicios de la Guía de Procesos.

¡Gracias!

Recuerden leer los comentarios adjuntos
en cada clase por aclaraciones.