



Documento anónimo

SISTEMAS OPERATIVOS GUIA DE EJERCICIOS 2005. Resueltos.pdf

Ejercicios SO Resueltos



2º Sistemas Operativos



Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de
Telecomunicación
Universidad de Granada



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.



DESPUÉS DE LOS FINALES... **¡RELÁJATE!**

- **480** cartas resultado de mucho amor y birra
- **80 cartas especiales** para animar tus fiestas
- **A partir de 16 años**, de 3 a 15 jugadores
- Perfecto para largas noches de risas con amigos



GUA
TA
FAC

GUA
TA
FAC

GUA
TA
FAC

SISTEMAS OPERATIVOS

GUIA DE EJERCICIOS 2005

Resueltos

WUOLAH

Después de los finales, ¡RELÁJATE CON GUATAFAC!

Ejercicio 2

Enunciado

Un sistema operativo utiliza un algoritmo de planificación de CPU basado en una cola multinivel de dos niveles. El primer nivel se gestiona mediante un algoritmo de prioridades puras y en él se introducen las rutinas de servicio del sistema operativo. El segundo nivel se gestiona mediante un algoritmo Round Robin de quantum 3 unidades de tiempo, y en él se introducen los procesos de usuario. Al terminar cada operación de entrada salida se produce una interrupción que utiliza la CPU durante 2 unidades de tiempo, y este es el único tipo de interrupciones que se deben considerar en el problema. En un instante dado, la carga del sistema es la siguiente:

<i>Proceso</i>	<i>Prioridad</i>	<i>Secuencia de Operaciones</i>
<i>P1</i>	3	CPU(4) - E/S(2) - CPU(4) - E/S(2) - CPU(4)
<i>P2</i>	1	CPU(2) - E/S(3) - CPU(3)
<i>P3</i>	3	CPU(7)
<i>P4</i>	1	CPU(3) - E/S(1) - CPU(3) - E/S(2) - CPU(3)
<i>P5</i>	2	CPU(2) - E/S(2) - CPU(2)

Donde CPU(n) indica la ejecución de n unidades de tiempo de CPU y E/S(n) indica uso de la E/S durante n unidades de tiempo. Un mismo proceso puede tener varias ráfagas de CPU y/o E/S. A menor número le corresponde mayor prioridad (1 es más que 3). En este sistema se pide:

- Realizar un seguimiento de la ejecución de todos los procesos, indicando la ocupación de CPU y E/S en cada instante
- Ventajas y/o desventajas que aporta en este sistema el uso de Round Robin virtual frente al Round Robin clásico.

Marco Teórico del ejercicio:

La idea del ejercicio es permitir ver cómo se ejecutan los procesos enumerados con las características que se destacan en la tabla. Recordemos un poco de qué se trata la planificación de procesos: Existen cuatro tipos de planificaciones:

- Una planificación a extra largo plazo que consiste en planificaciones externas que se hacen en el centro de cómputo y están determinadas por las políticas del mismo
- Una planificación a largo plazo la que se encarga de determinar y organizar los nuevos programas que van a ser admitidos al sistema
- Una a mediano plazo encargada del intercambio de los procesos, entre la memoria principal y la virtual
- Una planificación a corto plazo o “dispatcher”, que determina cuál es el próximo proceso que va a tomar la CPU

De estas cuatro categorías podemos decir que las tres últimas se tratan de planificaciones internas al procesador, pero si hablamos de planificación interna tenemos que tener en cuenta también otra categoría, que sería la planificación de Entrada / Salida.

En lo que hace al desarrollo del ejercicio nos interesa netamente la planificación a corto plazo, o sea, aquella de la que se encarga el dispatcher. El planificador a corto plazo se ejecuta cuando ocurre un suceso que puede conducir a la interrupción del proceso actual o que ofrece la oportunidad de expulsar al proceso actual a favor de otro respondiendo a políticas de uso del Procesador. La planificación de CPU consiste en decidir a cuál de los procesos situados en la cola de preparados para la ejecución se le debe asignar CPU (tener en cuenta el modelo de 5 estados como ejemplo graficador. Remitirse al NOTAS – Módulo 2 – Fig 2.11).

En nuestro ejercicio en particular, tenemos que distinguir entre la secuencia de operaciones indicada en la tabla, aquellas que mencionan CPU de aquellas que mencionan E/S, se trata de dos cosas distintas. Durante la ejecución, un proceso alterna entre períodos de uso de CPU llamados CPU burst y períodos de uso de E/S o I/O burst.

Para la planificación de las diferentes operaciones del proceso en el corto plazo, se cuenta con una siere de algoritmos que enunciamos a continuación: FCFS – Round Robin (en su dos variantes, tradicional y virtual) – SPN – SRT – Por Prioridades – Con Retroalimentación – Múltiples Colas Fijas / Dinámicas

De entre ellos los hay en versiones Preenptive (o no apropiativos del procesador) y Non Preemptive (propativos del procesador).

En nuestro caso nos iteresan tres tipos, ya que el ejercicio los menciona:

Cola Multinivel de dos niveles: Se particiona la cola de listos en dos niveles, cada una con su propio algoritmo de planificación. Cada proceso perteneces a una cola y no cambia.

Luego nos dice que cada cola tiene un algoritmo como corresponde, de la siguiente manera:

Prioridades: Cada proceso tiene una prioridad y se orden en la cola de mayor a menor prioridad y pasan a usar la CPU en ese orden. En el caso de ser apropiativo, el proceso que pasa a ser uso del procesador queda ahí hasta que se bloquee o finalice. En caso de ser no apropiativo, si llega un proceso con mayor prioridad, desaloja al que actualmente está haciendo uso del procesador, y este nuevo proceso toma la CPU.

Round Robin: Viene siempre acompañado por un quantum de tiempo que tienen los procesos para ejecutar; finalizado ese tiempo el proceso abandona la CPU. Debido a esta característica, el algoritmo es no apropiativo. Obviamente que el proceso puede abandonar antes la CPU en caso de que finalice o se bloquee por una E/S; caso contrario (que se le termina el tiempo), vuelve a la cola de listos.

Vista esta introducción teórica, pasemos de lleno a la resolución del ejercicio.

Resolución Práctica:

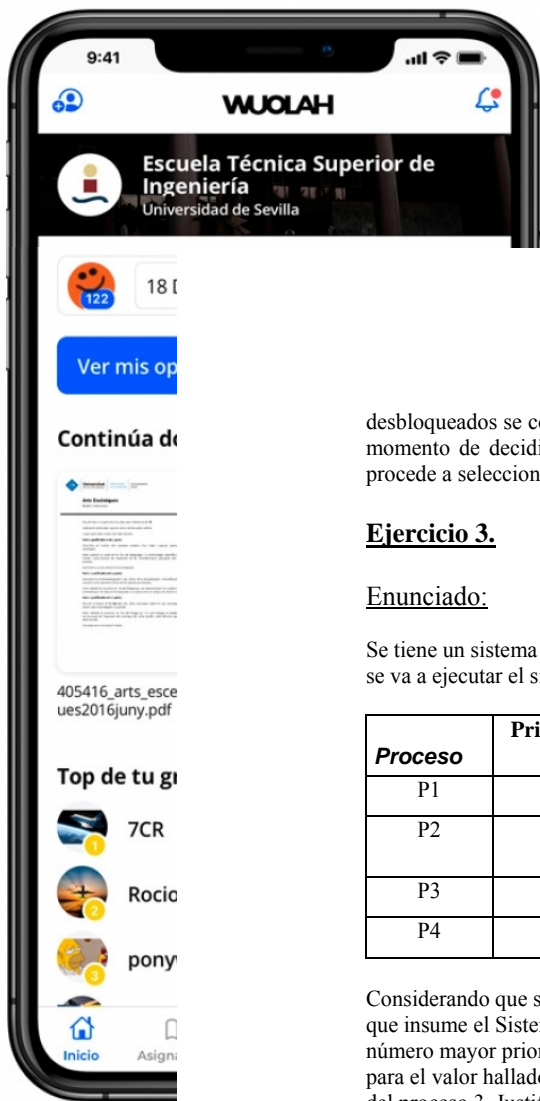
a)

Int.									Ex	Ex					Ex	Ex				Ex	Ex	Ex	Ex					
P5													Ex			Ex												
P4										Ex	Ex	Ex	I/O				I/O	I/O										
P3					Ex	Ex	Ex																			Ex		
P2				Ex	Ex	I/O	I/O	I/O									Ex							Ex	Ex			
P1	Ex	Ex	Ex														Ex	Q	I/O	I/O								
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25		
	1 q																					Fin P2						

Int.								Ex	Ex											Ex	Ex						
P5						Ex	Ex																				
P4			Ex	Ex	Ex	I/O	I/O							Ex	Ex	Ex											
P3	Ex	Ex											Ex														
P2																											
P1											Ex	Ex	Ex				Ex	I/O	I/O			Ex	Ex	Ex	Ex		
	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50		
							Fin P5						Fin P3			Fin P4									Fin P1		

Referencias: Ex = Ejecutando
I/O = Entrada / Salida
Q = Queued (por I/O)

b) Round Robin es efectivo en sistemas de tiempo compartido de propósitos generales pero presenta dificultades si existe una mezcla de procesos limitados por CPU y procesos limitados por E/S ya que los ultimos tienen CPU burst muy cortas por lo que usan el procesador muy poco tiempo y quedan bloqueados, luego se ejecuta un proceso limitado por CPU usando todo su quantum y volviendo a la cola de listos, mientras que el de E/S pierde su turno, produciendo baja performance. Este problema se soluciona implementando el RR Virtual, el cual funciona de manera similar aunque los procesos que son



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.



desbloqueados se colocan en una cola FIFO auxiliar, la cual tiene prioridad por sobre la cola principal; al momento de decidir que proceso pasa a ejecución se consulta a esta cola y si hay procesos en ella se procede a seleccionarlos.

Ejercicio 3.

Enunciado:

Se tiene un sistema el cual utiliza un Short Term Scheduler por prioridades puras. En un momento dado se va a ejecutar el siguiente set de procesos:

Proceso	Prioridad	T. Llegada	CPU	I/O	CPU	I/O	CPU
P1	40	0	2	IMP (4)	3	DISCO (4)	2
P2	30	0	3	PLOTTER (2)	X		
P3	20	1	5	PLOTTER (3)	6	DISCO (2)	2
P4	10	3	2	IMP (2)	1		

Considerando que se tienen tres dispositivos de entrada / salida, los cuales planifican FIFO, que el tiempo que insume el Sistema Operativo en realizar un process switch es de un ciclo de CPU y que a menor número mayor prioridad. Se pide hallar el valor de X y los tiempos de finalización de todos los procesos para el valor hallado, sabiendo que el Turnaround Time del proceso 2 es menor que el Turnaround Time del proceso 3. Justifique su respuesta.

Nota: Para la resolución de este ejercicio tenga en cuenta que no se considera el inicio del mismo como un process switch y que ante la simultaneidad de eventos deberá elegir por FIFO al proceso a ejecutar.

Teoría:

Long term Scheduler o Planificación a largo plazo: se encarga de determinar y organizar los nuevos programas que van a ser admitidos al sistema.

Middle term Scheduler o Planificación a mediano plazo: encargada del swapping (intercambio) de los procesos, entre la memoria principal y la virtual.

Short Term Scheduler o Planificación a corto plazo: consiste de un “dispatcher”, que determina cual es el próximo proceso que va a hacer uso de la CPU.

Prioridades Puras (o no apropiativo): Cada proceso tiene una prioridad y se ordena en la cola de mayor a menor prioridad y pasan a usar la CPU en ese orden. Si llega un proceso con mayor prioridad, desaloja al que actualmente está haciendo uso del procesador, y este nuevo proceso toma la CPU.

Dispositivo con planificación FIFO: los procesos usan el dispositivo de E / S en el orden que llegaron y por ser un algoritmo apropiativo, no lo abandonan hasta que finalice su uso.

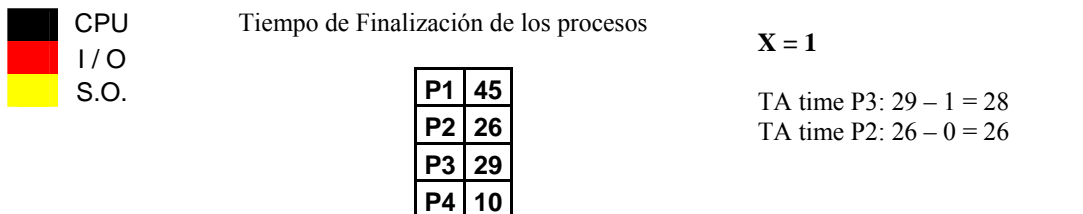
Process Switch o Context Switch: Conmutación del procesador entre un proceso y otro.

El Context Switch es Overhead puro (pérdida de tiempo) en:

1. Preservar el estado del proceso viejo (guardar en el stack su PCB).
2. Recuperar el estado del proceso nuevo (recuperar su PCB).
3. Bifurcar a la dirección donde había sido suspendido el nuevo proceso.

Turnaround Time o tiempo de servicio: Tiempo de ejecución desde que se somete el nuevo trabajo hasta que el proceso muere.

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.



Para calcular el valor de X pedido en el enunciado, lo que debemos tener en cuenta es que en el instante 24, el proceso P3 requiere una operación de E / S, por lo tanto, mientras la lleva a cabo, el S.O. realiza un context switch para otorgarle el uso del CPU a otro proceso, que en este caso es el proceso P2. Como la operación de E/S que efectúa P3 consume 2 ciclos de CPU y luego este mismo proceso requiere nuevamente el uso de CPU, la cantidad de ciclos de CPU que se le otorgará a P2 es de 1, ya que se consumió un ciclo en hacer el context switch y luego deberá conmutar nuevamente hacia P3 para que este utilice CPU, debido a que P3 posee más prioridad que P2. Por lo tanto, el valor de X pedido en el enunciado (que sería el tiempo que consume CPU el proceso P2 luego de realizar su operación de E / S) será igual a 1, ya que si luego se le volviera a otorgar CPU al proceso P2 luego de que finaliza su uso P3, el Turnaround time de P2 sería mayor al Turnaround time de P3, cosa que no es factible porque el enunciado dice justamente lo contrario.

Ejercicio 5.

Enunciado:

Sea un Sistema Operativo con Virtual Round Robin (VRR) como Short Term Scheduler y quantum de 4 unidades de tiempo. En dicho Sistema se ejecutará el siguiente SET de instrucciones:

Proceso	Inst. Llegada	Tamaño (Kb)	Ciclo CPU	Ciclo E/S	Ciclo CPU	Ciclo E/S	Ciclo CPU
P1	0	100	7	3	4	2	5
P2	1	160	4	2	1	1	3
P3	2	70	1	3	5	2	4
P4	2	120	8	2	6	3	4

Sabiendo que existe un único dispositivo de entrada / salida, que un ciclo de CPU o E/S se corresponde a una unidad de tiempo y que la gestión de memoria de los procesos se realiza mediante particiones dinámicas, se pide calcular el Turnaround Time e indicar el tiempo de finalización de cada proceso. Justifique su respuesta mediante la confección en forma clara y detallada de un diagrama de GANTT.

Teoría:

Short Term Scheduler o Planificación a corto plazo: consiste de un “dispatcher”, que determina cual es el próximo proceso que va a hacer uso de la CPU.

Algoritmos Preemptive: Estos algoritmos liberan al procesador del proceso que lo esta usando para seleccionar uno de la cola de listos como reemplazo. Dado que el proceso por si mismo no deja el uso del procesador, se debe incorporar un mecanismo para forzar un context switch. Generalmente se recurre a una interrupción, por ejemplo, de reloj que requiere una atención del Kernel y éste aprovecha esta circunstancia para cambiar el PCB del proceso en uso del procesador por otro PCB de la cola de listos.

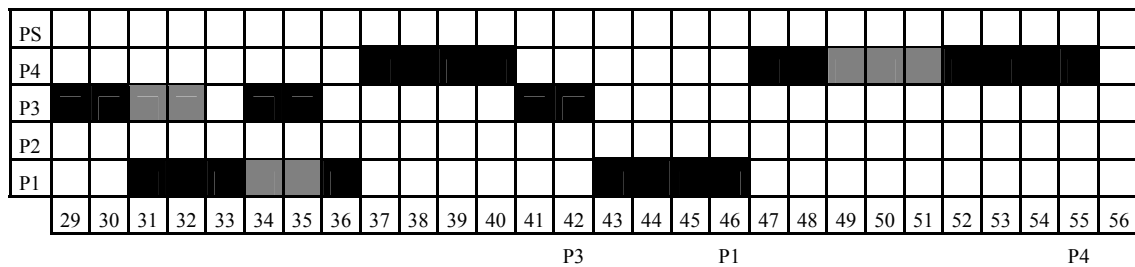
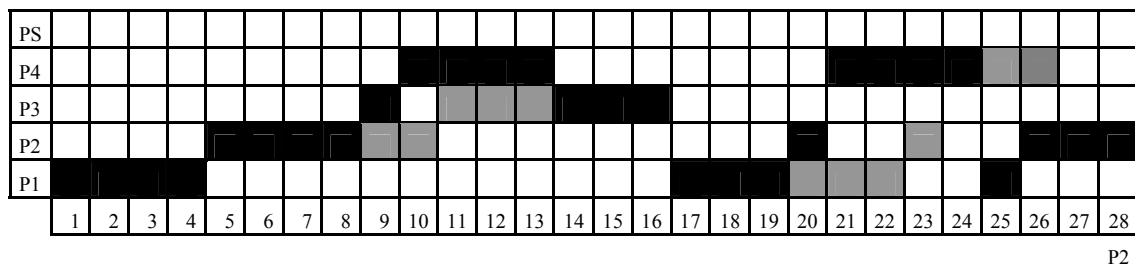
Algoritmo Round Robin: Se define una pequeña unidad de tiempo llamada quantum de tiempo o time slice (QT). La cola de listos se trata como una cola circular. El planificador de CPU recorre la cola, asignando procesador a cada proceso durante un intervalo de tiempo de hasta un quantum. La cola se mantiene como una cola de procesos FIFO. Los procesos nuevos, se añaden al final de la cola.

Algoritmo Virtual Round Robin: Este algoritmo funciona de forma similar al Round Robin, aunque los procesos que son desbloqueados se colocan en una cola FIFO auxiliar, la cual tiene prioridad por sobre la cola principal. Al momento de decidir que proceso pasa a ejecución se consulta esta cola auxiliar y si hay procesos en ella se procede a seleccionarlos primero. El quantum de tiempo para esta cola auxiliar se calcula como el quantum de tiempo de la cola principal menos el tiempo que el proceso utilizó la de CPU.

Dispositivo con planificación FIFO: los procesos usan el dispositivo de E/S en el orden que llegaron y por ser un algoritmo apropiativo, no lo abandonan hasta que finalice su uso.

Turnaround Time o tiempo de servicio: Tiempo de ejecución desde que se somete el nuevo trabajo hasta que el proceso termina su ejecución.

Resolución



Turnaround Time:

P1: 46
P2: 27
P3: 40
P4: 53

Dado que el enunciado no especifica un tamaño máximo de memoria disponible, asumimos que los procesos se pueden cargar en memoria sin problemas.

El Turnaround Time se calcula como el tiempo de finalización del proceso menos el instante en el que entró al sistema. Así por ejemplo, el Turnaround Time para el proceso P3 es $42 - 2 = 40$

El enunciado no menciona demora por parte del procesador para realizar el context switch de los procesos, por lo que lo consideramos despreciable.

Todos los procesos comienzan y terminan su ejecución con ráfagas de CPU.

¿Clases de Inglés telefónicas con Profesores Nativos dónde y cuándo quieras?

25 minutos de conversación en inglés por teléfono desde 3,25 €/sesión.
Conoce más en www.ringteacher.com

ringteacher

Ejercicio 9

Enunciado

	Recursos Asignados				Máximo pedido de recursos			
	A	B	C	D	A	B	C	D
PO	0	0	1	2	0	0	1	2
P1	2	0	0	0	2	7	5	0
P2	0	0	3	4	6	6	5	6
P3	2	3	5	4	4	3	5	6
P4	0	3	3	2	0	6	5	2

Recursos Disponibles			
A	B	C	D
2	1	0	0

Dada la situación planteada de procesos y recursos:

- Es este un estado seguro? Detallar los pasos que lo llevan a la conclusión. Justificar su respuesta
- Suponga que P2 realiza, inmediatamente después de concluir con su ejecución, el próximo pedido de recursos y este es: (0,3,0,0) respectivamente. Los restantes procesos se encuentran en la situación que les corresponde de acuerdo al grado de avance. En que estado (deadlock, seguro, inseguro) quedaría el sistema si se otorgara este pedido.

Marco Teórico para la Resolución:

Este ejercicio nos plantea una secuencia de procesos (P_n) y nos facilita dos matrices, o tablas: la tabla de "Recursos Asignados" que denota la cantidad de unidades del recurso A, B, C o D posee en un preciso instante, cada proceso del sistema. Por otro lado, la tabla de "Máximo Pedido de Recursos" que denota la cantidad máxima de un recurso que requerirá un proceso determinado, para finalizar su ejecución. Por otro parte encontramos la tabla de "Recursos Disponibles" que denota la cantidad de unidades que hay en el sistema actualmente disponibles de cada recurso que los procesos poseen y/o requerirán.

El ejercicio nos pide que demos si existe forma alguna en que estos procesos se ejecuten permaneciendo en un estado seguro, sin caer en deadlock, de manera que la secuencia de procesos, de alguna determinada manera, concluya.

