**Informe - Sistemas Operativos 2020**

**Castro, Martin - LU 92179**

**Vazquez, Santiago - LU 108751**

**Experimentación**

**CONJUNTO DE TAREAS**

Para el ejercicio “Conjunto de tareas” se decidió tener un proceso coordinador (proceso creado al momento de ejecutar el programa) y tres procesos, A, B y C, que representan cada uno un tipo de tarea a realizar. Al momento de seleccionar si se realizarán 4, 5 o 6 tareas, el coordinador genera una estructura de tareas con lo que debe realizar cada proceso tarea. Para comunicarse el proceso padre con cada proceso tarea se utilizará un pipe para cada uno, de manera tal que el coordinador escribirá en cada pipe un mensaje con la estructura de tareas a realizar y cada proceso tarea leerá el mensaje.

Un proceso tarea recibirá un mensaje que leerá. De acuerdo a la cantidad de tareas que se deban realizar se crearán 2 hilos para el proceso A, uno o dos hilos para el proceso B, y cero o dos hilos para el proceso C.

Cada hilo, según corresponda, chequeará de qué tipo es el mensaje recibido. Para solucionar que solo una tarea se pueda realizar por vez se utiliza un semáforo de exclusión mutua. Mediante un pipe compartido por todos los hilos se enviará una estructura de mensaje al proceso coordinador con la tarea que fue realizada.

El proceso coordinador leerá la estructura de mensajes que fue enviada cuando terminó cada tarea y luego se imprimirá el mensaje contenido en dicha estructura.

**MINI SHELL**

La idea principal de este programa es simular el funcionamiento de una consola. Cuando el usuario ingresa una instrucción, el proceso se encarga de separar la cadena de caracteres ingresada en varias subcadenas. Para separar el comando ingresado y sus argumentos se utilizó la función ***\*strtok(char \*str, const char \*delim),*** dentro de nuestra función ***separarStrings(char \*comandos)*** que se encarga de dividir el comando y su posible parámetro***.*** Luego se discrimina la operación a realizar, dentro de las indicadas en el enunciado. Una vez obtenido el comando a ejecutar, se realiza la creación de un proceso mediante una bifurcación hacia un hijo y se carga el comando a ejecutar por medio de un ***execv(),*** reemplazando la imagen actual del proceso por una nueva, y se ejecuta la tarea correspondiente a cada comando. Este procedimiento se repite en bucle hasta que se decida proceder con la finalización de la tarea por medio de un ***exit()***.

Los comandos a utilizar son:

* *mkdir*: Crear un directorio con el nombre ingresado por parámetro.
* *rmdir*: Eliminar directorio con el nombre pasado por parámetro.
* *touch*: Crear archivo con el nombre pasado por parámetro.
* *ls*: Mostrar una lista de archivos de un directorio pasado por parámetro. Se utiliza la librería dirent.h. Funciona únicamente si los nombres de las carpetas de la dirección pasada por parámetro no contienen algún espacio.
* *cat*: Mostrar el contenido de un archivo pasado por parámetro.
* *chmod*: Modificar los permisos de un archivo.

**DEMASIADAS BOTELLAS DE LECHE**

Para el ejercicio “Demasiadas botellas de leche” sincronizado con **hilos** **y semáforos** se optó por un usar tres semáforos y dos hilos.

Se usó un semáforo para sincronizar las botellas de leche que hay en la heladera. En el ejercicio no está explícito la cantidad de las mismas por lo que se asumió que haya como máximo diez. Otro de los semáforos utilizados es binario, el cual indicará si se puede o no ir a comprar leche en caso que haya faltante de la misma, por lo cual si eventualmente alguien quiere tomar leche pero no hay y ya hay alguien comprando, no vaya y simplemente vuelva más tarde a chequear si tiene disponible una botella de leche para tomar. Por último y no menos importante, el semáforo binario de exclusión mutua para sincronizar la heladera, donde si hay alguien en la heladera no podrá otra persona acercarse a chequear si puede tomar una botella y si se está reponiendo la leche no pueda venir otra persona y tomar una botella.

