Proyecto SO 2020

Alumnos: Aitor Ortuño Rossetto 113603., Lucas Cervelli Haderne 119909

A continuación se brindará detalles sobre las implementaciones de los ejercicios provistos por la cátedra:

- BotellasLecheSemaforo:
    - La solución encontrada para el problema de "**demasiadas botellas de leche**" se estructura de la siguiente forma, se cuentan con 3 semáforos (s_comprando (Modela cuando algún compañero va a comprar las botellas de leche al supermercado), s_heladera_llena es un semáforo contador (Modela la cantidad de botellas de leche que hay en la heladera), s_heladera_cerrada (Modela cuando alguien tiene la exclusividad de la heladera)), e hilos que ejecutan la función compañero, el primer compañero que va hacia la heladera se queda con la exclusividad de la misma (haciendo un wait al semáforo heladera_cerrada), revisa si no hay más leches en la heladera(trywait al semáforo heladera_llena), en caso de que no haya leches en la heladera libero la heladera (para que el otro compañero la pueda usar), para verificar si alguien fue a comprar (trywait al semáforo comprando) en caso de que alguien ya haya ido a comprar el compañero se queda esperando(wait heladera_llena) una vez que le avisa el compañero que fue a comprar que ya volvió del supermercado el compañero que se quedó esperando toma una leche de la heladera. En caso de que nadie haya ido a comprar todavía este compañero va a comprar las leches en el supermercado, paso siguiente repone las botellas en la heladera(iterando la cantidad de botellas haciendo post al semaforo heladera_llena), una vez que terminó de reponer avisa que ya volvió de comprar, en caso de que al momento de revisar en la heladera se encuentren botellas de leche, el compañero toma una botella.

    - BotellasLecheCola:
    - La solución encontrada para el problema de "**demasiadas botellas de leche**" ahora con N compañeros, ahora se cuenta con los siguientes procesos:
    - "**botellaLecheCola**" proceso que crea la cola y la llave en la cual se van a ir almacenando los mensajes con los tipos de mensajes que se detallan más adelante, además este proceso crea los procesos de los compañeros y se le cambia la imagen para que ejecuten la implementación de "companiero.c" el proceso padre está a la espera de la finalización de sus procesos hijos, los tipos de mensaje que se utilizan son:
    - "**TYPE_ALGUIEN_COMPRANDO 1L**" tipo de mensaje el cual modela que un compañero fue a comprar botellas de leche en el supermercado.
    - "**TYPE_HELADERA_LLENA 2L**" tipo de mensaje el cual modela la cantidad de botellas de leche que hay en la heladera.
    - Los procesos compañeros modelan el comportamiento de un compañero, el proceso verifica que no haya leche en la heladera, si no hay leche en la

heladera verificará si alguien fue a comprar, en caso de que alguien haya ido a comprar, este proceso se quedara esperando a que repongan una leche(msgrcv del tipo heladera_llena de forma bloqueante) , luego de que repongan al menos una de las leches consume una leche de la heladera, en el caso en que nadie haya ido a comprar aun entonces irá este proceso a comprar luego repone todas las leches en la heladera(msgsnd del tipo heladera_llena) y por último avisa que ya volvió de comprar(msgsnd del tipo alguien_comprando). En el caso en que haya leche en la heladera en el momento en que un compañero fue a revisar, este tomará una botella de leche de la heladera.

- ComidaRapidaSemaforo:
    - La solución encontrada para el problema de la comida rápida con semáforos planteado, está estructurada de la siguiente forma, tenemos 6 semáforos, para la sincronización, y contamos con 4 hilos cada uno implementa una función distinta, que representa a los distintos participantes mencionados en el enunciado, se cuenta además con cuatro funciones las cuales serán realizadas por los hilos antes mencionados., ellas son:
        - “limpiador” Cuando al limpiador le avisan que hay una mesa sucia, el mismo va a la mesa y la limpia.
        - “camarero” Espera a que le hagan un pedido, y busca en la cola de comida y retira un plato de la misma, luego se la entrega al cliente.
        - “cliente” Espera a que este una mesa libre, una vez que se sienta en una mesa realiza un pedido, espera a que le traigan la comida, luego que el camarero le trae el plato, come y se va del restaurante dejando una mesa sucia.
        - “cocinero” Espera a que haya un lugar en la cola de comidas, luego de que haya un lugar libre en la cola de comidas crea una nueva comida y lo agrega a la cola.

II):

- Explique el comportamiento que presenta su solución. ¿Encuentra algún problema?
- Consideramos que un aspecto desfavorable de la solución presentada sería que el cocinero no cocina a demanda, esto llevándolo a la vida real no tiene mucho sentido, otro aspecto negativo es que puede ocurrir que el primer cliente que entra sea el último atendido y por último como otro aspecto desfavorable sería que un cliente que ya comió en el restaurante y se fue vuelve a entrar

    - ComidaRapidaCola:
        - La solución que se encontró para este problema, se estructura de la siguiente manera,:
        - Se cuenta con un struct de mensaje que contiene además del tipo , una variable entera el cual modela el tipo de menú solicitado por un cliente,(Si el tipoMenú es 1 es “**MENU_DE_CARNE**”, y 2

"**MENU_VEGETARIANO**"), gracias a que esta variable entera está en el mensaje el camarero podrá identificar cual es el tipo de pedido del cliente.