Para permitir lograr la ejecución de estos procesos debe existir sincronización entre los mismos a la hora de utilizar los recursos, el objetivo que se persigue es poder asegurar la exclusión mutua, es decir, que solo un proceso pueda entrar utilizar un recurso crítico, o sea, aquel que solo puede ser accedido por un recurso a la vez.

En este caso particular deberemos hacer uso del llamado Algoritmo del Banquero. Este algoritmo nos permite determinar si un sistema queda en un estado seguro o inseguro analizando las peticiones que van a realizar los procesos, por lo que el mismo requiere conocer de antemano las solicitudes de recursos de cada uno de ellos.

Un sistema, queda en estado seguro, si existe una traza de ejecución en la cual todos los procesos finalizan, es decir, si tiene los recursos disponibles para poder satisfacer de alguna manera, las solicitudes de los procesos. Si no se puede realizar, o hallar una traza en la cual todos los procesos finalicen, el estado del sistema es inseguro (observación: que un estado sea inseguro no determina que los procesos están en deadlock). El estado inseguro alerta que si la secuencia se ejecuta de una manera determinada, los procesos pueden llegar a quedar interbloqueados, pero no significa que si o si va a suceder.

Por lo tanto, ahora nos dipondremos a hallar si existe una posible traza en la cual los procesos dados, con las solicitudes y los recursos dados, logran finalizar en su totalidad.

Resolución Práctica:

a)

1º)

	PEDIDOS			
	A	B	C	D
P0	0	0	0	0
P1	2	7	5	0
P2	6	6	5	6
P3	4	3	5	6
P4	0	6	5	2

	ASIGNADOS			
	A	B	C	D
P0	0	0	0	0
P1	2	0	0	0
P2	0	0	3	4
P3	2	3	5	4
P4	0	3	3	2

DISPONIBLES			
A	B	C	D
2	1	1	2

EJECUTA P0

2º)

	PEDIDOS			
	A	B	C	D
P0	0	0	0	0
P1	2	7	5	0
P2	6	6	5	6
P3	0	0	0	0
P4	0	6	5	2

	ASIGNADOS			
	A	B	C	D
P0	0	0	0	0
P1	2	0	0	0
P2	0	0	3	4
P3	0	0	0	0
P4	0	3	3	2

DISPONIBLES			
A	B	C	D
4	5	6	6

EJECUTA P3

3º)

	PEDIDOS			
	A	B	C	D
P0	0	0	0	0
P1	2	7	5	0
P2	6	6	5	6
P3	0	0	0	0
P4	0	0	0	0

	ASIGNADOS			
	A	B	C	D
P0	0	0	0	0
P1	2	0	0	0
P2	0	0	3	4
P3	0	0	0	0
P4	0	0	0	0

DISPONIBLES			
A	B	C	D
4	8	9	8

EJECUTA P4

4º)

	PEDIDOS			
	A	B	C	D
P0	0	0	0	0
P1	0	0	0	0
P2	6	6	5	6
P3	0	0	0	0
P4	0	0	0	0

	ASIGNADOS			
	A	B	C	D
P0	0	0	0	0
P1	0	0	0	0
P2	0	0	3	4
P3	0	0	0	0
P4	0	0	0	0

DISPONIBLES			
A	B	C	D
6	8	9	8

EJECUTA P1

Como 5º y último paso, P2 recibe los recursos solicitados, y posteriormente culmina la ejecución de los procesos dados, y el estado final de los recursos es el siguiente:

DISPONIBLES			
A	B	C	D
6	8	12	12

Para concluir, la secuencia de ejecución ha sido: P0 => P3 => P4 => P1 => P2

Por lo tanto, podemos ver que como se llega a la ejecución de toda la secuencia, se trata de un estado seguro.

b) Llegado el caso de que P2 luego de concluir su ejecución un pedido de recursos del tipo (0,3,0,0), léase esto como "A=0 , B=3 , C=0 , D=0" , podemos ver que

tranquilamente el pedido puede ser satisfecho ya que hay recursos suficientes como para otorgárselos a P2 en ese instante, por lo cual, el estado será *seguro*.

Ejercicio 13.

Enunciado:

Suponga que los siguientes 2 procesos, Foo y Bar, son ejecutados concurrentemente en un S.O. preemptive y comparten los contadores generales de los semáforos A y B (ambos inicializados en 1) y la variable entero count (inicializada en 0):

Proceso Foo	Proceso Bar
repeat	repeat
wait (A);	while(random (1) < 0,5 && count != 0) {
writeln("algo");	wait(B);
if(count > 0) then	writeln("mas cosas");
writeln("otra cosa");	wait(A);
wait(B);	count := count - 1;
count := count + 1;	writeln(count);
signal(A);	signal(A);
signal(B);	signal(B);
forever	}
	forever

1. ¿Puede ocurrir deadlock en la ejecución concurrente de los procesos, Foo y Bar?. En caso afirmativo indicar una secuencia de ejecución que resulte en deadlock, en caso negativo, explique porque no.
2. ¿Puede la ejecución de los dos procesos resultar en starvation de alguno de los dos procesos? En caso afirmativo, indicar una secuencia de ejecución que justifique su respuesta, en caso negativo, explique porque no.
3. ¿Considera que el acceso a los recursos críticos se encuentra correctamente sincronizado?

Teoría

Los programas pueden ser clasificados en secuenciales y en concurrentes.

Programa secuencial: especifica una secuencia de instrucciones que se ejecutan sobre un procesador que definimos como proceso o tarea. Se caracteriza por no ser dependiente de la velocidad de ejecución y de producir el mismo resultado para un mismo conjunto de datos de entrada

Programa concurrente (o lógicamente paralelo): especifica dos o más procesos secuenciales que pueden ejecutarse concurrentemente como tareas paralelas. Las actividades están superpuestas en el tiempo (una operación puede ser comenzada en función de la ocurrencia de algún evento, antes de que termine la operación que se estaba ejecutando).

La programación concurrente requiere de mecanismos de sincronización y comunicación entre los procesos.

Preemptive (No apropiativo): Los procesos usan al procesador hasta que quieren o hasta que el S.O. decide sustituirlos por otro proceso.

Semáforo: herramienta genérica de sincronización de procesos. Permite el ordenamiento de las operaciones que realizan los procesos en el tiempo. Es una especie de bandera (señal o flag) que indica la posibilidad de acceder o no a un recurso. Es una función o un arreglo dentro del Kernel. Un semáforo es una variable protegida cuyo valor puede ser accedido y alterado tan sólo por las dos primitivas independientes y atómicas y una operación de inicialización del semáforo.

Un semáforo S es una variable entera "e(s)" llamada valor del semáforo con las siguientes propiedades:

1. La variable puede tomar cualquier valor entero (positivo, negativo o nulo) en "e(s)" (valor del semáforo).
2. Se crea mediante un system call o una declaración, donde se especifica el valor inicial del semáforo que, por definición, debe ser un entero no negativo.

3. Sólo es accesible mediante dos operadores primitivos atómicos: `down()` y `up()` (o sus sinónimos, `P(s)` (Probaren) y `V(s)` (Verhogen), `wait()` y `signal()`, `pedir()` y `soltar()`, etc.)

La operación `wait` en el semáforo `S`, (se escribe `wait(S)`), opera de la siguiente manera:

```
if S > 0
    then S = S - 1
else (espera en S)
```

La operación `signal` en el semáforo `S`, (se escribe `signal(S)`), opera de la siguiente manera:

```
if (uno o más procesos están en espera en S)
    then (deja proseguir a uno de estos procesos) y
else (sigue en secuencia) luego hace S = S + 1
```

Supondremos que hay una disciplina de colas del primero en entrar - primero en salir (FIFO) para los procesos que esperan a completar un `wait(S)`. La ejecución de las instrucciones `wait` y `signal` son indivisibles. La mutua exclusión en el semáforo `S`, es aplicada en `wait(S)` y `signal(S)`. Si varios procesos intentan ejecutar `wait(S)` al mismo tiempo, sólo uno podrá proseguir; los otros permanecerán en espera. Los semáforos y las operaciones de semáforos pueden implementarse en software o hardware. En general, se implementan en el Kernel del sistema operativo, donde se controlan los cambios de estado de un proceso.

Starvation (Inanición, Postergación o Aplazamiento Indefinido): Consiste en el hecho de que uno o varios procesos nunca reciban el suficiente tiempo de ejecución para terminar su tarea. Por ejemplo, que un proceso ocupe un recurso y lo marque como 'ocupado' y que termine sin marcarlo como 'desocupado'. Si algún otro proceso pide ese recurso, lo verá 'ocupado' y esperará indefinidamente a que se 'desocupe'.

Deadlock (Abrazo Mortal, Interbloqueo, Bloque Mutuo o Estancamiento): se define como el estado en el que dos o más procesos esperan por condiciones que no se dan y que deben producirse por los procesos de ese conjunto. Se dice que dos o más procesos se encuentran en estado de deadlock cuando están esperando por condiciones que nunca se van a cumplir.

Los puntos de entrada de un recurso indican la cantidad de procesos que pueden utilizar simultáneamente al mismo.

Recurso crítico (o no compartible): Un recurso con 1 punto de entrada.

Región crítica de un proceso: es la fase o etapa en la vida de ese proceso concurrente en la cual accede a un recurso crítico para modificarlo o alterarlo.

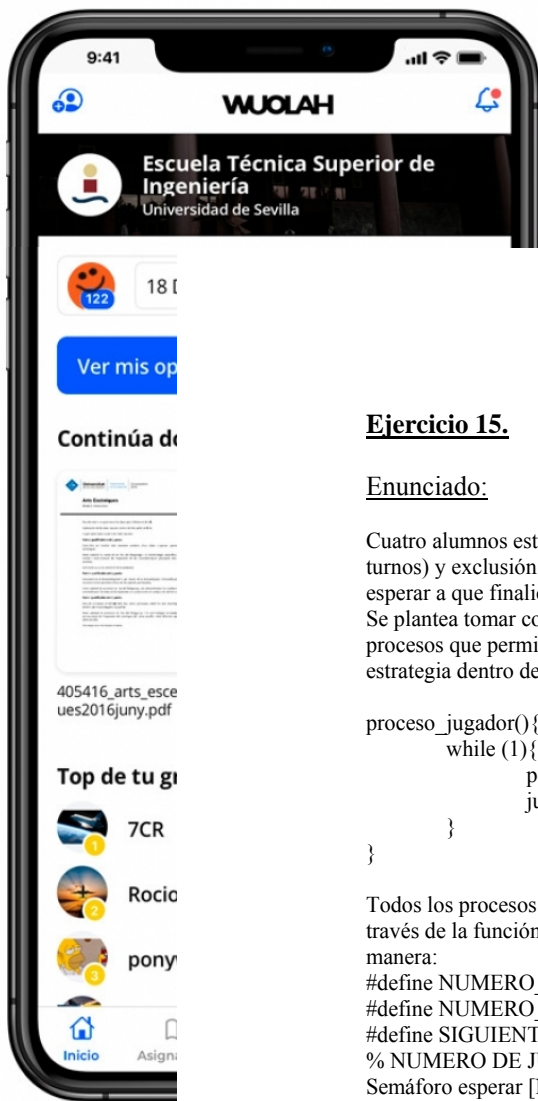
Respuestas:

1. Sí, puede ocurrir deadlock. El caso sería que se esté ejecutando el proceso Foo y no ingrese al while el proceso Bar por no cumplirse la condición. Y supongamos que ingresa al while (el proceso Bar) una vez que el proceso Foo comienza a repetir su ciclo.

El proceso Bar quedaría bloqueado en A (encolado) y el proceso Foo, por consecuencia, al seguir su ejecución quedaría bloqueado en B (encolado).

2. Si, puede ocurrir Starvation. Es difícil que ocurra, pero la posibilidad de que el proceso Bar no ingrese al ciclo del while por no cumplir su condición, existe.

3. En realidad no es lo más óptimo. Podría liberarse cada recurso una vez que finaliza su uso y antes de utilizar otro, es decir, que cada proceso pueda utilizar un recurso, sólo cuando ya no esté utilizando algún otro. También se podría cambiar la condición de acceso al while en el proceso Bar.



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.



Ejercicio 15.

Enunciado:

Cuatro alumnos están jugando en clase un juego de cartas que implica coordinación (no saltarse los turnos) y exclusión mutua (mientras uno toma una carta o la cambia con el compañero el resto debe esperar a que finalice para no quitarse la carta de la mano).

Se plantea tomar como base esta experiencia para implementar una arquitectura de comunicación entre procesos que permita realizar una práctica de inteligencia artificial en la que cada alumno codificara su estrategia dentro de las siguientes funciones aisladas:

```
proceso_jugador(){
    while (1){
        pensar_la_jugada();    // va calculando la mejor opción
        jugar();                // intercambia rápidamente las cartas
    }
}
```

Todos los procesos serán hijos de un proceso principal denominado partida, y por lo tanto heredarán a través de la función fork() y exec() correspondiente una serie de semáforos inicializados de la siguiente manera:

```
#define NUMERO_DE_JUGADORES 4
#define NUMERO_DE_CARTAS 40
#define SIGUIENTE (IDENTIFICACIÓN_DE_JUGADOR) ((IDENTIFICACIÓN_DE_JUGADOR + 1)
% NUMERO DE JUGADORES)
Semáforo esperar [NUMERO_DE_JUGADORES]
User_ID cartas [NUMERO_DE_CARTAS]
```

Se pide:

- Intercalar en el código propuesto para proceso_jugador() las funciones wait y signal necesarias sobre los semáforos esperar[IDENTIFICADOR_DE_JUGADOR] y esperar[SIGUIENTE (IDENTIFICADOR_DE_JUGADOR)] para asegurar el correcto funcionamiento del mismo, es decir, que cada jugador deba esperar su turno antes de poder ejecutar jugar().
- ¿Cómo debe estar inicializado el array de semáforos para que funcione, es decir, empiece el primer jugador y ceda el turno al siguiente y así sucesivamente, quedando bloqueado cada cual hasta que le toque el turno?
- En el caso que jugar() no se ejecute de forma atómica, ya que tendrá que modificar al menos dos entradas de la tabla carta. ¿Habría que incorporar algo para garantizar la exclusión mutua en la ejecución de jugar()? De ser necesario, escribir el código. De no serlo, explicar por que.

Teoría:

Sincronización de procesos: Es el ordenamiento de las operaciones en el tiempo debido a las condiciones de carrera (acceder a los diversos recursos asincrónicamente. Ejemplo, dos procesos seleccionan la misma impresora simultáneamente lo que daría listados mezclados). Permite que un proceso continúe su ejecución después de la ocurrencia de un determinado evento.

Exclusión mutua: Los recursos no son compartibles, es decir que sólo un proceso a la vez puede usar el recurso. El recurso no debe ser compartido.

Semáforos: Herramienta genérica de sincronización de procesos. Permite el ordenamiento de las operaciones que realizan los procesos en el tiempo. Es una variable protegida que indica la posibilidad de acceder a un recurso. El valor de los semáforos puede ser accedido y modificado únicamente por las primitivas wait() y signal(). La ejecución de estas primitivas es indivisible o atómica.

Resolución:

a)

```
proceso_jugador(){
    while (1){
        pensar_la_jugada();    // va calculando la mejor opción
        wait(esperar[IDENTIFICACION_DE_JUGADOR]);
        jugar();                // intercambia rápidamente las cartas
        signal(esperar[SIGUIENTE(IDENTIFICADOR_DE_JUGADOR)]);
    }
}
```

b) esperar[1,0,0,0]

c) No es necesario porque wait() y signal() se ejecutan en forma atómica, por lo tanto se garantiza la exclusión mutua en la ejecución de jugar(). Para que se ejecute esa sentencia cada proceso debe esperar a que se habilite el respectivo semáforo del cual está haciendo el wait().

P0	P1	P2	P3	Esperar			
				0	1	2	3
pensar_la_jugada()	pensar_la_jugada()	pensar_la_jugada()	pensar_la_jugada()	1	0	0	0
wait(esperar[0])	wait(esperar[1])	wait(esperar[2])	wait(esperar[3])	0	0	0	0
jugar()				0	0	0	0
signal(esperar[1])				0	1	0	0
pensar_la_jugada()				0	0	0	0
wait(esperar[0])	jugar()			0	0	0	0
	signal(esperar[2])			0	0	1	0
	pensar_la_jugada()			0	0	0	0
	wait(esperar[1])	jugar()		0	0	0	0
		signal(esperar[3])		0	0	0	1
		pensar_la_jugada()		0	0	0	0
		wait(esperar[2])	jugar()	0	0	0	0
			signal(esperar[0])	1	0	0	0
			pensar_la_jugada()	0	0	0	0
jugar()			wait(esperar[3])	0	0	0	0

Problema 17

En un aeropuerto que se utiliza como base de operaciones de una flota de aviones, tenemos por un lado los aviones, las 10 pistas de aterrizaje / despegue y dos controladores aéreos encargados de gestionar todos los pedidos de los aviones. Uno de ellos se encarga de los pedidos de pistas para aterrizaje o despegue y otro de la liberación de las pistas cuando los aviones han finalizado dichas maniobras. Las pistas se pueden utilizar para despegar o aterrizar según se desee. Para poder utilizar una pista, los aviones deben solicitarla previamente al controlador de entrada y, una vez que hayan aterrizado o despegado, avisar al controlador de salida devolviéndole la pista al conjunto de pistas libres. Cada avión, en principio, se encuentra en el hangar realizando tareas de mantenimiento, para más tarde pasar a solicitar una pista a la torre de control.

El avión entonces usa la pista para despegar y avisa al controlador que dejó la pista libre. Cuando un avión decide aterrizar, realizará idénticas acciones a las que se acaban de describir. Teniendo en cuenta el siguiente código, se pide que lo sincronice convenientemente utilizando semáforos para que no se produzca ni deadlock, ni starvation, indicando el tipo y los valores iniciales de los mismos.

Avión	Controlador Entrada	Controlador Liberacion
<pre>while (true) { mantenimiento_en_hangar(); despegue(); vuela(); aterriza(); }</pre>	<pre>while (true) { pistas_libres--; }</pre>	<pre>while (true) { pistas_libres++; }</pre>

Deadlock

Estado en el que dos o más procesos esperan por condiciones que no se dan y que deben producirse por procesos de ese conjunto. Se dice que dos o más procesos se encuentran en estado de deadlock cuando están esperando por condiciones que nunca se van a cumplir

- Es el bloqueo permanente de un conjunto de procesos que o compiten por recursos del sistema o se comunican entre sí
- No tiene una solución eficiente
- Envuelve necesidades conflictivas por recursos entre dos o más procesos
- El problema es de naturaleza lógica.
- Mutua Exclusión
 - **sólo un proceso puede usar un recurso a la vez**
- Retener y esperar (Hold-and-wait)
 - **un proceso solicita todos los recursos requeridos al mismo tiempo**
- Expropiación (Preemptive)
 - **Si a un proceso que retiene ciertos recursos se le niega una solicitud, ese proceso debe liberar sus recursos originales**
 - **Si un proceso solicita un recurso que está siendo retenido por otro proceso, el sistema operativo puede instar al segundo proceso a que libere sus recursos**
- Espera circular
 - **Prevenida definiendo un orden lineal de los tipos de recurso**

```

wait( iSemaforo)

{
    while (iSemaforo <=0);

    iSemaforo--;
}

```

```

signal( iSemaforo)

{
    iSemaforo++;
}

```

Valores iniciales de los semáforos

SEMAFOROS Mutex

Utilizar=0; Liberar=0; Disponible=0;

SEMAFORO Contador

Pistas_libre = 10;

Proceso Avion()	Proceso Controlador ()	Proceso Liberación()
<pre> while(true) { mantenimiento_en_hangar(); signal(Utilizar); wait(Disponible); despegar(); signal(Liberar); volar(); signal(Utilizar); wait(Disponible); aterrizar(); signal(Liberar); } </pre>	<pre> while (true) { wait(Utilizar); wait(Pistas_libres); signal(Disponible); } </pre>	<pre> while (true) { wait(Liberar) signal(Pistas_Libres); } </pre>

Semáforos ,estado de pistas, y acciones

Semaforos						Acciones de un avion
Pistas_Libres	10	9	9	9	10	Signal(Utilizar)
Utilizar	0	1	0	0	0	Signal(Disponible)
Disponible	0	0	1	0	0	despegar() - Signal (Liberar)
Liberar	0	0	0	1	0	volar()
Pistas_Libres	10	9	9	9	10	Signal(Utilizar)
Utilizar	0	1	0	0	0	Signal(Disponible)
Disponible	0	0	1	0	0	aterizar() - Signal (Liberar)
Liberar	0	0	0	1	0	mantenimiento_en_hangar();

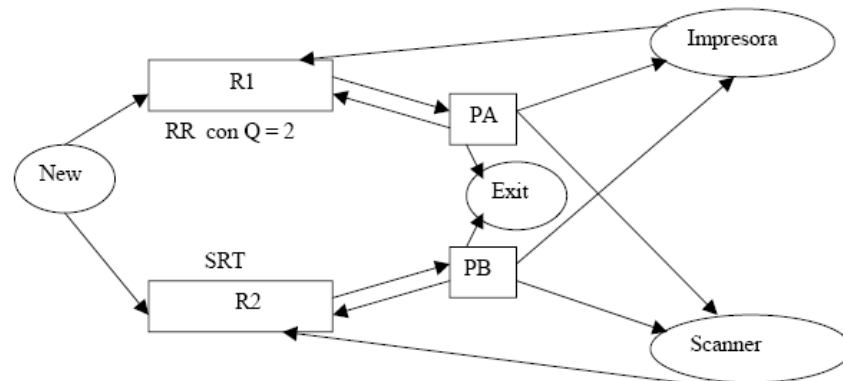
¿Clases de Inglés telefónicas con Profesores Nativos dónde y cuándo quieras?