*Pseudocódigo:*

sem leche = 10;

sem supermercado = 1

sem heladera = 1

mirarHeladera() {

si (try\_wait(heladera) == 0) *//heladera libre*

si(try\_wait(leche) == 0) *//leche disponible*

tomarLeche();

signal(heladera);

sino *//no hay leche*

si(try\_wait(supermercado) == 0) *//nadie está comprando leche*

signal(heladera); //se va de la heladera al supermercado

comprarLeche();

wait(heladera); //vuelve a la heladera a reponer leche

reponerLeche();

signal(supermercado)

signal(heladera)

sino

noHayLecheYestanComprando();

signal(heladera);

sino

heladeraOcupada();

En el caso de la sincronización con **colas de mensajes** el problema es análogo a la sincronización con procesos e hilos, donde cada semáforo en este caso es una cola. Así, se tendrá una cola para la sincronización de las botellas de leche, una para la sincronización de la heladera y una para sincronizar si es posible ir al supermercado.

Inicialmente, la cola de botellas de leche se llenará con diez mensajes, cada uno representando una botella; a la cola de la heladera y a la del supermercado se les enviará un mensaje a cada una, de manera que sea posible acercarse a la heladera y comprar en caso que haga falta.

El pasaje de semáforos a mensajes es directo, uno a uno, por lo que simplemente hay que reemplazar las operaciones por las que corresponden según la siguiente tabla:

|  |  |  |
| --- | --- | --- |
| **OPERACIÓN EN SEMÁFOROS** | **OPERACIÓN EN COLAS DE MENSAJES** | **CONSIDERACIONES** |
| post(s) | send(msj) |  |
| wait(s) | receive(msj) | En este caso, wait es bloqueante. El flag para el pasaje de mensajes será seteado en 0 |
| try\_wait(s) | receive(msj) | El flag para el pasaje de mensajes será seteado en IPC\_NOWAIT |

**COMIDA RÁPIDA**

Para el ejercicio “comida rápida” sincronizado con **hilos** se optó por tener un hilo para cada cocinero (3 en total), uno p=ra el camarero, un hilo para el limpiador y un hilo por cada cliente (50 en total). En lo que respecta a los semáforos, se tiene uno de tipo contador para las mesas que están disponibles para ser usadas; otro semáforo contador para las mesas que quedan sucias al retirarse el cliente, un semáforo para exclusión mutua en el caso que se solicite al camarero, uno contador para el número máximo de platos que pueden haber cocinados, uno contador para los platos cocinados, y un semáforo de exclusión mutual para indicar el plato que se le entrega a un cliente.

*Pseudocódigo:*

sem mesaLimpia = 30;

sem mesaSucia = 0;

sem camarero = 0;

sem cola = 10;

sem = comidaDisponible = 0;

sem comidaLista = 0

*cliente()* {

wait(mesaLimpia);

sentarse();

pedirComida();

signal(camarero);

wait(comidaLista);

comer();

retirarse();

signal(mesaSucia);

}

*camarero()* {

wait(camarero);

buscarComida();

wait(comidaDisponible);

signal(cola);

llevarComida();

signal(comidaLista);

}

*limpiador()* {

wait(mesaSucia);

limpiarMesa();

signal(mesaLimpia);

}

*cocinero()* {

wait(cola);

cocinar();

signal(comidaDisponible);

}

En el caso de la sincronización con **colas de mensajes** se optó por tener siete colas:

* cola de mesas
* cola de camarero
* cola de pedidos
* cola de lugar disponible
* cola de comida lista
* cola de comida con carne
* cola de comida vegetariana

Las colas de mesas y de lugares disponibles son inicializadas con treinta y diez mensajes respectivamente, de manera tal que inicialmente hay treinta mesas disponibles y diez lugares para despachar comida.

Este ejercicio se modela de la misma manera que el de botellas de leche, el pasaje de semáforos a colas de mensajes se realiza de la misma manera. La dificultad que tiene es el adicional que ahora hay dos tipos de comida, de manera que el cliente al elegir el menú puede ser de tipo carne o vegetariano. En este caso se envía un mensaje a la cola de pedidos con el tipo que corresponde, un cocinero recibe el mensaje, cocina y coloca en la cola de comida con carne o vegetariana según corresponda. Luego el camarero tomará de la cola que corresponda la comida y se la entregará al cliente.