- se cuenta con los siguientes hilos:
  - "**Clientes** (la cantidad es 50)" , estos hilos ejecutan la función de cliente (más adelante se detalla sobre esta función)
  - "**Cocineros** (la cantidad es 30)", estos hilos ejecutan la función de cocinero (más adelante se detalla sobre esta función)
  - "**Camareros** (la cantidad es 1)", este hilo ejecuta la función de camarero (más adelante se detalla sobre esta función)
  - "**Limpiador** (la cantidad es 1)", este hilo ejecuta la función de limpiador(más adelante se detalla sobre esta función)
  - Además se cuenta con una cola de mensajes en la cual los hilos escribirán sus mensajes respectivos en cada una de las funciones que ejecutan, la idea que se llevó a cabo es que la cola de mensajes tiene un tamaño máximo fijo de cantidad de platos de cada uno de los tipos de menu disponibles (Carne y vegetariano) en concreto la mitad de la cola serán platos de carne y la otra mitad restante será de platos vegetarianos, (Consideramos que esta implementación es útil ya que la cola siempre contendrá platos de ambos menús evitando así un posible escenario en el cual la cola se encuentre con platos de un solo tipo de menú), luego de esta aclaración se detalla acerca de los tipos de mensajes que se utilizan:
  - " **TYPE_CANT_CARNE_VACIA 1L** " tipo de mensaje el cual modela la cantidad de platos de carne que se pueden meter a la cola.
  - "**TYPE_CANT_VEG_VACIA 2L**" tipo de mensaje el cual modela la cantidad de platos vegetarianos que se pueden meter en la cola.
  - "**TYPE_PEDIDO_MENU 3L**" tipo de mensaje el cual modela el pedido de un menú (carne o vegetariano) de un cliente.
  - "**TYPE_MESA_LIBRE 4L**" tipo de mensaje el cual modela la cantidad de mesas libres del restaurante.
  - "**TYPE_MESA_SUCIA 5L**" tipo de mensaje el cual modela la cantidad de mesas sucias que hay en el restaurante (las cuales van a ser limpiadas por el limpiador).
  - "**TYPE_ENTREGADO_CARNE 6L**" tipo de mensaje el cual modela que el camarero entregó un plato de carne a el cliente que lo solicitó.
  - "**TYPE_ENTREGADO_VEG 7L**" tipo de mensaje el cual modela que el camarero entregó un plato vegetariano a el cliente que lo solicitó.

- "**TYPE_CANT_CARNE_LLENO 8L**" tipo de mensaje el cual modela la cantidad de platos de carne que se encuentran en la cola.
- "**TYPE_CANT_VEG_LLENO 9L**" tipo de mensaje el cual modela la cantidad de platos vegetarianos que se encuentran en la cola.
- Las funciones que son ejecutados por los hilos son los siguientes:
- "**limpiador**" función la cual modela el comportamiento de un limpiador. El hilo esperará a que haya una mesa sucia (realizando un msgrcv del tipo mesa sucia de forma bloqueante), una vez que haya una mesa sucia la limpia y deja un mensaje en la cola del tipo mesa libre, indicando que una mesa más se encuentra disponible en el restaurante.
- "**camarero**" función la cual modela el comportamiento de un camarero. El hilo esperará a que le envien un mensaje del tipo pedido_menu(de forma bloqueante), luego miro si en el mensaje la variable tipoMenu es del tipo "**MENU_DE_CARNE**" , si es así entonces el camarero esperará a que en la cola de comidas se encuentre un plato de carne(con un msgrcv del tipo cant_carne_lleno de forma bloqueante), luego retirara de la cola de comidas el plato de carne, además agrega en la cola un mensaje de tipo cant_carne_vacia indicando que hay un lugar más disponible en la cola de comidas para platos de carne y luego entregará el plato al cliente que lo solicitó, para el caso en que la variable del mensaje tipoMenu es del tipo "**MENU_VEGETARIANO**" ocurre lo mismo que el pedido de carne pero ahora con plato vegetariano.
- "**Cliente**" función la cual modela el comportamiento de un cliente, este esperará a que haya un lugar disponible en el restaurante (realizando un msgrcv del tipo mesa_libre de forma bloqueante) , una vez que se encuentre una mesa libre si el random del tipoPedido es 1 entonces el cliente agrega en la cola un mensaje del tipo "**PEDIDO_MENU**", con la variable del mensaje tipoMenu en "**MENU_DE_CARNE**", luego esperará a que el camarero le traiga el plato que carne(msgrcv del tipo entregado_carne de forma bloqueante), cuando le traen el plato el cliente come su plato y por último deja un mensaje en la cola del tipo mesa_sucia indicando que hay una mesa a la cual limpiar, en el caso en que el random del tipoPedido sea 2 es similar al caso del plato de carne pero con la diferencia que la variable del tipoMenu es "**MENU_VEGETARIANO**" mensaje a insertar en la cola, y que esperara a que el camarero le traiga el plato vegetariano.

- "**Cocinero**" función la cual modela el comportamiento de un cocinero del restaurante, el mismo esperará a recibir un mensaje indicando que hay un lugar en la cola de comidas, una vez recibido el mensaje verifica a qué tipo de plato hay lugar en cola de comida(platos de carne o platos vegetarianos), del tipo de plato que hay disponible para agregar, el cocinero cocinara un plato de este tipo y lo agregara en la cola.

.

- Tareas:
  - La solución encontrada para el problema planteado, está estructurada de la siguiente forma, tenemos un proceso "Coordinador" el cual contiene un random con la cantidad de tareas a realizar (4 , 5 o 6 tareas), este proceso organiza la ejecución por ciclos, dependiendo de la cantidad que le tocó en el random llama a las funciones correspondientes (ejecutarCuatro, ejecutarCinco, o ejecutarSeis), luego el Coordinador está a la espera de que se escriba en el "pipeCoordinador" para indicarle que un ciclo ha finalizado, dando lugar a ejecutar un nuevo ciclo.
  - Las funciones ejecutarCuatro, ejecutarCinco o ejecutarSeis, crean los mensajes que se van a escribir en el pipe correspondiente, luego llamarán a la función inicializarMensaje correspondiente (se cuentan con 3 InicializarMensaje uno por cada tipo (A,B o C)), pasándole el mensaje por parámetro. El ejecutarCinco a diferencia de el ejecutarCuatro y ejecutarSeis (los cuales antes de llamar al método incilizarMensaje para el mensaje B, setea el valor del entero "repeticion" en 2, dado que estos ciclos ejecutan 2 tareas de B), cambia el valor de "repeticion" en 1 dado que el ejecutarCinco solamente ejecuta 1 sola tarea de B.
  - Esta función inicializarMensaje contiene un random con el tipo de tarea a realizar, (si se encuentra en el tipo de mensajeA cuenta además con un random para el color, y si se encuentra en el tipo de mensajeC cuenta además con un random para las ruedas a reparar si el tipo que tocó en el random es reparacion de llanta).
  - EL programa cuenta con 6 semáforos 3 para la sincronización de hilos (s_A, s_B, s_C) y 3 semáforos para indicar la finalización de una tarea(t_A, t_B, t_C), todos ellos inicializados en 0.
  - Además se cuenta con 3 procesos (tareaA, tareaB, tareaC), todos crean 2 hilos para la ejecución de cada tarea una sola vez antes de comenzar el ciclo while logrando una mejor optimización de los recursos reutilizando los hilos ya creados previamente, cada uno de estos procesos leen de su pipe de forma bloqueante, lo que le escribió la función correspondiente (ejecutarCuatro, ejecutarCinco o ejecutarSeis), luego si nos encontramos tanto en la tareaA como en la tareaC haciendo uso del semaforo contador se le envía un post a los hilos para que ejecuten su función, luego de esto la tarea en cuestión se queda esperando a que finalice la ejecución de sus hilos, una vez recibida la señal de finalizacion de sus hilos se escribira en el