25 minutos de conversación en inglés por teléfono desde 3,25 €/sesión.
Conoce más en www.ringteacher.com

ringteacher

Problema 20

Sea un sistema que implementa el diagrama de transiciones para la administración de los procesos indicado en el siguiente gráfico, siendo PA y PB dos procesadores.



Se sabe que todo proceso nuevo que ingresa al sistema se ubica en la respectiva cola de Ready según su prioridad y en función de las velocidades de los procesadores, según la siguiente tabla:

Cola R1	Procesos con Prioridad 0, 1, 2 ó 3
Cola R2	Procesos con Prioridad 4, 5, 6 ó 7

En un instante dado la carga del sistema es la siguiente:

Proceso	Prioridad	T. Llegada	CPU	I / O	CPU	I / O	CPU
PW	0	0	5	Scanner: 2	2	Scanner: 3	5
PX	4	2	3	Impresora :1	1	Scanner: 1	3
PY	2	1	1	Impresora :6	4	Impresora :1	2
PZ	5	4	4	Impresora :3	2	Impresora :4	1

Teniendo en cuenta que cada tipo de dispositivo de E / S tiene una única instancia, y que planifican FIFO, se pide el orden de ejecución de los procesos y en que instantes finalizan, mediante la confección, en forma clara y detallada de un diagrama de GANTT.

Datos teóricos Adicionales

A modo de comentar un poco sobre Ráfagas del Procesador y estos planificadores (RR – SRT)

Durante su ejecución, un proceso alterna entre períodos de uso de CPU (**CPU bursts**) o ráfagas de CPU y períodos de uso de I/O (**I/O bursts**). La ejecución siempre comienza y termina por un CPU burst. En sistemas con procesos I/O bound (límite), la duración de los CPU bursts será muy corta. Los procesos alteran entre CPU burst y I/O burst durante su ejecución.

Turno rotatorio (Round-Robin)

- Emplea un fraccionamiento del tiempo para hacer que los procesos se limiten a ejecutar en ráfagas cortas de tiempo, rotando entre los procesos listos.
- La cuestión principal de diseño es la longitud del cuanto de tiempo (Quantum o Time Slice) o fracción que se va a usar.
- Si el cuanto es muy pequeño, los procesos cortos pasan por el sistema operativo, produciéndose una sobrecarga (overhead) en la gestión de interrupciones del reloj y en la ejecución de las funciones de planificación y expedición.
- El cuanto debe ser mayor que el tiempo necesario para una interacción.
- Es particularmente efectivo en sistemas de propósito general y de tiempo compartido o de proceso de transacciones.

SRT: Shortes Remanening Time

Es la Versión Expropiativa de (SJF- Shortest Process Firts) primero el proceso más corto

En este caso el planificador primero va a ejecutar al proceso con menor tiempo restante o remanente, o sea, selecciona al proceso al aquel le queda menos tiempo esperando de ejecución en el procesador. Es más eficiente que el RR.


Ventajas: Mejor performance ya que los proceso cortos toman de inmediato el control del CPU.

El siguiente diagrama representa los ciclos y ráfagas de cpu que cada proceso utiliza acorde al planificador, colas de espera de CPU del procesador A o B según corresponde, las colas de espera de uso de I/O, el tiempo o cantidad de ráfagas transcurridas para la finalización de cada proceso. El grafico del enunciado nos muestra que el proceso que termino de utilizar el PB y luego utiliza el scanner vuelve a la cola R1 cambiando de algoritmo, y el que termino de utilizar PA y utiliza l impresora vuelve a la cola R2 con algoritmo SRT.

Cola R1		PW							PX	PX	PY		PZ		PY								
Cola R2				PZ											PW	PW	PW						
Cola Imp						PX	PX	PX															
Cola Scanner												PX	PX										
PW	A	A		A	A	A	S	S		B	B	S	S	S	B				B	B	B	B	X
PY			A	I	I	I	I	I	I	A	A		A	A	I		A	A	X	X			
PX			B	B	B					I		A			S	B	B	B	X	X			
PZ						B	B	B	B		I	I	I		A	A	I	I	I	I	A	X	
Rafagas	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
																			XY			Z	W

A PA

B PB

 Scanner

 Impresora

☒

Fin del Proceso

Cola con algoritmo RR con Quantum =2

R1

Cola con algoritmo Shortest Remaining Times
Menor tiempo Restante

R2

PX	18
PY	18
PW	21
PZ	22

Tiempo de Finalizacion

Ejercicio 22

Enunciado:

Un sistema en el cual se utiliza memoria con particiones fijas para la organización de la memoria física de 10, 30, 42 y 60 Kb posee el siguiente juego de procesos:

Procesos	Proceso 1	Proceso 2	Proceso 3	Proceso 4	Proceso 5	Proceso 6	Proceso 7
	DOWN(A) UP(A) disco(5400) UP(C) UP(E) DOWN(H)	DOWN(G) disco(3457) UP(I) disco(4587) UP(J)	DOWN(G) DOWN(A) disco(4587) DOWN(C) DOWN(I) UP(C) disco(4888) disco(1244) UP(J)	DOWN(F) UP(A) disco(5400) DOWN(F) UP(B)	DOWN(B) disco(4587) UP(B) disco(4888) disco(1244) UP(G)	DOWN(J) UP(A) disco(5400) UP(G)	DOWN(E) UP(I) DOWN(C) Disco(358) UP(F)
T.Llegada	0	0	0	2	3	4	4
Espacio Memoria (Kb)	24	2	32	4	40	22	60

- Asuma que las operaciones atómicas insumen 1 ms en ejecutarse.
- A su vez, se tiene un disco rígido de 40 GB cuya velocidad de rotación es de 3000 RPM y el cabezal tarda 3 ms en pasar de una pista a otra. Las cabezas leen del sector mas chico hacia el mas grande (0,1,2,...,n). La configuración del disco es la siguiente: 100 cilindros, pistas de 20 sectores y 4 platos. Conformando un total de 8 cabezas. Las cabezas se encuentran en la dirección lógica 100, ascendiendo.
- El algoritmo de planificación de disco es el SSTF (Shortest Seek Time First)
- Considere que una vez que el proceso toma la memoria principal no la libera hasta que finalice o hasta que se bloquee por algún motivo y que el algoritmo para la selección de la víctima es el best fit.
- El estado inicial de los semáforos es el siguiente: A= B = E = G = I = J = 30 y C = F = H = 25
- Considere que se desprecia el tiempo de CPU que se insume en realizar un acceso al disco.

Teniendo en cuenta todos los datos mencionados se pide:

- a) La traza de ejecución de los procesos, en forma clara, el orden de finalización de los mismos y en que instante lo hacen. En caso que algún proceso no finalice debe fundamentar correctamente el motivo por el cual no lo hacen. No deje de indicar el estado de cada una de las colas en los diferentes instantes de tiempo.
- b) Tiempo medio de atención de los pedidos que se realizaron al disco y el orden en que fueron atendidos. Calcular el porcentaje de utilización de la CPU.

Marco Teórico del ejercicio:

El ejercicio enuncia que estamos tratando con un sistema que utiliza memoria con particiones fijas, esto implica que la memoria siempre tendrá un tamaño “n” el cual estará siempre subdividido en “i” particiones, cada una de ellas de tamaño fijo. En este caso están fijadas de antemano.

La asignación particionada permite el acceso del procesador a varios procesos simultáneamente ejecutando de a uno por vez según los algoritmos de planificación para su ejecución. Para poder repartir el procesador entre varios procesos, necesitamos disponer de ellos en la memoria central, para ellos la misma se divide en distintas regiones o particiones, cada una de las cuales mantiene un espacio de dirección para un solo proceso.

En el caso de particiones fijas la especificación de la partición la puede designar el usuario o sea puede incorporar en el SO. En este caso las particiones suponemos que fueron dadas por el SO ya que están prefijadas. la memoria estará subdividida en 4 particiones de 10, 30, 42 y 60Kb respectivamente.

Cada paso de trabajo proporcionado por un usuario debe especificar la máxima cantidad de memoria necesaria. Entonces se encuentra y asigna una partición de tamaño suficiente. En este ejercicio la

ubicación de los procesos en las particiones correspondientes será mediante la aplicación del algoritmo Best Fit, el cual busca en toda la memoria una partición, pero no se conforma con la primera que encuentra libre para ubicar el proceso, sino que ubica la partición libre cuyo tamaño encaje mejor con el tamaño del proceso, es decir, que se desperdicie menor cantidad de memoria en dicha partición.

Luego, nos mencionan la planificación del disco, dice que se utiliza un algoritmo de planificación de Shortest Seek Time First, con lo cual nos indican que dado una serie de pedidos simultáneos se atiende primero al pedido que más cercano está de la posición actual de la cabeza, es decir el que produce un menor desplazamiento del brazo. Pero puede producir inanición de aquellos pedidos que están en direcciones muy alejadas.

Primero que nada, en cuanto a los pedidos de E/S debemos poder traducir las direcciones lógicas en físicas para saber y contemplar los desplazamientos que tendrá que hacer el brazo y así tener en cuenta los tiempos de búsqueda que se demorará en acceder a una ubicación solicitada.

Para esto ya nos vienen dados ciertos datos de las características del disco, como ser que tiene 3000 RPM de velocidad de rotación y el tiempo que se tarda en pasar de pista a pista es de 3ms. A partir de estos tiempos podremos calcular los denominados tiempos de retardo rotacional:

3000 vueltas ____ 1 minuto

$$1 \text{ vuelta} \text{ ____ } 0.00033 \text{ min} = 0.02 \text{ seg} = \boxed{20 \text{ ms}}$$

Luego también tenemos otros datos físicos, como ser que tiene 100 cilindros y que hay pistas de 20 sectores contenidas en 4 platos.

A partir del tiempo que tarda en dar una vuelta podemos calcular lo que tarda en recorrer un sector de la siguiente manera:

20 sectores ____ 1 vuelta (20ms)

$$1 \text{ sector} \text{ ____ } \boxed{1 \text{ ms}}$$

Por último, lo que necesitamos es saber como convertir las direcciones lógicas a las cuales referencian los procesos, en direcciones físicas, esto se logra teniendo en cuenta ciertas características físicas del disco y haciendo una serie de cuentas a saber:

$$\text{Sectores x Pista} = \boxed{20}$$

Sectores x Pista x Plato = 40 (ya que por cada plato tenemos pistas de ambas caras)

$$\text{Sectores x Pista x Plato x Cantidad de Platos} = 40 * 4 = \boxed{160}$$

A partir de esto podemos calcular las direcciones físicas a partir de las lógicas, mediante las siguientes divisiones sucesivas:

$$\text{Dirección Lógica} / \text{Sectores x Pista x Plato x Cantidad de Platos} = C1 + R1$$

C1 = Cilindro de la dirección

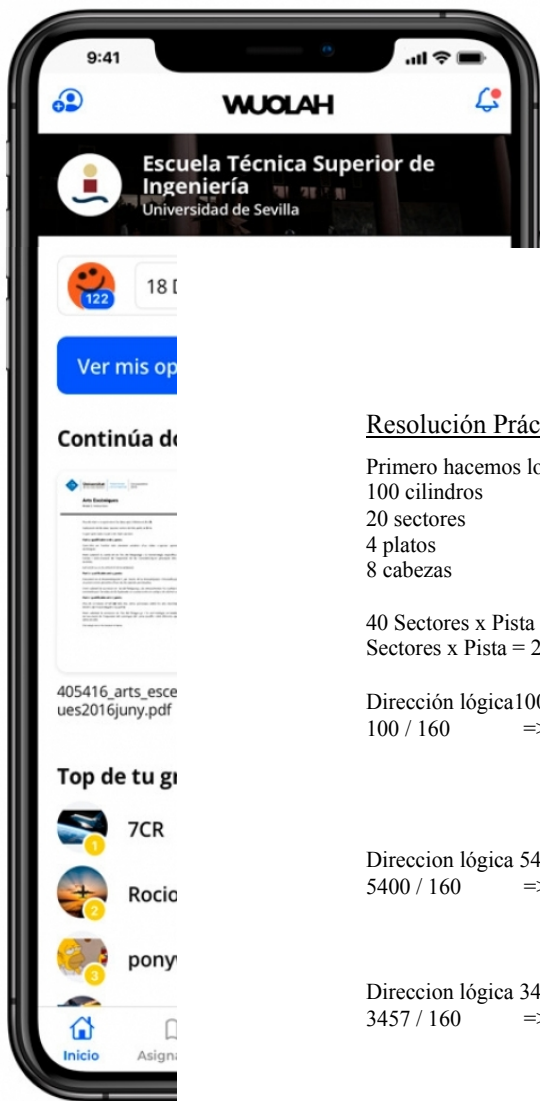
$$R1 / \text{Sectores x Pista} = C2 + R2$$

C2 = Cabeza de la dirección

$$R2 + 1 = \text{Sector de la dirección}$$

Conformando así una tupla del estilo (C1,C2;R2+1) = (Cilindro, Cabeza, Sector)

Esto se verá con mayor claridad en la resolución del ejercicio mismo.



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.



Resolución Práctica:

Primero hacemos los cálculos para los pedidos del disco:

100 cilindros

20 sectores

4 platos

8 cabezas

40 Sectores x Pista x Plato * 4 Platos = 160

Sectores x Pista = 20

Dirección lógica 100:	=>	(0,5,1)			
100 / 160	=>	C = 0			
		R = 100 =>	100 / 20 =>	C = 5	
					R = 5 + 1
Dirección lógica 5400:	=>	(33,6,1)			
5400 / 160	=>	C = 33			
		R = 120 =>	120 / 20 =>	C = 6	
					R = 0 + 1
Dirección lógica 3457:	=>	(21,2,18)			
3457 / 160	=>	C = 21			
		R = 97	=>	97 / 20	=>
				C = 4	
					R = 17 + 1
Dirección lógica 4587:	=>	(28,5,8)			
4587 / 160	=>	C = 28			
		R = 107 =>	107 / 20 =>	C = 5	
					R = 7 + 1
Dirección lógica 4888:	=>	(30,4,9)			
4888 / 160	=>	C = 30			
		R = 88	=>	88 / 20	=>
				C = 4	
					R = 8 + 1
Dirección lógica 1244:	=>	(7,6,5)			
1244 / 160	=>	C = 7			
		R = 124 =>	124 / 20 =>	C = 3	
					R = 4 + 1
Dirección lógica 358:	=>	(2,1,19)			
358 / 160	=>	C = 2			
		R = 38	=>	38 / 20	=>
				C = 1	
					R = 19

Ahora procedamos a seguir la traza de ejecución de los procesos a medida que los mismos van llegando, hay que tener en cuenta las locaciones en memoria, y los diferentes tiempos de las solicitudes de E/S que realicen los mismos.

P7																		EX	EX	EX	D
P6														EX	EX	D					
P5												EX	D								
P4									EX	EX	D										
P3						EX	EX	D													
P2				EX	D																

P1	EX	EX	D																	
TIEMPO	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
10Kb	P2	P2	P2	P2																
30Kb	P1	P1	P1	P4	P4	P4	P4	P4	P4	P4	P4									
42Kb	P3	P3	P3	P3	P3	P3	P3	P3	P6	P6	P6	P6	P6	P6	P6	P6				
60Kb					P5	P5	P5	P5	P5	P5	P5	P5	P5	P7	P7	P7	P7	P7	P7	P7

Hasta el instante 18 todos los procesos pudieron ejecutar sin bloquearse, salvo que en ese instante todos estarán esperando E/S, con lo cual solo nos queda analizar que sucede entonces con el disco a partir del momento en que llega el primer pedido y éste es atendido:

En el instante 0 el disco estaba en (0,5,1) pero cuando le llega el primer pedido será en el instante 3ms, con lo cual el disco habrá recorrido 3 sectores y se encontrará en (0,5,4).

En los primeros 13ms de ejecución, al disco llegarán los siguientes pedidos:

P1(5400) – P2(3457) – P3(4587) – P4(5400) – P5(4587) – P6(5400) – P7(358)

El primer pedido en llegar es el (33,6,1) y como no tiene otro pendiente en ese instante procede a atenderlo, pese a que en el tiempo que tarda en rotar y posicionarse van a ir llegando los otros, allí es donde entra en juego el algoritmo SSTF. Por ahora solo nos interesa saber cuanto tardará en cumplimentar el primer pedido:

(0,5,4) -> (33,6,1) => 33pistas x 3ms = 99ms => 99ms = 4 vueltas + 19 sectores
 (33,_,3) => Tiene que llegar de nuevo hasta 1 => 17ms + 1ms de lectura => **117ms**

Observación: En el transcurso en que el disco procedía a atender el pedido, le llegaron más pedidos a la misma dirección con lo cual los satisface a todos (P1 – P4 – P6)

P7							
P6						EX	FIN
P5							
P4				EX	EX	FIN	
P3							
P2							
P1	EX	EX	EX	FIN			
TIEMPO	120	121	122	123	124	125	126
10Kb	P4	P4	P4	P4	P4		
30Kb	P1	P1	P1				
42Kb	P6	P6	P6	P6	P6	P6	
60Kb							

A todo esto, en el disco había una serie de pedidos pendientes:

P2(3457) – P3(4587) – P5(4587) – P7(358)

La cabeza estaba en la posición (33,6,2) tras el acceso anteriormente satisfecho entonces es aquí donde entra en juego la aplicación del algoritmo SSTF. Desde la posición actual, el pedido más cercano a satisfacer es 4587, que vemos también lo tienen varios procesos (P3-P5)

(33,6,2) -> (28,5,8) => 5 pistas x 3ms = 15ms => 15 sectores
 (28,_,17) => 3 sectores + 8 sectores => 11ms + 1ms de lectura => **27ms**

P7						
P6						
P5						

P4					EX	D
P3	EX	EX	EX	D		
P2						
P1						
TIEMPO	147	148	149	150	151	152
10Kb	P3	P3	P3	P3		
30Kb						
42Kb	P5	P5	P5	P5	P5	P5
60Kb						

Estamos en la posición (28,5,9) y tenemos por procesar los pedidos pendientes:

P2(3457) – P7(358)

Los iniciamos a procesar en el instante 147, recién en el 150 aparecen nuevos pedidos, por lo tanto atendemos primero al pedido de P2 que está más cercano que el de P7.

(28,5,9) -> (21,2,18) => 7 pistas x 3ms = 21ms => 1 vuelta + 1 sector
 (21,_,10) => tengo que avanzar hasta 18 => 8ms + 1ms de lectura => **30ms**

P7		
P6		
P5		
P4		
P3		
P2	EX	D
P1		
TIEMPO	177	178
10Kb	P2	P2
30Kb		
42Kb		
60Kb		

Ahora, en el instante en que se libera el disco, el 177, tendremos los siguientes pedidos en cola:

P7(358) – P3(4888) – P5(4888)

De la posición actual (21,2,19) el más cercano es sin dudas el 4888:
 (21,2,19) -> (30,4,9) => 9 pistas x 3ms = 27ms => 1 vuelta + 7 sectores
 (30,_,6) => tengo que llegar hasta el 9 => 3ms + 1ms de lectura => **31ms**

En el instante 208 que es cuando se cumplen estos pedidos, tanto P3 como P5 piden otro acceso a disco, mientras que en la cola ya teníamos los pedidos:

P7(358) – P2(4587)

Luego de aceptar atender al más cercano de la cola, se ingresan los pedidos de P3 y P5. Por tanto de entre el pedido de P2 y de P7 nos quedamos con el más cercano, o sea, con el de P2

(30,4,10) -> (28,5,8) => 2 pistas x 3ms = 6ms
 (28,_,16) => tengo que dar la vuelta hasta 8 => 4ms + 8ms + 1ms de lectura => **19ms**

Durante estos 19ms se encolan los pedidos de P3 y P5

Luego de satisfacerse este pedido de P2 ejecuta un instante de tiempo más y finaliza en el instante 228.

Mientras tanto en disco está la siguiente situación:

P7(358) – P3(1244) – P5(1244)

Por tanto estamos en (28,8,9), atendemos el pedido más cercano que será 1244:

(28,8,9) -> (7,6,5) => 21 pistas x 3ms = 63ms => 3 vueltas + 3 sectores

(7,_,12) => tiene que dar una vuelta y llegar a 5 => 8ms + 5ms + 1ms de lectura => **77ms**

En el instante 304 se satisfacen ambos pedidos, ejecuta luego P3 y finaliza, para luego dar lugar a la ejecución de P5 y su posterior finalización.

Por último en disco solo queda pendiente el pedido 357:

(7,6,7) -> (2,1,19) => 5 pistas x 3ms = 15ms

(2,_,2) => tiene que llegar a 19 => 15ms + 1ms de lectura => **31ms**

En el instante 335 se satisface el pedido y P7 finaliza luego de ejecutar un instante más.