**Problemas**

Artículo elegido: “Android Operating System Architecture”

**a) Describa los componentes de la arquitectura.**

La arquitectura de este sistema operativo se basa en cuatro capas:

* Kernel de Linux
* Librerías nativas, conteniendo la librería Android y Android Runtime
* Framework de aplicaciones
* Aplicaciones

La capa del **kernel de Linux** provee la funcionalidad estándar del sistema. Este kernel fue modificado por Google para atender las necesidades que un sistema móvil requiere. Así, se contará con el driver de alarma para manejar de manera correcta ciertos eventos. También tendrá los controladores Binder para manejar la comunicación entre procesos debido a que un dispositivo móvil tiene los recursos más limitados que una PC. La modificación del kernel de Linux incorpora el binder permite que se solucionen ciertos problemas de costos de procesamiento y de seguridad. El kernel también tendrá a su cargo la gestión de energía, sumamente importante para celulares, tablets y todo dispositivo que tenga este sistema operativo y para funcionar no dependa de estar conectado a la corriente, por lo que tiene que estar a la altura de las circunstancias y manejar de la mejor manera posible los recursos. En lo que respecta a los controladores de memoria, se debe hacer una eficiente gestión de los procesos, por lo que si no hay memoria suficiente, existe un manejador de memoria que finaliza de manera inteligente aquellos procesos que corren en segundo plano. Cuando un dispositivo presenta poca memoria, la memoria compartida permite tener un mejor soporte y su gestión será más eficiente. Otra parte importante de este kernel es su debugger, que permite mantener de manera correcta el funcionamiento de Android en caso que un tercero haga cambios en el sistema.

Luego está la capa de **librerías**, también llamada capa nativa, que incluye un conjunto de librerías en C/C++ usadas en varios componentes de Android. Están compiladas en código nativo del procesador. Muchas de las librerías utilizan proyectos de código abierto. En lo que respecta a las librerías, está libc, WebKit, SQLite, el framework de media (audio, video y fotografía), el administrador de superficies, librerías 3D y SSL. Libc proporciona funcionalidad básica para la ejecución de las aplicaciones; es una versión adaptada de BSD, pero adaptada para sistemas embebidos de Linux. Luego está WebKit, el motor web utilizado por el navegador, que permite visualizar correctamente contenido HTML. En lo que respecta a SQLite, esta base de datos relacional es usada en Android para almacenar datos y se encuentra disponible para todas las aplicaciones que la requieran. El framework de media soporta codecs de reproducción y grabación de multitud de formatos de audio y video e imágenes. El administrador de superficies maneja el acceso al subsistema de representación gráfica en 2D y 3D.

La capa de librerías contiene una parte dedicada al entorno de ejecución. El componente principal del entorno de ejecución de Android es la máquina virtual Dalvik, componente que ejecuta todas y cada una de las aplicaciones no nativas de Android. Las aplicaciones se compilan en un formato específico para la máquina virtual Dalvik, que es la que las ejecuta.

Sobre la capa de librerías se encuentra la capa de **framework de aplicaciones**. Esta la forman las clases y servicios que usan directamente las aplicaciones para realizar sus funciones. La mayoría de los componentes son bibliotecas Java. Así, la componen el administrador de actividades, de ventanas, el proveedor de contenidos, el administrador de paquetes, el de recursos, las vistas, administrador de telefonía, servicios de localización, entre otros. El administrador de actividades va a controlar el ciclo de vida de las mismas; estas actividades son como las “ventanas” que se muestran en una aplicación. Para mostrar en la pantalla se requerirá del administrador de ventanas, que generará superficies que pueden ser rellenadas por las actividades. El proveedor de contenidos por otro lado permite encapsular los datos que serán compartidos entre aplicaciones, como SQLite y el sistema de archivos por ejemplo. El manejador de paquetes permite obtener información de los paquetes que actualmente se encuentran instalados en el dispositivo. El administrador de telefonía proporciona el acceso a los servicios de telefonía del dispositivo; permite realizar llamadas y enviar y recibir SMS. El administrador de ubicaciones permite determinar la posición geográfica del dispositivo.

Por último, la capa superior está formada por las **aplicaciones**. Se incluye a las aplicaciones del dispositivo, tanto las pre instaladas como aquellas que el usuario instala.