"pipeCoordinador" para avisar al Coordinador que se terminaron las tareas de ese ciclo. En el caso de la tareaB también se crea una sola vez sus dos hilos antes de la ejecución del while, luego es similar a la tareaA y tareaC ya que espera a que se escriba en el pipeCoordiandor, pero a diferencia de estos el mensaje del tipo B cuenta con una variable entera "repeticion" el cual indica la cantidad de tareas a realizar por este proceso.

- Minishell:
    - Para la implementación de la shell propuesta por la cátedra adoptamos la siguiente implementación, la shell una vez ejecutada esta a la espera de que el usuario ingrese el comando deseado, una vez ingresado el comando, la shell ejecuta un fork y al proceso hijo se le cambia la imagen ejecutable ("execv") con el archivo vinculado al comando ingresado (por ejemplo si el usuario ejecuta mkdir, el hijo del fork ejecutará el código que se encuentra en ese archivo "mkdir.c"), y el proceso padre está a la espera a que el hijo finalice su ejecución. Consideramos una buena implementación modularizar los diferentes comandos en diferentes archivos, en cada uno de estos archivos se ejecutan usando funciones de librerías.
    - Los comandos aceptados por la shell implementada son los siguientes:
        - "mkdir {nombre del directorio}" crea un nuevo archivo en la ruta donde se encuentra la minishell.
        - "rmdir {nombre del directorio}" Elimina un directorio en la ruta donde se encuentra la minishell. Observación solo elimina directorios vacíos.
        - "open {nombre del archivo}" Crea un nuevo archivo en la ruta donde se encuentra la minishell
        - "ls" Lista todos los archivos y carpetas en el directorio en la ruta especificada
        - "fopen {nombre del archivo}" Muestra el contenido de un archivo en la ruta donde se encuentra la minishell. Observación solo se mostrará el contenido de un archivo sólo si este contiene los permisos de lectura.
        - "man {comando}" Muestra una ayuda y detalles sobre un comando específico
        - "chmod {nombre del archivo}" Modifica los permisos del archivo.

A partir de la lectura del artıculo:
"Android Operating System Architecture"
Realice los siguientes puntos. La evaluación de este punto incluirá el orden y elaboración de las ideas como la presentación de las mismas.

a) Describa los componentes de la arquitectura.

La arquitectura del sistema operativo de android está dividido en 4 capas, ellas son:

1. Kernel
2. Librerías nativas
3. Aplicaciones de Frameworks
4. Aplicaciones

Están compuestas de la siguiente manera:
1. Drivers(pantalla, wifi, bluetooth, binder ipc), manejo de procesos, administración de memoria, manejo de la fuente de alimentación.

- Drivers de dispositivos:
  El kernel de linux tiene un modelo de drivers bien documentado e incluye una gran cantidad de drivers soportando varios CPUs, controladores, buses y dispositivos. Aunque están incrementando constantemente los recursos de los dispositivos móviles, siguen siendo bastante limitados y mantener controlado su uso es necesario en un sistema con estas características para que sean usables.
- Binder driver maneja la comunicación entre procesos (IPC) de manera eficiente. Hace que las llamadas de mensajería sean eficientes (aumentando el rendimiento) a través de la memoria compartida, no hay replicación de objeto ya que los objetos se pasan por referencia(es una de las principales características claves para el uso eficaz de la memoria). Surge la necesidad de tener un Binder debido a que el IPC de Linux tradicional puede introducir sobrecargas de procesamiento y agujeros de seguridad.
- Administración de memoria: Como el SO Android está orientado a correr en dispositivos móviles que si bien han tenido una mejora de sus características de hardware siguen siendo limitados si los comparamos con un ordenador, entonces la memoria que es un recurso muy importante para gestionar los procesos que corren en el sistema necesita optimizarse lo mejor que se pueda. La única forma que tiene de liberar memoria de una app es liberando aquellas referencias a objetos que la app tiene en un momento determinado para que el garbage collector “limpie” esas referencias de la memoria y deje espacio para otros. Debido a esta limitación de recurso, android cuenta con un sistema llamado “low memory killer daemon” el cual monitorea el estado de la memoria de un sistema que está corriendo y reacciona cuando se presenta un alto uso de memoria matando los procesos menos esenciales, o alguna otra política definida por el usuario en el sistema para mantener el uso en niveles aceptables. Esa política es pasada al kernel para también matar procesos en espera en segundo plano.
- Manejo de la fuente de alimentación (batería):
  Android incluye su propio gestor de uso de la batería para cumplir con los requerimientos específicos de un sistema móvil. Por ejemplo algunos gestores introducen los siguientes cambios:
    - Las apps en segundo plano no tienen acceso a redes.
    - Los servicios de ubicación pueden inhabilitarse cuando la pantalla está apagada.
    - Se aplican límites de ejecución en segundo plano a todas las apps, sin importar su nivel de API.

    - El sistema pone las apps en el modo de espera de forma más agresiva, en lugar de esperar que queden inactivas.

2. Librerías Android: SQLlite, WebKit, OpenGL ES, FreeType, Media Framework, etc.
Android Runtime: Core libraries, Dalvik Virtual Machine.

