udp Escuela de Informática y Telecomunicaciones
FACULTAD DE INGENIERÍA Y CIENCIAS

Tutoría para Examen de Grado

Sesión 4: Sistemas Operativos

Docente: Mauricio Hidalgo Barrientos

Temario de la sesión

- Recordar algunos conceptos introductorios sobre Sistemas Operativos
- Recordar aspectos relacionados a Procesos
- Recordar los algoritmos de Scheduling
- Recordar aspectos relacionados a Threads y Sincronización de Procesos
- Recordar conceptos y tópicos vinculados a Memoria y Sistema de Archivos

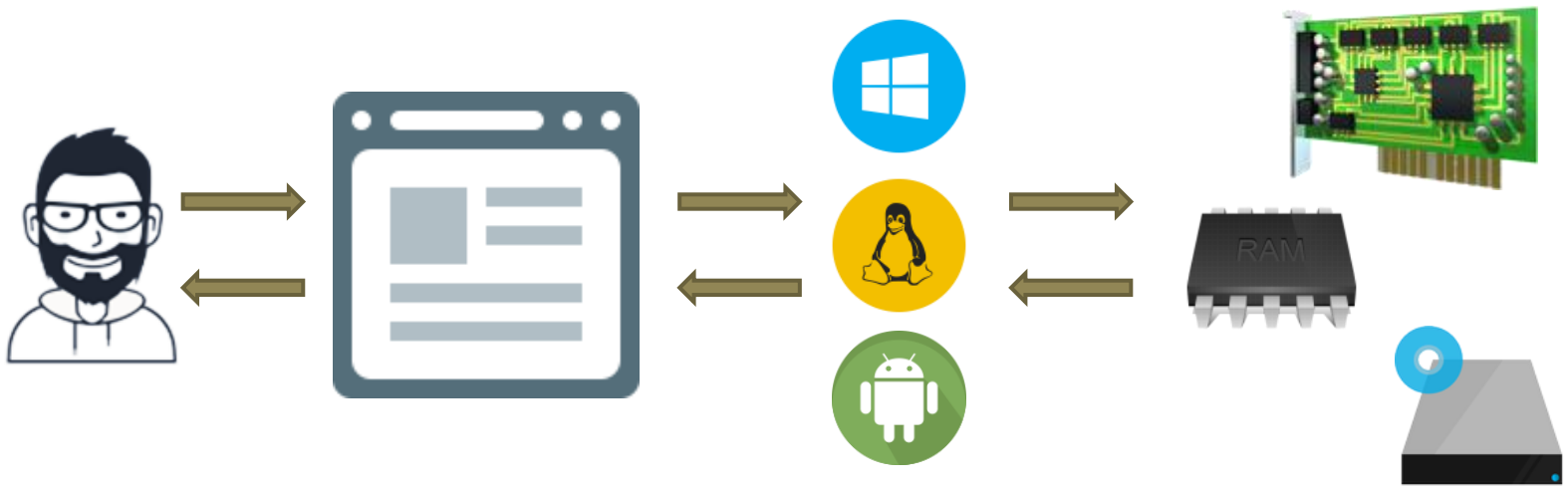
Conceptos Introductorios

Definiciones iniciales

Un Sistema Operativo es un programa o conjunto de programas de un sistema informático que:

- Gestiona los recursos de hardware y
- provee servicios a los programas de aplicación de software

Se ejecuta en modo privilegiado respecto de los restantes programas (aunque puede que parte de él se ejecute en espacio de usuario)



Definiciones iniciales

Proceso

Un proceso es un programa en ejecución que abarca un espacio en la memoria y que posee la capacidad para leer, escribir datos. Además, contiene a su stack de ejecución y tiene acceso a una serie de recursos (archivos, alarmas, PC y SP, procesos relacionados, etc.)

Direcciones de memoria

Es el espacio físico (y virtual) que ser compartido por los distintos programas. Posee mecanismos de protección de SW hacia el HW y direccionamiento (independiente del tamaño, hasta cierto punto).

Archivos

Son espacios de almacenamiento en memoria secundaria que son manejados por el SO a través de syscalls y que están organizados típicamente en forma de árbol de directorios (los procesos también están organizados como árbol).

Definiciones iniciales

Input/Output - I/O

Se refiere a dispositivos que le ingresan datos al SO y que producen salidas. Así mismo, los drivers son una interfaz fundamental para dichos dispositivos.

Kernel

Es el Software que dirige el funcionamiento fundamental del SO (núcleo del SO) y que ejecuta en modo privilegiado (a diferencia de cualquier otro programa que no forme parte del kernel)

Shell

Es la interfaz de comandos de UNIX y aunque no forma parte del SO*, utiliza syscalls. Existen diversas opciones de Shell para sistemas base UNIX (sh, csh, tcsh, ksh, bash, zsh) y así como en también existen para Sistemas Windows (command.com, cmd.exe, PowerShell)

Procesos

Procesos vs Programa

Un proceso una actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado (actual) y recursos asociados. Es decir, un proceso es la ejecución de un programa.

¿Cuál es la diferencia entre un programa y un proceso?

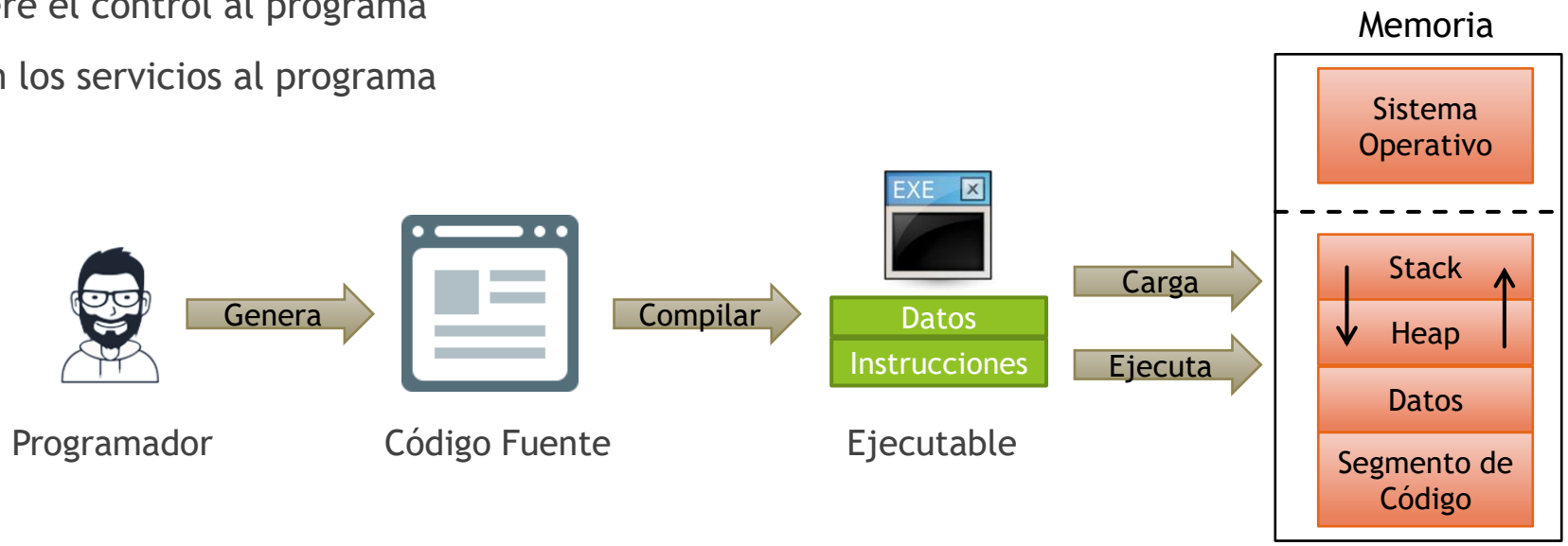
```c
#include <stdio.h>

int main()
{
        printf("Hello, Assembly World.");
        return(0);
}
```

```
Lcfi2:
        .cfi_def_cfa_register %rbp
        subq    $16, %rsp
        leaq    L_.str(%rip), %rdi
        movl    $0, -4(%rbp)
        movb    $0, %al
        callq   _printf
        xorl    %ecx, %ecx
        movl    %eax, -8(%rbp)          ## 4-byte Spill
        movl    %ecx, %eax
        addq    $16, %rsp
        popq    %rbp
        retq
        .cfi_endproc
                                        ## -- End function
        .section        __TEXT,__cstring,cstring_literals
L_.str:                                 ## @.str
        .asciz  "Hello, Assembly World."
```

Ejecución de un programa (proceso)

La ejecución de un programa se realiza a través de los siguientes pasos:

- Se cargan las instrucciones y los datos en un archivo ejecutable en la memoria
- Se crea un Stack* y un Heap*
- Se transfiere el control al programa
- Se proveen los servicios al programa



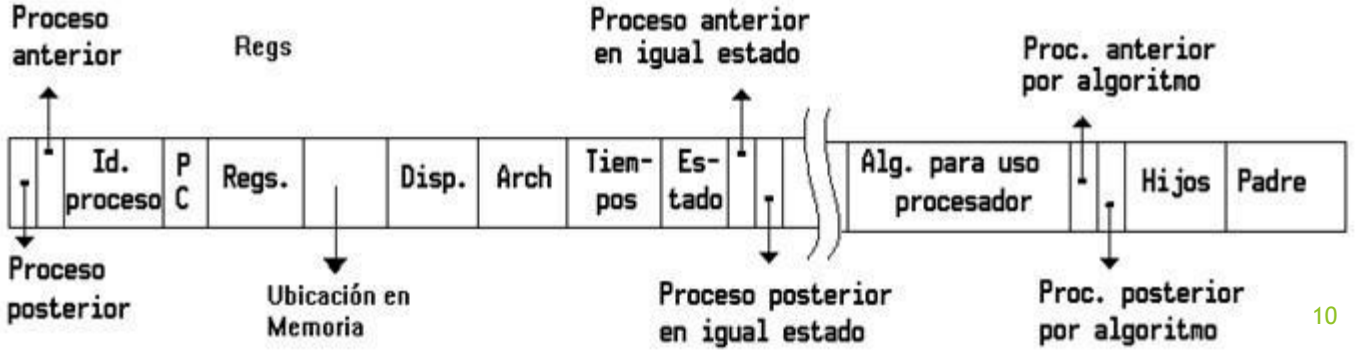
** En el manejo de memoria un Stack es la memoria reservada para **un hilo** (thread) de ejecución y el Heap es la memoria reservada para la ejecución dinámica.*

Bloque de Control de Procesos (PCB)

El PCB “es un registro especial donde el sistema operativo agrupa toda la información que necesita conocer respecto a un proceso particular. Cada vez que se crea un proceso el sistema operativo crea el PCB correspondiente para que sirva como descripción en tiempo de ejecución durante toda la vida del proceso”.

Cada vez que un nuevo proceso aparece se crea un PCB. Luego:

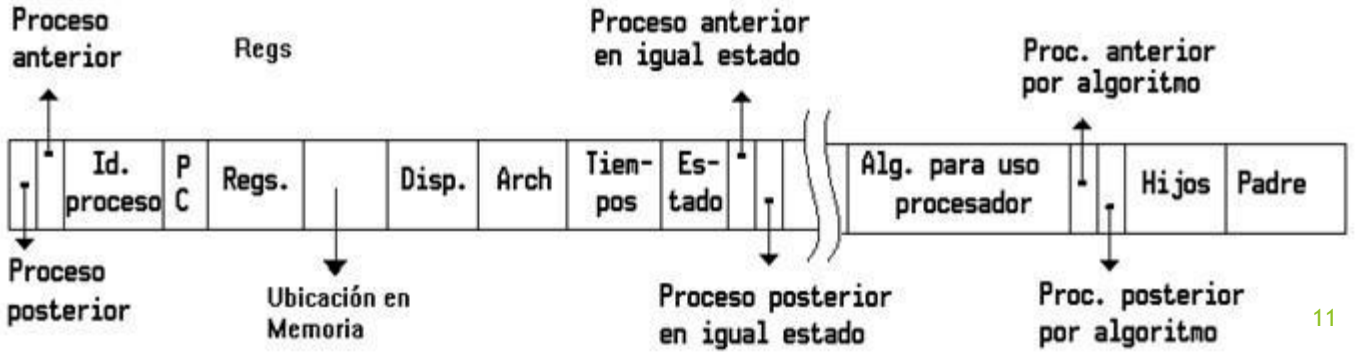
- El proceso es reconocido para el sistema operativo.
- Se hace elegible para competir por los recursos del sistema (dado que el PCB está activo y asociado a dicho proceso).
- Cuando el proceso termina, el PCB es eliminado para dejar espacio libre en el registro (y poder almacenar otro PCB).



Información almacenada en un PCB

Aunque esto difiere según el S.O, existen algunos aspectos comunes. Entre ellos:

- Identificador del proceso (PID) y Estado del proceso (Activo, Bloqueado, etc.)
- Dirección de la próxima instrucción a ejecutar (Program Counter).
- Registros: Información sobre en qué lugar se encontraba la ejecución del proceso, así como el lugar en el que se detuvo la ejecución del proceso anterior (cada una en su respectivo BCP).
- Espacio de direcciones de memoria y Lista de recursos asignados (ej. Archivos abiertos).
- Relación entre procesos padre e hijos.
- Permisos asignados, Tiempos asociados, Signals pendientes, Prioridad.



Estados fundamentales de un proceso

Nuevo o en Espera

Antes de tener su primer estado “preparado” los procesos pasan por un estado de espera. Esto es desde que son creados hasta que están en condiciones de ser ejecutados. Es similar a pensar en la primera “carga” del proceso.

Activo

El proceso está utilizando la CPU (se encuentra en ejecución). Dependiendo del número de procesadores, podemos tener tantos procesos activos como disponibilidad de cada uno de dichos recursos.

Preparado

Esto se refiere a que el proceso no está en ejecución, pero podría pasar a activo. No tiene limitantes para poder cambiar de estado. El planificador es el que se encargará de dicha transición.

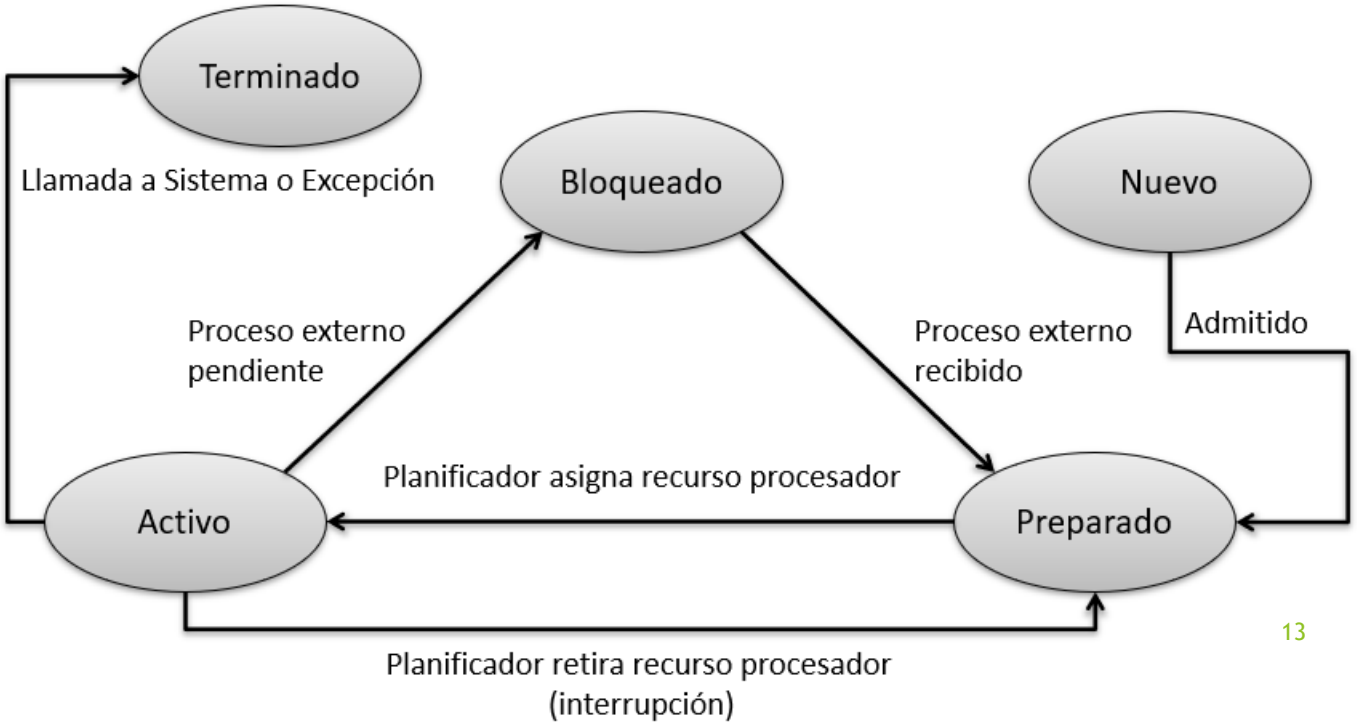
Bloqueado

Este estado se da cuando un proceso está a la espera de un evento. Cuando dicho “evento” ha terminado (ya sea un HW o un SW) emite un aviso mediante una interrupción para que el planificador pueda revisar los procesos bloqueados, identificando a y aquellos involucrados con este evento para poder cambiar su estado.