**b) Identifique elementos que son representativos para este tipo de sistemas operativos**

En el caso de este sistema operativo que actualmente lidera el mercado en móviles, tiene un un kernel de Linux que está modificado por Google. Esta modificación se da debido a que dicho kernel debe ser adaptado a los usos que puede llegar a tener un móvil. Android tiene sus propias librerías como son SQLite y Bionic (anteriormente libc) para vincular y compilar. Vale también aclarar que tiene su propio entorno de programación, el runtime con librerías y las máquinas virtuales para poder programar. El contar con estas herramientas permite que un desarrollador tenga a su alcance lo necesario para producir software de calidad, adaptado a las necesidades del usuario.

**c) Realice un comentario general del artículo.**

Según lo visto en las clases de la materia, Android es un sistema operativo híbrido al igual que los sistemas operativos actuales porque la estructura de los mismos no es puramente de un tipo.

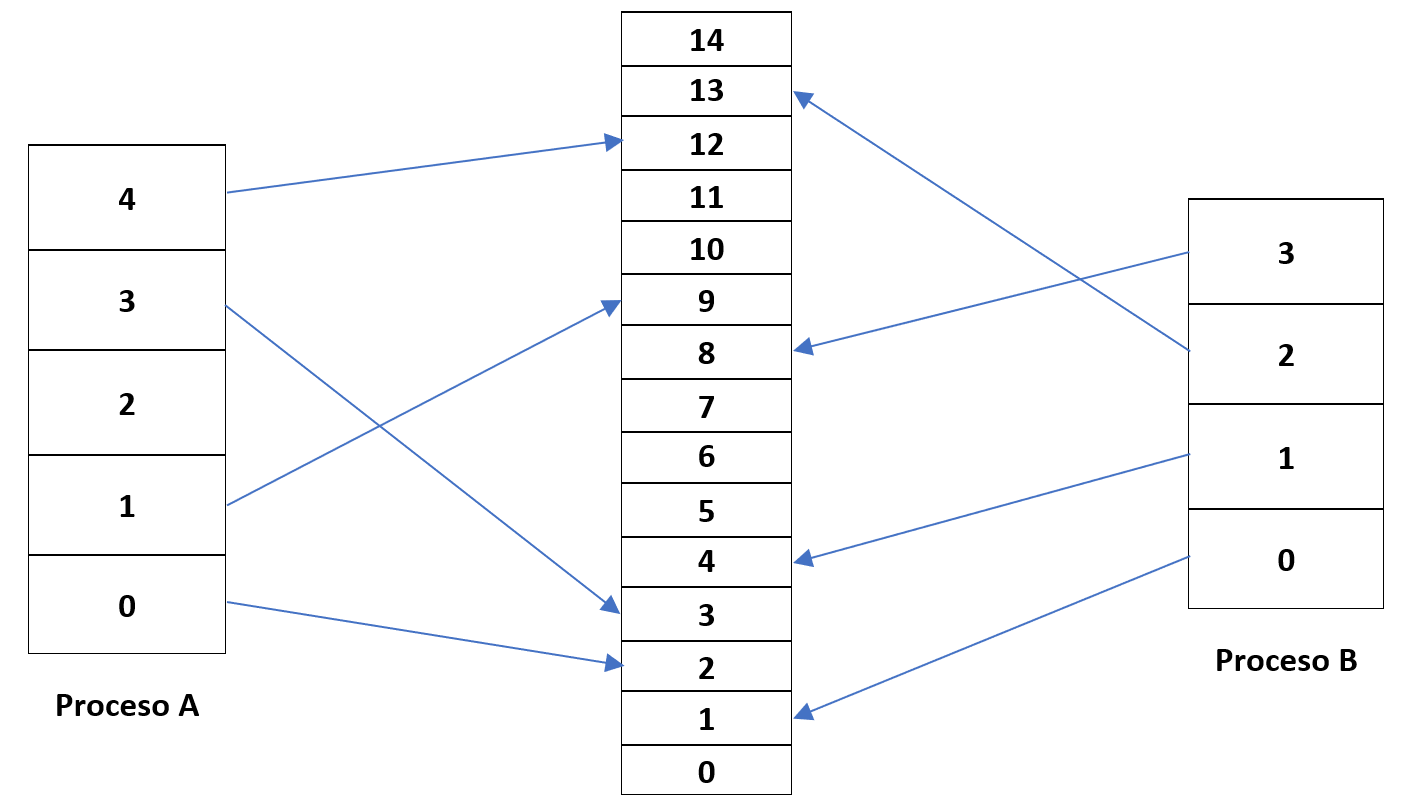
En la última década hubo un avance exponencial en materia de tecnología, que ha ayudado en el día a día de las personas. Tal es así que el uso de dispositivos móviles ha superado en gran cantidad a las computadoras personales en parte porque su utilidad es similar. Con un teléfono inteligente no solo se realiza llamadas o se envía y recibe mensajes de texto como en aquellos viejos dispositivos de la década de 1990 y 2000 sino que llegan a ser una extensión del humano; permite tener (casi) todo en la palma de la mano. Sea un dispositivo con Android o iOS, es posible realizar tareas, trámites, como también entretenerse.