- Android utiliza SQLlite (base de datos relacional) para almacenar los datos que se encuentran disponibles para todas aquellas aplicaciones de recuperación y almacenado de datos en función de la necesidad de estos.
- Dado que GNU libs(glicb) es de gran tamaño y muy complicado para los dispositivos móviles, Android decidió implementar su propia versión de "libc bionica" el cual es de un tamaño muy inferior y menos complicado en comparación con el GNU libs mencionado.
- Android usa "Media framework" para la reproduccion y grabacion de audio, video, y imagenes en muchos formatos , algunos de los que incluye son:
    - **MPEG4**
    - **MP3**
    - **AME**
    - **H.264**
    - **AAC**
    - **PNG**
    - **JPG**
- Android utiliza un componente llamado "SurfaceFlinger" el cual se encarga de aceptar buffers de dos formas: BufferQueue y SurfaceControl, cuando una aplicación pasa a primer plano le pide buffers de WindowsManager, este luego solita a SurfaceFlinger una capa(combinación de una superficie , que contiene BufferQueue, y un SurfaceControl , que contiene los metadatos de la misma), esté la crea y se la envía a WindowManager, cuando recibe la capa se la envía a la app, manipulando la apariencia de la app en la pantalla del dispositivo móvil.
- Android para mostrar contenido HTML utiliza el motor de exploración "WebKit".

3. Administradores de notificaciones, recursos, ventanas. Proveedores de contenido, vista del sistema.

- Administradores de dispositivos:
    - Android utiliza interfaces de programación de aplicaciones (API) para que los desarrolladores de las aplicaciones puedan reutilizar componentes a través del uso de las mismas (estas APIS se encuentran de forma abierta para que cualquier desarrollador las pueda utilizar en caso de necesitarlas para crear apps de Android), además también utiliza servicios de nivel superior en clases de Java.
    - Android utiliza un componente para las apps denominado "Activity" (el cual representa una sola pantalla con una interfaz de usuario), estas activities trabajan en conjunto para poder ofrecer una buena experiencia de usuario. Generalmente en una app de múltiples activaciones, una activity se considera

como la activity principal que es presentada al usuario cuando se inicia la app por primera vez. Cuando se desea realizar diferentes acciones una activity puede iniciar otra activity (la activty anterior guarda el estado en una pila para su uso posterior)y la nueva activity comienza.
    - Un service es un componente de una aplicación que puede realizar operaciones de larga ejecución en segundo plano y que no proporciona una interfaz de usuario. Otro componente de la aplicación puede iniciar un servicio y continuar ejecutándose en segundo plano aunque el usuario cambie a otra aplicación. Además, un componente puede enlazarse con un servicio para interactuar con él e incluso realizar una comunicación entre procesos (IPC).
    - El Content Providers es el encargado de gestionar cómo acceder a los datos de otras apps, estos datos se podrían encontrar en SQLite (base de datos) , o también cualquier otra ubicación de almacenamiento que una aplicación pueda tener fácil acceso. Un proveedor forma parte de una aplicación para Android y a menudo proporciona su propia UI para trabajar con los datos.
    - El sistema Android utiliza una extensión para los paquetes denominada APK (paquete de Android), el package manager realiza un seguimiento de todos los paquetes que se encuentran disponibles en un sistema y los servicios que pueden ofrecer estos paquetes.
    - El Hardware Services de Android interactúa con la capa de abstracción de hardware para acceder a los dispositivos ya que el acceso directo no está permitido existen distintos managers(administradores) para obtener el acceso a esos dispositivos.

4. Aplicaciones nativas de Android, aplicaciones de terceros.

- En la arquitectura Android en la capa de aplicación, se encuentran aquellas aplicaciones provistas por los fabricantes del teléfono en la versión de android que incorporen en sus dispositivos, puede ser una versión de android personalizada con sus propias apps nativas o la versión original de Google, que facilitan a los smartphones a cumplir con las actividades básicas como realizar llamadas, navegar por internet, tomar fotos, etc.
- Adicionalmente, los smartphones permiten instalar aplicaciones desarrolladas por terceros, generalmente son “funciones adicionales” que pueden adquirir, es decir que no son necesarias para cumplir con las funciones básicas pero sí que brindan funcionalidades extras muy útiles y que potencian las posibilidades de los dispositivos al máximo.

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

- Dentro de los elementos que consideramos que son representativos para Android tenemos los siguientes:
    - Low memory killer
    - Dalvik virtual machine(DVM)

- Zygote
- Kernel de linux
- Seguridad

Low memory killer:

- Cuando se ejecutan varios proceso en paralelo en un sistema Android puede darse el caso en que los procesos que requieren más memoria del que está utilizando sufren de retrasos a causa de que la memoria del sistema se agota, para solucionar este problema cuando Android detecta que el sistema tiene una alta presión de memoria elige un procesos y lo mata, el proceso es seleccionado por la política que está definida por el usuario del dispositivo, para que el sistema siga trabajando en unos niveles estables.

Dalvik virtual machine:

- En lugar de usar la JVM, a pesar de Java ser uno de los lenguajes más portables, necesita algo más específico que presente ventajas para sistemas más pequeños, con menor CPU y RAM. Por este motivo se introduce DVM, diseñada específicamente para este tipo de sistemas. Utiliza un formato de archivos específico (.dex) el cual está optimizado para dejar la menor huella posible en la memoria. DVM provee a las aplicaciones portabilidad y consistencia en tiempo de ejecución además de permitir a las apps correr sus propios procesos, con su propia instancia de DVM.

Zygote:

- Es el primer proceso de la máquina virtual en ejecutarse en el booteo del sistema. Para inicializar un nuevo proceso de aplicación, el sistema hace un fork desde Zygote y corre el código de la app en un nuevo proceso. Este enfoque permite que la mayoría de las páginas de la memoria RAM alocadas para el código de un framework o los recursos puedan ser compartidos por todos los procesos de aplicaciones logrando una huella en la memoria más baja y tiempo menor de inicio.