Estados fundamentales de un proceso

Terminado

Esto ocurre cuando un proceso realiza una llamada a sistema (o excepción) solicitando su propio término y posterior eliminación. El proceso seguirá existiendo (en estado terminado) hasta que todas las estructuras de datos asociadas a dicho proceso sean liberadas para que, posterior a ello, el proceso sea eliminado.



Transiciones en los estados de un proceso

Existen tres casos se puede realizar la transición de los procesos:

Tiempo

Se cambia el estado del proceso si es que se encuentra activo y excede el tiempo máximo de asignación.

Finalización

Simplemente es cuando termina la ejecución del código asociado a un proceso.

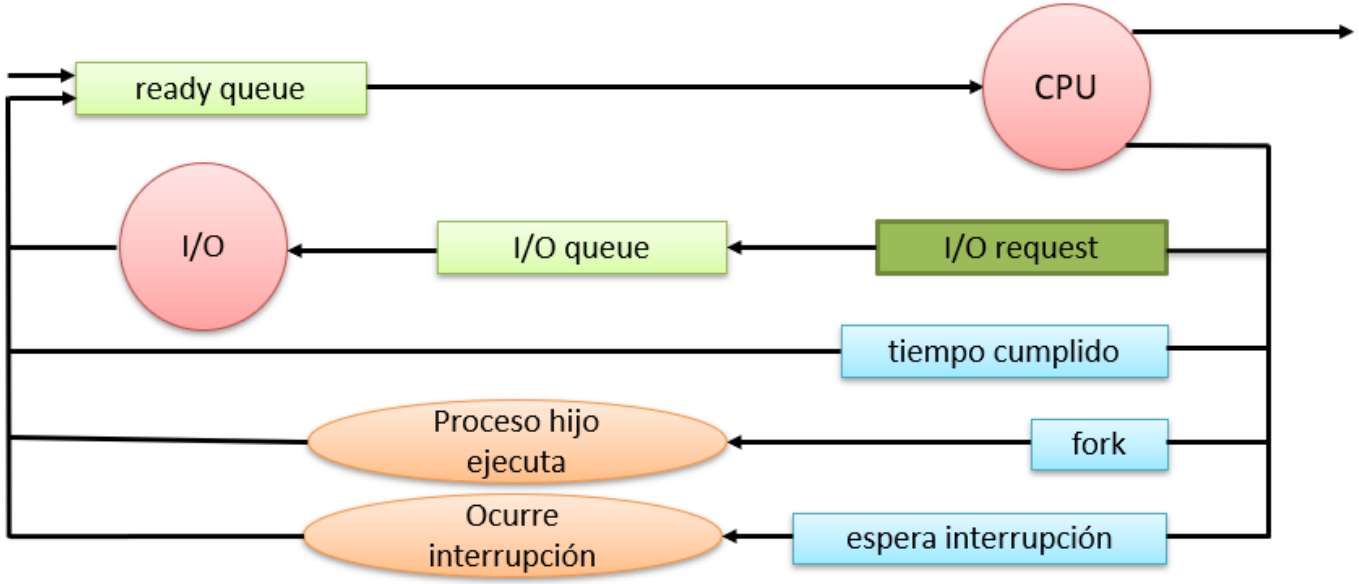
Bloqueo

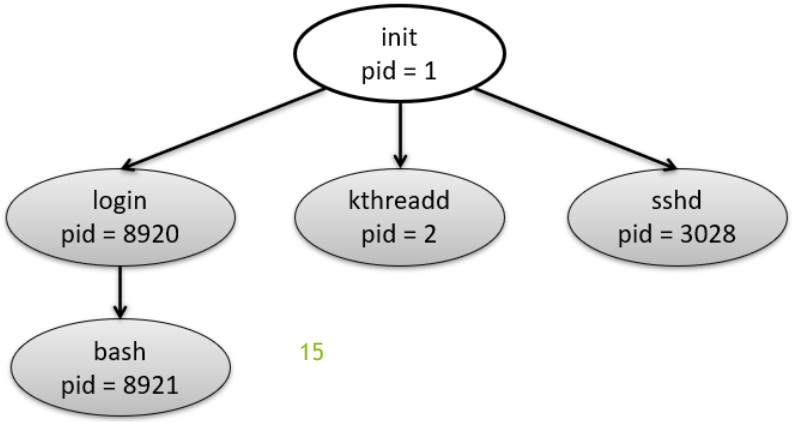
Ocurre cuando un proceso activo hace una llamada a sistema dado que requiere realizar o esperar una acción determinada. Por ejemplo: Lectura (read), escritura (write), espera (wait), entre otras. Cuando esto ocurre, el planificador asigna el procesador a otro proceso (si es que existe uno en estado Preparado).

Árbol de procesos y dinámica de ejecución

Los procesos se vinculan en forma de un árbol donde cada proceso tiene un padre y puede tener 0 o más procesos hijos.

La dinámica de ejecución de los procesos en CPU sigue típicamente la lógica descrita por el diagrama:





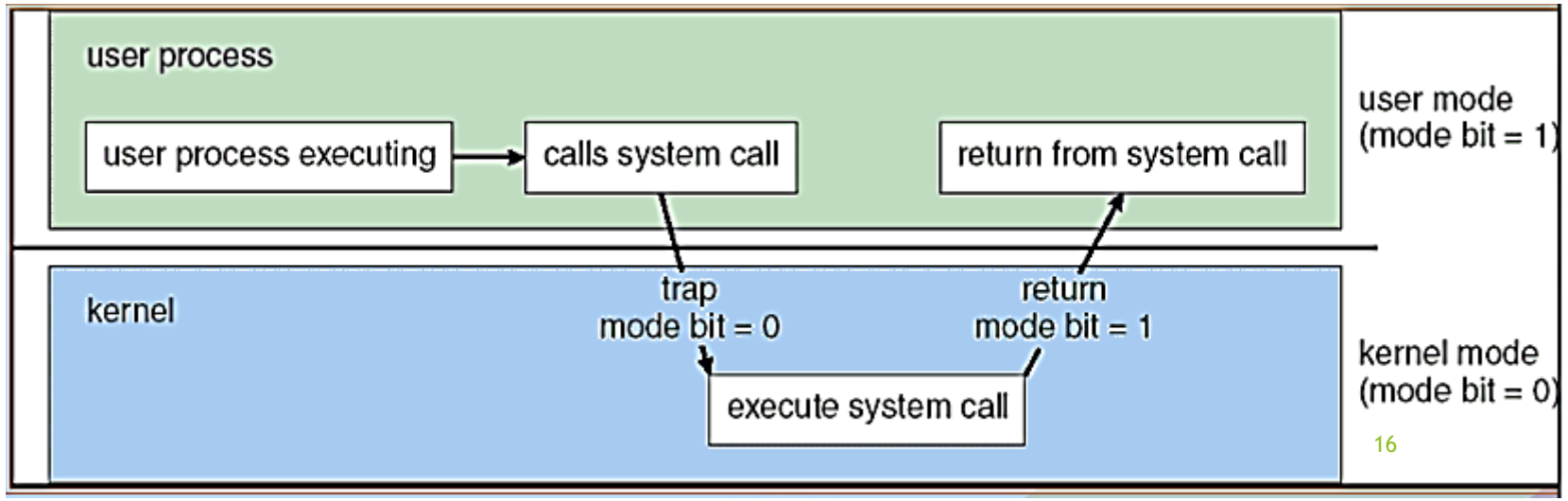
Modo Dual de Operación

El sistema operativo posee dos modos fundamentales para su operación:

- Modo Kernel (privilegiado): En este modo se pueden realizar todas las operaciones.
- Modo Usuario (no privilegiado): Genera una excepción cuando se intenta realizar una instrucción privilegiada.

Cambio de modo

El sistema comienza en Modo Kernel y cambia a Modo usuario cuando cede el control (al usuario) y recupera el control **solo** cuando ocurre una llamada a sistema, interrupción o excepción.



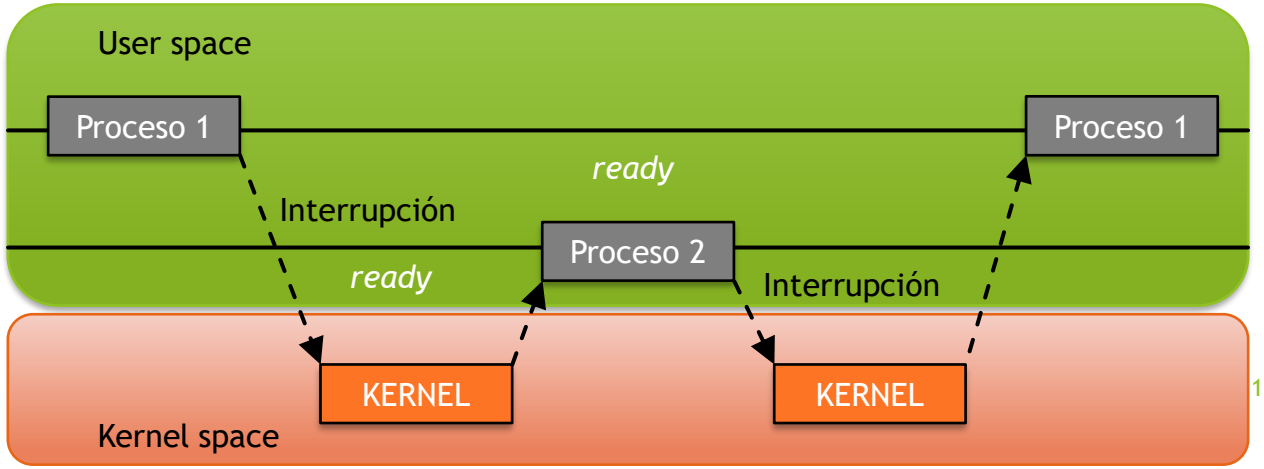
Elementos de control de procesos

El sistema operativo posee distintos elementos para controlar los procesos:

- Interrupciones de HW y de SW (syscalls y excepciones)
- Scheduling
- Sincronización (hablaremos de Threads en esta parte)

Interrupción

Es una señal recibida por el procesador de una computadora, para indicar que debe “interrumpir” el curso de ejecución actual y pasar a ejecutar código específico para tratar esta situación.



Interrupciones

Excepciones

Son una interrupción sincrónica dada por un error en la ejecución de un programa. Ejemplo: Driver incompatible.

Syscalls

Ocurre cuando un programa en ejecución llega a una instrucción que requiere del sistema operativo para alguna tarea. Ejemplo: Leer un archivo en el disco. Veamos algunas syscalls:

- Open: Se utiliza cuando un programa inicializa el acceso a un archivo. Usualmente requiere de dos argumentos:
  - La ruta del archivo
  - El tipo de acceso que se desea (lectura, escritura, etc.)
- Read: Se utiliza cuando un programa requiere acceder a los datos de un archivo almacenado en el sistema de archivos.
- Write: Se utiliza cuando se desea escribir información desde un buffer declarado por el usuario a un componente específico (por ejemplo, un archivo).

Interrupciones

- Close: Se utiliza cuando un programa finaliza el acceso a un archivo.

- Fork: Es una llamada que permite crear una copia (hijo) de un proceso determinado (padre). Ambos tendrán distinto identificador de proceso (PID) y el padre tendrá “conocimiento” del PID del hijo, más el hijo no (en esa variable tendrá un cero).

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]){

    pid_t me, childpid;
    me = getpid();
    printf("%d\n", me);
    childpid = fork();
    printf("%d\n", childpid);

    return 0;
}
```

```
Tin@tinslair:Programas$ ./fork1
22659
22660
0
```

Interrupciones

- Exec: Permite poner en ejecución un proceso. También se relaciona con la “superposición”: Ejecutar un proceso donde se está (o estaba) ejecutando otro.

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]){

    pid_t me, childpid;
    me = getpid();
    printf("%d\n", me);
    childpid = fork();

    if(childpid == 0)
    {
        printf("Proceso hijo: pronto a llamar a exec().\n");
        execlp("./wazza","",NULL);
        printf("Proceso hijo: exec() terminado.\n");
    }
    else if(childpid > 0)
    {
        printf("Proceso padre: el PID del proceso hijo es: %d\n", childpid);
    }

    return 0;
}
```

```
[Tin@tinslair:Programas$ ./exec1
 22920
 Proceso padre: el PID del proceso hijo es: 22921
 Proceso hijo: pronto a llamar a exec().
 WAZZAAAAAAAAAA!!!
```

Interrupciones

- Wait: Es una llamada que permite que un proceso espere a que se ejecute otro proceso. El proceso que “espera” está pendiente de una señal (signal) que le permita indicar que puede reanudar su ejecución.

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]){

    pid_t me, childpid;
    me = getpid();
    printf("%d\n", me);
    childpid = fork();

    if(childpid == 0)
    {
        printf("Proceso hijo: pronto a llamar a exec().\n");
        execlp("./wazza","",NULL);
        printf("Proceso hijo: exec() terminado.\n");
    }
    else if(childpid > 0)
    {
        printf("Proceso padre: el PID del proceso hijo es: %d\n", childpid);
        wait(NULL);
    }

    printf("Sugar daddy: El proceso hijo termino.\n");

    return 0;
}
```

```
[Tin@tinslair:Programas$ ./wait1
 23069
 Proceso padre: el PID del proceso hijo es: 23070
 Proceso hijo: pronto a llamar a exec().
 WAZZAAAAAAAAAA!!!
 Sugar daddy: El proceso hijo termino.
```

Interrupciones

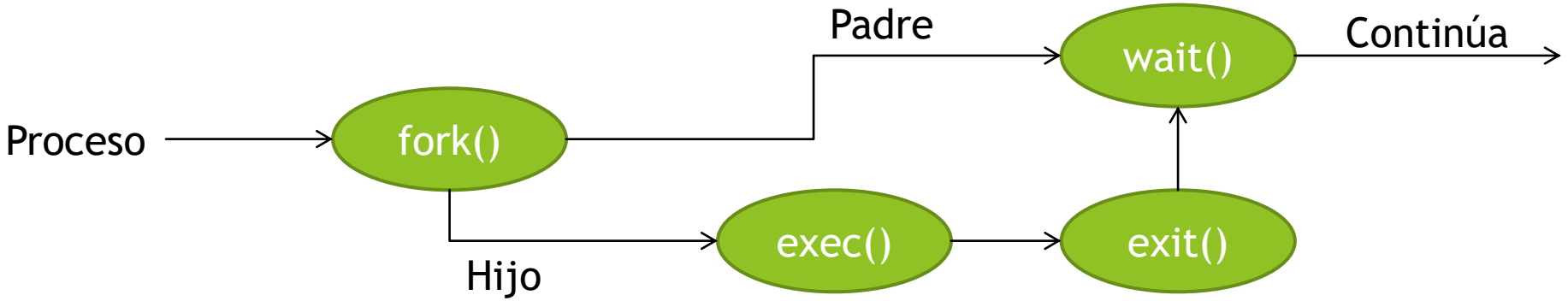
- Exit: Termina la ejecución de un proceso en el contexto ideal.

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]){

    int status;
    pid_t me, childpid;
    me = getpid();
    printf("%d\n", me);
    childpid = fork();

    if(childpid == 0)
    {
        printf("Proceso hijo: Ejecutando.\n");
        printf("Bienvenidos a Temuco.\n");
        exit(9);
    }
    else if(childpid > 0)
    {
        printf("Proceso padre: el PID del proceso hijo es: %d\n", childpid);
        wait(&status);
    }

    printf("Sugar daddy: El proceso hijo termino.\n");
    printf("Status de salida: %d\n", WEXITSTATUS(status));

    return 0;
}
```

```
Tin@tinslair:Programas$ ./exit1
23145
Proceso padre: el PID del proceso hijo es: 23146
Proceso hijo: Ejecutando.
Bienvenidos a Temuco.
Sugar daddy: El proceso hijo termino.
Status de salida: 9
```

Interrupciones

- Kill: Permite terminar con la ejecución de un proceso independiente de su punto de ejecución.