En primer lugar, se puede decir que la arquitectura estructurada en capas en Android permite que cada capa resuelva un único conjunto de problemas, y el usuario final sólo interactuará con la capa “más superficial”, es decir la capa de aplicaciones, centrada en la ejecución, comunicación y estabilidad de toda aplicación preinstalada por cada fabricante como de las aplicaciones que el usuario va a instalar. Esta capa permite tener un gran nivel de comprensión y simplicidad para el usuario. Por otro lado, un desarrollador construirá las aplicaciones que se encontrarán en dicha capa utilizando las herramientas y abstracciones que provee la capa de framework de aplicaciones, la cual simplifica el acceso a componentes de bajo nivel al crear API’s sobre librerías nativas. En la capa de librerías nativas y de entorno de ejecución se utiliza lenguajes de bajo nivel; es la capa que optimiza los dispositivos móviles. Esto garantiza que el código que desarrolla un programador pueda ejecutarse sin problemas a pesar de las limitaciones del dispositivo. En la parte inferior se encuentra la capa perteneciente al kernel de Linux, que interactúa con los componentes de hardware comunes en los dispositivos Android.

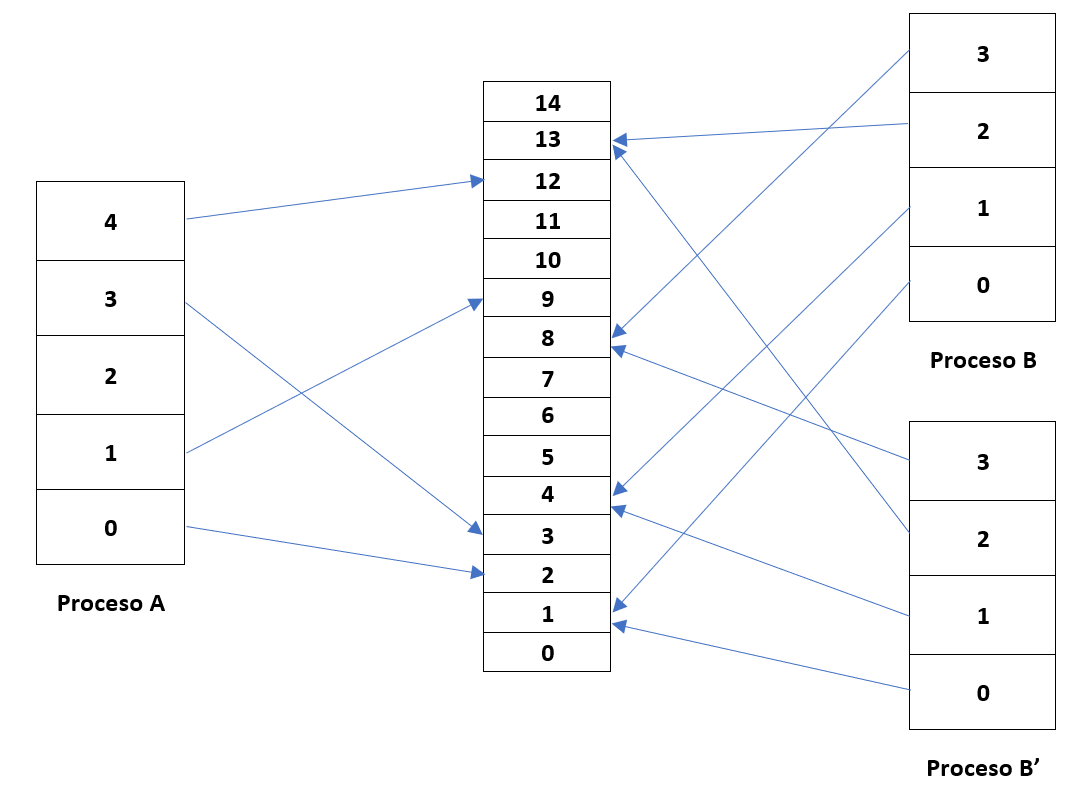
Este sistema de código abierto está presente en un abanico de marcas, las cuales deben adaptarlos a sus dispositivos. Las actualizaciones de seguridad están aseguradas por un tiempo limitado para cada dispositivo, por lo que se puede sufrir de ataques a los que el sistema puede llegar a ser muy vulnerable. En lo que respecta a la seguridad, cabe destacar que toda aplicación requiere de permisos para cada funcionalidad que posea: otorgar permisos que la aplicación no requiera puede ser motivo para preocuparse por la integridad del dispositivo involucrado.