Kernel de linux:

- El kernel en que se basa Android es en el kernel de Linux el cual utiliza una arquitectura monolítica en donde los componentes son interconectados y se encuentran compuestos todos en una sola pieza. Las componentes del núcleo (por ejemplo memoria compartida, unidades del dispositivo, etc) son ejecutados en el kernel en donde no hay aislamiento entre ellos. Una desventaja de la utilización de este kernel es que cualquier kernel del cual se intenta aprovechar al máximo (explotación) permite a un atacante modificar la memoria del kernel pudiendo lograr un efecto muy dañino.

Seguridad:

- El sistema operativo Android no es reconocido por su seguridad, algunos de los más conocidos riesgos de seguridad que hacen a Android vulnerables a ataques

son el tiempo de demora en las actualizaciones de seguridad, una vez que se reporta un bug o debilidad al repositorio público de Android y hasta que se lanza la actualización que soluciona dicho problema el sistema es inseguro, otro riesgo de seguridad es que Android usa el kernel de linux (arquitectura monolítica) puede darse el caso en que cualquier persona que logre tener acceso al kernel tendrá la posibilidad de modificar la memoria del kernel.
Android tiene un problema de seguridad relacionado con el formulario de permisos el cual se solicita al instalar una aplicación, el problema es que no deja denegar acceso a algún servicio que no se quiera permitir el acceso, y en ese caso debe optarse por no instalar la app. Debería haber un sistema o mecanismo alternativo que si no se quiere autorizar ciertos permisos, la app se encuentre en un modo limitado, que permita instalarla y sólo pueda desenvolver aquellas acciones para la que tiene los permisos garantizados.

c) Realice un comentario general del artículo (al menos 500 palabras).

Luego de haber leido el articulo ”Android Operating System Architecture” propuesto por la cátedra procederemos a hacer unos breves comentarios respecto a las observaciones que consideramos nos resultaron relevantes al hacer la lectura del artículo.
Nos parece pertinente destacar que el artículo fue escrito hace 6 años, y si bien las tecnologías móviles empezaban a cumplir un rol muy importante en el mundo interconectado, no es el mismo contexto que se presenta a día de hoy donde los entornos virtuales cobraron una importancia vital en la vida de todo el mundo y sobre todo algo tan importante como el comercio electrónico y la forma de vender o promocionar por internet, especialmente en las apps que utilizamos a nivel mundial en nuestros smartphones guían poco a poco cómo evolucionan. Qué componentes de hardware van variando, y todos los componentes que se agreguen necesitan integrarse al sistema operativo y la forma en que se haga dependerá de su arquitectura y cómo se comunicará con la capa de kernel de Android.
El artículo nos brinda información acerca de el funcionamiento y cómo está compuesta la arquitectura del sistema operativo Android. El enfoque que tiene este sistema operativo es hacia teléfonos inteligentes y tabletas utilizando el kernel de linux como el principal componente base, está orientado a dispositivos de baja potencia los cuales funcionan a base de bateria, este recurso es el principal limitante a la hora hablar de la performance que pueden dar estos dispositivos, tanto celulares como tablets. De todas formas no es el único recurso que restringe qué hacer y cómo hacerlo, por ejemplo para poder mantener el sistema en niveles usables nos llama poderosamente la atención la estrategia utilizada en low memory killer, cómo es que el sistema puede estar tan al borde del colapso o de registrar un nivel de ocupación de los registros de la RAM tan altos que no le queda más remedio que matar un proceso que se encuentra en ejecución.
Los buenos cimientos que genera la arquitectura propuesta hace posible algunos aspectos que son muy ventajosos, uno de ellos por ejemplo es la capa de Frameworks, facilitando el desarrollo de las aplicaciones de una manera más general y haciendo que el sistema operativo sea quien se encargue de establecer la comunicación entre software y hardware, esto permite que los desarrolladores puedan obtener un cierto nivel de abstracción y no

tengan que volverse expertos en cada pieza de hardware y cómo integrarán sus aplicaciones móviles con estos componentes
Una de las principales características que llamó nuestra atención fue que Android no utiliza la JVM sino que utiliza una diferente que es la Dalvik virtual machine ya que como se mencionó anteriormente brinda ventajas con respecto a la anterior, ofreciendo portabilidad , consistencia en tiempo de ejecución y uso optimizado de memoria, logrando que el sistema operativo consuma la menor cantidad de recursos posibles.
Otra característica que destacamos fue cómo Android maneja la comunicación entre procesos (IPC) de manera eficiente, haciendo uso de Binder el cual brinda eficiencia en velocidad y memoria, además de un muy buen rendimiento gracias a la memoria compartida (principal característica del uso eficaz de la memoria).