Scheduling

Scheduling

Schedulling

Es el conjunto de políticas y mecanismos construidos dentro del sistema operativo que gobiernan la forma de ejecutar los procesos. Entre ellas:

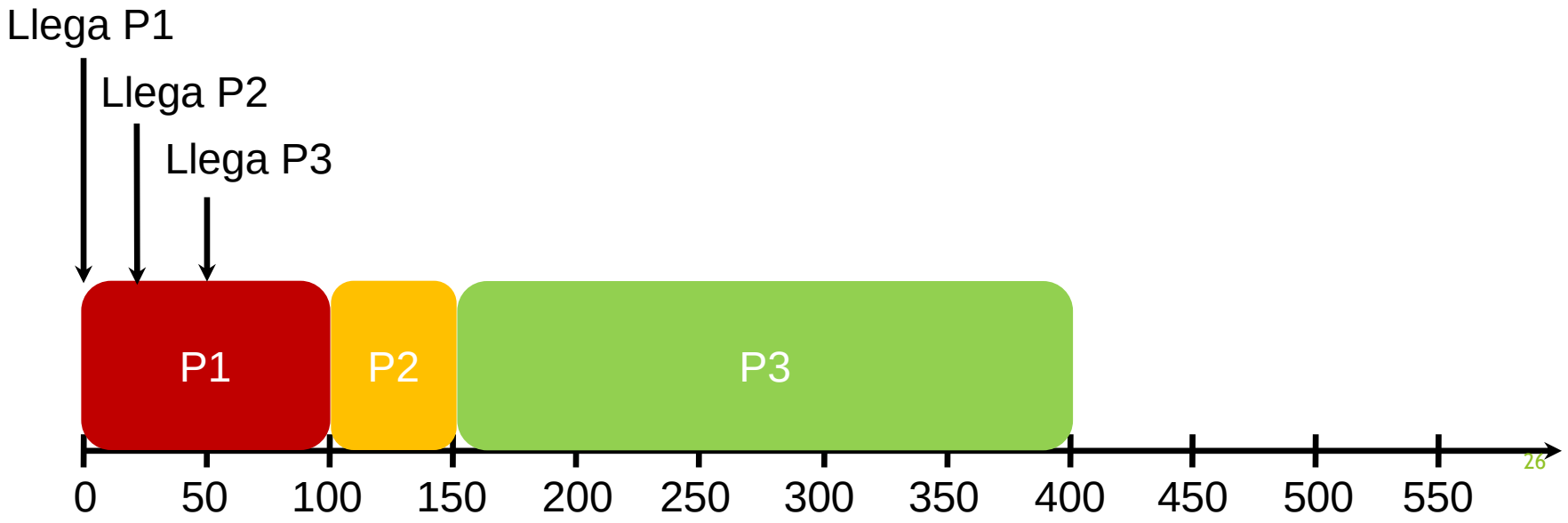
- Scheduling expropiativo
- Scheduling no-expropiativo
- Batch scheduling
- Interactive scheduling
- Real-time scheduling

Existen diferentes algoritmos de Scheduling

Algoritmos de Scheduling

Algoritmo de Calendarización 1: FCFS

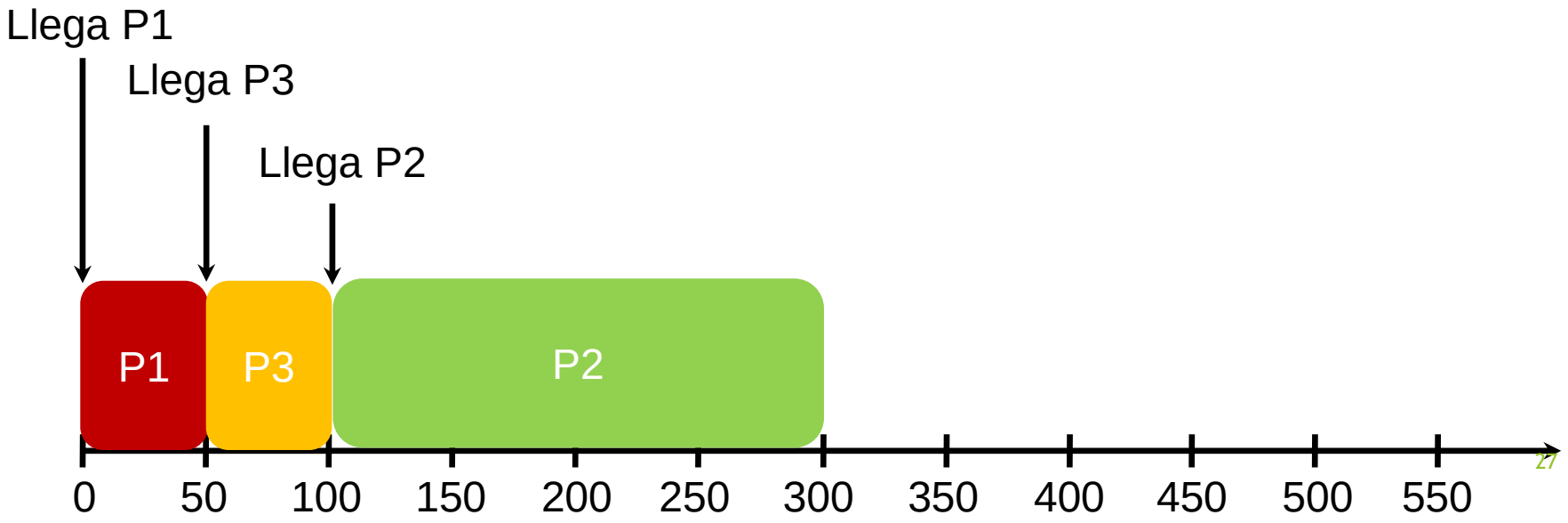
Process	Arrival time	CPU burst	End time	Turnaround
P1	0	100	100	100
P2	20	50	150	130
P3	50	250	400	350



Algoritmos de Scheduling

Algoritmo de Calendarización 2: SJF - Shortest Job First (sin expropiación)

Process	Arrival time	CPU burst	End time	Turnaround
P1	0	50	50	50
P2	100	200	300	200
P3	50	50	100	50



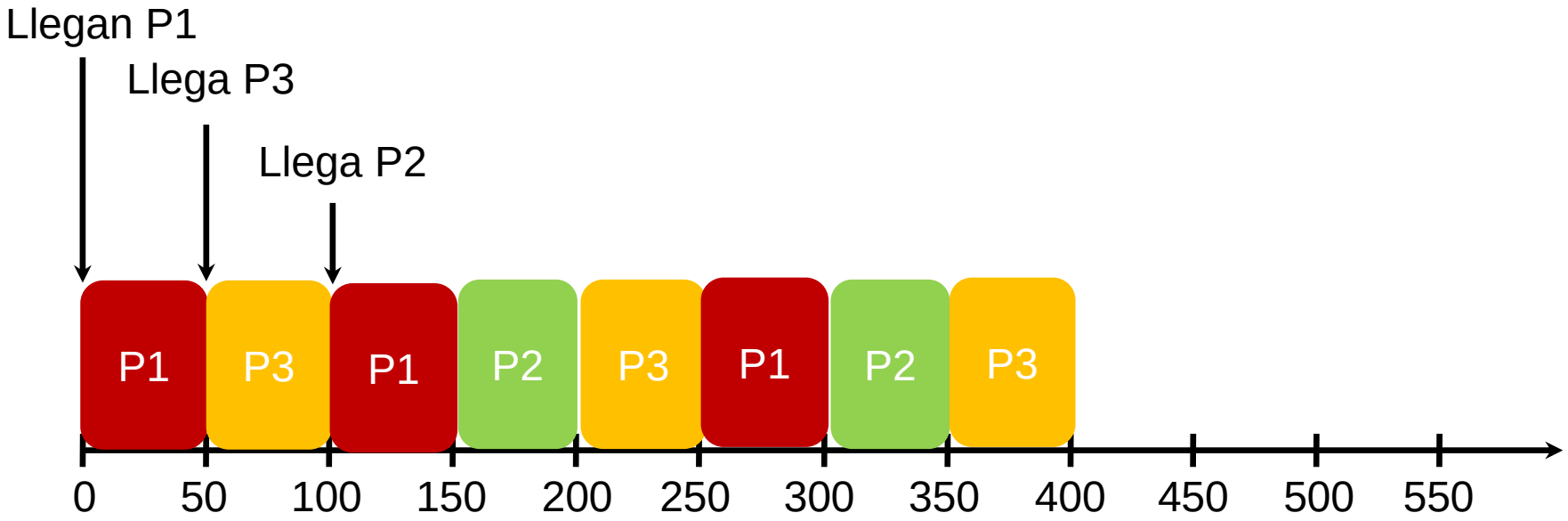
Algoritmos de Scheduling

Algoritmo de Calendarización 3: Round Robin

Cada proceso recibe un quantum de tiempo en la CPU, debiendo abandonarla al término del mismo.

Ejemplo: q = 50

Process	Arrival time	CPU burst	End time	Turnaround	Execution start	Response time
P1	0	150	350	350	0	0
P2	100	100	300	200	100	0
P3	50	150	400	350	50	0

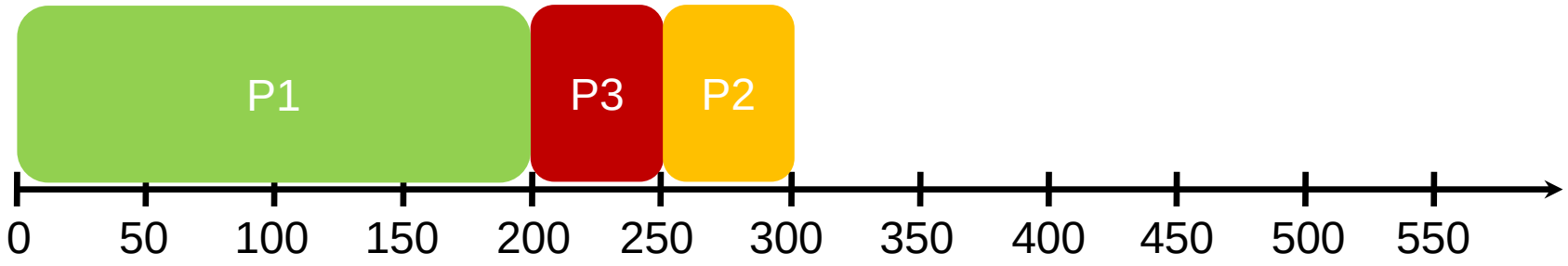


Algoritmos de Scheduling

Algoritmo de Calendarización 4: Por prioridad (sin expropiación)

Process	Arrival time	CPU burst	End time	Turnaround	Execution start	Response time	Priority
P1	0	200	200	200	0	0	63
P2	0.1	50	300	299.9	250	249.9	24
P3	0.2	50	250	249.8	200	199.8	50

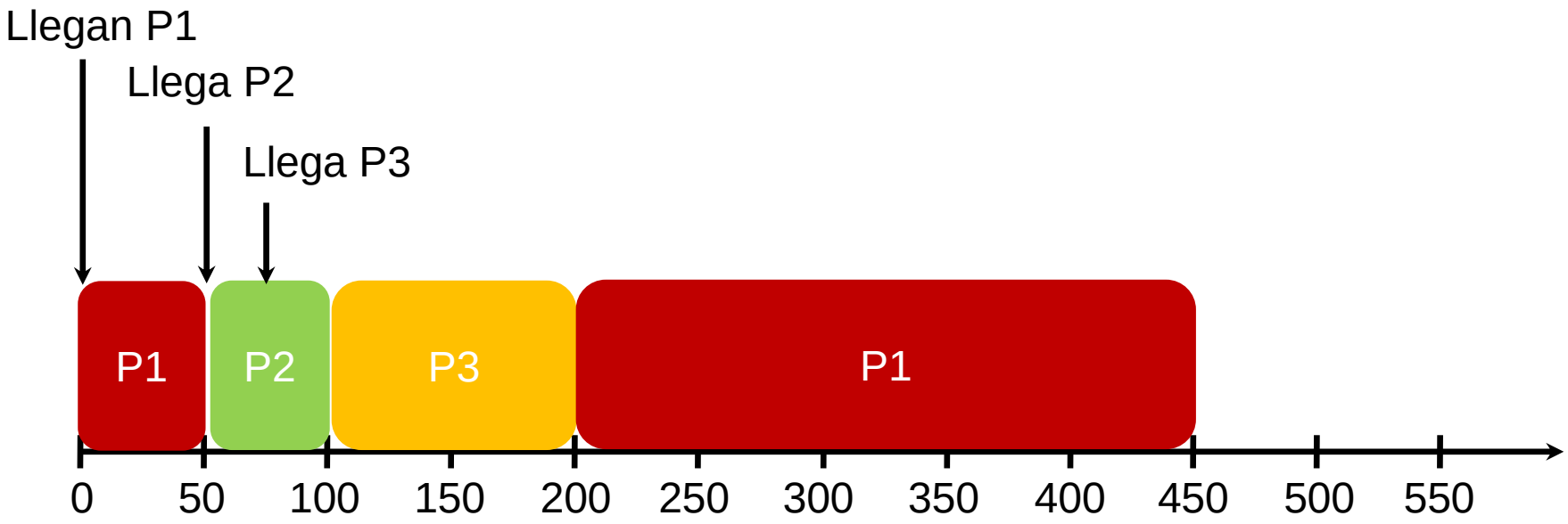
¿Es correcto este ejemplo?



Algoritmos de Scheduling

Algoritmo de Calendarización 5: Earliest Deadline First

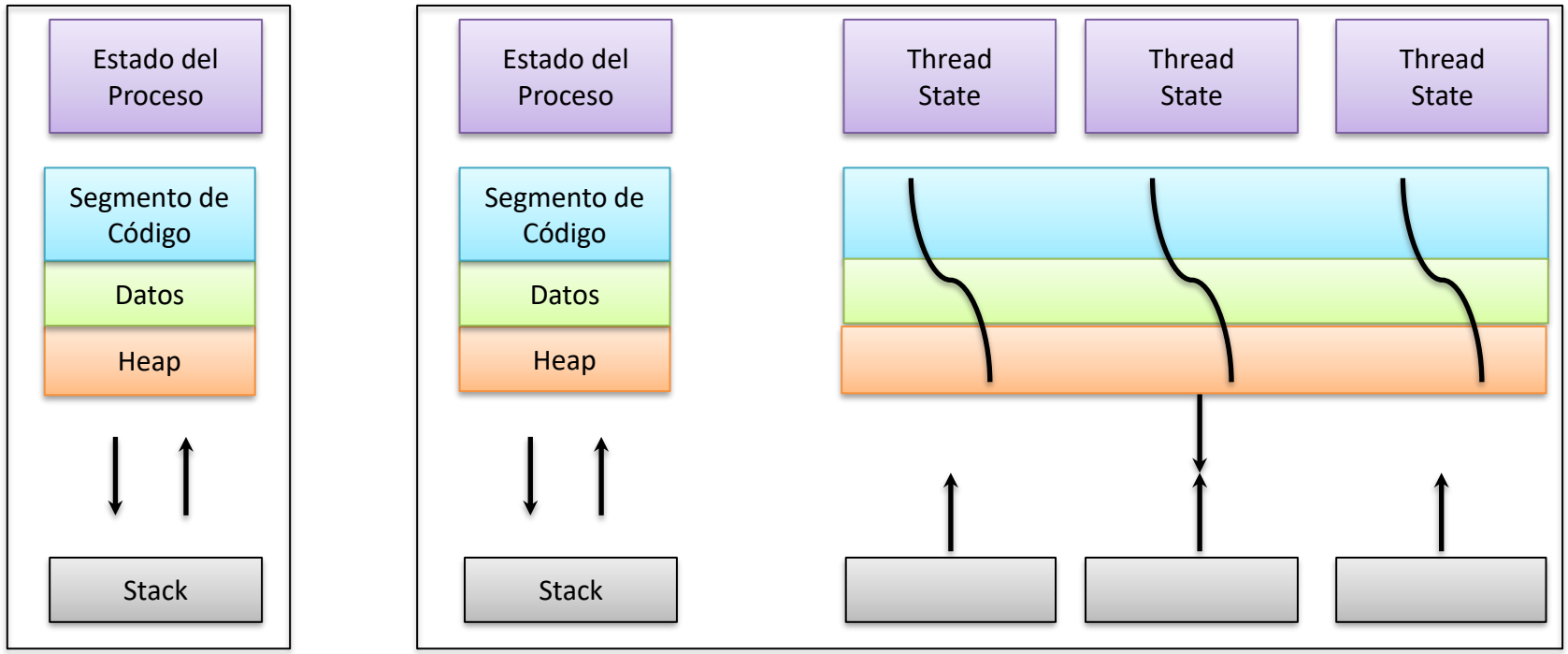
Process	Arrival time	CPU burst	End time	Turnaround	Execution start	Response time	Deadline
P1	0	300	450	450	0	0	500
P2	50	50	100	50	50	0	350
P3	75	100	200	125	100	25	450



Threads y Sincronización

Threads

Un Thread (hilo o hebra de ejecución) es la unidad mínima que se puede planificar en un Sistema Operativo y se refiere a un contexto individual único de ejecución (una tarea que puede ser ejecutada al mismo tiempo con otra tarea).