Para comprender mejor la ejecución de los procesos a continuación se detalla el Diagrama Gantt completo:

P7																	EX	EX	EX	D
P6													EX	EX	D					
P5											EX	D								
P4								EX	EX	D										
P3						EX	EX	D												
P2				EX	D															
P1	EX	EX	D																	
TIEMPO	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
10Kb	P2	P2	P2	P2																
30Kb	P1	P1	P1	P4	P4	P4	P4	P4	P4	P4	P4									
42Kb	P3	P3	P3	P3	P3	P3	P3	P3	P6	P6	P6	P6	P6	P6	P6	P6				
60Kb					P5	P5	P5	P5	P5	P5	P5	P5	P5	P5	P7	P7	P7	P7	P7	P7

P7																				
P6						EX	FIN													
P5																				
P4				EX	EX	FIN							EX	D						D
P3									EX	EX	EX	D							D	
P2																EX	D			
P1	EX	EX	EX	FIN																
TIEMPO	120	121	122	123	124	125	126	[...]	147	148	149	150	151	152	[...]	177	178	[...]	208	209
10Kb	P4	P4	P4	P4	P4				P3	P3	P3	P3				P2	P2			
30Kb	P1	P1	P1																	
42Kb	P6	P6	P6	P6	P6	P6			P5	P5	P5	P5	P5	P5					P3	
60Kb																			P5	P5

¿Clases de Inglés telefónicas con Profesores Nativos dónde y cuándo quieras?

25 minutos de conversación en inglés por teléfono desde 3,25 €/sesión.
Conoce más en www.ringteacher.com

ringteacher

P7									EX	F
P6										
P5						EX	FIN			
P4										
P3					EX	FIN				
P2		EX	FIN							
P1										
TIEMPO	[...]	227	228	[...]	304	305	306	[...]	335	336
10Kb		P2								
30Kb										
42Kb					P3					
60Kb					P5	P5			P7	

Ejercicio 23.

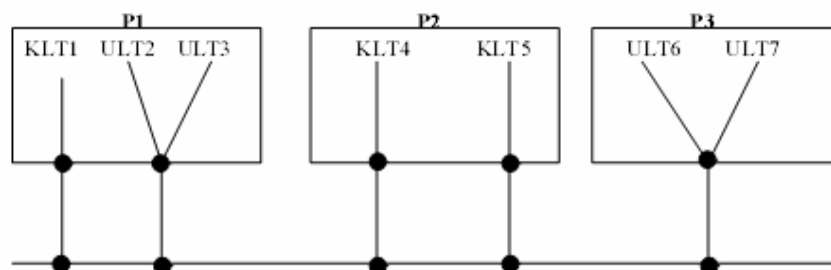
Enunciado:

Sea un sistema operativo que utiliza un Scheduler Round Robin con quantum de 2 unidades de tiempo, el cual deberá planificar a los procesos P1 (tiempo de llegada: Instante 0), P2 (tiempo de llegada: instante 3) y P3 (tiempo de llegada: instante 8). Los tres procesos fueron programados por el famosísimo Billy Puertas como parte de una poderosa aplicación que utilizan los servicios secretos. Como es su costumbre, Billy planifica sus hilos con SRT (Shortest Remaining Time).

Considerando, además, que existe un único dispositivo de I/O, se solicita indicar por medio de un diagrama de GANTT la traza de ejecución de los procesos, teniendo en cuenta la siguiente estructura de los mismos:

(Observación: KLT = Kernel Level Threads; ULT = User Level Threads)

	CPU	I/O	CPU	I/O	CPU
KLT1	5	1	3		
ULT2	3	2	2		
ULT3	4	4	3	4	1
KLT4	3	5	2	1	1
KLT5	2	3	3		
ULT6	2	5	3		
ULT7	2	1	1	1	3



Teoría:

Round Robin: El Round Robin viene siempre acompañado por un slice de tiempo, es decir, un tiempo determinado que tienen los procesos para ejecutar; finalizado ese tiempo el proceso abandona la CPU. Debido a esta característica, el algoritmo es no apropiativo. Obviamente que el

proceso puede abandonar antes la CPU en caso de que finalice o se bloquee por una E/S; caso contrario (que se le termina el tiempo), vuelve a la cola de listos.

Shortest Remaining Time (SRT): Es muy similar al anterior, pero este sí es un algoritmo no apropiativo. En la cola se siguen ordenando según quien tenga el menor burst de CPU, pero si se da el caso de que en un momento hay un proceso ejecutando y llega uno nuevo, se determina si el proceso que está usando la CPU actualmente le falta menos del uso que va a hacer el nuevo proceso; si es menor, sigue ejecutando, si le queda más, el proceso es desalojado del procesador y llevado nuevamente a la cola de listos.

Hilo (Thread o Hebra): unidad elemental de uso del procesador. Cada hilo posee un Contador de Programa (PC - Program Counter), un juego de Registros del procesador (Register Set) y una Pila (Stack). Son como pequeños miniprocesos.

Cada thread se ejecuta en forma estrictamente secuencial compartiendo el procesador de la misma forma que lo hacen los procesos, solo en un multiprocesador o en una arquitectura acorde se pueden realizar en paralelo.

No existe protección entre los hilos debido a que es imposible y no es necesario, ya que generalmente cooperan entre sí la mayoría de las veces. Aparte del espacio de direcciones, comparten el mismo conjunto de archivos abiertos, procesos hijos, relojes, señales, etc.

Hilos a Nivel de Usuario (ULT): Todo el trabajo del hilo es manejado por la aplicación, el kernel ni se entera de la existencia de los hilos. Cualquier aplicación puede ser programada para ser multithreaded mediante el uso de threads library (paquete de rutinas para ULT en el compilador). Las bibliotecas contienen código para crear y destruir hilos, pasar mensajes y datos entre hilos, ejecución planificada de hilos y para guardar y restablecer contextos de hilos. Entonces la generación de los ULT se hace en el momento de compilación y no se requiere la intervención del Kernel.

Ventajas:

1. El cambio de hilo no requiere el modo kernel, porque todas las estructuras de datos están dentro del espacio usuario. Es más, el proceso no cambia al modo kernel para manejar el hilo.
2. El algoritmo de planificación puede ser adaptado sin molestar la planificación del SO.
3. ULT puede correr en cualquier SO.
4. Es muy rápido en la ejecución

Desventajas:

1. En un SO típico, la mayoría de los system call son bloqueantes. Cuando un hilo ejecuta un system call no sólo se bloquea ese hilo, sino que también se bloquean todos los hilos del proceso.
2. En una estrategia pura de ULT, una aplicación multithreaded no puede tomar ventaja del multiprocesamiento. Un kernel asigna un proceso a sólo un procesador por vez.

Hilos a nivel de Kernel (KLT): Todo el trabajo de manejo de hilos es hecho por el kernel. No hay código de manejo de hilo en el área de aplicación. Cualquier aplicación puede ser programada para ser multithreaded. Todos los hilos dentro de una aplicación son soportados dentro de un solo proceso. El kernel mantiene la información de contexto para el proceso e individualmente para los hilos dentro del proceso.

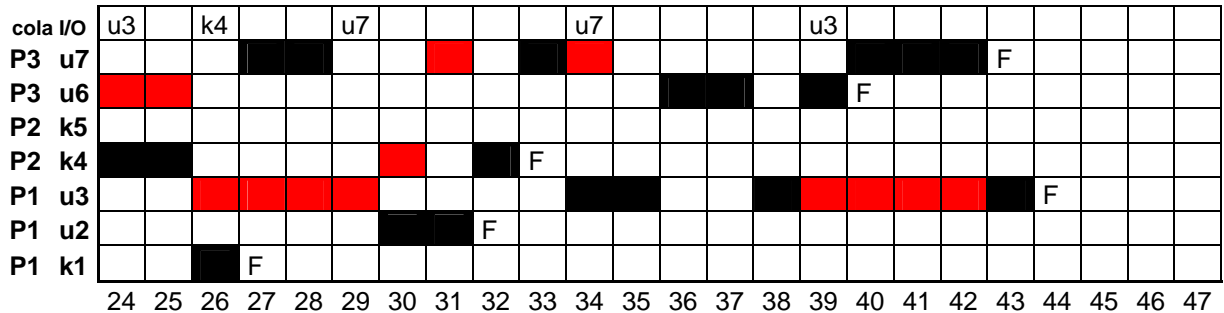
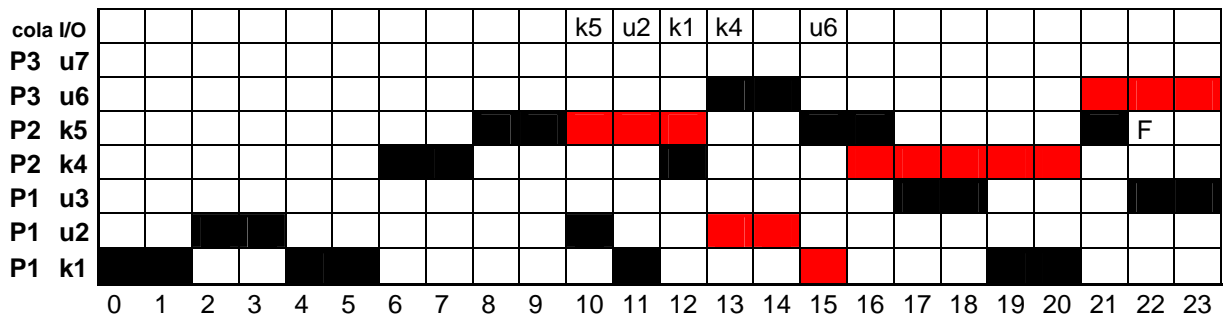
Ventajas:

1. Simultáneamente el kernel puede planificar múltiples hilos del mismo proceso en múltiples procesadores.
2. Si un hilo de un proceso se bloquea, el kernel puede planificar otro hilo del mismo proceso.
3. Las rutinas mismas del kernel pueden ser multithreaded.

Desventaja:

La transferencia de control de un hilo a otro dentro del mismo proceso le requiere al kernel un cambio de modo (context switch).

Resolución:



T. Llegada	P	CPU	I/O	CPU	I/O	CPU
0	k1	5	1	3		
	u2	3	2	2		
	u3	4	4	3	4	1
3	k4	3	5	2	1	1
	k5	2	3	3		
8	u6	2	5	3		
	u7	2	1	1	1	3

RR con q = 2
hilos SRT

 CPU
 I/O

T. Finalizacion	
P1	44
P2	33
P3	43

R Q = k1 k2 k1 k4 k5 k2 k1 k4 k6 k5 k2 k1 k5 k2 k4 k1 k6 k2 k4 k6
k2 k6 k2 k6 k6 k6 k2

I/O Q= k5 k2 k1 k4 k6 k2 k4 k6 k6 k2

Ejercicio 24

Enunciado:

Debido al gran deterioro que existe en el Puente Velez Sarsfield que une la Capital Federal con la provincia de Buenos Aires, el Gobierno de la provincia ha decidido tomar algunas precauciones para evitar accidentes fatales en dicho puente. Para ello se ha implementado un semáforo en cada una de las entradas al puente para que únicamente pasen los autos en un solo sentido. Cuando el semáforo se encuentra en verde en cualesquiera de las dos entradas significa que los autos pueden circular, si se encuentra en rojo no lo podrán hacer.

Los Ingenieros Civiles de la UTN fueron convocados para realizar un análisis y llegaron a la conclusión que no solamente deben circular los autos en un solo sentido, sino que también el peso máximo que soporta el puente es de 20 toneladas, siendo el peso promedio de un vehículo que circula por esa zona de 500 kilos. Viendo la gran problemática que ha surgido y que no existe presupuesto alguno para poder arreglar el puente, el Gobierno lo a convocado a Ud., futuro Ingeniero en Sistemas de Información de la UTN para que trate de resolver dicho inconveniente, implementado el pseudo código de los autos,

utilizando semáforos, para que impida que se produzcan accidentes fatales y que a su vez circulen los autos en forma

equitativa, ya que en horas pico todos deberían llegar en horario a sus trabajos, tanto las personas que trabajan en Capital como en Provincia.

Tenga en cuenta que deberá de realizar dos pseudo códigos, uno para los autos que provengan de Capital y otro para los de Provincia y que el semáforo cambia de estado (de rojo a verde y viceversa) cada 1 minuto.

Para la resolución considerar únicamente las siguientes funciones: Entrar_puente(); Cruzar_Puente(); y Salir_Puente().

Teoría:

Semáforos: Herramienta genérica de sincronización de procesos. Permite el ordenamiento de las operaciones que realizan los procesos en el tiempo. Es una variable protegida que indica la posibilidad de acceder a un recurso. El valor de los semáforos puede ser accedido y modificado únicamente por las primitivas wait() y signal(). La ejecución de estas primitivas es indivisible o atómica.

Wait: La operación wait disminuye el valor del semáforo. Si el valor se hace negativo, el proceso que ejecuta el wait se bloquea.

Signal: La operación signal incrementa el valor del semáforo. Si el valor no es positivo, se desbloquea un proceso bloqueado por una operación wait.

Resolución:

cant_provincia y cant_capital hacen referencia a la cantidad de autos que circulan desde provincia hacia capital y desde capital hacia provincia, respectivamente. Para que los autos que se encuentren circulando en el sentido que va desde capital hacia provincia deberán aguardar que sem_capital sea igual a 1 (o sea que el semáforo esté en verde), y que cant_provincia sea igual a 0 (es decir que todos los autos que se encontraban circulando por el puente en sentido contrario lo hayan terminado de cruzar). Lo mismo ocurre para el caso contrario.

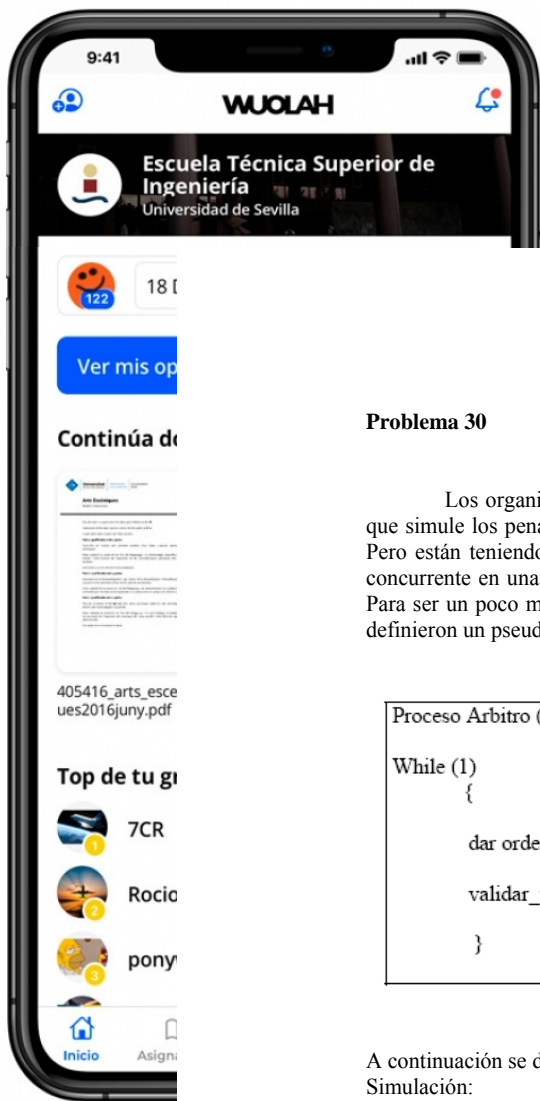
cant_provincia = 0; cant_capital = 0;

cant_autos = 40;

entrar = 1; salir = 1;

semaf_capital = 0 ó 1; semaf_provincia = 0 ó 1; (semaf_capital + semaf_provincia = 1)

<pre> while (true){ while (semaf_capital == 1) && (cant_provincia == 0){ wait (cant_autos); incremento cant_capital ; wait (entrar); Entrar_puente; signal (entrar); Cruzar_puente; wait (salir); Salir_puente; signal (salir); decremento cant_capital; signal (cant_autos); } } </pre>	<pre> while (true){ while (semaf_provincia ==1 && cant_capital == 0){ wait (cant_autos); incremento cant_provincia; wait (entrar); Entrar_puente; signal (entrar); Cruzar_puente; wait (salir); Salir_puente; signal (salir); decremento cant_provincia; signal (cant_autos); } } </pre>
--	--



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.



Problema 30

Los organizadores del próximo mundial de fútbol 5 se encuentran desarrollando una aplicación que simule los penales que se producirán en los partidos que se llevarán a cabo durante el Campeonato. Pero están teniendo inconvenientes, ya que han definido 3 tipos de procesos que se ejecutan en forma concurrente en una Pentium IV con 2 GB de memoria RAM y no logran poder sincronizar los mismos. Para ser un poco más ordenados, se han definido una serie de reglas que se deberán cumplir, y a su vez definieron un pseudo código que es una primera aproximación para poder resolver la simulación

Proceso Arbitro ()	Proceso Jugador ()	Proceso Arquero ()
<pre>While (1) { dar orden () validar_tiro () }</pre>	<pre>While (1) { posicionarse () patear () if (GOL = TRUE) festejar () else lamentarse () }</pre>	<pre>While (1) { ubicarse () atajar () if (GOL = TRUE) lamentarse () else festejar () }</pre>

A continuación se detallan las reglas que se tienen que cumplir para que puedan lograr una correcta Simulación:

- Existen cinco jugadores y un arquero
- Los jugadores no pueden patear si el árbitro no lo indico
- El arquero no puede atajar si el árbitro no da la orden
- Solo se deben usar semáforos, indicando el tipo y los valores iniciales
- Se debe sincronizar el orden en que patean los jugadores
- El árbitro no debe dar el pitido hasta que los dos jugadores no estén posicionados en sus respectivos lugares
- Existe una variable global GOL que indica si el último penal fue pateado fue gol o no
- Una vez que se valide el penal, se le pasara el turno al próximo jugador
- Se tiene una función “Siguiente ()” que cuando es invocada por los procesos retorna el identificador del próximo jugador y la función “Actual ()” que retorna el identificador del jugador actual

Con todos los datos propuestos se le pide que ayude a los organizadores del evento a resolver su problemática. Cabe destacar que la solución propuesta por Ud. debe incluir al pseudo código el uso de SEMÁFOROS ÚNICAMENTE, de forma tal que se puedan desarrollar los penales normalmente y que no quede un jugador sin poder patear un penal en forma permanente.

Los semáforos son una herramienta par sincronizar la ejecución de procesos concurrentes por medio de flags o

Señales que se ejecutan atómicamente e indican la posibilidad de acceder o no al uso de un recurso.

Algunos tipos son:

Semáforos Mutex: Solo pueden tomar valores 1, 0 y negativos.

Semáforos Binarios: Solo pueden tomar valores 0 o negativos.

Semáforos Contadores: Pueden tomar valores enteros no negativos, 0 y negativos.

Solución: Ocho Semáforos Mutex, y un Semáforo Contador

Estas es la solución implementada con un semáforo contador que se utiliza como índice por ejemplo de un vector de (turnos) para sincronizar el turno de cada jugador, y que estos no se salteen, el resto de los semáforos mutex solo son para la sincronización de la acción de cada proceso durante el penal.

SEMAFOROS Mutex

```
JugEnPos = 0,
ArqEnPos = 0
Orden=0
Tiro = 0
Atajada = 0
ValidacionJ = 0,
ValidacionA=0
```

SEMAFORO Contador

```
Turno = 5;
```

PROCESO ARBITRO ()	PROCESO JUGADOR ()	PROCESO ARQUERO ()
<pre>while (1) { wait (ArqEnPos) wait (JugEnPos) darOrden (); signal (orden); wait (Atajada); validarTiro (); signal (ValidaciónA) signal (ValidacionJ); }</pre>	<pre>while (1) { wait (Turno); posicionarse (); signal (JugEnPos); wait (Orden); patear (); signal (Tiro); wait (ValidacionJ); if(GOL = TRUE) Festejar (); else Lamentar(); if (Turno=0) exit; }</pre>	<pre>while (1) { ubicarse (); signal (ArqEnPos); wait(Tiro); Atajar (); signal (Atajada); wait (ValidacionA); if (GOL = TRUE) Lamentar (); else Festejar (); }</pre>

Proceso Arbitro							
ArqEnPos	1	0	0	0	0	0	0
JugEnPos	0	1	0	0	0	0	0
Orden	0	0	1	0	0	0	0
Atajada	0	0	0	1	0	0	0
Validacion A	0	0	0	0	1	0	0
Validacion J	0	0	0	0	0	1	0

Proceso Arquero					
ArqEnPos	1	0	0	0	0
Tiro	0	1	0	0	0
Atajada	0	0	1	0	0
Validacion A	0	0	0	1	0

Proceso Jugador						
Turno	4	4	4	4	4	3
ArqEnPos	0	1	0	0	0	0
Orden	0	0	1	0	0	0
Tiro	0	0	0	1	0	0
Validacion J	0	0	0	0	1	0

Semaforos				Acciones	
Turno	5	4	4	PosicionarA() - Signal(ArqEnPos)	
ArqEnPos	0	1	0	PosicionarJ() -Signal (JugEnPos)	
JugEnPos	0	1	0	Dar Orden() - Signal(Orden)	
Orden	0	1	0	Patear() - Signal(Tiro)	
Tiro	0	1	0	Atajar() - Signal(Atajada)	
Atajada	0	1	0	ValidaTiro() - Signal (ValidacionA) Signal (ValidacionJ)	
ValidacionA y J	0	1	0	Lamentar() - Festejar() - Signal(Turno)	

Ejercicio 35

Enunciado

Con los siguientes datos:

Direcciones: decimales – ref. a su propio espacio de memoria para cada proceso

Cantidad: 8 (0 ...7) – 3 frames p/proceso

Tamaño de pagina: 512 bytes

Semáforos (Estado Inicial) => Mutex		
A	B	C
0	1	0

Frames (Estado Inicial)		Referencias Anteriores	
P1	P2	P1	P2
0	0	0	2
1	2	2	0
2	3	1	3

Y los siguientes procesos:

P1	P2
Read (1400)	Down(B)
Down(A)	Write(2048)
Write(1536)	Up(A)
Up(C)	Write(3070)
Read(1024)	Down(C)
Down(B)	Read(560)
Read(512)	Up(B)
Up(A)	Write(1050)
Write(3000)	Down(A)
Down(C)	Read(2560)
Read(1500)	Up(C)
	Read(4096)

1. ¿Cuántos fallos de página se producen para cada proceso?

a. LRU – Asignación fija – Alcance Global

b. LRU – Asignación fija – Alcance Local

2. ¿Están en deadlock? Justifique

3. ¿Terminan de ejecutar? Justifique

Marco Teórico del ejercicio:

Estamos ante un ejercicio integrado donde se ven mezclados tanto la sincronización de procesos como la administración de memoria de los mismos. En este caso la sincronización es lograda mediante semáforos, recordemos que un semáforo es un mecanismo proveído por un lenguaje de programación, éstos no son más que variables mediante las cuales nuestros procesos P1 y P2 se envían señales. Si recordamos, éstos se manejan con tres operaciones básicas implementadas atómicamente: *una para inicializarlo, una para incrementarlo Up() y otra para decrementarlo Down()*.