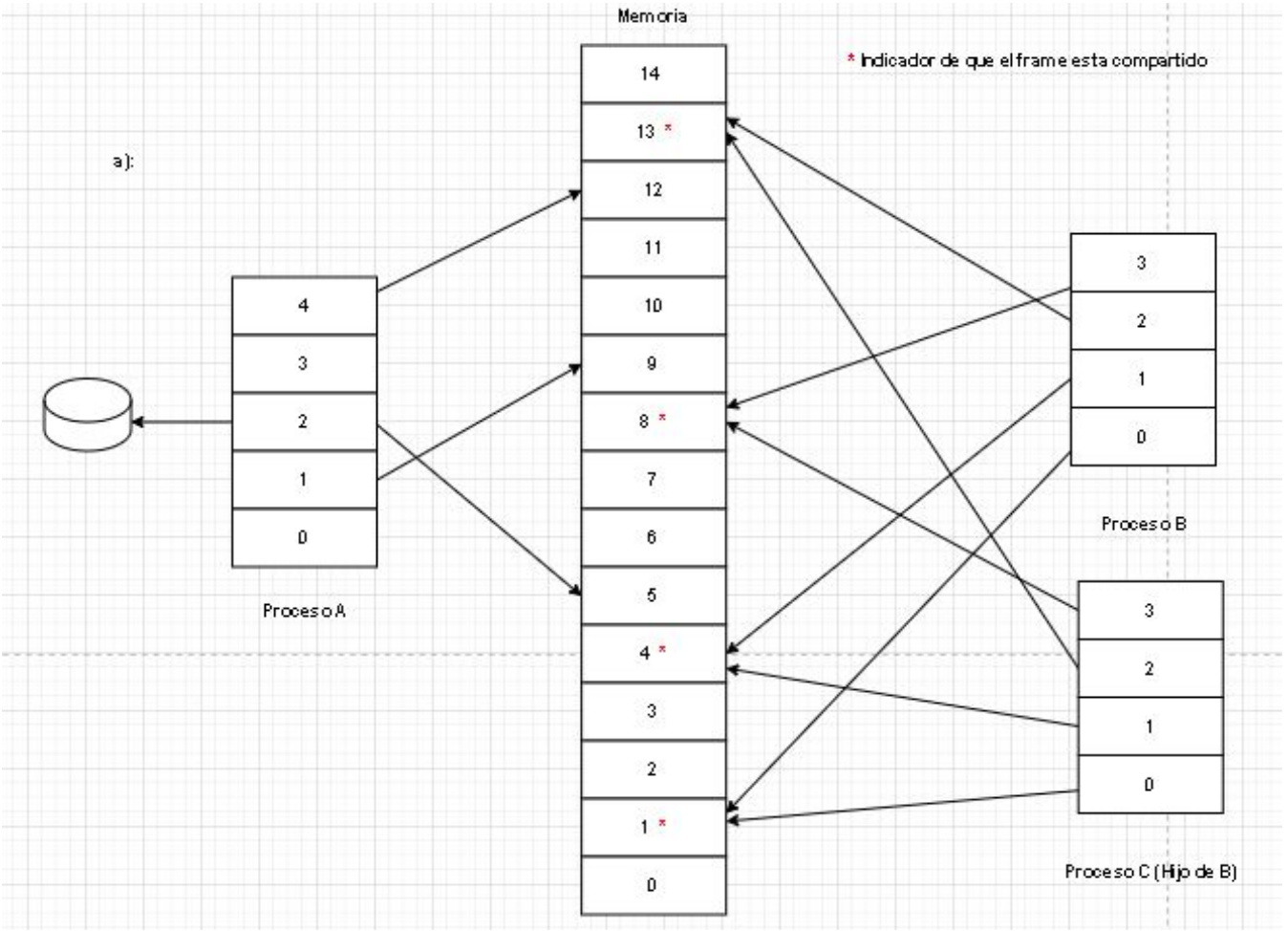
Ejercicios de Memoria del proyecto:
La figura 1 muestra la asignación de páginas en un sistema Unix que ejecuta los procesos A y B. Las páginas son de 4 KB.
a) Suponga que el proceso B invoca fork y que el kernel utiliza la estrategia copy-on-write para implementar fork. Realice un nuevo diagrama para la asignación de páginas después de invocar fork y luego de que el proceso B modificó la página 2.

a):

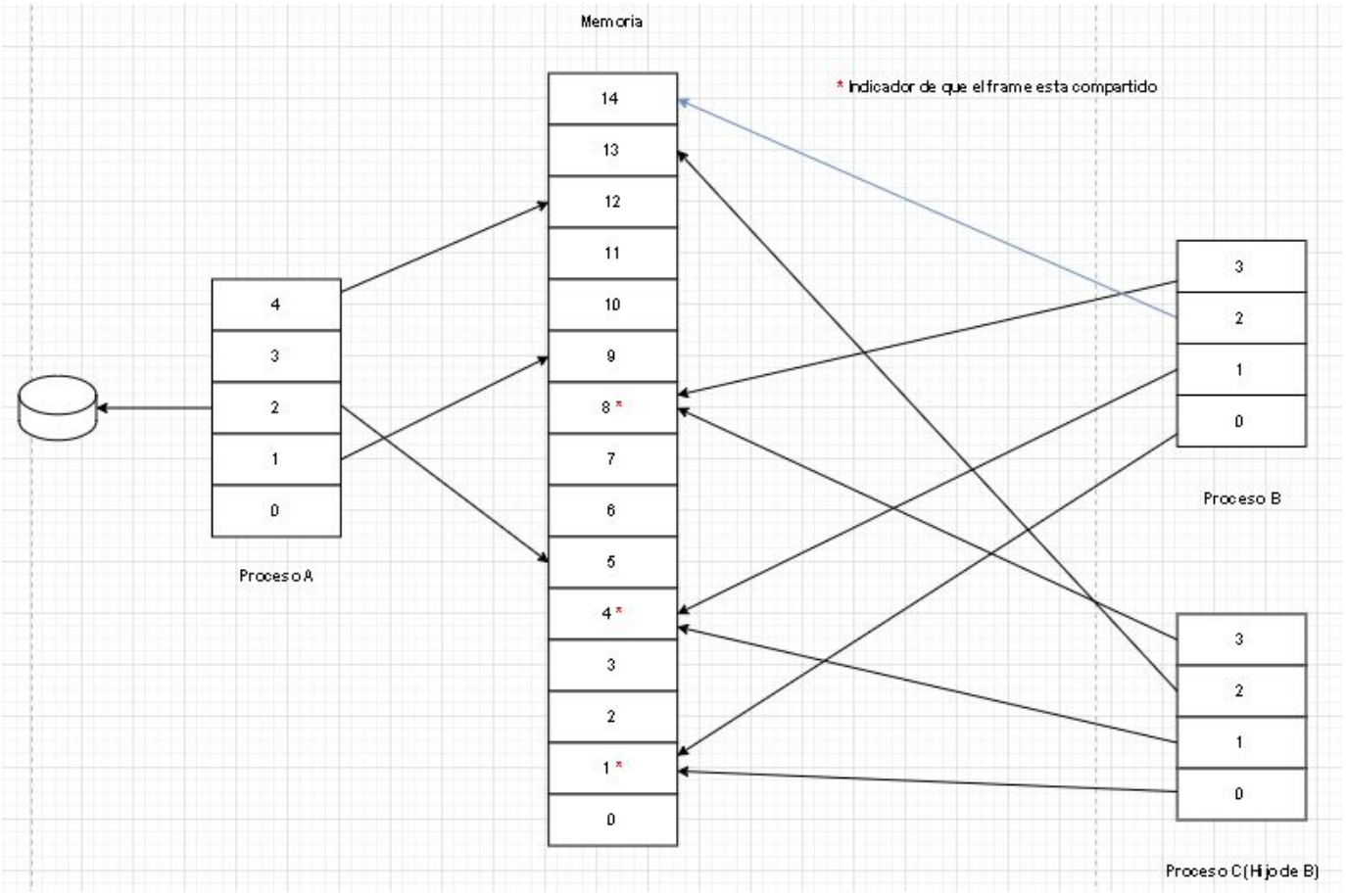
Estrategia copy-on-write:

- En la estrategia copy-on-write cuando se hace el fork se comparten las páginas de un proceso padre al hijo. Cuando un proceso escribe o modifica ese proceso deja de apuntar a la página donde estaba apuntando y ahora apunta a una copia de la página con las modificaciones hechas.