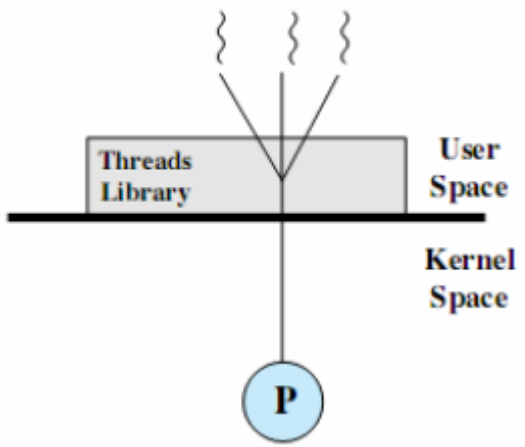


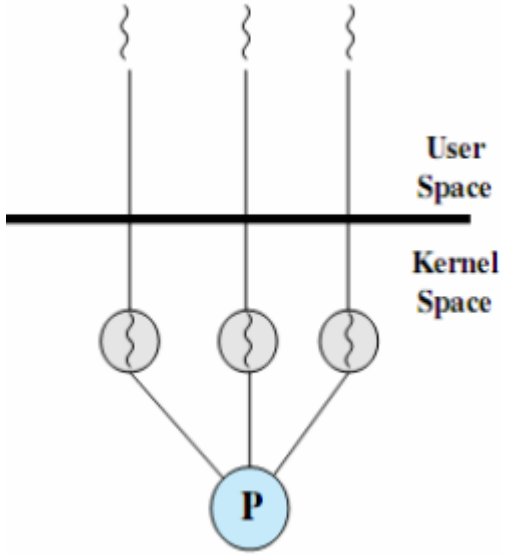
El thread contiene solo la información necesaria como el stack (para variables locales, argumentos, retornos, etc.), una copia de los registros, program counter y data específica del thread para que sean calendarizados individualmente. Toda la demás data es compartida entre todos los threads del proceso.

Tipos de Threads

Threads a nivel de usuario (ULT)

El Sistema operativo maneja procesos por lo que no tiene “conocimiento” de la existencia de Threads. El manejo de ellos se realiza mediante una Biblioteca a nivel de usuario.





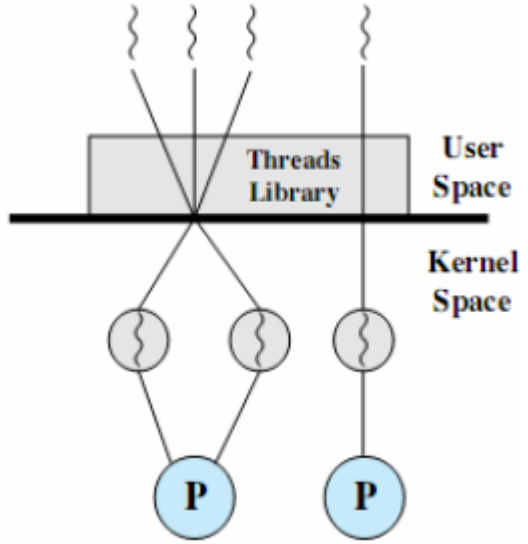
Threads a nivel de kernel (KLT)

El Sistema operativo maneja los procesos como un todo por lo que mantiene información de cada proceso y sus Threads. Así, la planificación es a nivel de procesos por lo hay un Thread de Kernel por cada Thread de usuario.

Tipos de Threads

Esquema combinado

Busca aprovechar las ventajas de ULT y KLT lo que implica que la aplicación es escrita al nivel del paralelismo lógico más conveniente y ejecutada a nivel de paralelismo físico más conveniente. Luego, la aplicación utiliza Threads de usuario y el S.O. planifica Threads de Kernel.



PUNTOS IMPORTANTES

- *CONDICIÓN DE CARRERA*
- *SECCIÓN CRÍTICA*
- *EXCLUSIÓN MUTUA*
- *DEADLOCK*

Sincronización de Procesos: Semáforos

Un semáforo es un Tipo Abstracto de datos que permite el uso de un recurso de forma exclusiva, en el caso de que existan múltiples procesos que requieran acceso a éste (ofrece una solución al problema de la sección crítica) y se refiere a un algoritmo que maneja un entero con tres diferencias:

- Cuando se crea un semáforo, se pueden inicializar su valor con cualquier entero, pero después siempre cambia sumando o restando 1 (una unidad).
- No se puede leer el valor actual del semáforo.
- Cuando un Thread intenta disminuir el semáforo y el valor de dicho semáforo es negativo, el subproceso se bloquea a sí mismo y no puede continuar hasta que otro subproceso incremente el semáforo. Así, cuando un Thread incrementa el semáforo, si hay otros threads esperando, uno de ellos se desbloquea.

AL MARGEN

El concepto de semáforo fue inventando por Esdger Dijkstra

TDA Semáforo

Este TDA cumple con:

- El valor del semáforo cuenta cuántos procesos pueden usar el recurso.
- Posee una operación de inicialización: init()
- Posee dos operaciones para su funcionamiento: wait() y signal()

Init

Inicia el semáforo con el número máximo de procesos que tiene derecho a acceder al recurso. Si este semáforo se inicializa con 1, se convierte en un semáforo binario.

Wait

Si el valor del semáforo es mayor a cero, indica que un proceso más puede acceder al recurso, el contador del semáforo se disminuye en uno y el proceso continúa con éxito. Si el semáforo es cero, el proceso queda esperando hasta que es despertado por otro proceso.

Signal

Cuando el proceso ha terminado de utilizar el recurso, éste lo indica al semáforo mediante signal(), el que aumenta el valor del semáforo en 1. Si existe algún proceso esperando su turno para utilizar el recurso, el proceso que finaliza lo puede despertar* o, de lo contrario, se incrementa el contador.

Ejemplo de Semáforo

```
from threading import Thread, Semaphore
import time, random


class MiThread(Thread):
    def __init__(self, numId, semaforo):
        Thread.__init__(self)
        self.semaforo=semaforo
        self.numId = numId

    def run(self):
        semaforo.acquire()
        print ("\nEntra el thread "+str(self.numId))
        d.append(self.numId)
        time.sleep(random.randrange(1,6,1))
        print ("\nTermina el thread " + str(self.numId))
        semaforo.release()

d = []

semaforo = Semaphore(3) #Genera un Semáforo cuya capacidad es 3
for i in range(0,6):
    thread=MiThread(i+1,semaforo)
    thread.start()
    print ("\nIniciando el thread "+str(i+1))

print(d)
```

Sincronización de Procesos: Monitor

Definición 1

Los monitores son estructuras de un lenguaje de programación que ofrecen una funcionalidad equivalente a la de los semáforos , pero son más fáciles de controlar.

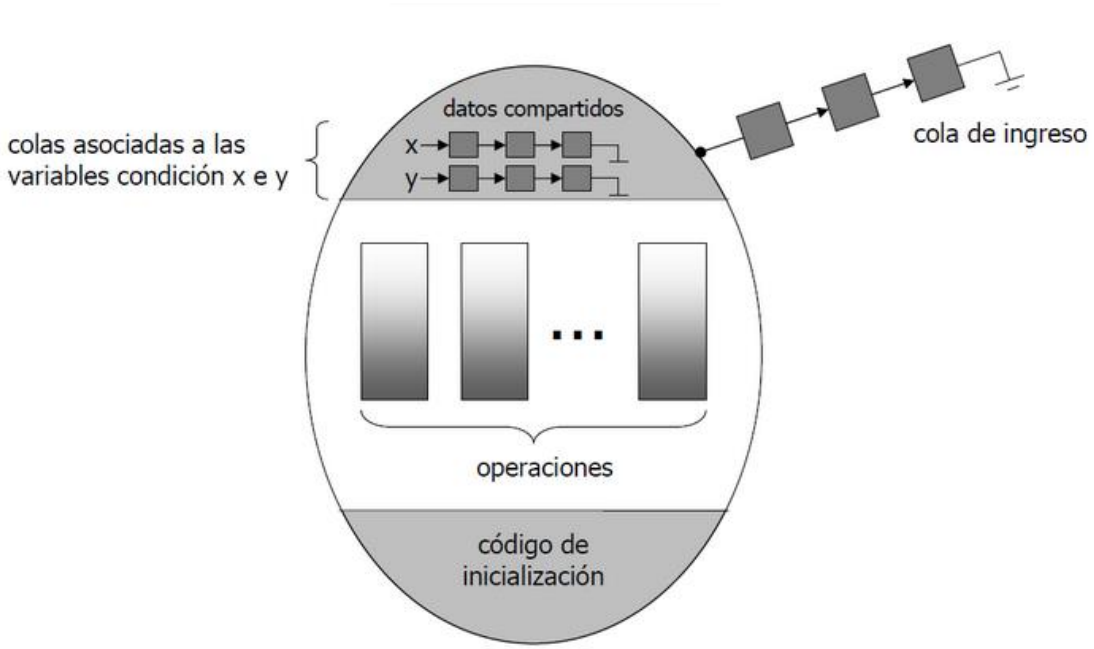
Definición 2

Es un mecanismo de sincronización que permite asignar un recurso (o grupo de recursos) en forma concurrente a un grupo de procesos.

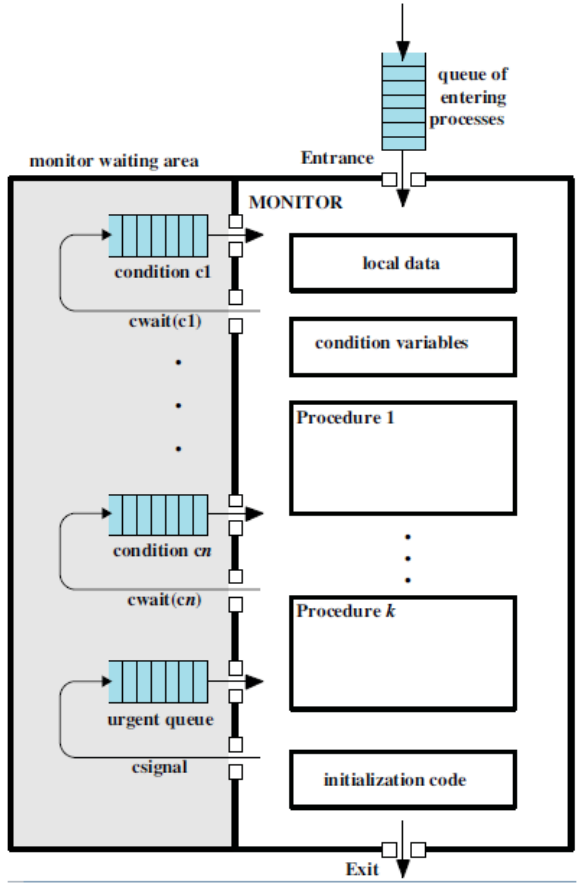
¿Cómo funciona?

Similar a los semáforos, pero con la diferencia que su operación de “Wait” y “Signal” se basan en alguna condición

Sincronización de Procesos: Monitor



Características de un Monitor



Estructura de un Monitor

Fuente imágenes: *http://wiki.inf.utfsm.cl/index.php?title=Monitores*

Operaciones de un Monitor

cwait(x)

Suspende la ejecución del proceso bajo la condición x dejando el monitor disponible para otros procesos

csignal(x)

Reanuda la ejecución de algún proceso suspendido bajo la condición x. Si un proceso ejecuta csignal(x) y no hay procesos suspendidos bajo x el csignal se pierde.

¿Qué política de ingreso al Monitor permite evitar la inanición?

La cola es, en simple, un buffer con una capacidad limitada.

Para controlar esto, el monitor posee procesos internos llamados add y take .

Pseudo ejemplo de un Monitor

```
class  PrinterRoom():

  def  __init__(self):
         self.printerBusy = False
         self.printerLock = Lock()
         self.printerAvailable = Condition(self.printerLock)

  def  waitforPrinter(self):
         with printerLock:                    #acquire the monitor lock
           while printerBusy:          #check if you need to wait
             printerAvailable.wait()          #wait until somebody notifies you
           printerBusy = True                 #set printerBusy to True, to make others wait

  def  donewithPrinter(self):
         with printerLock:                    #acquire the monitor lock
           printerBusy = False                #set printerBusy to False, when you are done
           printerAvailable.notify()      #notify one person waiting for the printer to be available

class  Student():

  def  printdocument(self):
         pr = PrinterRoom()
         while True:
           pr.waitforPrinter()
           use printer now
           pr.donewithPrinter()
           chat with friends and hang out
```


Fuente Ejemplo: *http://www.cs.cornell.edu/courses/cs4410/2011fa/mp1-faq.php*

El problema de la sección crítica

En este problema tenemos *N* procesos *P = {P1,P2, … , Pn}* en el que cada uno tiene un fragmento de código, que se denomina *sección crítica*. Dentro de la sección crítica, los procesos pueden acceder y modificar recursos compartidos como: Variables comunes, Registros de una base de datos, Archivos, entre otros

La característica más importante de este sistema es que cuando un proceso se encuentra ejecutando código de la sección crítica, ningún otro proceso puede ejecutar en su sección. Luego, cualquier solución que se utilice para resolver este problema debe cumplir con 3 aspectos:

- **Exclusión mutua:** Si un proceso está ejecutando código de la sección crítica, ningún otro proceso lo podrá hacer.
- **Progreso**: Si no hay procesos ejecutando dentro de la sección crítica, la decisión de qué proceso entra en la sección se hará sobre los procesos que requieren ingresar.
- **Espera acotada**: Debe haber un límite en el número de veces que se permite que los demás procesos entren a ejecutar código de la sección crítica después de que un proceso haya efectuado una solicitud de entrada y antes de que se conceda la suya.

El problema de la sección crítica

El procedimiento general para resolver este problema se basa en:

1. Cada proceso debe solicitar permiso para entrar en la sección crítica mediante algún fragmento de código, que se denomina de forma genérica **entrada en la sección crítica**.

2. Cuando un proceso sale de la sección crítica debe indicarlo mediante otro fragmento de código, que se denomina **salida de la sección crítica**. Este fragmento permitirá que otros procesos ingresen para ejecutar el código de la sección crítica.

La estructura general, por tanto, de cualquier mecanismo que pretenda resolver el problema de la sección crítica es la siguiente:

Entrada en la sección crítica

Código de la sección crítica

Salida de la sección crítica

El problema del Productor – Consumidor

El problema del Productor/Consumidor es uno de los más comunes cuando se programan aplicaciones utilizando procesos concurrentes. Se caracteriza porque uno o más procesos "productores" generan datos que utilizarán otros procesos llamados "consumidores".