Con el uso de estas variables y las mencionadas primitivas, los procesos proceden a utilizar los recursos incrementando y decrementando los semáforos. Cuando se decrementa el valor de un semáforo quedando el mismo por debajo de 0, implica que los recursos controlados por dicho semáforo estaban siendo utilizados por lo tanto el proceso debe esperar a que alguno se desocupe, por ende, se bloquea. Cuando algún proceso incrementa el valor de un semáforo quedando el mismo en estado positivo o nulo, se desencola a alguno de los procesos que estaban a la espera del recurso.

Se destacan varios tipos de semáforos: los *binarios* que solo pueden tomar valores 0 y 1, los *contadores* que pueden tomar valores enteros positivos, nulos y negativos, y los *mutex* que solo pueden tomar valores 1, 0 y negativos. En el ejercicio haremos uso de los últimos.

Por otra parte mencionamos que en este ejercicio tendremos que evaluar la administración de memoria además de la sincronización de los procesos mediante los semáforos ya explicados.

Si leemos algunos de los datos que nos menciona el ejercicio veremos que nos habla de frames y de páginas, por lo cual podemos deducir que estamos hablando de un sistema con memoriapaginada.

La paginación es una técnica de gestión de memoria que permite asignar memoria en forma no contigua. Se procede a dividir el espacio de direccionamiento de cada programa que se pretende ejecutar en porciones iguales llamados páginas, e igualmente se divide la memoria física en porciones del mismo tamaño llamados frames. Recordemos que cuando se trabaja con esta metodología, existe una tabla de páginas por cada proceso activo, y un mapa de frames, único para todo el sistema.

Cada proceso tendrá una cantidad de frames asignados donde podrá “encajar” sus páginas, pero he aquí una distinción, si se trata de Paginación Pura, el proceso deberá cargar todas sus páginas en frames no contiguos, pero todas al fin. En contraposición, existe la Paginación por Demanda, la cual no requiere que se cargue todo un proceso en memoria central.

Estamos hablando en este caso, de Memoria Virtual, la cual da la impresión de que el sistema tiene más memoria central haciendo posible que existan una cantidad total de memoria asignada por programa mayor a la memoria central.

Dada esta circunstancia, existe la posibilidad que ciertas páginas a las cuales el proceso hace referencia no se encuentren en memoria central en dicho momento, en tal circunstancia, el hardware de mapeo de memoria genera una interrupción por Page Fault, por lo cual el SO realiza el trabajo de cargar las páginas requeridas desde el almacenamiento secundario y actualiza las entradas de la tabla de páginas del proceso.

Cuando se traen páginas desde memoria secundaria, suele suceder que hay que escoger una víctima a ser reemplazada en el caso de no tener frames libres en memoria física, es decir, alguna de las páginas asignadas en frames de memoria central, deberá ser reemplazada por la página traída desde memoria secundaria. Para elegir a esta víctima existen diferentes algoritmos que citamos a continuación: Óptimo, FIFO, LFU, Reloj y LRU.

Nuestro ejercicio hace referencia al LRU o Least Recently Used, el cual consta de escoger para sustituir las páginas que no se han utilizado durante el mayor período de tiempo, el cual se basa en el principio de que si una página ha sido usada será requerida nuevamente muy pronto, por eso reemplaza la que menos se usó recientemente.

Hay que tener en cuenta que las políticas de reemplazo se realizan sobre el conjunto residente de un proceso, esto es, las páginas que actualmente se ubican en memoria. Este conjunto residente puede ser fijo (el número de páginas siempre es el mismo) o variable (puede tener más o menos páginas en memoria). Por otro lado, el alcance del reemplazo puede ser local (se reemplazan las páginas del proceso que produjo el fallo) o global (se consideran todas las páginas residentes en memoria). Por lo cual tenemos las siguientes posibilidades:

- Asignación fija, alcance local: Se reemplazan alguna de las páginas del proceso que produjo el fallo.
- Asignación variable, alcance local: Se consideran las páginas del proceso que produjo el fallo, pero se utilizan diferentes criterios (como el PFF) para determinar si le quito un frame al proceso o le sumo uno más (le agrego una página mas en memoria).
- Asignación fija, alcance global: Imposible de implementar. Como hemos visto, si la asignación es fija el número de páginas del proceso no puede cambiar. Si consideramos todas las páginas de memoria, puede pasar que al proceso se le sume o se le quite un frame más, lo cual contradice lo anterior.
- Asignación variable, alcance global: Se toman en cuenta todas las páginas residentes en memoria (no solo las del proceso que produjo el Page Fault) y se reemplaza alguna según el algoritmo utilizado. El número de frames asignados al proceso puede crecer o puede disminuir

¿Clases de Inglés telefónicas con Profesores Nativos dónde y cuándo quieras?

25 minutos de conversación en inglés por teléfono desde 3,25 €/sesión.
Conoce más en www.ringteacher.com

ringteacher

Resolución práctica:

Primero veamos como se corresponderían las páginas de memoria con las direcciones referenciadas, si las mismas tienen un tamaño dado de 512Kb

0	0 a 511
1	512 a 1023
2	1024 a 1535
3	1536 a 2047
4	2048 a 2559
5	2560 a 3071
6	3072 a 3583
7	3584 a 4095

1.a) Imposible de implementar. Ver último párrafo de *Marco Teórico*

1.b)

P1	Referencias	1400	1536	1024	512	3000	1500
	Páginas	2	3	2	1	5	2
Frames (Estado Inicial)	0	0	3	3	3	3	3
	1	1	1	1	1	5	5
	2	2	2	2	2	2	2
CANTIDAD DE PAGE FAULT = 2							

P2	Referencias	2048	3070	560	1050	2560	4096
	Páginas	4	5	1	2	5	8?
Frames (Estado Inicial)	0	0	5	5	5	5	Direccion no Válida
	2	4	4	4	2	2	
	3	3	3	1	1	1	
CANTIDAD DE PAGE FAULT = 4							

2) Como podemos observar en el cuadro de la página siguiente, los procesos no están en Deadlock, ya que pueden ejecutar normalmente.

3) El proceso P1 termina su ejecución, no así el proceso P2 ya que en su última referencia a memoria, intenta acceder a una dirección de memoria no válida, por lo tanto, es abortado.

Proceso	Accion	Semáforos			Resultado
		A	B	C	
P1	Read(1400)	0	1	0	
P1	Down(A)	-1	1	0	P1 se bloquea en A
P2	Down(B)	-1	0	0	
P2	Write(2048)	-1	0	0	
P2	Up(A)	0	0	0	P1 se desbloquea en A
P2	Write(3700)	0	0	0	
P2	Down(C)	0	0	-1	P2 se bloquea en C
P1	Write(1536)	0	0	-1	
P1	Up(C)	0	0	0	P2 se desbloquea en C
P1	Read(4024)	0	0	0	
P1	Down(B)	0	-1	0	P1 se bloquea en B
P2	Read(560)	0	-1	0	
P2	Up(B)	0	0	0	P1 se desbloquea en B
P2	Write(1050)	0	0	0	
P2	Down(A)	-1	0	0	P2 se bloquea en A
P1	Read(512)	-1	0	0	
P1	Up(A)	0	0	0	P2 se desbloquea en A
P1	Write(3000)	0	0	0	
P1	Down(C)	0	0	-1	P1 se bloquea en C
P2	Read(2560)	0	0	-1	
P2	Up(C)	0	0	0	P1 se desbloquea en C
P2	Read(4096)	0	0	0	Página Inexistente
P1	Read(1500)	0	0	0	Fin P1

Ejercicio 37.

Enunciado:

Sea un sistema operativo que utiliza memoria virtual con reemplazo local y el algoritmo LRU para la elección de la víctima. Las direcciones de memoria son de 32 bits con 12 bits de offset. Sabiendo que el proceso posee dos frames (20 y 50), el primero para el código y el segundo para los datos, y que se va a procesar el siguiente código:

```
int mat [10] [1024];
for (i = 0; i < 1024; i++)
    for (j = 0; j < 10; j++)
        mat [j] [i] = 1;
```

Se pide:

- Calcular la cantidad de fallos de página que produce el proceso.
- Traducción de las siguientes direcciones lógicas: Mat [1] [5]; Mat [5] [5] y Mat [7] [5] a sus respectivas direcciones físicas.

Nota: Para la resolución de este ejercicio tenga en cuenta que el tamaño de un entero es igual a 4 bytes y que no existen errores de compilación en el código.

Teoría:

Memoria Virtual: El enfoque de memoria virtual utiliza el S.O. para producir la ilusión de una memoria sumamente grande. En memoria virtual se utiliza ya sea la técnica de paginación, de segmentación o una combinación entre ambas. En cualquier de los casos, a las tablas de segmentos o de paginación de los procesos se les suele incorporar un bit de presencia y un bit de modificación.

Cuando se hace referencia a una página (o segmento) que no tiene frame asignado, es decir, la página no está cargada en memoria, se produce un PAGE FAULT (fallo de página). Es en este momento cuando la página es traída de disco a memoria y el bit de presencia de la entrada a esa página es modificado. Lo mismo sucede con el bit de modificación.

Frame: Para la administración de memoria paginada se divide el espacio de direccionamiento de cada programa que se pretende ejecutar en porciones iguales llamados páginas, e igualmente se divide la memoria física en porciones del mismo tamaño llamados bloques, marcos o frames.

Paginación: el proceso se divide en páginas y la memoria en frames; tanto los frames como las páginas tienen el mismo tamaño y suelen ser potencias de 2. A cada proceso se le adhiere una tabla de páginas donde la entrada de cada una de las páginas contiene el número de frame en el cual están ubicadas.

Segmentación: es similar a la paginación, con la diferencia de que el programa se divide en segmentos que suelen tener diferentes tamaños. Los procesos tienen también una tabla de segmentos, y cada entrada de segmento dice la dirección base y el tamaño de dicho segmento. Si el desplazamiento referenciado en la instrucción es mayor al tamaño del segmento, la dirección se toma como inválida; caso contrario se suma a la dirección base del segmento y se obtiene la dirección real.

Básicamente:

Direcciones lógicas: Emitidos por el proceso cuando direcciona su espacio de memoria. Su valor más bien, no refleja la organización de la memoria física de la computadora. (#Página / #Offset)

Direcciones físicas: Proviene de adaptar la dirección lógica emitida por el programa en ejecución al espacio de direccionamiento real de la computadora. (#Frame / #Offset)

Teniendo en cuenta este concepto podemos deducir que el tamaño de cada página (o el tamaño del frame en memoria) estará dado por el tamaño del desplazamiento; es decir, si el desplazamiento ocupa 8 bits, entonces el tamaño de cada página será de $2^8 = 256$ bytes.

Políticas de reemplazo: se realizan sobre el conjunto residente de un proceso, esto es, las páginas que actualmente se ubican en memoria. Este conjunto residente puede ser fijo (el número de páginas siempre es el mismo) o variable (puede tener más o menos páginas en memoria). Por otro lado, el alcance del reemplazo puede ser local (se reemplazan las páginas del proceso que produjo el fallo) o global (se consideran todas las páginas residentes en memoria).

Algoritmo LRU (Last Recently Used): Escoge las páginas que mas tiempo han permanecido en memoria.

Resolución:

Como la matriz se guarda primero por filas y después por columnas, entonces se toma la columna 0 y la recorre completa.

Si cada posición de la memoria ocupa 4 bytes $\Rightarrow 4 \times 1024$ es lo que ocupa cada fila \Rightarrow en cada página entra una fila.

Como el código recorre una por columna y en memoria las guarda por filas \Rightarrow los fallos de página son de $1024 \times 10 = 10240$ ya que por cada posición que modifico de la matriz me va a producir un PF.

Cantidad de páginas direccionables: $2^{20} = 1048576$

Tamaño de cada página: $2^{12} = 4096$

Traducción de direcciones lógicas a físicas:

dir fis = (nro de marco $\times 2^m$) \times t + desp

t = tamaño en bytes de cada posición = 4 bytes

desp = posición del índice j \times t

m = cant de bits menos significativos = 12

#pagina | #offset

Mat[1][5] = 1 x 1024 bytes x 4 bytes + 5 x 4 bytes = 4116 es la dirección lógica. (1 | 20)

Mat[5][5] = 5 x 1024 bytes x 4 bytes + 5 x 4 bytes = 20500 es la dirección lógica. (5 | 20)

Mat[7][5] = 7 x 1024 bytes x 4 bytes + 5 x 4 bytes = 28692 es la dirección lógica. (7 | 20)

La dirección física es la misma para las 3 debido a que sólo se posee un frame para los datos (50) y el desplazamiento es el mismo.

#frame | #offset 50 | 20

50 x 1024 bytes x 4 bytes + 5 x 4 bytes = 204820

Ejercicio 39

Enunciado:

Considere una máquina con direcciones de 32 bits, con 10 bits de offset y dos procesos, cuyas referencias a memoria se detallan a continuación:

Proceso 1: 4242 - 8749 - 5555 - 324574 - 104 - 56422 (decimal)

Proceso 2: 85547 - 2100 - 4571 - 65537 - 321 - 10247 - 47888 (decimal)

El proceso 1 tiene asignados 3 frames de memoria (10, 20, 50) y el proceso 2 cuenta con dos frames (30, 40), los mismos se encuentran inicialmente vacíos. El Sistema Operativo implementa como algoritmo de elección de la víctima el LRU con asignación fija y alcance local. Se sabe también que el sistema posee un disco rígido de 80 Gb, cuya velocidad de rotación es de 6400 RPM. Asumiendo que consideramos que el disco rígido que se posee en el sistema es lento, ¿sería conveniente reemplazarlo por uno más rápido?. Justifique ampliamente su respuesta.

Resolución:

Proceso 1

4242 = 00000000000000000000100//0010010010

2. No se soluciona nada reasignando frames o agregando memoria, cada pedido de alguna dirección de memoria te va a generar un page fault pase lo que pase (sería lo mismo si cada proceso tuviera un frame solo).
3. Si se colocaría un disco más rápido ayudaría a bajar el tiempo de lectura de la página desde el disco.

Problema 41

Se tiene una máquina con direcciones de 16 bits, con 10 bits de offset, y un proceso P1 cuyas referencias a memoria son las siguientes: 535 – 3451 – 7548 – 487 – 2087 – 3147 – 9843 – 3521 – 6841 – 8544 – 6024 – 987 (decimal)

El proceso P1 tiene asignados tres frames de memoria (0, 1 y 2), los mismos se encuentran inicialmente vacíos.

A su vez se posee un disco rígido con las siguientes características: 3 platos, formando un total de 6 cabezas, 100 sectores por pista, un total de 500 cilindros y sectores de 512 bytes. Suponga que las direcciones lógicas de las referencias a memoria son las mismas que las referencias a disco. Ejemplo: la dirección lógica de memoria 100 se encuentra en la dirección lógica 100 del disco y siempre se mantiene esa relación.

Se utiliza LRU con reemplazo local como política de elección de la víctima y el LOOK como política de planificación de disco.

Sabiendo que la cabeza se encuentra en la dirección lógica 50 descendiendo, al respecto se pide:

- El estado de la memoria en cada instante de tiempo, indicando en cada uno de ellos, si fuera necesario, los fallos de páginas correspondientes y la traducción de las direcciones lógicas a físicas de las referencias del proceso P1.
- El tiempo promedio de atención de los pedidos a disco y el orden en que fueron atendidos.

Aplicación de la teoría para la resolución del ejercicio:

Algoritmo LRU- :

Objetivo:

- Reemplaza la página que menos ha sido referenciada recientemente.

Funcionamiento:

- Se basa en el principio en que si una página ha sido usada, será requerida nuevamente muy pronto, entonces las sustitutas a ser reemplazadas, son las que menos se ha usado recientemente.
- Asocia a cada página el instante de tiempo que se usó por última vez.

Algoritmo de LOOK-UP:

La cabeza lectora comienza en un extremo del disco, y se mueve en dirección hacia el otro extremo del disco, sirviendo los requerimientos una vez servido el último requerimiento en esa dirección sin llegar al extremo se dirige hacia en dirección al otro extremo.

Se necesita un software con un BIT que indica la dirección en que se está leyendo y que los requerimientos estén ordenados.

La diferencia de este Algoritmo con respecto al resto es que mira primero el requerimiento próximo en esa dirección si no hay cambia de invierte sentido de dirección sin llegar al extremo.

Direcciones: 16bits

nro de pág. 6 bits \rightarrow Cant de Pág.: $2^6 \rightarrow 64$ páginas

Offset: 10 bits \rightarrow Tamaño de Pág.: $2^{10} \rightarrow 1024$ Bytes

Dirección Lógica = (cilindro, cabeza, sector)

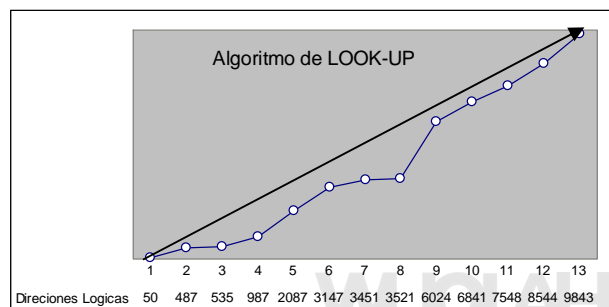
Dirección Lógica = (0 .. 499 , 0 .. 6 , 1.. 100)

Datos Disco Rígido:

Platos = 3 Cabezas = Platos x lado

Cabezas = 6 (0 .. 5)

Sectores = 100/Pista



Cilindros = 500 (0 ..499)

Sectores/Cilindro = Sec/Pista * pistas/cilindro

Resolución:

La memoria principal queda dividida en pequeñas porciones idénticas de tamaño fijo llamados **page frames** (marcos de paginas). Cada proceso queda dividido en **páginas** del mismo tamaño que los page frames. El sistema operativo carga los procesos en Memoria Principal distribuyéndolos en páginas preferentemente continuas. Se lleva una lista de paginas libres y una lista de paginas por proceso que dice en que page frame esta cada página del proceso, o que no esta en ningún page frame para las paginas no cargadas. Estas listas son las que permiten distribuir a los procesos en páginas discontinuas.

Si se usa un tamaño de página que sea potencia de 2 la dirección relativa será idéntica a la lógica (nro_de_pag: desplazamiento) y para convertirla a dirección física se siguen estos pasos:

- Se extrae el nro. de página de los bits más a la izquierda.
- Se usa el nro. de página como índice en la tabla de páginas del proceso, así se obtiene el nro. de page frame.
- Este nro. de page frame se multiplica por el tamaño de page frame y se le suma el desplazamiento (la parte derecha de la dirección lógica).

Orden		16 Bits																	Frame	Direccion
Atencion	Referencia	P a g i n a								O f f s e t								P a g i n a	Asignado	Fisica Disco
1	487	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	1	0	0	(0,4,88)
2	535	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	1	0	0	(0,5,36)
3	987	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	1	1	1	(1,3,88)
4	2087	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	1	2	1	(3,2,88)
5	3147	0	0	0	0	1	1	0	0	0	1	0	0	1	0	1	1	3	2	(5,1,48)
6	3451	0	0	0	0	1	1	0	1	0	1	1	1	1	0	1	1	3	1	(5,4,52)
7	3521	0	0	0	0	1	1	0	1	1	1	1	0	0	0	0	0	3	2	(5,5,22)
8	6024	0	0	0	1	0	1	1	1	1	1	0	0	0	1	0	0	5	2	(10,0,25)
9	6841	0	0	0	1	1	0	1	0	1	0	1	1	1	1	0	0	6	1	(11,2,42)
10	7548	0	0	0	1	1	1	0	1	0	1	1	1	1	1	1	0	7	2	(13,3,49)
11	8544	0	0	1	0	0	0	0	1	0	1	1	0	0	0	0	0	8	0	(14,1,45)
12	9843	0	0	1	0	0	1	1	0	0	1	1	1	0	0	1	1	9	0	(16,2,44)

Proceso (1)	Referencias	535	3451	7548	487	2087	3147	9843	3521	6841	8544	6024	987
	Páginas	0	3	7	0	2	3	9	3	6	8	5	0
Frames	0	0	0	0	0	0	0	9	9	9	8	8	8
inicial	1		3	3	3	2	2	2	2	6	6	6	0
	2			7	7	7	3	3	3	3	3	5	5
9 page fault		1	2	3		4		5		6	7	8	9

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

(*) El estado final de la Memoria .