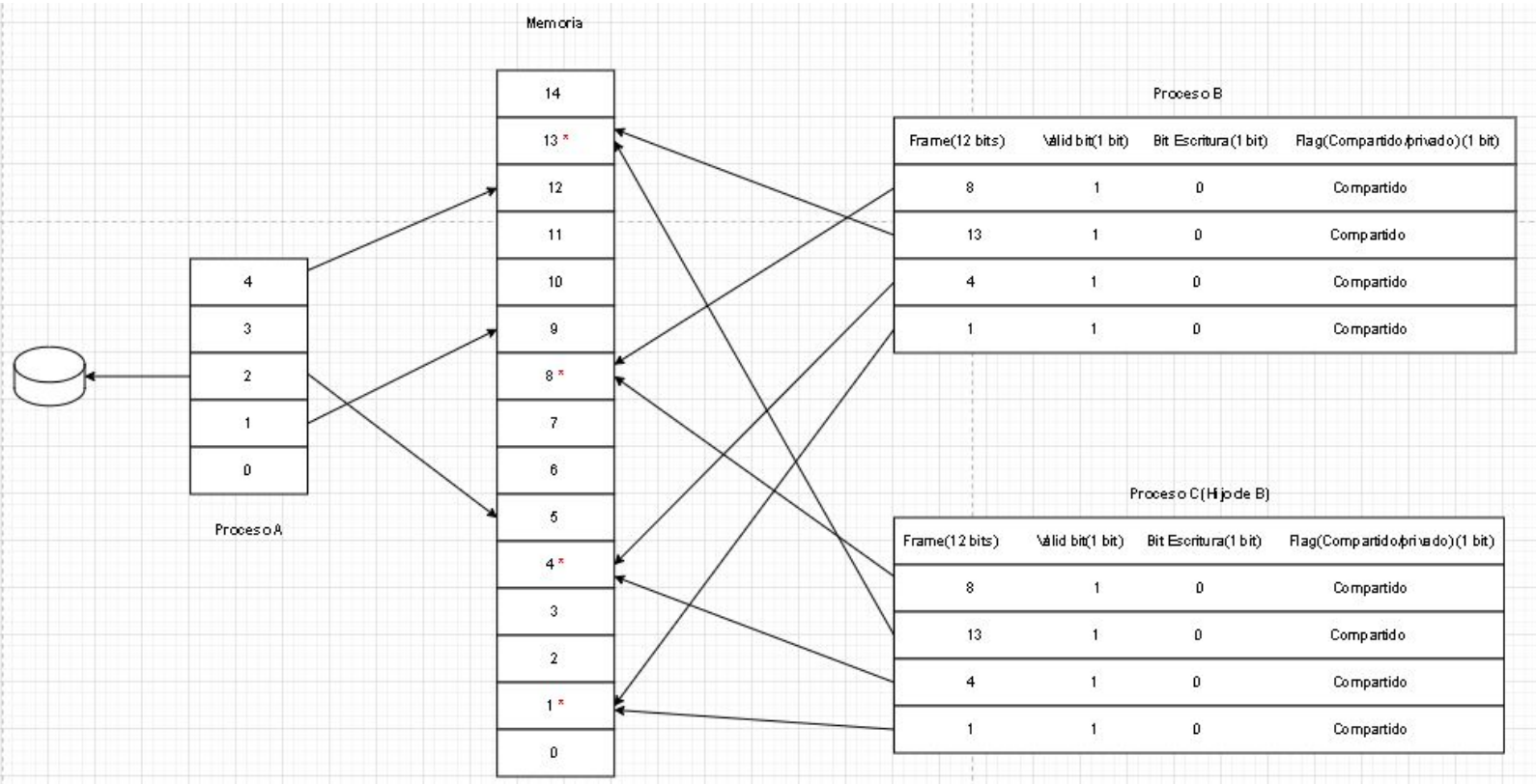
  Cuando se determina que una página va a ser duplicada usando copy-on-write, es importante notar la locación en la cual la nueva página va a ser alocada. Muchos SO proveen un pool de páginas disponibles para tales pedidos. Estas páginas disponibles típicamente se utilizan para cuando el stack o heap deben expandirse.



La página 2 del proceso B apunta a el frame 14 asumiendo que es un frame libre y es una copia de la 13.



b) Construya la tabla de páginas para el padre y el hijo indicando los atributos de cada página (bits de válido y escritura).
Pudimos obtener el tamaño del frame ya que sabemos que el tamaño de una página es 4kb entonces el tamaño del frame pesa lo mismo que el tamaño de una página.