Para resolver este problema debemos tener un mecanismo de comunicación que:

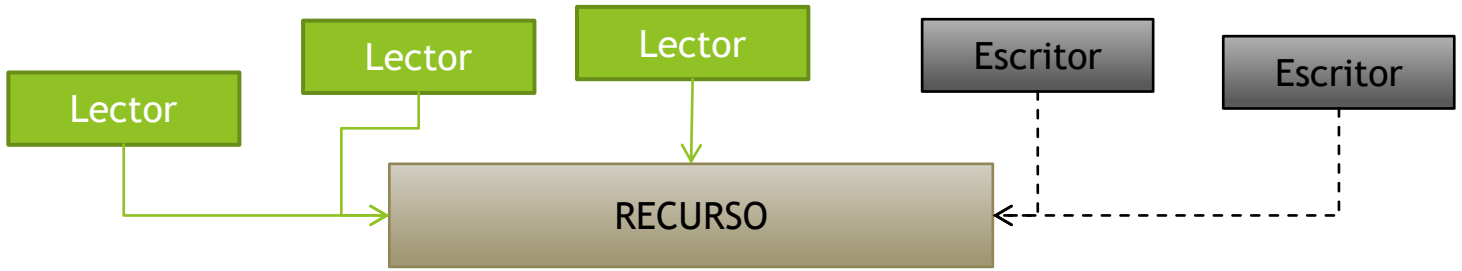
- Permita a los productores y consumidores intercambiar información.
- Ambos procesos deben sincronizar su acceso al mecanismo de comunicación para que la interacción entre ellos no sea problemática:
  - *Cuando el mecanismo de comunicación se llene, el proceso productor se bloquea hasta que haya espacio para seguir insertando elementos.*
  - *El proceso consumidor se bloquea cuando el mecanismo de comunicación este vacío (al no disponer de información).*

En resumen, este tipo de problema requiere servicios para que los procesos se puedan comunicar y servicios para que se sincronicen a la hora de acceder al mecanismo de comunicación.

El problema de Lectura - Escritura

En este problema existe un determinado objeto (ej. un archivo) que va a ser utilizado y compartido por una serie de procesos concurrentes. En el contexto:

- Algunos procesos – lectores – sólo van a acceder al objeto (sin modificar).
- Otros procesos – escritores – van a acceder al objeto para modificar su contenido.

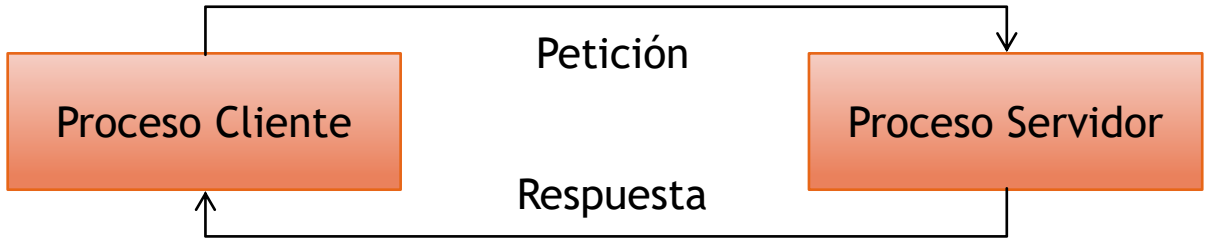


Para resolver este tipo de problema existe una serie de restricciones:

- Sólo se permite que un escritor tenga acceso al objeto al en un momento del tiempo.
- Mientras el escritor esté accediendo al objeto, ningún otro proceso lector ni escritor podrá acceder a él.
- Se permite, sin embargo, que múltiples lectores tengan acceso al objeto (porque no van a modificar).

El problema del modelo Cliente - Servidor

En el modelo cliente-servidor, los procesos llamados servidores ofrecen una serie de servicios a otros procesos que se denominan clientes.



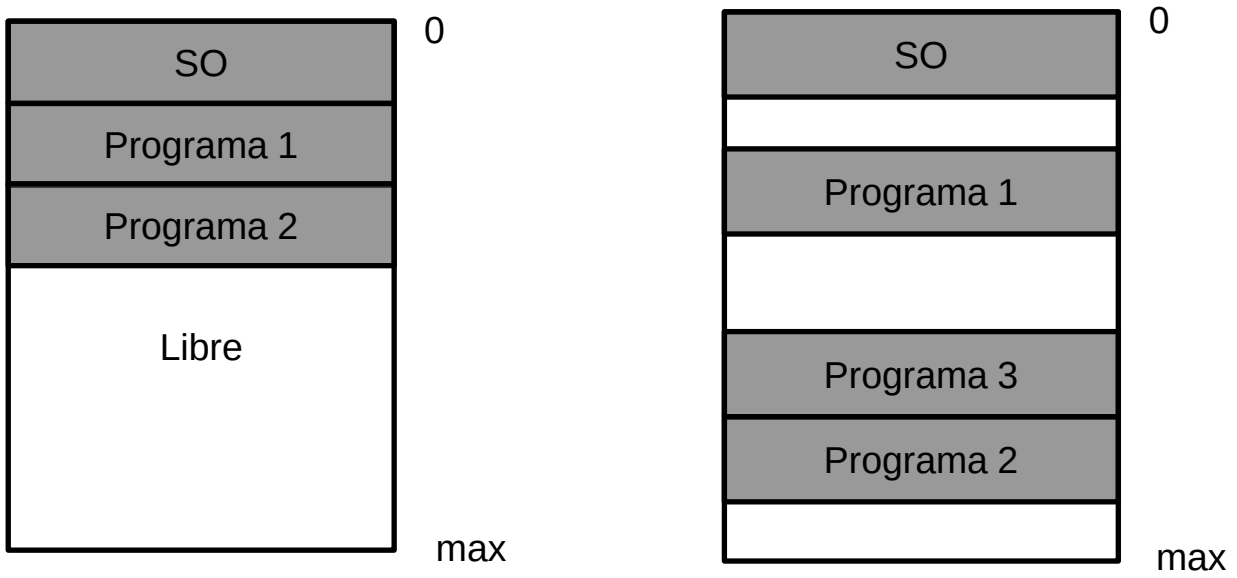
En este tipo de aplicaciones es necesario que el sistema operativo ofrezca servicios que permitan la comunicación entre los procesos cliente y servidor.

- Cuando los procesos ejecutan en la misma máquina, se pueden emplear técnicas basadas en memoria compartida o archivos.
- Cuando las aplicaciones que ejecutan en equipos diferentes (no comparten memoria), se usan técnicas basadas en paso de mensajes.

Administración de Memoria

Memoria

La memoria es, en concreto, el grupo de circuitos que permiten almacenar y recuperar la información. En esto,la Random Access Memory (RAM) es la memoria utilizada por un procesador para recibir instrucciones y guardar los resultados.

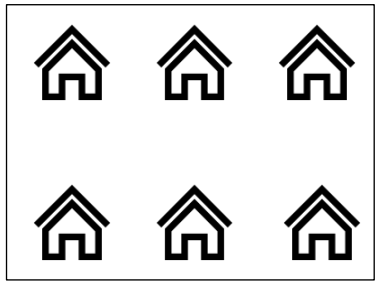


Contexto ideal **Contexto real**

Espacio de Direcciones

El espacio de direcciones del procesador se refiere a un rango de direcciones discretas, donde cada una puede corresponder a un registro de memoria físico o virtual, un dispositivo periférico, un host de red, un sector de disco, u otra entidad lógica o física.

Dirección física



Espacio de Direcciones

Múltiples espacios de Direcciones

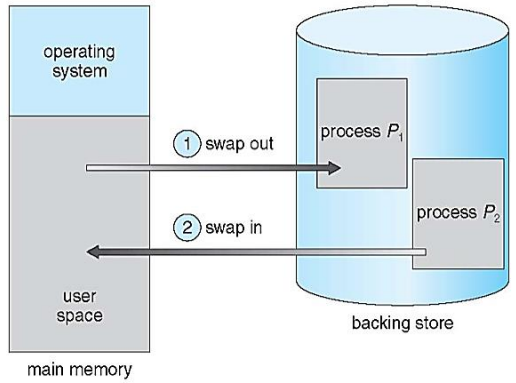
La MMU (Memory Management Unit) es el hardware necesario para producir la traducción de memoria lógica a memoria física. Este proceso, llamado paginación, contempla:

- El espacio de direcciones se divide en unidades llamadas páginas.
- Las unidades en la memoria física que les corresponden a las páginas se denominan marcos.
- Se utiliza una tabla de páginas para mantener la relación entre las direcciones virtuales y la memoria física.
- Las unidades de las páginas y de los marcos son siempre del mismo tamaño.

Swapping y Segmentación

El swapping es el procedimiento de mover temporalmente un proceso (o parte de él) desde la memoria principal (RAM) a un dispositivo secundario de almacenamiento (Disco Duro) para luego devolverlo a la memoria principal. Tiene dos operaciones:

- Swap-in: Es cuando retornamos desde la memoria secundaria a la principal
- Swap-out: Es cuando movemos desde la memoria principal a la memoria secundaria



Estrategias de Swapping

Para ubicar los procesos en memoria, al haber dinamismo, es necesario contar con estrategias que permitan "facilitar" el procedimiento.

- Desfragmentación (compactación)
- First-fit
- Best-fit
- Worst-fit

Desfragmentación

Es el proceso conveniente mediante el cual se acomodan los archivos en un disco para que no se aprecien fragmentos de cada uno de ellos, de tal manera que quede contiguo el archivo y sin espacios dentro del mismo. Existen dos tipos de fragmentación:

- ***Interna***

Es la pérdida de espacio en disco debido a que el tamaño de un determinado archivo sea inferior al tamaño del clúster, ya que teóricamente el archivo estaría obligado a ser referenciado como un clúster completo.

- ***Externa***

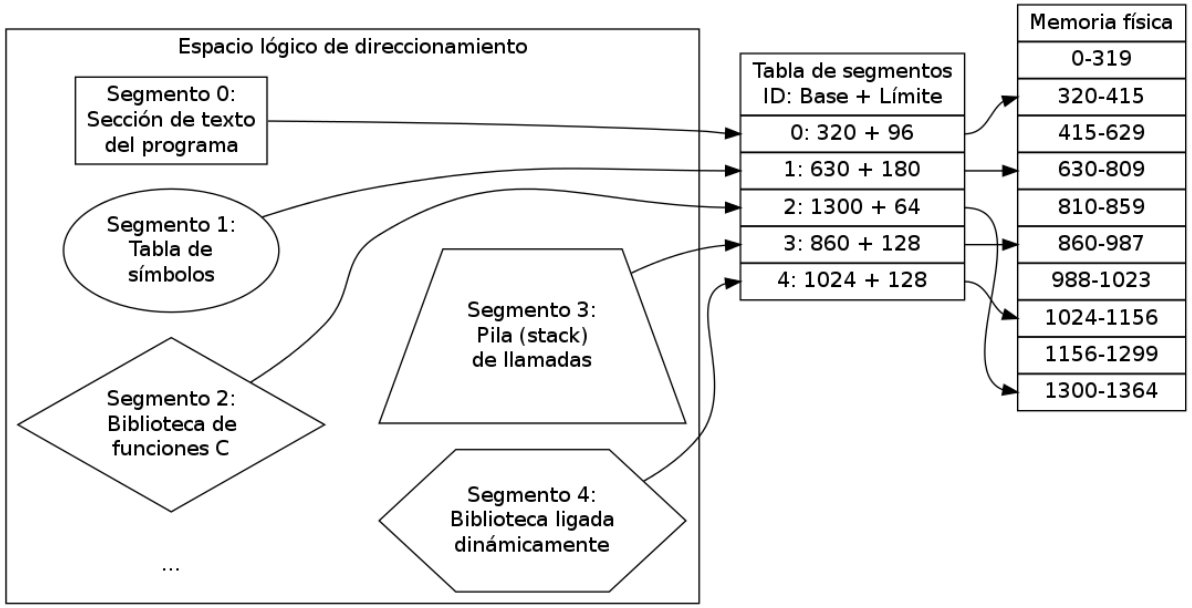
Es consecuencia de las distintas políticas de ajuste de bloques que tiene un sistema de ficheros, o al utilizar asignaciones dinámicas de bloques en el caso de la memoria.

IMPORTANTE

El problema de la fragmentación se relaciona con otro problema: Asignación de bloques de memoria a un proceso.

Segmentación

Es la división de la memoria asignada a un proceso en segmentos más pequeños y persigue que la asignación a espacios en memoria sea más fácil. La MMU (Memory Management Unit) se encarga de llevar registro de esa asignación y almacena la Segment Table (en que cada segmento tiene base y limit); en simple, en vez de dar una dirección lineal, damos al procesador una dirección de segmento y un desplazamiento dentro de dicho segmento.



Segmentation fault

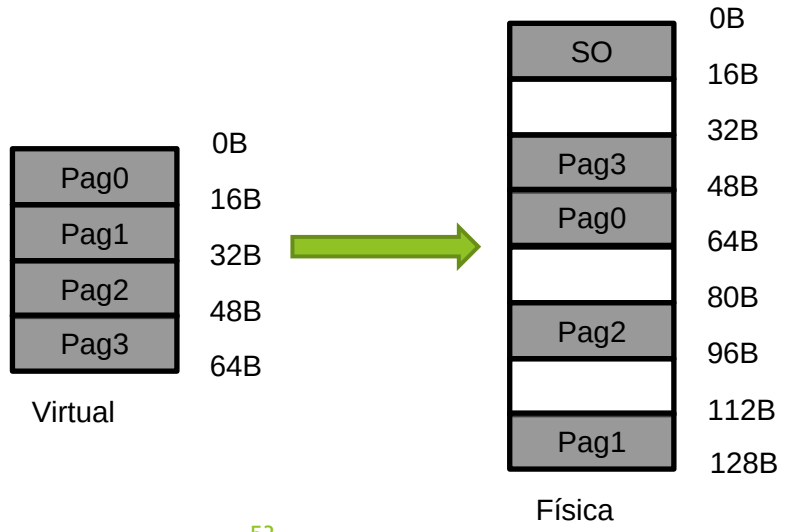
Se refiere al intento fallido de acceso a información (o programas) a los que no se tiene autorización para ver o modificar.

Paginación

La paginación es una estrategia de organización de la memoria física que consiste en dividir la memoria en porciones de igual tamaño. A dichas porciones se las conoce como páginas físicas o marcos los que están identificados por un número, conocido como número de página física. Cada página física se asigna en exclusividad a un proceso.

PUNTOS IMPORTANTES

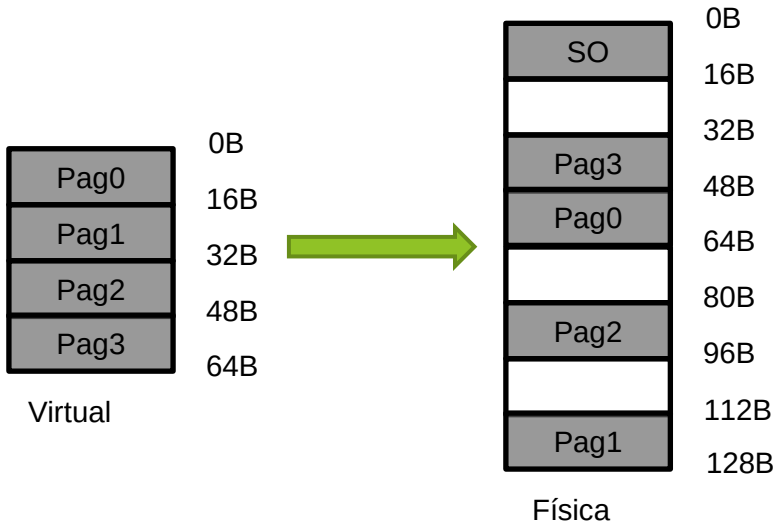
- Si dejamos segmentos como espacios del mismo tamaño obtenemos páginas de memoria
- Las páginas se ubicarán en el espacio “virtual”
- Sus homólogos de espacio físico se llamarán frames o marcos.
- La relación entre el Marco y la Página será 1:1 (tienen el mismo tamaño)
- El registro de las Páginas se lleva, a través de la MMU, en una Tabla de Páginas.