Los tiempos de atención no se pueden calcular, ya que no tenemos los datos de tiempos de atención ni velocidad del disco rígido.(RPM).

Direccion	Logica	=	50			
50	600		50	100		
50	0		50	0		
	Cilindro		Sector	Cabeza		
Direccion Fisica = (0 , 0 , 51)						

Direccion	Logica	=	3451			
3451	600		451	100		
451	5		51	4		
	Cilindro		Sector	Cabeza		
Direccion Fisica = (5 , 4 , 52)						

Direccion	Logica	=	7548			
7548	600		348	100		
348	13		48	3		
	Cilindro		Sector	Cabeza		
Direccion Fisica = (13 , 3 , 49)						

Direccion	Logica	=	487			
487	600		487	100		
487	0		87	4		
	Cilindro		Sector	Cabeza		
Direccion Fisica = (0 , 4 , 88)						

Direccion	Logica	=	3147			
3147	600		147	100		
147	5		47	1		
	Cilindro		Sector	Cabeza		
Direccion Fisica = (5 , 1 , 48)						

Direccion	Logica	=	2087			
2087	600		287	100		
287	3		87	2		
	Cilindro		Sector	Cabeza		
Direccion Fisica = (3 , 2 , 88)						

Direccion	Logica	=	9843			
9843	600		243	100		
243	16		43	2		
	Cilindro		Sector	Cabeza		
Direccion Fisica = (16 , 2 , 44)						

Direccion	Logica	=	3521			
3521	600		521	100		
521	5		21	5		
	Cilindro		Sector	Cabeza		
Direccion Fisica = (5 , 5 , 22)						

Direccion	Logica	=	6841			
6841	600		241	100		
241	11		41	2		
	Cilindro		Sector	Cabeza		
Direccion Fisica = (11 , 2 , 42)						

Direccion	Logica	=	8544			
8544	600		144	100		
144	14		44	1		
	Cilindro		Sector	Cabeza		
Direccion Fisica = (14 , 1 , 45)						

Direccion	Logica	=	6024			
6024	600		24	100		
24	10		24	0		
	Cilindro		Sector	Cabeza		
Direccion Fisica = (10 , 0 , 25)						

Direccion	Logica	=	987			
987	600		387	100		
387	1		87	3		
	Cilindro		Sector	Cabeza		
Direccion Fisica = (1 , 3 , 88)						

Direccion	Logica	=	535			
535	600		535	100		
535	0		35	5		
	Cilindro		Sector	Cabeza		
Direccion Fisica = (0 , 5 , 36)						

¿Clases de Inglés telefónicas con Profesores Nativos dónde y cuándo quieras?

25 minutos de conversación en inglés por teléfono desde 3,25 €/sesión.
Conoce más en www.ringteacher.com

ringteacher

Ejercicio 42

Enunciado

Un bloque de 128 KB de memoria utiliza el algoritmo de asignación de memoria Buddy System (o Sistema amigable). Después de un tiempo el bloque queda de la siguiente manera, donde los bloques de memoria identificados con X están asignados y el resto están libres.

Mem	X	X	X			X	X		X		X	X	X	X		
Dir	0	8	16	24	32	40	48	56	64	72	80	88	96	104	112	120

Luego, estos bloques son liberados de la siguiente manera:

Dirección	Tamaño
64K	8KB
0K	24KB
96K	16KB

Se pide que muestre como son los estados intermedios y el final del bloque de memoria.

Marco Teórico del Ejercicio:

En este ejercicio veremos como funciona la técnica de administración de memoria llamada Buddy System. Es un algoritmo de manejo de memoria que hace que sea muy rápida la unión de huecos adyacentes cuando un proceso termina o es llevado a disco. Posee un hardware sencillo y hay poco overhead en las funciones de liberar o asignar el espacio.

Por otro lado, hay mal aprovechamiento de la memoria por los grandes fragmentos que aparecen en la misma.

Para comprender mejor esta técnica lo más simple y comprensible será verlo mismo sobre el ejercicio.

Resolución Práctica:

Tenemos un bloque de memoria de la siguiente manera:

Estado	X	X	X			X	X		X		X	X	X	X		
Dir (K)	0	8	16	24	32	40	48	56	64	72	80	88	96	104	112	120

Según dice nuestro ejercicio, se liberan inicialmente de la dirección de memoria 64K, 8Kb de la misma, es decir, que se libera ese bloque, ya que como vemos, cada bloque tiene 8Kb, quedando de la siguiente manera:

Estado	X	X	X			X	X				X	X	X	X		
Dir (K)	0	8	16	24	32	40	48	56	64	72	80	88	96	104	112	120

Como vemos se quedarán libres varios bloques contiguos, pero como podemos asociarlos solo formando bloques con tamaños que sean potencias de 2, no podemos junta los bloques 56, 64 y 72, por tanto, asociamos dos bloques, 64 y 72 pasando a tener un bloque de mayor tamaño en la dirección 64:

Estado	X	X	X			X	X				X	X	X	X		
Dir (K)	0	8	16	24	32	40	48	56	64	72	80	88	96	104	112	120

Posteriormente vemos que se libera en la dirección 0K un bloque de 24Kb, por tanto se asociarán con el bloque libre de la dirección 24K, quedando de la siguiente forma::

Estado						X	X				X	X	X	X		
Dir (K)						32	40	48	56	64	80	88	96	104	112	120

Por último, se liberarían de la dirección 96K un bloque de 16Kb, el cual puede asociarse a los bloques libres de las direcciones 112K y 120K, quedando finalmente la memoria de la siguiente forma:

Estado						X	X				X	X				
Dir (K)						32	40	48	56	64	80	88			96	

Ejercicio 45.

Enunciado:

El proceso X tiene asignados 4 frames de memoria y comenzó a ejecutar en $t = 0$. En el momento $t = 10$ los frames se encuentran de la siguiente forma:

Frame	Referenciado en t
2	9
5	4
1	6
3	8

Si se utiliza LRU como política de reemplazo, y sabiendo que los pedidos restantes se realizaron en el siguiente orden:

4 – 5 – 2 – 1 – 4 – 6 – 4 – 1 – 2

Indicar la cantidad de page faults que ocurrieron, señalando los mismos al igual que el estado final de memoria.

Teoría:

Frame: Para la administración de memoria paginada se divide el espacio de direccionamiento de cada programa que se pretende ejecutar en porciones iguales llamados páginas, e igualmente se divide la memoria física en porciones del mismo tamaño llamados bloques, marcos o frames.

Algoritmo LRU (Last Recently Used): Escoge las páginas que mas tiempo han permanecido en memoria.

Fallo de página: Cuando se hace referencia a una página (o segmento) que no tiene frame asignado, es decir, la página no está cargada en memoria, se produce un PAGE FAULT (fallo de página). Es en este momento cuando la página es traída de disco a memoria.

Resolución:

Pedidos: 4 – 5 – 2 – 1 – 4 – 6 – 4 – 1 – 2

4

en t=10

Frame	Ref en t
2	9
5	4
1	6
3	8

PF

4

en t=11

Frame	Ref en t
2	9
4	11
1	6
3	8

PF

5

en t=12

Frame	Ref en t
2	9
4	11
5	12
3	8

PF

2

en t=13

Frame	Ref en t
2	13
4	11
5	12
3	8

1

en t=14

Frame	Ref en t
2	13
4	11
5	12
1	14

PF

4

en t=15

Frame	Ref en t
2	13
4	15
5	12
1	14

PF

6

en t=16

Frame	Ref en t
2	13
4	15
6	16
1	14

PF

4

en t=17

Frame	Ref en t
2	13
4	17
6	16
1	14

1 en t=18		2 en t=19	
Frame	Ref en t	Frame	Ref en t
2	13	2	19
4	17	4	17
6	16	6	16
1	18	1	18

Se produjeron 4 fallos de página (Page Faults – PF) en los pedidos restantes.
El estado final de memoria es el siguiente:

en t=19

Frame	Ref en t
2	19
4	17
6	16
1	18

Ejercicio 46

Enunciado:

Considere una máquina con direcciones de 16 bits, con 10 bits de offset y un proceso P1 cuyas referencias a memoria son las siguientes: 535 - 3451 - 7548 - 487 - 2087 - 147 - 9843 - 521 - 65535 - 8544 - 1024 - 65530 (decimal).

El proceso P1 tiene asignados 3 frames de memoria (0, 1, 2), los mismos se encuentran inicialmente vacíos. Si se utiliza LRU con reemplazo local como política de elección de la víctima, se pide:

- a) El estado de la memoria en cada instante de tiempo indicando en cada uno de ellos, si fuera necesario, los fallos de página correspondientes.
- b) La traducción de las direcciones lógicas a físicas de las referencias del proceso P1.
- c) Enunciar si cambiaría su respuesta a) en el caso de utilizar el algoritmo LRU con asignación fija y alcance global.

Teoría:

Para la administración de memoria paginada se divide el espacio de direccionamiento de cada programa que se pretende ejecutar en porciones iguales llamados páginas, e igualmente se divide la memoria física en porciones del mismo tamaño llamados bloques, marcos o frames.

Entonces, proporcionando una posibilidad adecuada de mapeo del hardware se puede colocar una página en cualquier frame libre existente. Las páginas permanecen lógicamente contiguas (es decir, para el programa usuario) aunque los bloques correspondientes en la memoria física no necesariamente lo sean.

Para que el hardware realice el mapeo del espacio de direcciones lógicas a la memoria física, debe haber un registro para cada página por separado que establezca la correspondencia lógico-física. Generalmente se llama tablas de página o tablas de mapas de página MPT. Dichos mapas pueden ser registros especiales de hardware o una porción reservada de la memoria central que se destina para ese fin.

Ya que es posible reubicar cada página por separado, no se necesita que el área que ocupa un trabajo esté completamente contigua en memoria. El único caso de contigüedad deben ser las posiciones en una sola página.

Si el tamaño del espacio de direcciones es 2^a , y el tamaño de la página es 2^n unidades de direccionamiento (bits), el orden superior $a-n$ bits de direcciones lógicas estarán designadas al número de páginas, y el orden inferior n al desplazamiento (offset) de la página. Por ejemplo, para este ejercicio $a=16$ y $n=10$ bits, se tendrá una longitud de página de 1K (2^{10}) y una cantidad de 64 páginas (2^6).

El algoritmo LAST RECENTLY USED (LRU) elige la página que no ha sido referenciada por mas tiempo. Este algoritmo es casi óptimo, pero también es muy caro porque implica: asociar a cada página un campo que diga 'cuando fue la última vez que fue accedida' o mantener una pila de referencias a páginas.

Page Faults (fallo de página). Se produce cuando se intenta acceder a una página que no se encuentra cargada en memoria central.



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.



Resolución:

Para obtener el nro. de página que se debe cargar en memoria, se divide la dirección lógica por el tamaño de página, y se toma la parte entera:

Dir. Lógica	Nro. De Página
535	0
3451	3
7548	7
487	0
2087	2
147	0
9843	9
521	0
65535	63
8544	8
1024	1
65530	63

Para el reemplazo de páginas usamos el algoritmo LRU:

Pág.	0	3	7	0	2	0	9	0	63	8	1	63
F.1	0	0	0	0	0	0	0	0	0	0	1	1
F.2		3	3	3	2	2	2	2	63	63	63	63
F.3			7	7	7	7	9	9	9	8	8	8
P.F.	*	*	*		*		*		*	*	*	(1)

(1) Estado final de la Memoria.

Sustitución global: Asignación dinámica de frames.

- El proceso puede tomar un frame a sustituir de cualquier lugar de memoria central aunque el frame esté asociado a otro proceso.
- Puede suceder que la cantidad de frames de un proceso crezcan o disminuyan. Es posible que un proceso seleccione solamente frames de otros procesos, incrementando el número de frames que tiene asignado (suponiendo que otros procesos no elijan sus frames para reemplazarlo).
- Los procesos no son responsables ni pueden controlar su propio page fault rate.
- El conjunto de páginas en memoria para un proceso no depende solamente del comportamiento de ese proceso, sino también del de otros.

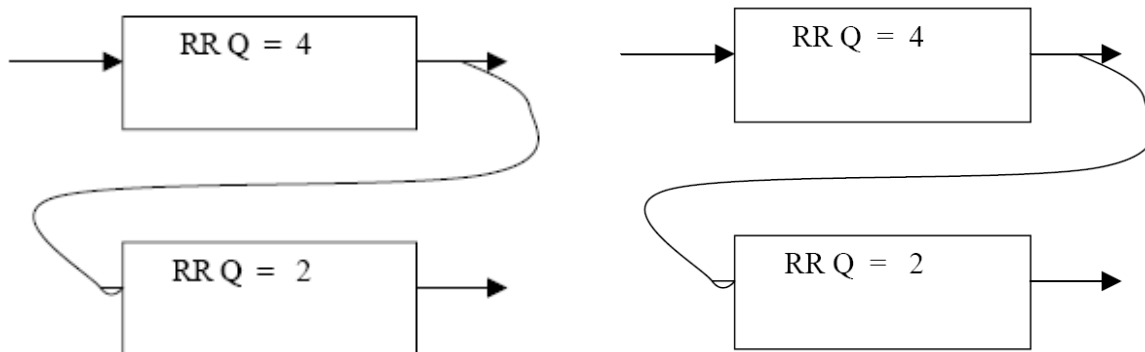
Por lo anteriormente descrito, utilizar el algoritmo LRU con sustitución global, podría disminuir el número de page faults, no siendo el caso para el set de direcciones lógicas dado.

Problema 51

Las correcciones en las fechas de final son un asunto que preocupa a la cátedra de Sistemas Operativos. En algunas ocasiones se presentan situaciones impensadas y se requiere su ayuda como futuro Ingeniero de Sistemas de Información para relevarlas. Citaremos un ejemplo ocurrido en una fecha en Medrano 951 – Aula 402 – 15:30 hs.

Es una fecha complicada. Los integrantes de la cátedra deben repartirse entre las entregas del TP y la corrección de finales. Es el momento de corregir y solo hay 3 ayudantes (Inter., maty y rodri) y dos profesores (Pepe y Jose Luis)

Para corregir se debe utilizar una lapicera roja, pero entre los 5 solo tienen una, por lo tanto deciden compartirla de la siguiente manera:



Pero en distintos instantes, ayudantes y profesores deben ir a buscar mas finales para corregir. Para ello, deben solicitárselos al Jefe de Cátedra, acto que les implica tiempo relativo a corregir 3 finales mas.

- Q es la cantidad de finales que pueden corregir antes que deban dejar de usar la lapicera
- La planificación elegida para compartir la lapicera es una cola multinivel con realimentación y prioridades PREEMPTIVE entre los niveles
- Cuando una persona deja la lapicera forzado por haber cumplido su Quantum en la cola de mayor prioridad, pasa a la cola de menor prioridad.
- En caso que una persona tenga que volver a tomar la lapicera después de ir a tomar café o de obtener mas finales, se sitúa en la cola de mayor prioridad debido a que se encuentra mas descansado que los demás.

El Jefe de cátedra solo atiende de a uno por vez, según el orden en el que llegan. Como corregir es una tarea agotadora, entre ellos deciden ponerse de acuerdo en turnarse para ir a tomar un café, yendo a lo

- Q es la cantidad de finales que pueden corregir antes que deban dejar de usar la lapicera
- La planificación elegida para compartir la lapicera es una cola multinivel con realimentación y prioridades PREEMPTIVE entre los niveles
- Cuando una persona deja la lapicera forzado por haber cumplido su Quantum en la cola de mayor prioridad, pasa a la cola de menor prioridad.
- En caso que una persona tenga que volver a tomar la lapicera después de ir a tomar café o de obtener mas finales, se sitúa en la cola de mayor prioridad debido a que se encuentra mas descansado que los demás.

RR Q = 4 (Algoritmos Round Robin con Quantum = 4)

RR Q = 2 (Algoritmos Round Robin con Quantum = 2)

El Jefe de cátedra solo atiende de a uno por vez, según el orden en el que llegan. Como corregir es una tarea agotadora, entre ellos deciden ponerse de acuerdo en turnarse para ir a tomar un café, yendo a lo sumo de a dos por vez, en el orden en que lo estipular sin prioridades. A fines estadísticos, se determina que el tiempo promedio para corregir un final es de 8 minutos, por lo tanto es importante relevar también el tiempo que tarda cada uno en ir a tomar un café a fin de saber cuantos finales se podrían haber corregido en ese tiempo

Nombre	Hora Llegada	Cant. Finales a corregir	Solicitar mas finales	Cant. Finales a corregir	Tomar Café	Cant. Finales a corregir
Pepe	15:30	5	X	3	32 min	6
Jose Luis	15:54	3	X	4	24 min	3
Inter.	15:46	4	X	8	16 min	4
Matias	15:38	2	X	5	40 min	3
Rodrigo	16:02	4	X	8	24 min	6

Teniendo en cuenta los datos mencionados, se le pide que realice un Diagrama de Gantt Informando el Turn Around Time de cada profesor / ayudante

Ejercicio 56

Enunciado

Dado un servidor que posee cinco discos rígidos de 23 GB cada uno, formando con ellos un RAID tipo 5, el que se posee el File System Black Thunder. La única restricción de este file system basado en Inodos (con dos punteros directos, uno indirecto y otro doble indirecto) es que puede direccionar como máximo 10.000.000 de bloques de datos de 12 Kb cada uno.

- ¿Cuál es el tamaño máximo de una partición de este file system en el servidor?
- ¿Cuál es el tamaño máximo teórico de un archivo?
- ¿Cuál es el tamaño máximo teórico que contiene un directorio?

Ejercicio 57

Enunciado

Dado un servidor que posee seis discos rígidos de 40 GB cada uno, formando un RAID 1, en el que se posee el File System Helter-Dehaes_Skelter. La única restricción de este file system basado en i-nodos (con tres punteros directos, uno de in dirección simple, uno de in dirección doble y uno de in dirección triple) es que se puede direccionar como máximo 20 millones de bloques de datos de 24 KB cada uno. Hallar:

- Tamaño máximo físico de una partición de este file system
- Tamaño máximo teórico de un archivo

Marco Teórico del ejercicio:

Ambos ejercicios nos mencionan que se trata de File System basados en inodos. UNIX maneja todos los archivos mediante inodos.

Un inodo(nodo índice o de información) es la estructura que contiene la información clave para manejar un archivo. Contiene atributos, permisos y otra información de control del archivo, incluyendo direcciones. Muchos archivos pueden ser asociados a un inodo, pero un inodo activo es asociado exactamente con un solo archivo y cada archivo es controlado por un inodo.

Podemos ver que el tamaño máximo de un archivo estará dado por la capacidad de direccionamiento que tenga el inodo, ya que los mismos, son de tamaño fijo. En cambio, el tamaño máximo de la partición del File System estará relacionado con la capacidad máxima que yo tenga de almacenamiento.

En UNIX la asignación es dinámica, en base a bloques no necesariamente contiguos. Se le sigue la pista a los archivos mediante el método indexado, con la primera parte del índice en el inodo. Allí hay un vector de 13 elementos de 3 bytes c/u. Los 10 primeros contienen la dirección de los 10 primeros bloques del archivo. Si el archivo ocupa más de 10 bloques, uno o más niveles de indirección son usados.

El elemento 11° apunta a un bloque dedicado a indicar el archivo, solo contiene entradas de bloques de información. Este bloque es llamado 'de indirección simple'. Si el archivo necesita un índice más grande se usará el 12° elemento.

El elemento 12° apunta a un bloque que contiene entradas que solo apuntan a bloques de indirección simple. Este bloque es llamado 'de indirección doble'.

El elemento 13° apunta a un bloque que contiene entradas que solo apuntan a bloques de indirección doble. Este bloque es llamado 'de indirección triple'.

Este planteo presenta las siguientes ventajas:

- Un inodo es de tamaño fijo y relativamente pequeño por lo cual puede ser mantenido en memoria por mayor cantidad de tiempo
- Archivos pequeños pueden ser accedidos mediante una única indirección, lo cual reduce el tiempo de procesamiento y el tiempo de acceso al disco

¿Clases de Inglés telefónicas con Profesores Nativos dónde y cuándo quieras?

25 minutos de conversación en inglés por teléfono desde 3,25 €/sesión.
Conoce más en www.ringteacher.com

ringteacher



- El tamaño máximo teórico de un archivo es lo suficientemente grande como para satisfacer todas las aplicaciones virtuales.

De éste último punto podemos rescatar que el *tamaño teórico máximo de un archivo* puede ser mayor que el tamaño real del filesystem ya que el mismo está relacionado con la capacidad de direccionamiento que se obtiene a partir de los tamaños de punteros y de bloques elegidos.

La capacidad que representa esta organización dependerá del tamaño de bloque elegido para el sistema. Si como ocurre en UNIX SVR4, el tamaño de bloque es 1 Kbyte...

Nivel	Número de Bloques	Número de Bytes
Directo	10	10K
Indirecto Simple	256	256K
Indirecto Doble	$256 * 256 = 65K$	65M
Indirecto Triple	$256 * 65K = 16M$	16G

De esta manera el tamaño máximo para un archivo sería algo mas de 16 G.