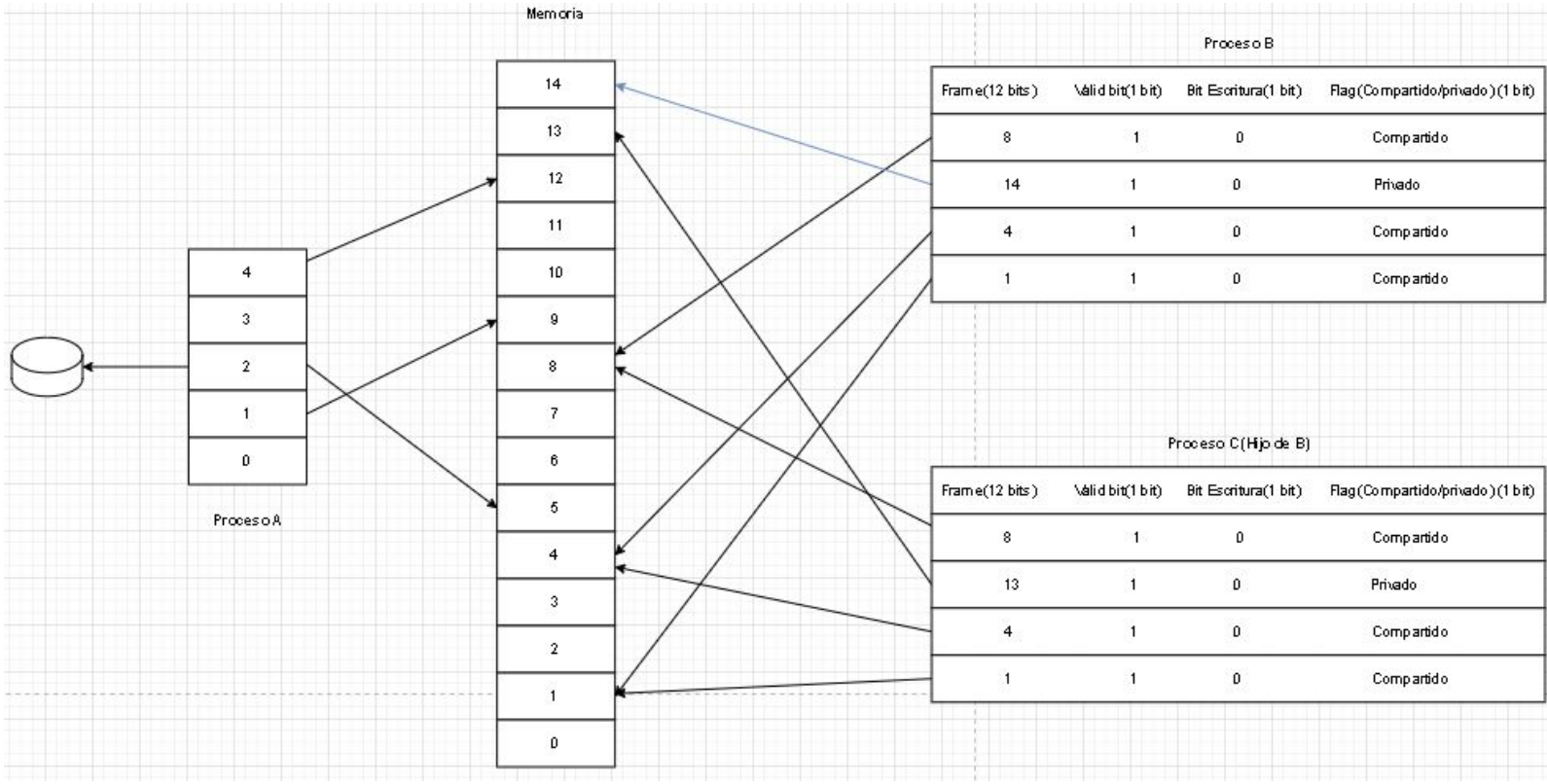
Proceso B

Frame(12 bits)	Valid bit(1 bit)	Bit Escritura(1 bit)	Flag(Compartido/privado)(1 bit)
8	1	0	Compartido
13	1	0	Compartido
4	1	0	Compartido
1	1	0	Compartido

Proceso C(Hijo de B)

Frame(12 bits)	Valid bit(1 bit)	Bit Escritura(1 bit)	Flag(Compartido/privado)(1 bit)
8	1	0	Compartido
13	1	0	Compartido
4	1	0	Compartido
1	1	0	Compartido

Para que el proceso B pueda modificar la página 2, tiene que buscar un frame libre para hacerla la copia del frame 13, (ya que tanto el proceso B como el proceso C referencian a el mismo frame), en este caso un frame libre sería el frame 14, entonces los valores de los flags compartido/privado cambian su valor a privado (en ambos procesos), dado que ahora el frame 13 y el frame 14 ya no son compartidos por más de un proceso.
Las tablas de páginas de los procesos A y B quedan de las siguiente manera:



Adjuntamos un pdf con el nombre “ejerciciosMemoria.pdf” con las imágenes que se mostraron antes por si no se llega a ver bien las imágenes.

2):

a)

Nombre	tipo	fecha	nro bloque
Actividades.txt	f	29-09-2020	3
Act-Labo1.txt	f	10-10-2020	1
Act-Labo2.txt	f	15-10-2020	2
Prueba	d	15-10-2020	6
Informe	d	18-10-2020	7

.

FAT

1	*	16	17
2	4	17	*
3	15	18	
4	5	19	
5	*	20	
6	*	21	
7	8	22	
8	9	23	
9	10	24	
10	11	25	
11	*	26	
12		27	
13		28	
14		29	
15	16	30	

Se tomo en cuenta el tamaño del puntero el cual es **4Bytes**

Observación:

- Asumimos que el tamaño del puntero es de **4Bytes**.
- Asumimos que el tamaño del EOF es muy pequeño y no necesita de un bloque aparte solo para marcarlo.

b):

Sucesión del bitmap: **00000000000111000111111111111**

Con tener 30 bits sería suficiente espacio para almacenar el bitmap.

Bit en 0 → Espacio alocado

Bit en 1 → Espacio libre

c):
Si se utiliza una lista enlazada para la gestión del espacio libre, ocuparía un espacio de:
4 Bytes
(Asumiendo que el tamaño del puntero pesa 4 bytes), dado que el primer puntero apunta al primer bloque libre y este ya apunta al próximo libre, esto es así gracias a que se aplica sobre una tabla “FAT”.

Si se pierde el primer puntero de la lista de espacio libre, el sistema operativo podría reconstruirla pero esto implicaría realizar una"recolección de residuos" o pasar el garbage collector lo cual requiere buscar en la estructura entera del directorio para determinar cuales paginas ya estan alocadas. Aquellas que permanecen sin alocar pueden ser reconectadas como los componentes libres de lista.

3):