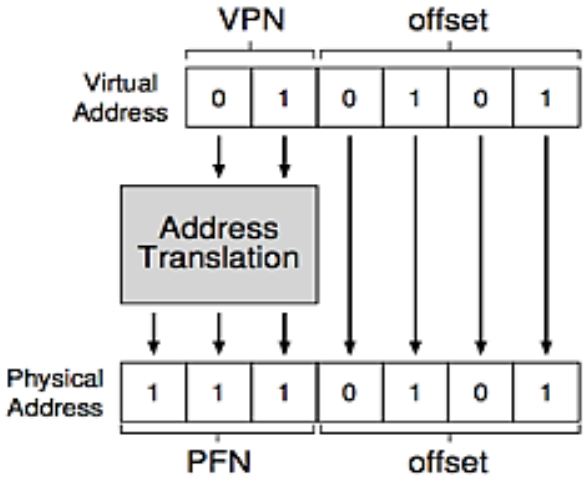
Ejemplo de Paginación

De la imagen, podemos ver que tenemos un espacio virtual de 64B, físico de 128B y páginas de 16B. Luego:

- Direcciones posibles en una página: 16 → 4 bits (esto da el mayor offset posible en una página)
- Espacio virtual de 64B → 6 bits. Dado que 4 bits determinan el offset, los 2 restantes indican el número de páginas (hay 4).
- Espacio físico de 128B → 7 bits. Al igual que para el caso anterior, el offset es determinado por 4 bits, por lo que hay 3 bits para los marcos (son 8).



Se vería así



Ejemplo de Paginación

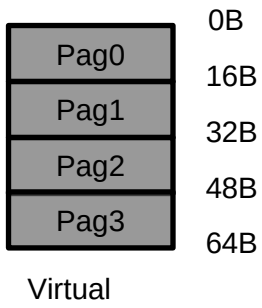
Tomemos por ejemplo la dirección virtual 21 (010101):

- ✓ Página 1 (01)
- ✓ Offset: 5 (0101)
- ✓ Según la page table, página 1 (01) corresponde a frame 7 (111)

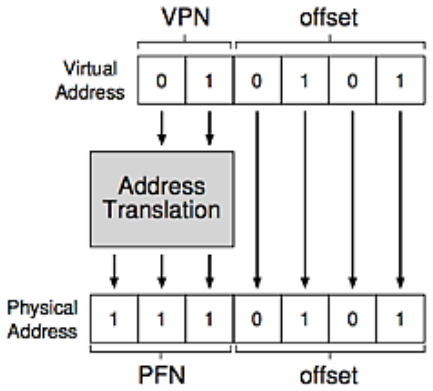
Luego la dirección física es 117 (1110101)

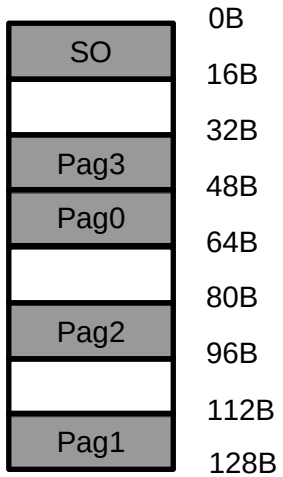
Page	Frame
00 (0)	011 (3)
01 (1)	111 (7)
10 (2)	101 (5)
11 (3)	010 (2)

Page Table



Virtual





Física

Algoritmos de Administración de Memoria

¿Cómo se gestiona un Page Fault?

- Se produce una interrupción
- Se busca la página en almacenamiento secundario (disco)
- Se busca un marco libre, se lee la página de almacenamiento secundario y se copia en el marco seleccionado.
- Se actualiza la tabla de páginas
- Reiniciamos en la instrucción interrumpida

La ocurrencia de una Page Fault es algo “malo” pues implica el acceso a la memoria secundaría… La idea es evitar al máximo que ocurran. Para esto, se necesitan algoritmos y políticas que dirijan la dinámica de paginación:

- Algoritmo óptimo (MIN)
- Algoritmo FIFO
- Algoritmo LRU
- Algoritmo del Reloj

Algoritmo óptimo (MIN)

Es el algoritmo que implica que, al cambiar una página, se debe reemplazar la página que no se va a usar durante más tiempo... IMPRACTICABLE EN LA REALIDAD.

Ejemplo: Supongamos que poseemos la siguiente secuencia de solicitudes y **7 0 1 2 0 3 0 4 2 3 0 3 2** que contamos con 4 espacios de memoria. Contemos los page faults y apliquemos algoritmo:

- 4 page faults de inicio (7 0 1 2)
- 0 page faults del 0
- 1 page fault del 3 -> reemplaza al 7 -> (3 0 1 2)
- 0 page faults del 0
- 1 page fault del 4 -> reemplaza al 1 -> (3 0 4 2)
- 0 page faults hasta el final

Total: 6 page faults

Algoritmo FIFO

Se reemplaza la página que lleva más tiempo en memoria... Esto tiene un costo:

- El rendimiento no siempre es bueno (se pueden sustituir páginas muy usadas)
- Se puede presentar la anomalía de Belady (más marcos no implica que haya menos fallos)

Ejemplo: Supongamos que poseemos la siguiente secuencia de solicitudes **0 1 2 0 1 3 0 3 1 2 1** y que contamos con 3 espacios de memoria. Contemos los page faults y apliquemos algoritmo:

- 3 page faults de inicio (0 1 2)
- 0 page faults del 0 y del 1
- 1 page fault del 3-> reemplaza al 0
- 1 page fault del 0 -> reemplaza al 1
- 0 page faults del 3
- 1 page fault del 1 -> reemplaza al 2
- 1 page fault del 2 -> reemplaza al 3
- 0 page faults del 1

Total: 7 page faults

Algoritmo LRU (Least Recently Used)

Sustituye la página que más tiempo lleva sin ser usada (esto implica tener un contador) y:

- Cada vez que accedemos a memoria se incrementa su valor
- Se copia el valor del contador en la tabla de páginas asociado a la página a la que hemos accedido
- Se elimina la página que tiene el valor del contador más bajo

Ejemplo: Supongamos que poseemos la siguiente secuencia de solicitudes 7 0 1 2 0 3 0 4 2 3 0 3 2 y que contamos con 4 espacios de memoria. Contemos los page faults y apliquemos algoritmo:

- 4 page faults de inicio (7 0 1 2)
- 0 page faults del 0
- 1 page fault del 3 -> reemplaza al 7
- 0 page faults del 0
- 1 page fault del 4 -> reemplaza al 1
- 0 page faults hasta el final

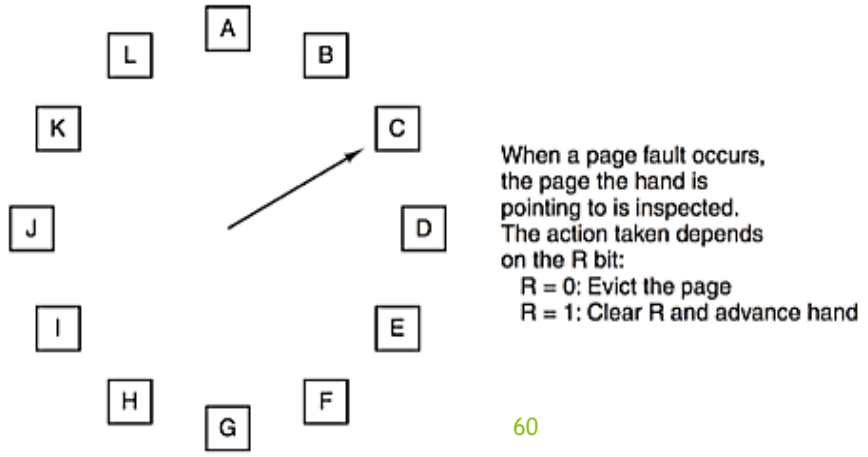
Total: 6 page faults

Algoritmo del Reloj

Considerando que el problema con LRU es que es muy costoso de implementar (ya sea por la estructura de datos que hay que mantener en memoria o bien por las comparaciones para elegir quién se reemplaza). Se aproxima utilizando el algoritmo del reloj: Utiliza un reference bit (que vale 1 si la página ha sido usada).

¿Cómo funciona?

- El algoritmo elige alguna página arbitrariamente:
- Si el reference bit = 1, se cambia a 0 y se pasa a la siguiente página (según el reloj).
- Si el reference bit = 0, se elige esa página.



Sistema de Archivos

Archivo

Es un conjunto de bits que son almacenados en un dispositivo. Un archivo es identificado por un nombre y la descripción de la carpeta o directorio que lo contiene. Los archivos, a vista de los usuarios, se organizan a través de “Rutas”. Sin perjuicio de lo anterior, tenemos que:

- La organización de los archivos es en directorios jerárquicos con “forma de árbol” y típicamente separados por un “/”.
- Los directorios (al igual que los archivos) son netamente una abstracción.
- Los directorios sirven para describir rutas, de las cuales hay dos tipos: Relativa y Absoluta (única)

```
D:\Data\Final\Infrastructure.gdb\Streets

..\..\Data\Final\Infrastructure.gdb\Streets
```

RAID

Contexto

- Supongamos que, en promedio, un SSD/HDD puede fallar cada 100000 horas (11 años aprox.)
- Luego, si consideramos un grupo de 100 discos, en promedio, se podría decir que habrá una falla cada 1000 horas (41.6 días).

¿Se podría considerar "grave" que ocurra una falla en este espacio de tiempo?

¿Qué medida precautoria podríamos utilizar?

Generalidades sobre los RAID

Se tratan de esquemas de almacenamiento que buscan mantener persistencia y seguridad sobre los datos guardados en HDD/SSD.

- La idea general es de tener conjuntos de SSD/HDD que trabajan coordinadamente.
- Si los datos se mantienen dentro del RAID, entonces se podrían recuperar los datos después de fallas físicas.
- Sin una estrategia, no hay RAID: JBOD - Just a Bunch Of Disks (no es RAID)

Es la estrategia básica de RAID y consiste en un “sistema espejo”. Es decir, se refiere a tener copias exactas de los discos, de modo que, si uno falla, es posible reconstruir los datos a partir de otra copia.

RAID 0

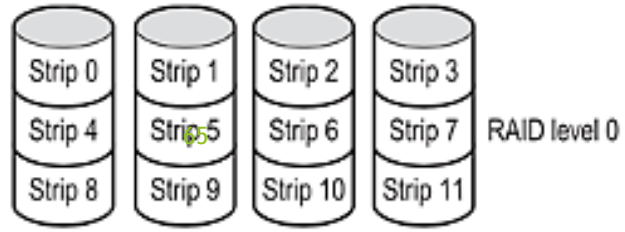
Es el “NO RAID” (ya que no es redundante) que distribuye los datos entre dos o más discos (usualmente se ocupa el mismo espacio en dos o más discos) sin información de paridad que proporcione redundancia. Es decir, se tratan muchos discos como si fuera “un solo gran disco”.

Ventajas

- Fácil de implementar
- Es muy conveniente para escritura (los datos se escriben en dos o más discos de forma paralela, pero el archivo solo está una vez en el conjunto)

Desventajas

- Si es creado con discos de tamaño diferente, el espacio total lo limitará el disco más pequeño
- No es tolerante a fallos



RAID 1 – Mirroring

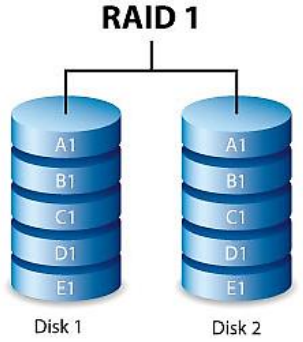
El RAID 1 consiste en crear una copia exacta (o espejo) de un conjunto de datos en dos o más discos.

Ventajas

- Fácil de implementar
- Hay tolerancia a fallas de 1 a N/2 (depende de cuáles discos fallen)

Desventajas

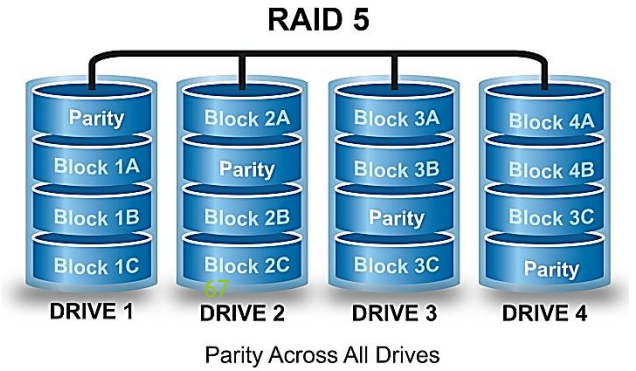
- Si es creado con discos de tamaño diferente, el espacio total lo limitará el disco más pequeño
- El rendimiento de lectura sube linealmente (en virtud de la cantidad de copias)
- Se desaprovecha la Capacidad de los Discos (C = N/2 x B)



RAID 5 – Striping with Parity across Drives

En esta estrategia que requiere un mínimo de 3 discos y consiste en almacenar la paridad de ciertos bloques del arreglo y además se distribuye el almacenamiento de la paridad. Es decir:

- Cada vez que un bloque de datos se escribe en un RAID 5, se genera un bloque de paridad dentro de la misma división (stripe).
- Un bloque se compone de varios sectores consecutivos de disco.
- Una serie de bloques (un bloque de cada uno de los discos del conjunto) se denomina stripe.
- Si otro bloque, o alguna porción de un bloque, es escrita en esa misma división, el bloque de paridad (o una parte del mismo) es recalculada y se vuelve a escribir.
- El disco utilizado por el bloque de paridad está escalonado de una división a la siguiente (por eso son bloques de paridad distribuidos).



Parity Across All Drives

RAID 5 – Striping with Parity across Drives

Ventajas

- *La lectura es muy veloz rápida*
- *Hay tolerancia a fallas de 1 disco sin interrupción de servicio (el sistema sigue funcionando, aunque el disco esté siendo reemplazado).*

Desventajas

- *La escritura es más lenta que la lectura ya que debemos calcular la paridad.*

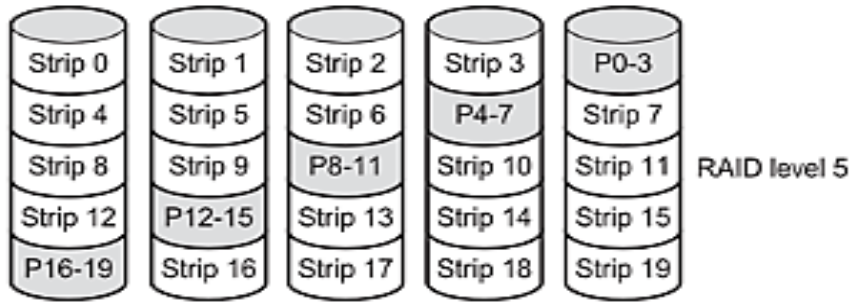


Tabla comparativa de los RAID

	RAID-0	RAID-1	RAID-5
Capacity	$N \cdot B$	$(N \cdot B)/2$	$(N-1) \cdot B$
Reliability	0	1 (for sure) $\frac{N}{2}$ (if lucky)	1
Throughput			
Sequential Read	$N \cdot S$	$(N/2) \cdot S$	$(N-1) \cdot S$
Sequential Write	$N \cdot S$	$(N/2) \cdot S$	$(N-1) \cdot S$
Random Read	$N \cdot R$	$N \cdot R$	$N \cdot R$
Random Write	$N \cdot R$	$(N/2) \cdot R$	$\frac{N}{4} R$
Latency			
Read	T	T	T
Write	T	T	$2T$

Métodos de Asignación de Bloques

Definición

Los métodos de asignación de bloques determinan cómo los bloques de datos son asignados a los archivos en un sistema de archivos. Los métodos más comunes son:

- ***Asignación Contigua***

En esta asignación los bloques de un archivo se almacenan de manera secuencial en ubicaciones contiguas en el disco.

```
Archivo A: Bloques 0, 1, 2, 3
Archivo B: Bloques 4, 5, 6, 7
```

Ventajas

- ✓ Acceso Rápido: El acceso secuencial a los datos es muy rápido porque los bloques están físicamente juntos.
- ✓ Simplicidad: Es simple de implementar y gestionar.

Desventajas:

- ✓ Fragmentación Externa: Con el tiempo, el espacio libre puede fragmentarse, dificultando la asignación de grandes archivos contiguos.
- ✓ Dificultad para Crecimiento de Archivos: Si un archivo necesita crecer, puede no haber espacio contiguo disponible.

Métodos de Asignación de Bloques

- ***Asignación Enlazada***

En esta asignación los bloques de un archivo se almacenan en cualquier lugar del disco, y cada bloque contiene un puntero al siguiente bloque.

```
Archivo A: Bloque 0 -> Bloque 4 -> Bloque 8 -> Bloque 12
Archivo B: Bloque 1 -> Bloque 5 -> Bloque 9 -> Bloque 13
```

Ventajas

- ✓ No hay Fragmentación Externa: Los bloques pueden estar dispersos, eliminando la necesidad de grandes bloques contiguos de espacio libre.
- ✓ Facilidad para el Crecimiento de Archivos: Es fácil añadir más bloques a un archivo.