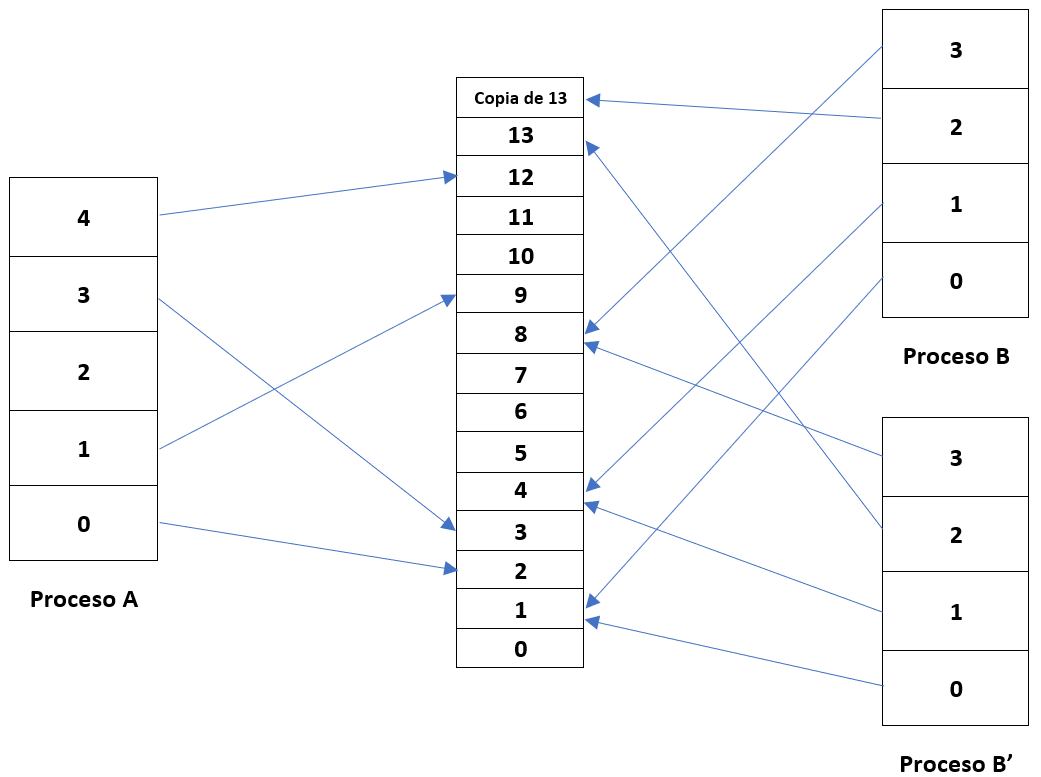
**Problemas Conceptuales**

**1- La figura 1 muestra la asignación de páginas en un sistema Unix que ejecuta los procesos A y B. Las páginas son de 4 KB.**

****

**a) Suponga que el proceso B invoca fork y que el kernel utiliza la estrategia copy-on-write para implementar fork. Realice un nuevo diagrama para la asignación de páginas después de invocar fork y luego de que el proceso B modificó la página 2.**

* **Después de invocar fork()**
* **Después que el proceso B modificó la página 2**



El proceso B modifica la página 2 de su espacio de direcciones, por lo tanto mapea a la dirección física 13 en memoria. Como el proceso B modifica la página 13, duplica la misma.