Para calcular la capacidad de direccionamiento del inodo, nos bastará con saber cuánto pueden direccionar los punteros directos, cuánto los indirectos, cuánto los doblemente directos y cuánto los trimplemente indirectos. Sumando todo, obtendríamos el tamaño máximo del archivo.

$$\sum_{i=0}^n \text{Cantidad de Punteros} * (\text{Cantidad Punteros por Bloque})^i * \text{Tamaño de Bloque}$$

Bloques referenciables

↖ Nivel de indireccion

Tam. máx. teórico del filesystem = cant. máx. de bloques direccionables • tam. de bloque

Tam. máx. real del filesystem = mín. (tam. máx. teórico del filesystem; espacio físico de almacenamiento)

El tamaño máximo de la partición del File System estará dado por la capacidad de almacenamiento que poseemos en nuestros discos, dependiendo del tipo de organización que tenga el mismo.

RAID es la sigla de Redundant Array of Independent Disks. Es un estándar de la industria que consiste en 6 niveles (0-5), admite distintas arquitecturas, todas ellas comparten las siguientes características: RAID es un conjunto de discos físicos vistos por el sistema operativo como un único disco lógico. La información es distribuida a través de los discos lógicos de un arreglo. La 'Facilidad de Disco Redundante' es usada para almacenar 'información de paridad', que garantiza la recuperación de datos en caso de falla de disco. Las dos ultimas características no son soportadas por el RAID nivel 0.

Su poder radica en permitir múltiples accesos simultáneos a lo que el sistema operativo ve como un único disco lógico. Para compensar los errores que pueden surgir de la actividad en paralelo, RAID usa información de paridad que permite recuperar información después de una falla de disco.

Resolución Práctica

Ej 56)

a) RAID 5 : Utiliza el equivalente a un disco físico para almacenar los stripes con bits de paridad, pero en lugar de utilizar "físicamente" un disco completo, distribuye los stripes a lo largo de todos los discos del conjunto.

Tamaño Real del File System = 5 discos * 23GB c/u = 115Gb

(pero como utiliza el equivalente a un disco por paridad)

Tamaño Real del File System = 115Gb - 23Gb = 92Gb

b) Para direccionar 10.000.000 de bloques se necesitan punteros de X bits

$$2^X = 10.000.000$$

$$X \log(2) = \log(10.000.000)$$

$$X = 23,25 \Rightarrow \boxed{X = 24 \text{ bits}} \Rightarrow \boxed{X = 3 \text{ bytes}}$$

Bloques 12 Kb = 12288 bytes \Rightarrow Cant. Punteros x Bloque = 12288bytes / 3bytes

$$\boxed{\text{Cant. Punteros x Bloque} = 4096}$$

$$\text{Tam. Mx. Terico} = \text{Cap.Direc.PD} + \text{Cap.Direc.PI} + \text{Cap.Direc.PDI}$$

* 2 punteros directos

$$\text{Cap.Direc.PD} = 2 * (4096)^0 * 12288\text{bytes} = 24576\text{bytes} (24\text{Kb})$$

* 1 puntero indirecto

$$\text{Cap.Direc.PI} = 1 * (4096)^1 * 12288\text{bytes} = 50331648 (48\text{Mb})$$

* 1 puntero doblemente indirecto

$$\text{Cap.Direc.PDI} = 1 * (4096)^2 * 12288\text{bytes} = 206158430208 (192\text{Gb})$$

Con lo cual:

$$\boxed{\text{Tamao Mximo Terico de un Archivo} = 206208786432 \text{ bytes}}$$

c) Idem anterior ya que los directorios, en UNIX se manejan tambin como archivos.

Ej 57)

a) RAID 1 : Utiliza la tcnica de espejo (mirroring), por lo cual duplica la informacin. Siempre habr un nmero par de discos, de los cuales solo estar disponible la mitad, ya que la otra mitad simplemente duplica la informacin existente en la primera mitad.

$$\text{Tamao Real del File System} = 6 \text{ discos} * 40\text{GB c/u} = 240\text{Gb}$$

(pero como utiliza la mitad para redundancia)

$$\boxed{\text{Tamao Real del File System} = 240\text{Gb} / 2 = 120\text{Gb}}$$

b) Para direccionar 20.000.000 de bloques se necesitan punteros de X bits

$$2^X = 20.000.000$$

$$X \log(2) = \log(20.000.000)$$

$$X = 24,25 \Rightarrow \boxed{X = 25 \text{ bits}} \Rightarrow \boxed{X = 4 \text{ bytes}}$$

$$\text{Bloques } 24\text{Kb} = 24576\text{bytes} \Rightarrow \text{Cant. Punteros x Bloque} = 24576\text{bytes} / 4\text{bytes}$$

$$\boxed{\text{Cant. Punteros x Bloque} = 6144}$$

$$\text{Tam. Mx. Terico} = \text{Cap.Direc.PD} + \text{Cap.Direc.PI} + \text{Cap.Direc.PDI} + \text{Cap.Direcc.PTI}$$

3 punteros directos

$$\text{Cap.Direc.PD} = 3 * (6144)^0 * 24576\text{bytes} = 73728\text{bytes} (72\text{Kb})$$

1 puntero indirecto

$$\text{Cap.Direc.PI} = 1 * (6144)^1 * 24576\text{bytes} = 150994944\text{bytes} (144\text{Mb})$$

1 puntero indirecto doble

$$\text{Cap.Direc.PDI} = 1 * (6144)^2 * 24576\text{bytes} = 927712935936 \text{ bytes} (864\text{Gb})$$

1 puntero indirecto triple

$$\text{Cap.Direc.PTI} = 1 * (6144)^3 * 24576\text{bytes} = 5699868278390784 \text{ bytes} (5184\text{Tb})$$

Con lo cual:

$$\boxed{\text{Tamao Mximo Terico de un Archivo} = 5700796142395392 \text{ bytes}}$$

Ejercicio 58.

Enunciado:

Un archivo en el disco rígido de una estación de trabajo (que tiene Windows con sistema vta), se encuentra fragmentado en los siguientes tracks: 72 – 30 – 85 – 160 – 50 – 23 – 17 – 102 – 298 – 10. Suponiendo que la cabeza esta en la posición 70 (ascendiendo), con una velocidad de 5400 RPM, un retardo de lectura entre tracks de 2 ms. y una capacidad de 300 cilindros. Se pide indicar en cada caso el orden de lectura y el tiempo total para el archivo si el algoritmo de planificación es:

a) C-SCAN

b) SSTF

(Considerar un giro de disco para la reubicación de la cabeza en el sector de arranque)

Teoría:

SSTF: Primero el de tiempo de servicio menor. Esta técnica consiste en seleccionar el pedido que requiere el menor movimiento del brazo del disco. Se minimiza el tiempo de búsqueda.

SCAN: La técnica SCAN evita esto. Aquí se requiere que el brazo se mueva en una sola dirección, satisfaciendo todos los pedidos pendientes, hasta que se llega a la última pista en esa dirección o hasta que no hay más pedidos en esa dirección. La dirección de servicio luego es revertida y el algoritmo procede en la otra dirección, de nuevo levantando pedidos en orden.

C-SCAN: SCAN circular. Esta política restringe el recorrido a una dirección solamente. De esta manera, cuando se llega a la última pista, el brazo se lleva a la parte opuesta y se comienza de nuevo con el recorrido. Esto reduce el tiempo de espera de los nuevos pedidos.

Resolución:

La secuencia para la recorrida con el algoritmo de planificación C-SCAN es la siguiente:

70	72	85	102	160	298	300	giro	0	10	17	23	30	50
Total recorrido:		280	pistas										
	Giro	11	ms										
Total:		571	ms.										

La secuencia para la recorrida con el algoritmo de planificación SSTF es la sig.:

70	72	85	102	50	30	23	17	10	100	298
Total recorrido:		348	pistas							
Total:		696	ms.							

EL GIRO se saca de las 5400 RPM

60 seg ----- 5400 RPM

1 seg ----- $X = 5400/60 = 90$, ➔ el disco da 90 revoluciones en 1 seg, pero como la unidad que necesitamos es ms, entonces:

90 RPS ----- 1 ms = 0,09 revoluciones

0,09 rev ----- 1 ms

1 rev ----- $1/0,09 = 11$ ms, entonces el tiempo de un GIRO es de 11 ms.

Ejercicio 59

Enunciado:

Se tiene un file system basado en i-nodos con las siguientes características: 5 punteros directos, 2 indirectos, 2 doblemente indirectos y 1 triple indirecto y cuyo tamaño de punteros es de 32 bits. Si los sectores del disco son de 512 kb, indicar:

- a) El tamaño máximo de un archivo
- b) ¿Cuántos bloques de disco hacen falta para leer el byte número 16911366, considerando que el inodo correspondiente al archivo ya se lo conoce?
- c) Explique porque es imprescindible para la existencia de “hard links” en Unix que los permisos sobre el archivo residan en el inodo. ¿Qué sucede con los “symbolic links”?

Teoría:

Inodos: un inodo (nodo de información) es una estructura de control que contiene la información clave que el sistema operativo necesita para un archivo particular. Varios archivos pueden estar asociados a un mismo inodo, pero un inodo activo está solo asociado a un archivo, y cada archivo es controlado por exactamente un inodo.

Los atributos del archivo, sus permisos y otra información de control es guardada en el inodo.

Inodo Estructura de control que contiene la información total de un archivo necesaria para el sistema operativo, salvo el nombre del mismo.

Los inodos, son estructuras que nos permiten trabajar con archivos en sistemas de archivos del tipo ext2, ext3, etc. Todos los atributos del archivo, permisos, información de control tales como la fecha de modificación etc. Un inodo contiene la "información administrativa" de un archivo. En sentido estricto los inodos son los archivos.

El inodo puede ser asociado con varios archivos, mejor dicho, varios nombres de archivo pueden hacer referencia al mismo número de inodo. Pero la recíproca no es válida, es decir, un solo inodo por cada archivo.

Nosotros podemos tranquilamente saber el número de inodo de un archivo en particular por medio de la utilidad ls(1), utilizando el parámetro -i, obtenemos lo siguiente:

```
[gen@gen][/]  
#ls -li refbooks.html  
34 refbooks.html
```

En este caso vemos que 34 es el número de inodo que posee el archivo (ordinario en este caso) refbooks.html.

Los inodos demás está decir son de tamaño fijo, ha habido diferencias entre los sistemas de archivos en cuanto a tamaños de nodo. Por ejemplo, en la versión 7 del Unix System V, el tamaño del inodo es de 64 bytes, la 4.3+BSD es de 128.

Ventajas:

Las ventajas de utilizar i-nodos es que este tipo de filesystems no utilizan FAT, sino que utilizan una lista de inodos, i-list, y el tamaño del bloque de datos por inodo depende de lo que yo elija en la instalación, dado esto no tenemos la limitación de una partición FAT que es de 2GB, todo lo contrario, incluso podíamos crear archivos hasta de 4GB hace unos años con este sistema, mas grande que una partición FAT16, hoy en día las limitaciones son menores todavía, y hablaremos de eso más adelante.

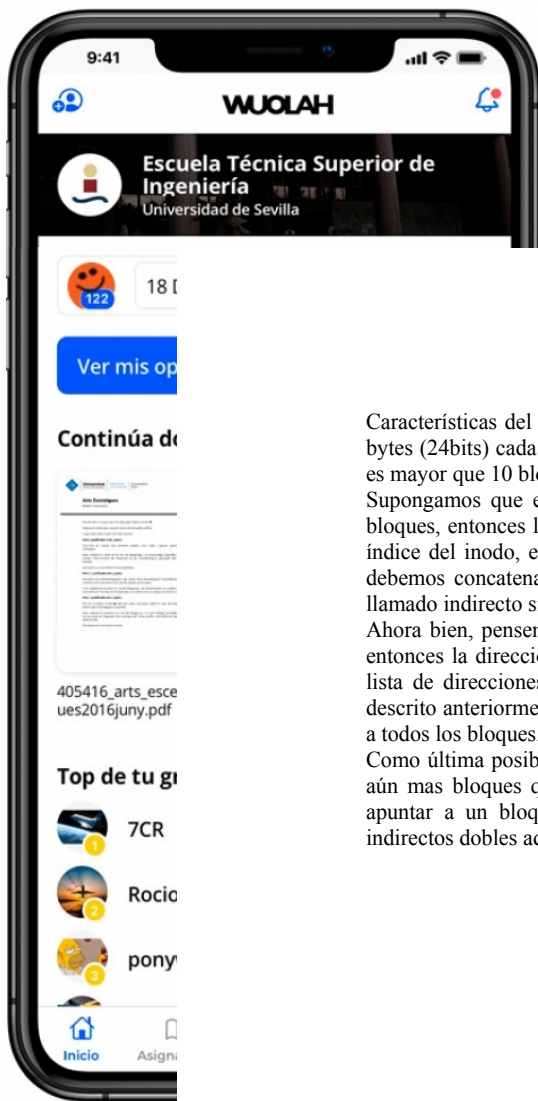
Otra de las ventajas es que podemos crear vínculos a través de todo el sistema de archivos, siempre y cuando no queramos vincular archivos de otro sistema (salvo que utilicemos symlinks, o sea, vínculos suaves).

El mover archivos dentro de la misma partición se vuelve mas veloz, dado que no es necesario copiar el contenido de los archivos (directorios), sino que basta con cambiar las referencias, o sea, como dijimos anteriormente, basta con crear una nueva entrada de directorio en el directorio destino y vincularla al nuevo archivo (o directorio) y desvincular la vieja entrada de directorios en el viejo directorio de origen.

Limitaciones:

El tamaño de los bloques por inodo es fijo cuando creamos el filesystem como mencionamos anteriormente, o sea que la única manera de incrementar la cantidad de inodos es incrementando el espacio en disco.

Asignación de archivos



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.

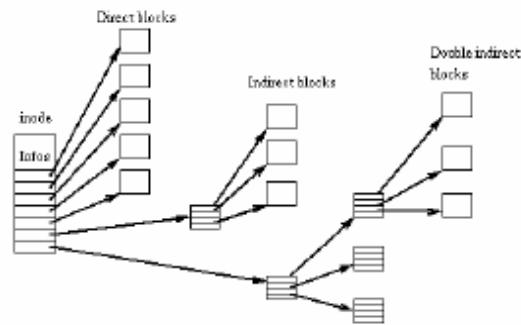


Características del inodo: posee 39 bytes de información de dirección, divididos en 13 direcciones de 3 bytes (24bits) cada una. Las primeras 10 apuntan a los primeros 10 bloques de datos del archivo. Si este es mayor que 10 bloques, se usan uno mas niveles indirectos.

Supongamos que estamos en la situación en que nos excedimos y nuestro archivo ocupa mas de 10 bloques, entonces la dirección 11 del inodo debe apuntar a un bloque que contiene la siguiente parte del índice del inodo, es decir, dado que el índice de un solo inodo no alcanza para direccionar un bloque, debemos concatenarlo con el índice de otro bloque, y así formamos el índice final. Este bloque es el llamado indirecto simple, y contiene los punteros a los siguientes bloques de archivo.

Ahora bien, pensemos que pasaría si nuestro archivo se excede mas aún y ocupa más bloques todavía, entonces la dirección 12 del inodo debe apuntar a un bloque indirecto doble, el cual debe contener una lista de direcciones de bloques indirectos simples, con los cuales en cada uno se utilizara el método descrito anteriormente para trabajar con los índices y poder así direccionar lo suficiente para poder llegar a todos los bloques.

Como última posibilidad consideremos la posibilidad de que ocurra el caso de que nuestro archivo tiene aún mas bloques que los considerados hasta ahora, entonces la dirección 13 y última del inodo debe apuntar a un bloque indirecto triple, que posee un tercer nivel de indexación, y apunta a bloques indirectos dobles adicionales los cuales apuntan a bloques indirectos simples.



Resolución:

a) tamaño máximo del archivo:

- 1- tenemos que obtener el número de puntero del bloque de disco que cabe en el bloque y se logra dividiendo el tamaño de bloque por el tamaño del puntero. $(512/32)$
- 2- por cada indirección multiplico la cantidad de punteros por el tamaño del bloque por la cantidad de punteros elevado al numero de indirección, en el caso de los directos seria 0 .

$$5 * 512 \text{ (directos)} + 2 * 512 * 512/32 \text{ (indirectos)} + 2 * 512 * (512/32)^2 \text{ (doblemente indirecto)} + 2 * 512 * (512/32)^3 \text{ (triplemente indirecto)} =$$

$$2560 + 16384 + 262144 + 4194304 = 4475392 \text{ Kb}$$

b) Usando la información de a) vemos que los bloques directos solamente cubren los primeros 2.5 Mb, el primer bloque indirecto cubre los próximos 8 Mb. El segundo bloque indirecto recién estaría cubriendo los próximos 8 Mb.

La posición pedida del archivo es 16 Mb.

Se ve claramente en el bloque en la segunda indirección. Esto requiere tres accesos al disco. Dos, una para el primer bloque indirecto, otra para el segundo y el tercer acceso al bloque que contiene el dato pedido.

c) Los Hard links crean una entrada en el directorio y la encadenan a un inodo ya existente. Los symbolic links hacen re

Ejercicio 60.

Enunciado:

Se tiene un disco con la siguiente geometría: 3 platos, 100 sectores por pista, un total de 500 cilindros y sectores de 512 bytes. En un momento determinado se leen los primeros 3 registros del archivo 1 y los cuatro primeros registros del archivo 2.

Sean los archivos con sus respectivos sectores lógicos:

Archivo 1 (tiene registros de 1024 bytes): 1301 / 1902 , 2503 / 3704 , 599 / 600

Archivo 2 (tiene registros de 512 bytes): 30023, 30024, 3800, 60102

Si se sabe que el brazo del disco se encuentra en el cilindro 4, graficar el orden de ejecución de los pedidos, asumiendo que están en la cola, para cada uno de los siguientes algoritmos:

a) SSTF

b) F-SCAN – el brazo esta ascendiendo

Teoría:

SSTF: Primero el de tiempo de servicio menor. Esta técnica consiste en seleccionar el pedido que requiere el menor movimiento del brazo del disco. Se minimiza el tiempo de búsqueda.

SCAN: La técnica SCAN evita esto. Aquí se requiere que el brazo se mueva en una sola dirección, satisfaciendo todos los pedidos pendientes, hasta que se llega a la última pista en esa dirección o hasta que no hay más pedidos en esa dirección. La dirección de servicio luego es revertida y el algoritmo procede en la otra dirección, de nuevo levantando pedidos en orden.

N-SCAN: Es el algoritmo SCAN pero no se tiene una única cola para encolar los pedidos, sino que se tienen N colas. Cuando se están atendiendo los pedidos de una cola, los nuevos que lleguen NO pueden ingresar a la cola en uso, se deben encolar en alguna de las otras colas. Cuando se atendieron todos los pedidos de una lista, se pasa a atender los pedidos de otra con el mismo criterio (no se pueden encolar nuevos pedidos en esta nueva cola atendida)

F-SCAN: Es un caso particular del N-SCAN; solo que tiene 2 colas. Los criterios a tener en cuenta son los mismos: cuando se atiende una cola, los nuevos pedidos se encolan en la otra.

Resolución:

En estos ejercicios lo único que se puede planificar es el movimiento del brazo que maneja la posición de las cabezas en los platos del disco.