Desventajas

- ✓ Acceso Lento: El acceso a los datos puede ser más lento debido a la necesidad de seguir los punteros.
- ✓ Fiabilidad: Si un bloque se corrompe, se pierde la cadena de bloques.

Métodos de Asignación de Bloques

- ***Ejemplo de Asignación Enlazada: FAT***

Tabla de Asignación de Archivos (FAT) una estructura de datos que se usa en sistemas de archivos de la familia DOS/Windows. Hay varias versiones, como FAT12, FAT16 y FAT32, que se refieren al tamaño de las entradas de la tabla FAT (12, 16, o 32 bits).

Asignación de Bloques

- ✓ Cada archivo se divide en bloques llamados clústeres.
- ✓ Cada entrada en la tabla FAT corresponde a un clúster en el disco y contiene un puntero al siguiente clúster del archivo.

Tabla de Asignación de Archivos

- ✓ La tabla FAT está almacenada en una sección reservada del disco.
- ✓ FAT actúa como una lista enlazada que rastrea la cadena de clústeres utilizados por un archivo.

Métodos de Asignación de Bloques

Ejemplo de FAT

Supongamos que tenemos un archivo example.txt que ocupa 4 clústeres y el sistema de archivos es FAT16 considerando:

Estructura del Archivo

- Clústeres utilizados: 2, 5, 7, 9

Tabla FAT

- Entrada del clúster 2: 5 (apunta al siguiente clúster)
- Entrada del clúster 5: 7
- Entrada del clúster 7: 9
- Entrada del clúster 9: EOF (End Of File, marcador de fin de archivo)

Directorio

- example.txt -> Primer clúster: 2

Clusters de Datos:

Cluster	Data
2	data of part 1
5	data of part 2
7	data of part 3
9	data of part 4

Tabla FAT:

Cluster	Next
2	5
3	0
4	0
5	7
6	0
7	9
8	0
9	EOF
...	...

Directorio:

File Name	Start Clus
example.txt	2

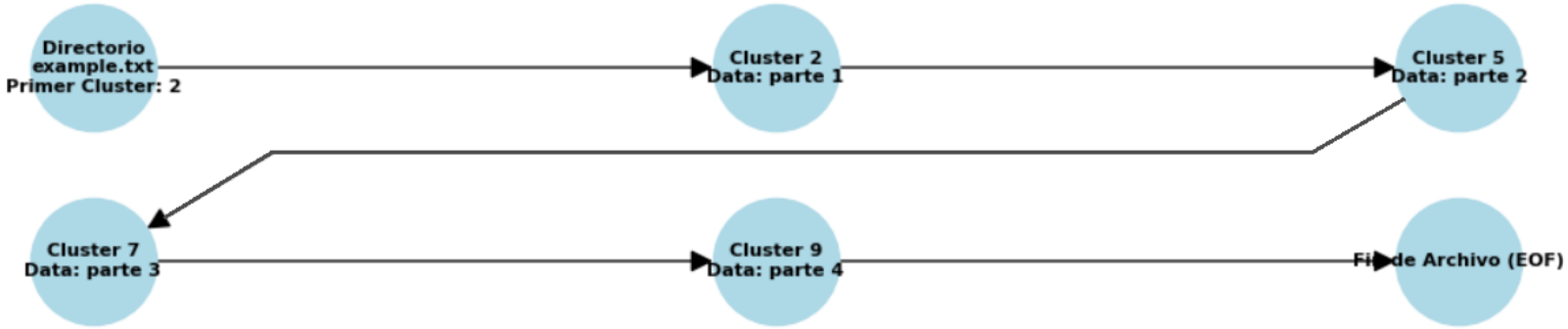
Métodos de Asignación de Bloques

Para acceder al archivo, el sistema considera lo siguiente:

- Ubicación Inicial: El directorio apunta al primer clúster del archivo (clúster 2).
- Seguimiento de Clústeres: Se lee la entrada en la tabla FAT correspondiente al primer clúster (2), que indica el siguiente clúster (5) y se continúa siguiendo los punteros en la tabla FAT hasta llegar al marcador de fin de archivo (EOF).
- Acceso a Datos: Se accede a los datos en los clústeres 2, 5, 7 y 9 en secuencia.

Entonces:

- Directorio: Contiene la entrada del archivo example.txt, indicando que el primer clúster del archivo es el clúster 2.
- Clúster 2: Contiene la primera parte de los datos del archivo y un puntero al siguiente clúster (clúster 5).
- Clúster 5: Contiene la segunda parte de los datos del archivo y un puntero al siguiente clúster (clúster 7).
- Clúster 7: Contiene la tercera parte de los datos del archivo y un puntero al siguiente clúster (clúster 9).
- Clúster 9: Contiene la cuarta parte de los datos del archivo y un puntero que marca el fin del archivo (EOF).



Métodos de Asignación de Bloques

- ***Asignación Indexada***

En esta asignación se utiliza un bloque de índice que contiene punteros a todos los bloques de datos del archivo.

```
Bloque de indice A: [Bloque 0, Bloque 4, Bloque 8, Bloque 12]
Bloque de indice B: [Bloque 1, Bloque 5, Bloque 9, Bloque 13]
```

Ventajas

- ✓ Acceso Directo: Permite acceso directo a cualquier bloque del archivo.
- ✓ Escalabilidad: Facilita el manejo de archivos grandes.

Desventajas

- ✓ Overhead del Índice: El bloque de índice añade overhead (recursos adicionales como tiempo, espacio, y complejidad).
- ✓ Complejidad: Es más complejo de implementar y gestionar.

HAY 3 TIPOS DE ASIGNACIÓN INDEXADA

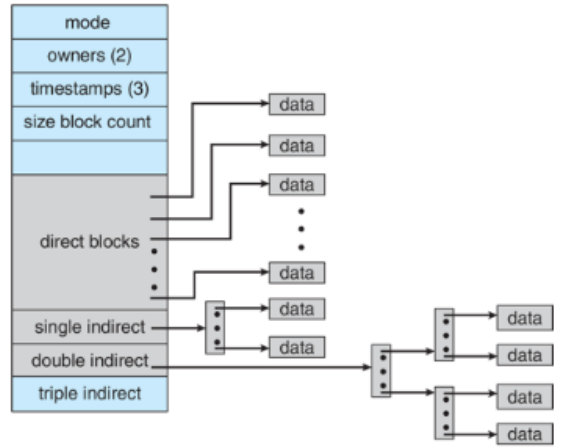
Métodos de Asignación de Bloques

Asignación Indexada: Esquema enlazado

- En este tipo de asignación indexada, cada bloque de índice (index block) contiene una serie de punteros, y el último puntero de cada bloque de índice apunta a otro bloque de índice.
- Este esquema se utiliza para manejar sistemas de archivos de gran tamaño donde la cantidad total de bloques de datos puede ser desconocida o muy grande.

Asignación Indexada: Multi-level

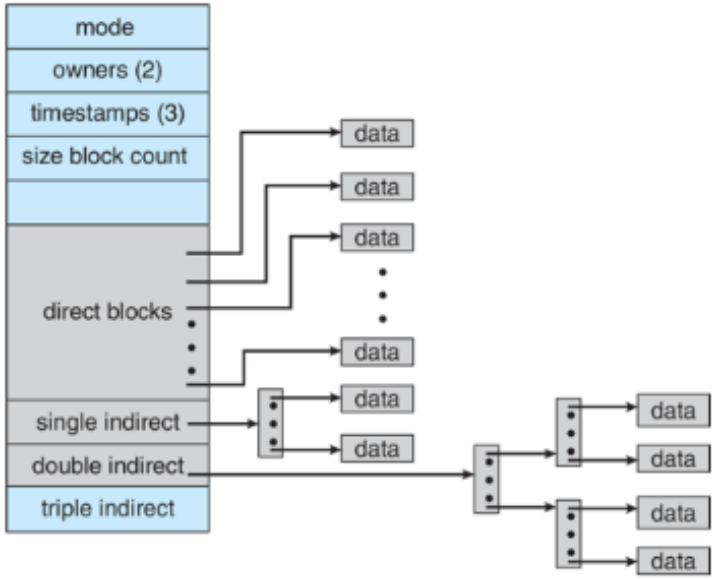
- Este tipo de asignación indexada funciona de manera similar a las tablas de páginas multinivel (Multi-level Page Tables) utilizadas en sistemas operativos para la gestión de memoria virtual.
- Se organiza en múltiples niveles de indirección, donde cada nivel tiene un conjunto de punteros que apuntan a los siguientes niveles de la jerarquía, hasta llegar a los bloques de datos finales.
- Este enfoque es eficiente para sistemas de archivos de gran tamaño y distribuidos



Métodos de Asignación de Bloques

Asignación Indexada: Esquema combinado

- El esquema de asignación indexada combinado combina características de los esquemas enlazados y multinivel.
- Utiliza una estructura enlazada para los bloques de índice superiores y luego pasa a una estructura multinivel para acceder a los bloques de datos finales.
- Este enfoque puede proporcionar una combinación de eficiencia y flexibilidad en la gestión de sistemas de archivos de gran tamaño.



Métodos de Asignación de Bloques

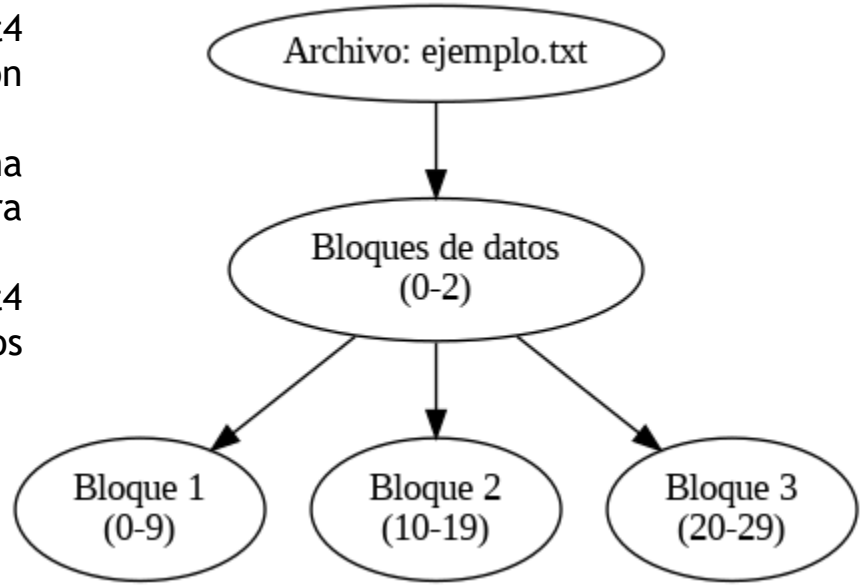
EXT3 y EXT4

Formalmente NO es asignación indexada, pero ext3 y ext4 utilizan un árbol de extensión (B-tree) para indexar los bloques de datos en el disco.

Ejemplo

Supongamos que creamos un archivo llamado "ejemplo.txt" en el sistema de archivos ext4 y escribimos datos en él. El sistema de archivos ext4 utilizará el árbol de extensión para asignar bloques de disco al archivo "ejemplo.txt".

1. Cuando se crea el archivo "ejemplo.txt", el sistema de archivos ext4 crea un nodo raíz en el árbol de extensión para mantener la información sobre el archivo.
2. A medida que se escriben datos en el archivo "ejemplo.txt", el sistema de archivos ext4 agrega nuevos nodos al árbol de extensión para mantener un registro de los bloques de datos utilizados por el archivo.
3. Cuando se lee el archivo "ejemplo.txt", el sistema de archivos ext4 utiliza el árbol de extensión para encontrar los bloques de datos necesarios para leer los datos del archivo.

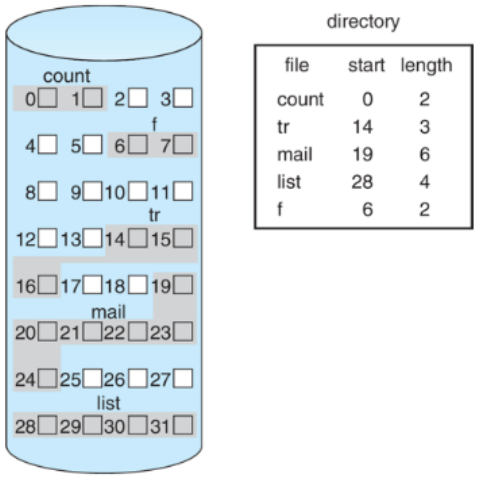


Métodos de Asignación de Bloques

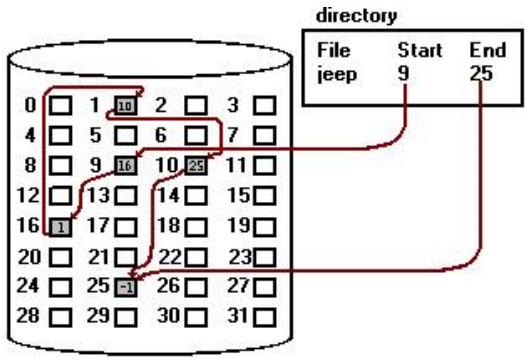
Comparación de los métodos

Método	Ventajas	Desventajas
Contigua	Acceso rápido, simplicidad	Fragmentación externa, dificultad para crecer
Enlazada	Sin fragmentación externa, crecimiento fácil	Acceso lento, fiabilidad
Indexada	Acceso directo, escalabilidad	Overhead del índice, complejidad

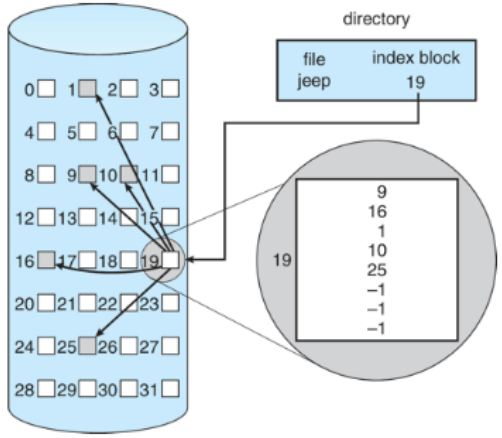
Asignación contigua



Asignación enlazada



Asignación indexada



Métodos de Asignación de Bloques

Ejemplo

Supongamos que tenemos tres archivos: A, B y C, y queremos almacenar sus datos en un disco utilizando los tres métodos de asignación considerando que poseen 4, 3 y 5 bloques respectivamente.

```
#Asignacion Contigua
Archivo A: Bloques 0, 1, 2, 3
Archivo B: Bloques 4, 5, 6
Archivo C: Bloques 7, 8, 9, 10, 11
```

```
#Asignacion enlazada
Archivo A: Bloque 0 -> Bloque 2 -> Bloque 4 -> Bloque 6
Archivo B: Bloque 1 -> Bloque 3 -> Bloque 5
Archivo C: Bloque 7 -> Bloque 8 -> Bloque 9 -> Bloque 10 -> Bloque 11
```

```
#Asignacion indexada
indice de Archivo A = [ Bloque 0, Bloque 2, Bloque 4, Bloque 6 ]
indice de Archivo B = [ Bloque 1, Bloque 3, Bloque 5 ]
indice de Archivo C = [ Bloque 7, Bloque 8, Bloque 9, Bloque 10, Bloque 11 ]
```

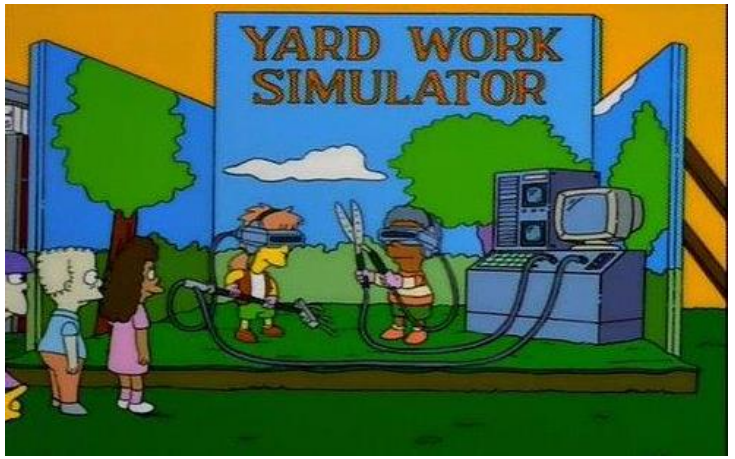
Problema de la Virtualización: Modo Dual

Recordando

- La virtualización permite que múltiples sistemas operativos (OS) y aplicaciones se ejecuten en el mismo hardware físico de manera aislada y eficiente.
- Para lograr esto, las tecnologías de virtualización deben gestionar cuidadosamente los recursos del sistema, como la CPU, la memoria y el almacenamiento.