**b) Construya la tabla de páginas para el padre y el hijo indicando los atributos de cada página (bits de válido y escritura).**

* **Tabla de páginas del proceso padre (Proceso B)**

**Proceso B:**

|  |  |  |
| --- | --- | --- |
| **Página** | **Bit de válido** | **Escritura** |
| **3** | **1** | **0** |
| **2** | **1** | **1** |
| **1** | **1** | **0** |
| **0** | **1** | **0** |

**Proceso B’:**

|  |  |  |
| --- | --- | --- |
| **Página** | **Bit de válido** | **Escritura** |
| **3** | **1** | **0** |
| **2** | **1** | **0** |
| **1** | **1** | **0** |
| **0** | **1** | **0** |

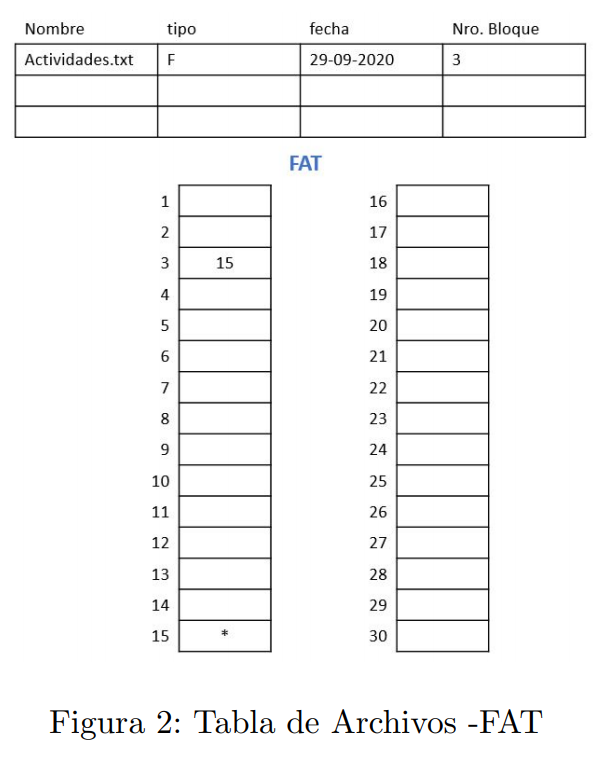
**2- En la figura 2 se representa una tabla FAT. Al borde de sus entradas se ha escrito, como ayuda de referencia, el número correspondiente al bloque en cuestión. También se ha representado la entrada de cierto directorio. Como simplificación del ejemplo, suponemos que en cada entrada del directorio se almacena: Nombre de archivo/directorio, el tipo (F=archivo, D=directorio), la fecha de creación y el número del bloque inicial.**