Datos disco rígido:

Platos = 3

Cabezas = 6 (0..5) “2 x plato (1 por lado)”

Sectores = 100 / pista

Cilindros = 500 (0..499)

Bytes / Sector = 512

Sectores / Cilindro = Sectores / Plato * Cantidad de Platos = 200 * 3 = 600

Posición inicial = Cilindro 4

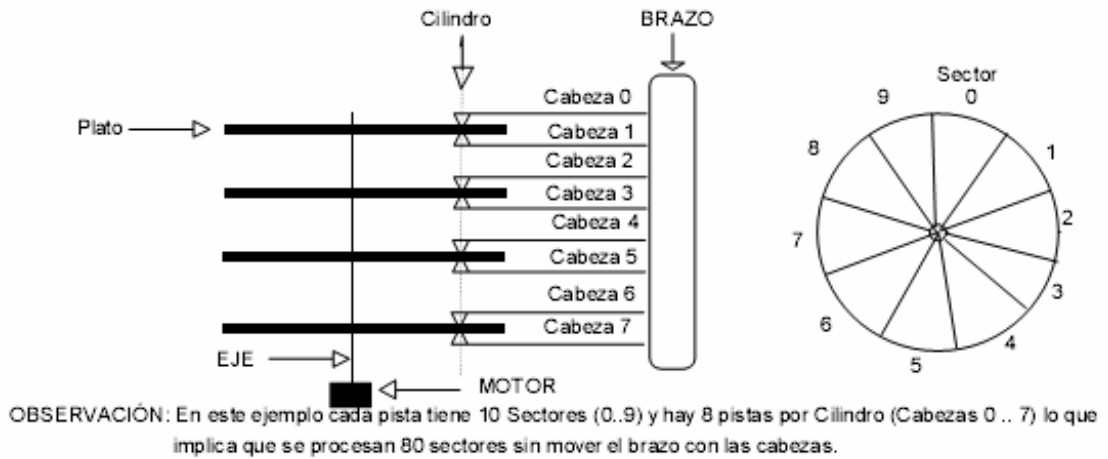
Dirección Física = (Cilindro, Cabeza, Sector)
(0 .. 499 , 0 .. 6, 1 .. 100)

Conversión de direcciones lógicas a direcciones físicas:

<p>Dirección Lógica = 1301</p> <div> <div>1301 600</div> <div>101 100</div> </div> <div> <div>101 2</div> <div>1 (+1) 1</div> <div>Cilindro Sector Cabeza</div> </div> <p>Dirección Física = (2 , 1 , 2)</p>	<p>Dirección Lógica = 1902</p> <div> <div>1902 600</div> <div>102 100</div> </div> <div> <div>102 3</div> <div>2 (+1) 1</div> <div>Cilindro Sector Cabeza</div> </div> <p>Dirección Física = (3 , 1 , 3)</p>
<p>Dirección Lógica = 2503</p> <div> <div>2503 600</div> <div>103 100</div> </div> <div> <div>103 4</div> <div>3 (+1) 1</div> <div>Cilindro Sector Cabeza</div> </div> <p>Dirección Física = (4 , 1 , 4)</p>	<p>Dirección Lógica = 3704</p> <div> <div>3704 600</div> <div>104 100</div> </div> <div> <div>104 6</div> <div>4 (+1) 1</div> <div>Cilindro Sector Cabeza</div> </div> <p>Dirección Física = (6 , 1 , 5)</p>
<p>Dirección Lógica = 599</p> <div> <div>599 600</div> <div>599 100</div> </div> <div> <div>599 0</div> <div>99 (+1) 5</div> <div>Cilindro Sector Cabeza</div> </div> <p>Dirección Física = (0 , 5 , 100)</p>	<p>Dirección Lógica = 600</p> <div> <div>600 600</div> <div>0 100</div> </div> <div> <div>0 1</div> <div>0 (+1) 0</div> <div>Cilindro Sector Cabeza</div> </div> <p>Dirección Física = (1 , 0 , 1)</p>
<p>Dirección Lógica = 30023</p> <div> <div>30023 600</div> <div>23 100</div> </div> <div> <div>23 50</div> <div>23 (+1) 0</div> <div>Cilindro Sector Cabeza</div> </div> <p>Dirección Física = (50 , 0 , 24)</p>	<p>Dirección Lógica = 30024</p> <div> <div>30024 600</div> <div>24 100</div> </div> <div> <div>24 50</div> <div>24 (+1) 0</div> <div>Cilindro Sector Cabeza</div> </div> <p>Dirección Física = (50 , 0 , 25)</p>
<p>Dirección Lógica = 3800</p> <div> <div>3800 600</div> <div>200 100</div> </div> <div> <div>200 6</div> <div>0 (+1) 2</div> <div>Cilindro Sector Cabeza</div> </div> <p>Dirección Física = (6 , 2 , 1)</p>	<p>Dirección Lógica = 60102</p> <div> <div>60102 600</div> <div>102 100</div> </div> <div> <div>102 100</div> <div>2 (+1) 1</div> <div>Cilindro Sector Cabeza</div> </div> <p>Dirección Física = (100 , 1 , 3)</p>

El enunciado del ejercicio dice que se asume que los pedidos ya están en la cola, es decir, que no van a llegar pedidos nuevos mientras se atiende los solicitados y además que el brazo del disco se encuentra en el cilindro 4. Al no especificar la velocidad de rotación del disco, se desprecia el tiempo que tarda el brazo del disco en ir desde una pista a la otra, por lo que si hay más de un pedido por pista, asumo que puede atender a cualquiera de los dos primero y luego al otro.

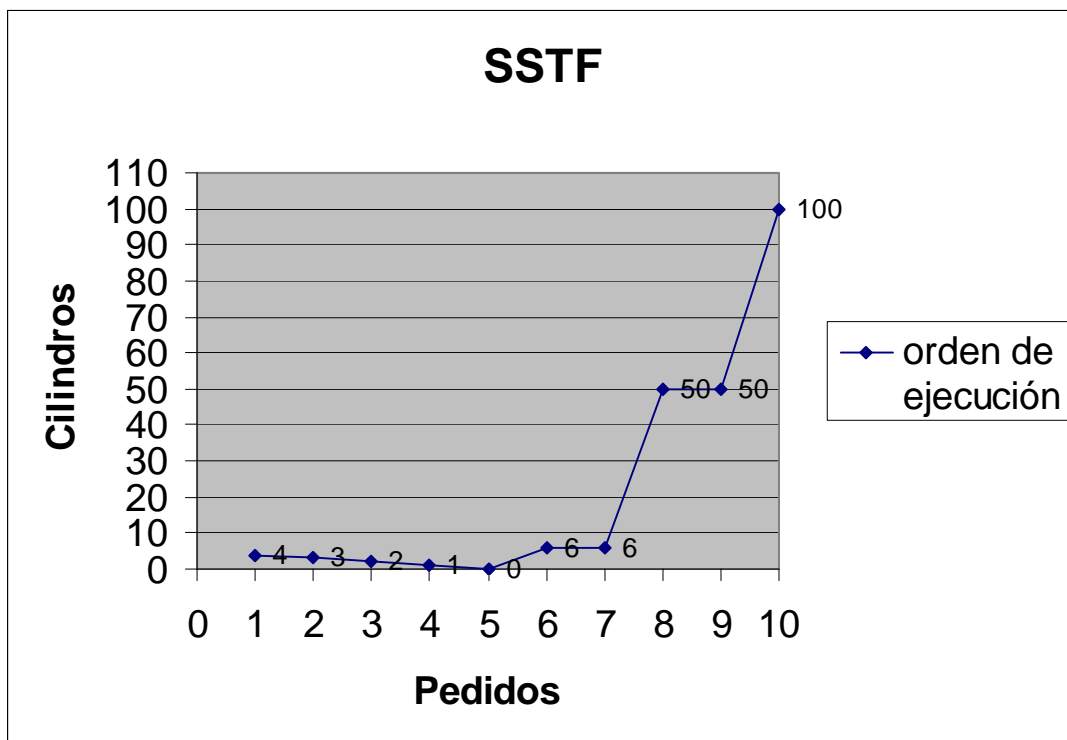
Ejemplo de cálculo de sectores por cilindro:



a) SSTF (Shortest Seek Time First)

El orden en que se encolan para su ejecución con este algoritmo es el siguiente:

4 - 3 - 2 - 1 - 0 - 6 (x2) - 50 (x2) - 100



b) F-SCAN

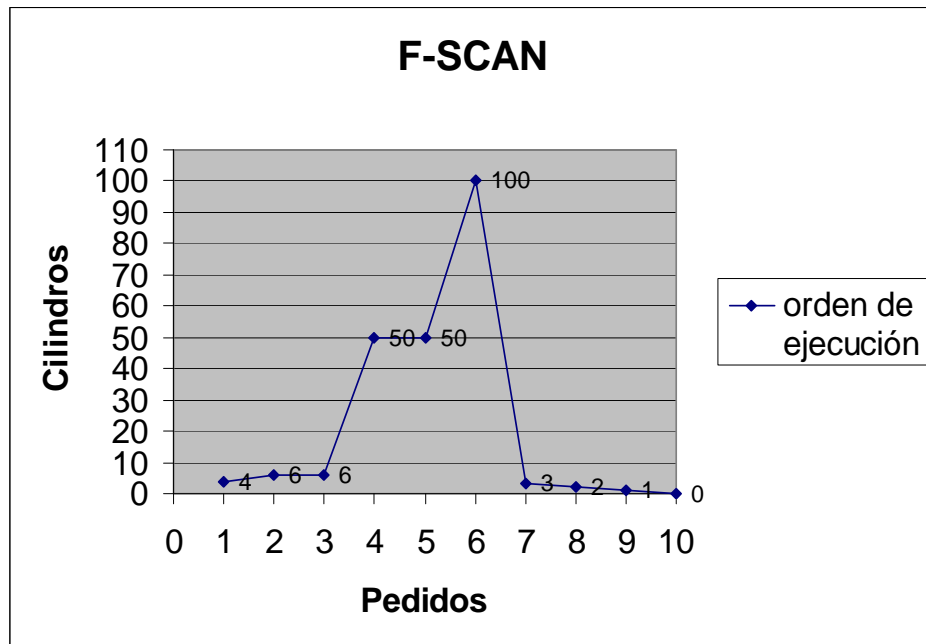
El orden en que se encolan para su ejecución con este algoritmo es el siguiente:

4 - 6 (x2) - 50 (x2) - 100 - 3 - 2 - 1 - 0

¿Clases de Inglés telefónicas con Profesores Nativos dónde y cuándo quieras?

25 minutos de conversación en inglés por teléfono desde 3,25 €/sesión.
Conoce más en www.ringteacher.com

ringteacher



Problema 64

Un Sistema Operativo utiliza organización encadenada para la gestión de los archivos. La maquina donde corre este sistema utiliza un disco rígido que gira a 6000 RPM (Revoluciones por minuto), el cual posee 3 platos, formando un total de 6 cabezas, 10 sectores por pista y un total de 300 cilindros. Considere que el tamaño de un registro es igual a un sector y es de 512 bytes. Los sectores se recorren en sentido ascendente.

Existe un proceso que en un momento dado deberá ejecutar el siguiente código (sin errores de compilación):

```
Archivo = fopen("/home/guest/final.bin", "wb");  
For (I= 0 ; i < (filesize(archivo) / size(REGISTRO)) ; i ++)  
Bytes = fread (dato, archivo, size (REGISTRO));
```

A su vez se cuenta con una FAT (File Allocation Table) como la que se describe a continuación:

File Name	Start Block (Dirección Lógica)	Lenght (En cantidad de Bloques)
File A	1089	10
final.bin	1000	5

El siguiente grafico muestra la organización del disco en un instante determinado:

957 / 3420	1000 / 1248	FREE	FREE
1089 / 1200	FREE	1200 / 4398	1248 / 5791
3420 / EOF	FREE	4398 / EOF	5791 / 957

TIP: La estructura de bloques se encuentre dada por la Dirección lógica del bloque / (Barra) Puntero de 32 bits cuyo contenido es una dirección lógica.

Sabiendo que el algoritmo de planificación del disco es el S-SCAN, que no se utiliza el programa "Disk Defragmenter" para la compactación del disco y que la cabeza se encuentra en la dirección lógica 5000 dirigiéndose a la 7100, se pide:

- El tiempo que tardo el proceso en ejecutar el código, considerando **UNICAMENTE** el tiempo que se insume en atender los pedidos a disco
- La traducción de las direcciones lógicas a físicas a las cuales hace referencia el proceso. **Deduzca** la formula que le permite realizar dichas traducciones.

Ejercicio 65

Enunciado:

Sea un Sistema Operativo tipo UNIX que gestiona bloques de datos de 24 bytes. Cada i-nodo, además de otra información, contiene 10 punteros directos a bloques de datos, un puntero de indirección simple y un puntero de indirección doble.

Suponiendo que tenemos una caché de 20 bloques de datos y otro de 20 i-nodos inicialmente vacíos, que el

i-nodo del directorio raíz se encuentra en memoria principal y que sólo ejecuta en el Sistema Operativo un proceso, se pregunta:

a) ¿Cuántos accesos al disco son necesarios para leer el byte 299, sabiendo que el tamaño máximo de un archivo es aproximadamente de 3,9 KB?

b) Teniendo en cuenta los datos calculados en el punto anterior. ¿Cuántos accesos al disco son necesarios para leer hasta el byte 299?

Justifique sus respuestas.

Teoría:

I-nodos: Un i-nodo (nodo índice o de información) es la estructura que contiene la información clave para manejar un archivo. Contiene atributos, permisos y otra información de control del archivo, incluyendo direcciones. Muchos nombres de archivos (contenidos en los directorios) pueden ser asociados (enlace duro) a un i-nodo, pero un archivo es controlado por un solo i-nodo y un i-nodo controla un solo archivo.

Bloque: Un bloque es generalmente una cantidad de longitud fija, determinada por el S.O. como la unidad mínima de posicionamiento. Puede ser 1 byte, 512 bytes, 1024 bytes, etc.

Asignación de Archivos

En UNIX la asignación es dinámica, en base a bloques no necesariamente contiguos. Se le sigue la pista a los archivos mediante el método indexado, con la primera parte del índice en el i-nodo. Allí hay un vector de 13 elementos de 3 bytes c/u. Los 10 primeros contienen la dirección de los 10 primeros bloques del archivo. Si el archivo ocupa más de 10 bloques, uno o más niveles de indirección son usados.

El elemento 11° apunta a un bloque dedicado a indicar el archivo, solo contiene entradas de bloques de información. Este bloque es llamado 'de indirección simple'. Si el archivo necesita un índice más grande se usará el 12° elemento.

El elemento 12° apunta a un bloque que contiene entradas que solo apuntan a bloques de indirección simple. Este bloque es llamado 'de indirección doble'.

El elemento 13° apunta a un bloque que contiene entradas que solo apuntan a bloques de indirección doble. Este bloque es llamado 'de indirección triple'.

La capacidad que representa esta organización dependerá del tamaño de bloque elegido para el sistema. Si como ocurre en UNIX SVR4, el tamaño de bloque es 1 Kbyte:

Nivel	Número de Bloques	Número de Bytes
Directo	10	10 K
Simple Indirecto	256	256 K
Doble Indirecto	$256 \times 256 = 65\text{ K}$	65 M
Triple Indirecto	$256 \times 65\text{ K} = 16\text{ M}$	16 G

De esta manera el tamaño máximo para un archivo sería de 16 G.

Resolución:

Direccionamientos:

10 directos, 1 simple y 1 doble.
Total del bloque de datos 24 bytes.

a) Tamaño Máximo de Archivo:

$$\sum_{i=0}^i \text{Cantidad de Punteros} * (\text{Tamaño de Bloque} / \text{Tamaño Puntero})^i * \text{Tamaño de Bloque}$$

Cantidad de Punteros (por bloque): Tamaño de bloque / Tamaño de puntero

P = Tamaño del Puntero

Tamaño del File System: $2^{(\text{Tamaño del Puntero})} * \text{Tamaño del Bloque} = \text{Resultado en bytes}$

$$10 * (24/P)^0 * 24 + 1 * (24/P)^1 * 24 + 1 * (24/P)^2 * 24 = 3,9 \text{ K}$$

$$240 + (576/P) + (13824 / P^2) = 3994 \text{ b } (3993,6 \text{ b} \rightarrow \text{Redondeo})$$

$$-3754 + (576/P) + (13824 / P^2) = 0$$

Sacamos las raíces de la cuadrática:

$$\Rightarrow P = -1,843 \text{ b o } P = 1,997 \text{ b} \rightarrow \text{por redondeo: } P = 2 \text{ b}$$

Cantidad de Punteros por Bloque:

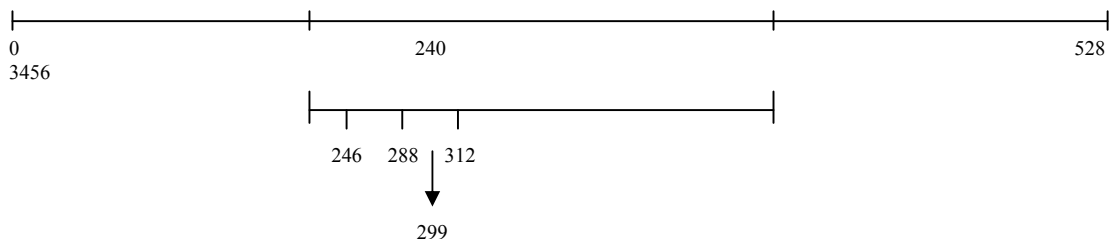
Tamaño de Bloque / Tamaño de Puntero

$$\Rightarrow 24/2 = 12 \rightarrow \text{Punteros por bloque}$$

Para averiguar cuantos accesos a disco son necesarios, hacemos la cuenta de tamaño max. De archivo:

$$10 * (12 \text{ b})^0 * 24 + 1 * (12 \text{ b})^1 * 24 + 1 * (12 \text{ b})^2 * 24 =$$

$$240 \text{ b} + 288 \text{ b} + 3456 \text{ b} =$$

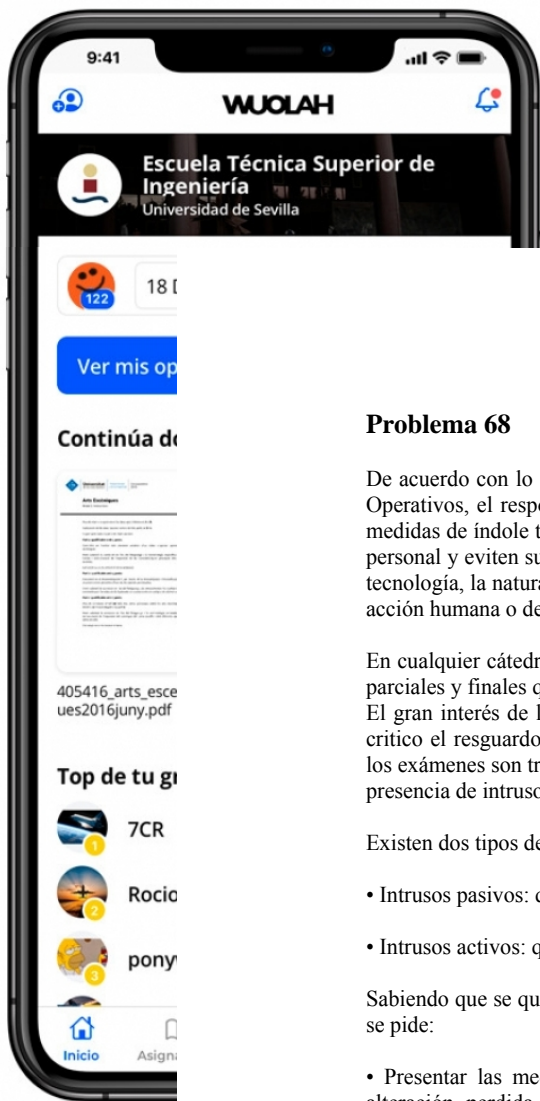


Nos damos cuenta que esta en la primera indirección (La Simple) ya que el valor es mayor que el del direccionamiento directo.

Por lo tanto para llegar hasta el bloque de disco 299, se accede por la indirección simple, luego por el i-nodo al que apunta ésta y luego al bloque 299.

Esto en total son 3 accesos a disco.

b) Teniendo en cuenta los resultados del punto anterior, como el i-nodo principal, y el de la primera indirección, están cargados en la caché de i-nodos, y el bloque que contiene el byte 299, ya se encuentra cargado en memoria. Los accesos a disco que son necesarios para leer “hasta” el byte 299 son 12.



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.



Problema 68

De acuerdo con lo establecido por el artículo 25 inciso 1ª de la Ley de Protección de Datos de Sistemas Operativos, el responsable del archivo y, en su defecto, el encargado del tratamiento deberá adoptar las medidas de índole técnicas y organizativas necesarias que garanticen la seguridad de los datos de carácter personal y eviten su alteración, pérdida, tratamiento o acceso no autorizado habida cuenta del estado de la tecnología, la naturaleza de los datos almacenados y los riesgos a que están expuestos, ya provengan de la acción humana o del medio físico o natural.

En cualquier cátedra de la facultad el mayor valor que se tiene es la información acerca de los exámenes parciales y finales que se van a tomar en determinadas fechas, entre otras.

El gran interés de los alumnos por querer obtener los exámenes con anticipación hace que sea aún más crítico el resguardo de dicha información. Por ello, se deben extremar las medidas de seguridad cuando los exámenes son transmitidos a través de las redes de comunicación y evitar el gran riesgo ante la posible presencia de intrusos que puedan hacerse de dicha información.

Existen dos tipos de intrusos:

- Intrusos pasivos: que son capaces de develar la información transmitida
- Intrusos activos: que son capaces de modificar esa información

Sabiendo que se quiere optimizar la eficiencia de las medidas técnicas y reducir los costos de las mismas, se pide:

- Presentar las medidas técnicas (mediante un esquema, para cada uno de los casos), para evitar la alteración, pérdida, tratamiento o acceso no autorizado de la información en la transmisión manteniendo la confidencialidad, autenticidad e integridad de los datos, sabiendo que los exámenes tienen que ser recibidos al menos por 5 integrantes de la cátedra para realizar su resolución.

Solución :

Una de las dos amenazas más conocidas a la seguridad (la otra es la de los virus) es el intruso, conocido en general como pirata informático. Los ataques de intrusos varían desde benignos hasta serios. En el extremo benigno de la escala, hay mucha gente que simplemente desea explorar las redes y ver que hay ahí fuera. En el extremo serio están los individuos que intentan leer datos privilegiados, realizar modificaciones no autorizadas a los datos o trastornar el sistema. El objetivo de los intrusos es obtener acceso a un sistema o aumentar el conjunto de privilegios accesibles en un sistema. Existen 2 tipos de intrusos: los pasivos (solo desean leer archivos y documentos y no están autorizados para hacerlo) y los intrusos activos (son los que desean hacer cambios a los datos y no están autorizados). Si se desea diseñar un sistema seguro contra ellos, es importante determinar el tipo de intruso contra el que se desea disponer de medidas de protección. Por su parte, la seguridad y protección de un sistema tienen como objetivos principales entre otros, prevenir y eliminar amenazas potenciales, disponer de un sistema seguro que mantenga la integridad, disponibilidad y privacidad de los datos, manipular datos del sistema (correctos, disponibles y privados), como así también asegurar la integridad de la información del centro de cómputos.

En todos los casos es fundamental el buen diseño de medidas técnicas que ayuden a evitar la alteración, pérdida, tratamiento o acceso no autorizado de la información en la transmisión, manteniendo la confidencialidad, autenticidad e integridad de los datos.