En esta clase, veremos cómo las tecnologías de virtualización interactúan con tres áreas clave gestionadas por el sistema operativo:

1. Scheduling (Planificación de Tareas)
2. Gestión de Memoria (RAM)
3. File Systems (Sistemas de Archivos)



Scheduling (Planificación de Tareas)

Recordando

La planificación de tareas en un sistema operativo implica decidir qué procesos o hilos de ejecución reciben tiempo de CPU y cuándo.

Planificación en entorno virtualizado

En un entorno virtualizado, el hipervisor o el gestor de virtualización debe manejar la planificación no solo para los procesos dentro de cada máquina virtual (VM), sino también entre las propias VMs.

Mecanismos

1. Planificación del Hipervisor: El hipervisor asigna tiempo de CPU a cada VM según ciertas políticas (por ejemplo, round-robin, prioridad, etc.). Esto es similar a cómo un OS planifica los procesos, pero a un nivel más alto.
2. Planificación en las VMs: Dentro de cada VM, el sistema operativo invitado realiza su propia planificación de tareas entre los procesos y los hilos, independientemente de las decisiones del hipervisor.

Impacto

1. Contención de Recursos: Puede ocurrir contención cuando varias VMs requieren tiempo de CPU simultáneamente. El hipervisor debe gestionar esta contención de manera efectiva para mantener el rendimiento.
2. Overhead: La planificación adicional realizada por el hipervisor introduce overhead (tiempo de ejecución), pero es necesario para garantizar el aislamiento y la gestión equitativa de los recursos.

Scheduling (Planificación de Tareas)

Ejemplo

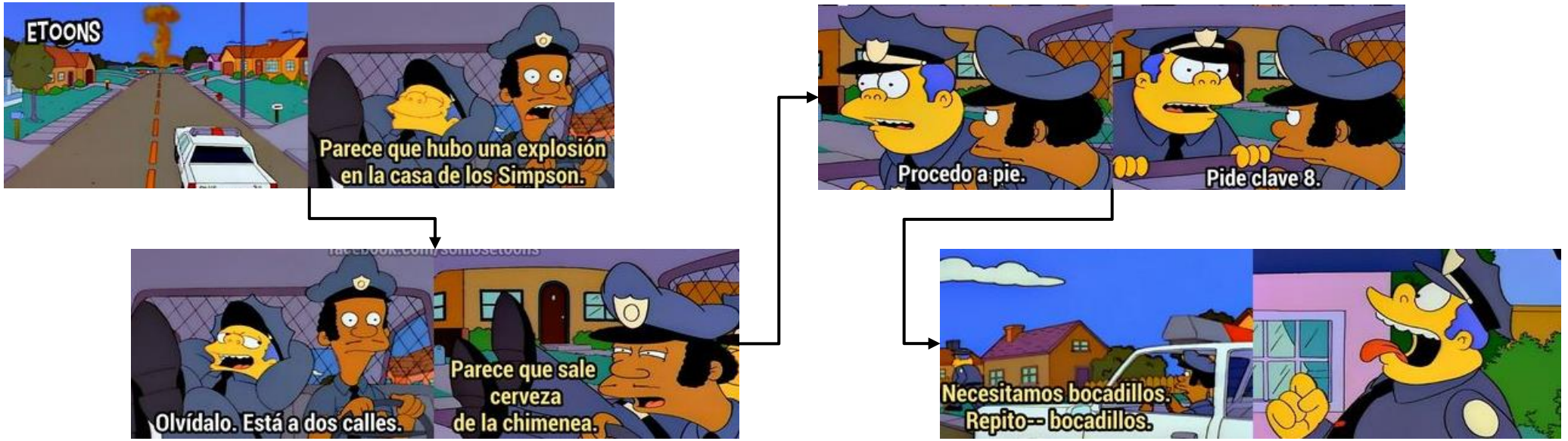
El Departamento de Policía de Springfield ha implementado la virtualización para mejorar la eficiencia en la gestión de sus recursos informáticos. El Jefe Gorgory ha asignado a cada uno de sus oficiales una máquina virtual para llevar a cabo sus tareas diarias de manera más eficiente.

1. Oficial Clancy Gorgory: Es el jefe del departamento y utiliza su máquina virtual para supervisar las operaciones policiales en toda la ciudad de Springfield. Su tarea principal es coordinar las actividades de los demás oficiales y responder a situaciones de emergencia.
2. Oficial Eddie: Eddie es uno de los oficiales de patrulla en Springfield. En su máquina virtual, utiliza herramientas de análisis de datos para identificar patrones delictivos, hacer seguimiento de criminales buscados y coordinar la respuesta a llamadas de emergencia.
3. Oficial Lou: Lou es otro oficial de patrulla en el departamento de policía. En su máquina virtual, gestiona la base de datos de evidencia, registra informes policiales y colabora con otros departamentos en investigaciones en curso.

Scheduling (Planificación de Tareas)

Funcionamiento de la Virtualización y Scheduling

1. El hipervisor, administrado por el Jefe, asigna recursos de CPU a cada máquina virtual según la importancia de la tarea y las necesidades del momento.
2. El Jefe Gorgory puede requerir una asignación más alta de CPU durante situaciones de emergencia o eventos importantes, mientras que Eddie y Lou pueden necesitar más recursos durante su patrulla diaria o durante investigaciones activas.
3. El hipervisor ajusta dinámicamente la asignación de recursos para garantizar que cada oficial tenga los recursos necesarios para llevar a cabo sus tareas de manera efectiva.



Gestión de Memoria (RAM)

Recordando

La gestión de memoria en un sistema operativo implica la asignación y liberación de memoria física (RAM) para los procesos, asegurando que cada uno tenga el espacio necesario para funcionar correctamente.

Virtualización y Gestión de Memoria

La gestión de memoria en un entorno virtualizado es más compleja, ya que el hipervisor debe gestionar la memoria física entre múltiples VMs y sus propios sistemas operativos invitados.

Mecanismos

1. Asignación de Memoria: El hipervisor asigna un bloque de memoria física a cada VM. Cada VM gestiona esta memoria como si fuera su propia memoria física.
2. Memoria Compartida y Swap: Técnicas como la deduplicación de páginas y el intercambio (swapping) permiten que el hipervisor optimice el uso de la memoria física. Por ejemplo, páginas de memoria idénticas entre diferentes VMs pueden ser compartidas para ahorrar espacio.
3. Ballooning: El hipervisor puede ajustar dinámicamente la cantidad de memoria asignada a cada VM utilizando una técnica llamada ballooning. Un controlador de "balloon" dentro de la VM puede "inflar" o "desinflar" para liberar o reclamar memoria según las necesidades.

Impacto

1. Aislamiento y Seguridad: La gestión de memoria debe asegurar que las VMs no puedan acceder a la memoria de otras VMs, manteniendo un alto nivel de aislamiento y seguridad.
2. Rendimiento: La sobrecarga adicional de la gestión de memoria puede afectar el rendimiento, pero es necesaria para la eficiencia y seguridad general del sistema.

Gestión de Memoria (RAM)

Ejemplo

El Canal 6 de Springfield ha decidido mejorar la eficiencia en la gestión de sus recursos por lo que cada una de las personalidades televisivas tiene asignada una máquina virtual para sus necesidades específicas.

1. Kent Brockman: Kent es el presentador principal del noticiero en el Canal 6. Utiliza su máquina virtual para preparar, editar y presentar las noticias. Sus aplicaciones de edición de video, investigación y presentación requieren una cantidad significativa de memoria para funcionar sin problemas.
2. Krusty el Payaso: Krusty es el famoso presentador de un programa de televisión infantil. Utiliza su máquina virtual para grabar y editar sketches, entrevistas y segmentos musicales para su programa. Sus aplicaciones de edición de video y producción de contenidos también son intensivas en memoria.
3. El Hombre Abejorro: El Hombre Abejorro es un conocido comediante latino que protagoniza su propio programa de comedia en el Canal 6. Utiliza su máquina virtual para grabar, editar y producir sketches cómicos, entrevistas y segmentos de entretenimiento para su programa. Sus aplicaciones de edición de video y producción de contenido requieren una cantidad considerable de memoria.

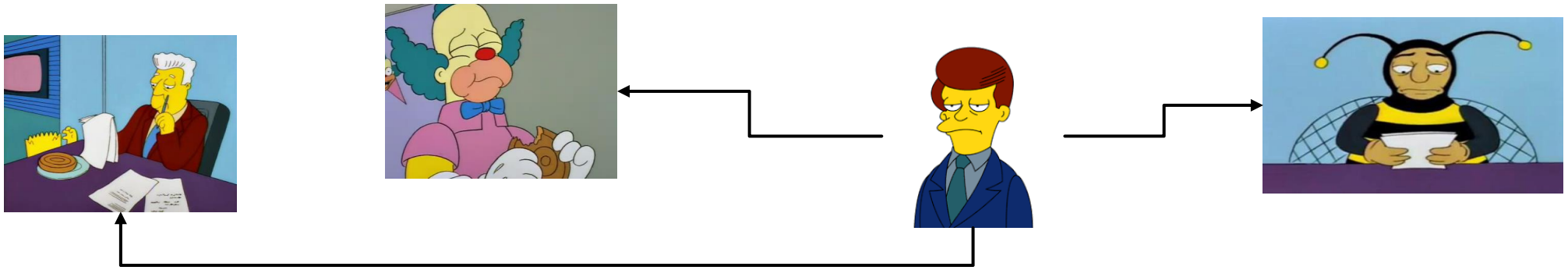
Gestión de Memoria (RAM)

Funcionamiento de la Virtualización y Gestión de Memoria

1. El hipervisor, administrado por un Director del Canal 6, asigna recursos de memoria a cada máquina virtual según las necesidades de cada personalidad y la disponibilidad de memoria en el sistema.
2. Kent, Krusty y el Hombre Abejorro reciben asignaciones de memoria ajustadas a sus respectivas necesidades. Debido a que trabajan con aplicaciones de edición de video y producción de contenido intensivas en memoria, cada uno recibe una asignación adecuada para garantizar un rendimiento óptimo.
3. El hipervisor garantiza que cada personalidad tenga suficiente memoria para llevar a cabo sus tareas de manera eficiente y sin problemas de rendimiento.

Situacionalmente

Mientras Kent está ocupado preparando el resumen de las noticias en su máquina virtual, Krusty está buscando un pan danés y el Hombre Abejorro está grabando un segmento de comedia para su próxima emisión. Cada personalidad puede utilizar su máquina virtual de manera independiente sin interferir con las actividades de los demás.



File Systems (Sistemas de Archivos)

Recordando

Un sistema de archivos gestiona cómo se almacenan y recuperan los datos en un disco. Proporciona una estructura organizada para los datos y mecanismos para acceder a ellos de manera eficiente y segura.

Virtualización y Sistemas de Archivos

En un entorno virtualizado, cada VM tiene su propio sistema de archivos, que puede ser gestionado de manera diferente por el hipervisor.

Mecanismos

1. Sistemas de Archivos Virtuales: Las VMs suelen utilizar discos virtuales (archivos de imagen de disco) que son gestionados por el hipervisor. Estos discos virtuales pueden ser archivos en el sistema de archivos del anfitrión o volúmenes gestionados por soluciones de almacenamiento.
2. Snapshots y Cloning: Los hipervisores pueden crear snapshots (copias del estado de una VM en un momento dado) y clonar VMs rápidamente utilizando técnicas de sistemas de archivos, lo cual es útil para pruebas y desarrollo.
3. Redirección de E/S: Las operaciones de entrada/salida de las VMs se redirigen a través del hipervisor, que puede optimizarlas y gestionarlas para mejorar el rendimiento y la eficiencia.

Impacto

1. Rendimiento de E/S: La redirección de operaciones de E/S a través del hipervisor puede introducir latencia y overhead, afectando el rendimiento de las aplicaciones que requieren acceso intensivo al disco.
2. Gestión de Almacenamiento: El hipervisor debe gestionar eficientemente el almacenamiento para evitar problemas como el desbordamiento del disco y asegurar que las VMs tengan suficiente espacio para operar.

File Systems (Sistemas de Archivos)

Ejemplo

La Taberna de Moe en Springfield está implementando la virtualización para mejorar la eficiencia en la gestión de sus recursos por lo que cada área de la taberna tiene asignada una máquina virtual para sus necesidades específicas.

1. Moe Szyslak: Moe es el propietario de la taberna y utiliza su máquina virtual para gestionar todas las operaciones del negocio, incluida la contabilidad, la gestión de inventario y las reservas de clientes. Sus aplicaciones requieren acceso rápido y eficiente a los sistemas de archivos para mantener registros precisos y gestionar las transacciones diarias.
2. Barney Gómez: Barney es un cliente habitual de la taberna y se ha convertido en el encargado informal de la música y el entretenimiento en vivo. Utiliza su máquina virtual para almacenar y organizar su extensa colección de música, así como para crear listas de reproducción para las actuaciones en vivo en la taberna.
3. Red Hot Chili Peppers: La banda de rock, Red Hot Chili Peppers, ocasionalmente se presenta en la taberna de Moe para conciertos exclusivos. Utilizan una máquina virtual para almacenar sus archivos de música, partituras y para ensayar antes de sus actuaciones en vivo en la taberna.

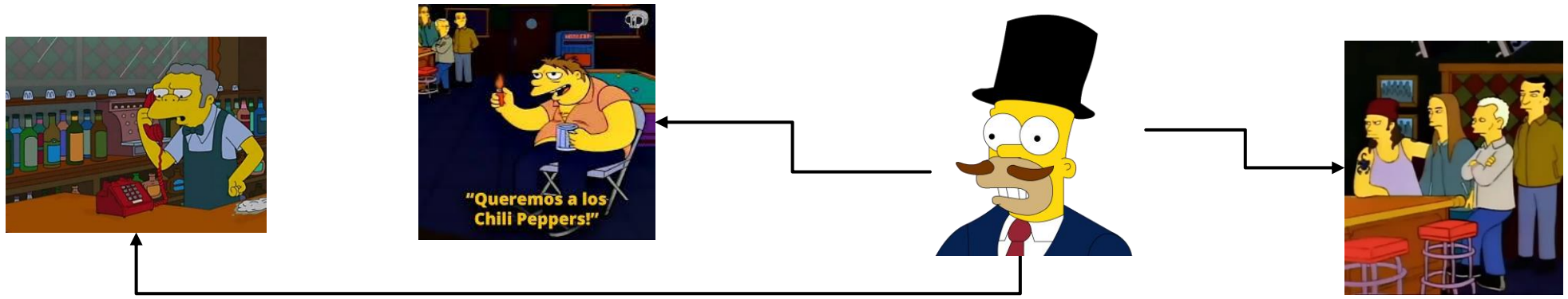
File Systems (Sistemas de Archivos)

Funcionamiento de la Virtualización y Sistemas de Archivos

1. El hipervisor, administrado por “El Cosme Fulanito”, asigna recursos de almacenamiento a cada máquina virtual según las necesidades de cada área y la disponibilidad en el sistema.
2. Moe, Barney y Red Hot Chili Peppers reciben asignaciones de almacenamiento ajustadas a sus respectivas necesidades. Moe necesita un espacio considerable para almacenar registros comerciales, mientras que Barney y la banda necesitan menos espacio, pero con un acceso rápido a los archivos de música y partituras.
3. El hipervisor garantiza que cada área tenga suficiente espacio de almacenamiento y un sistema de archivos eficiente para realizar sus tareas de manera efectiva y sin problemas.

Situacionalmente

Mientras Moe está tras la barra, Barney está esperando sentado a la banda en vivo y Red Hot Chili Peppers están escapando del lugar. Luego, cada área de la taberna puede utilizar su máquina virtual de manera independiente sin interferir con las actividades de las demás.



Cierre de la sesión