**Tenga en cuenta que:**

**- el tamaño de bloque es de 512 bytes**

**- el asterisco indica último bloque**

**- todo lo que está en blanco en la figura está libre.**



**a) Completa la tabla FAT con los siguientes datos.**

**a) Creación del archivo Act-Labo1.txt con fecha 10-10-2020, y tamaño de 120 bytes.**

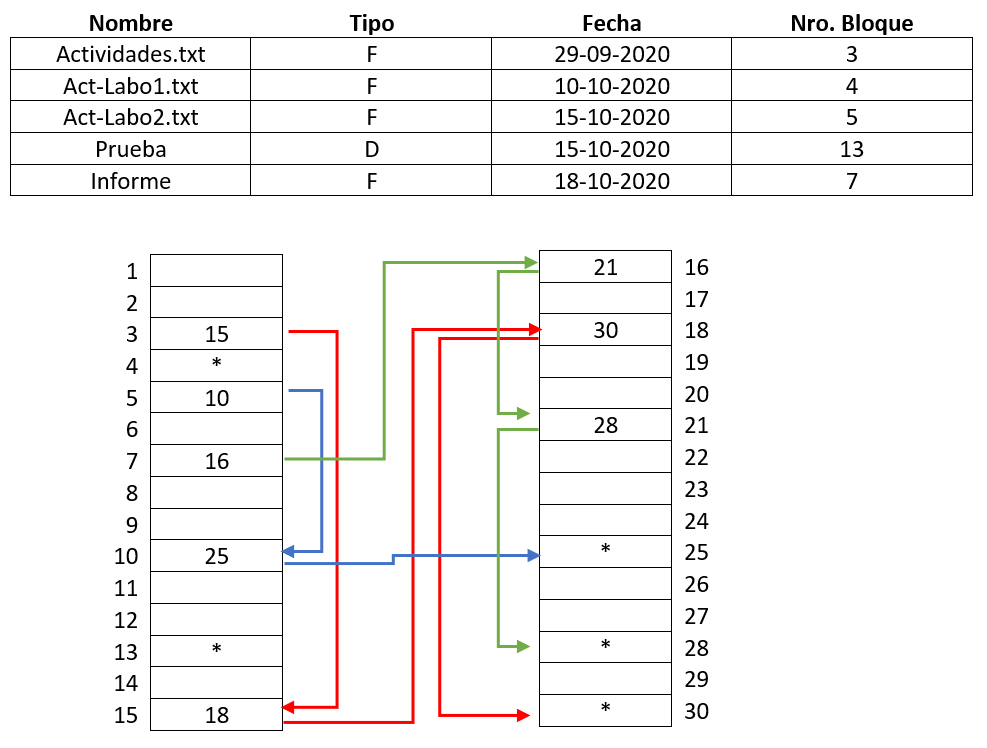
**b) Creación del archivo Act-Labo2.txt con fecha 15-10-2020, y tamaño 1200 bytes.**

**c) El archivo Actividades.txt aumenta de tamaño, necesitando 2 bloques más.**

**d) Creación del directorio Prueba, con fecha 15-10-2020, y tamaño 1 bloque.**

**e) Creación del archivo Informe con fecha 18-10-2020 y tamaño 2 kBytes.**

Consideración del ejercicio: según Silberschatz en la 10º edición de “Fundamentos de Sistemas Operativos”, una desventaja de la alocación enlazada es el espacio que requieren los punteros. Como en el enunciado del ejercicio no indica el tamaño de los mismos, se consideró despreciable.

****

a- El archivo Act-Labo1.txt ocupa 1 solo bloque, el 4

b- El archivo Act-Labo2.txt ocupa 3 bloques (⌈1200B/512B⌉=3)

c- El puntero al final del archivo ahora apunta a el nuevo bloque que se asigna

d- El directorio Prueba solo ocupa un bloque, el 13.

e- El archivo Informe ocupa 4 bloques (2kB/512B=4)

**b) Si utiliza un Mapa de Bits para la gestión del espacio libre, especifique la sucesión de bits que contendría respecto a los bloques dados. ¿Cuánto espacio ocuparía?**

Posiciones de la tabla que están alocadas: 3,4,5,7,10,13,15,16,18,21,25,28,30

Sucesión de bits: 110001011011010010110111011010

Espacio que ocuparía: (⌈(30 bits / 8 bits) \* 1 Byte⌉ = 4 Bytes

**c) Si utiliza una lista enlazada para la gestión del espacio libre, indique cuánto espacio requerirá. Si se pierde el primer puntero de la lista de espacio libre, ¿podría el Sistema Operativo reconstruirla? ¿Cómo?**

Èspacio requerido: 5 direccionamientos \* 17 bits libres = 85 bits

(⌈(85 bits / 8 bits) \* 1 Byte⌉ = 11 Bytes

Cuando se pierde el primer puntero de una lista de espacio libre, el Sistema Operativo podría reconstruirla asignando los grupos en clusters. Según Silberschatz, para poder reconstruir la lista de espacio libre se debería realizar una “recolección de basura”. Esto implicaría buscar en toda la estructura del directorio para determinar qué páginas ya están asignadas y las restantes sin asignar se podrían volver a vincular como una lista de espacio libre.

**3- Suponga que un proceso, P1, abre el archivo Actividad.txt en modo lectura/escritura y otro proceso, P2, abre el mismo archivo y con el mismo modo, y a continuación crea un proceso hijo que abre el archivo /usr/pepe/Informe en modo lectura/escritura. Represente toda la información relevante sobre el estado de las tablas de descriptores de archivos, tabla de archivos y tabla de i-nodos después de dichas operaciones.**

