1. Experimentación de Procesos y Threads con los Sistemas Operativos

1.1. Procesos, Threads y Comunicación

1.1.1. Conjunto de Tareas

El programa tiene un coordinador principal (el main) que se encarga de crear 3 procesos, coordinador A, coordinador B y coordinador C. Cada uno de estos coordinadores se encarga de crear 2 hilos, cada uno con su struct correspondiente y dichos hilos ejecutan la función correspondiente, si los crea el coordinador A, sus hijos ejecutan la tareaA y así sucesivamente.
Los coordinadores de tareas y los hilos están siempre ciclando, esperando por nuevas tareas.
El coordinador principal, dependiendo del caso en el que se encuentre,manda mediante pipes los mensajes correspondientes a los respectivos coordinadores de tareas. Dichos coordinadores, reciben a través de los pipes los mensajes que les corresponden y actualizan los structs que tienen sus hilos para pasarles la información nueva. Los hilos están sincronizados con semáforos, es decir que cuando el coordinador envie nueva información a través del struct, con un semáforo habilitará al hilo a ejecutar la tarea y cuando el hilo termine la tarea, le avisará al coordinador que terminó, también con un semáforo.
Cuando los coordinadores de tareas saben que su hijo terminó, le envían un mensaje al coordinador principal mediante un pipe y el coordinador principal no comenzará un caso nuevo hasta que todos los coordinadores de tareas le avisen que sus respectivos hilos completaron su tarea.

1.1.2. Mini Shell

Se utilizan ejecutables mediante la operación execv para ejecutar los comandos ingresador por consola.
Para saber qué tipo de comando se ingresó se asume que se puede ingresar hasta 3 cadenas de caracteres separadas por un espacio entre cada una de ellas. Con el comando "ayuda" se obtiene información sobre cada comando.

- mkdir [nombre del nuevo directorio]
- mkfil [nombre del nuevo archivo] Crea un archivo con permisos 0777 por defecto.
- modle [nombre del archivo] [nuevos permisos] los nuevos permisos se introducen en formato numérico.
- rmdir [nombre del directorio a borrar]
- showd [nombre de la carpeta] el nombre de la carpeta a revisar debe estar acompañado por la ruta para poder acceder a su contenido.
- showf [nombre del archivo] el nombre del archivo a revisar debe estar acompañado por la ruta para poder acceder a su contenido.

1.2. Sincronización

1.2.1. Demasiadas Botellas de Leche

1.2.1.a

Se crean los 2 semáforos y los 2 hilos “compañeros”. Inicialmente tenemos el semáforo de leches en 4 y el semáforo de compra en 1. Cuando un compañero abre la heladera (toma el mutex lock) se fija si hay leche:

- Si hay leche, consume una y libera la heladera (libera el mutex lock).
- Si no hay leche, se fija con el semáforo comprando si hay alguien comprando
    - Si no hay nadie comprando (el semáforo comprando está en 1), libera la heladera y va a comprar.
    - Si hay alguien comprando, libera la heladera
- El que fue a comprar, cuando llega a la casa con las leches avisa que ya volvió de comprar(vuelve a poner el semáforo de compra en 1), abre la heladera y guarda las leches (vuelve a poner el semáforo de las leches en 4), cuando termina libera la heladera.

1.2.1.b

Se crea la cola de mensajes y los N procesos “compañeros”. Inicialmente se envían 4 mensajes de Tipo_Leche a la cola y un mensaje Tipo_Compra. Cuando un compañero abre la heladera(se fija si hay algun mensaje de tipo Mutex_Lock) se fija si hay leches(si hay algún mensaje Tipo_Leche):

- Si hay leche, consume una y libera la heladera(carga un mensaje de tipo Mutex_Lock)
- Si no hay leche, se fija si hay alguien comprando
    - Si no hay nadie comprando (hay un mensaje de Tipo_Compra), libera la heladera y va a comprar.
    - Si hay alguien comprando, libera la heladera.
- El que fue a comprar, cuando llega a la casa, avisa que llego de comprar (carga un mensaje de Tipo_Compra en la cola), abre la heladera y guarda las 4 leches en la heladera (carga 4 mensajes de Tipo_Leche en la cola).

1.2.2. Comida Rápida

1.2.2.b

Se utilizan 7 semáforos para sincronizar las tareas entre los clientes, chefs, el mesero y el limpiador de mesas. Así mismo se crean 50 hilos para los clientes y 3 hilos para los cocineros.

Se inicializan los semáforos de la siguiente manera: cola en 10 lo cual indica que hay 10 espacios disponibles para crear platos de comida, mesas en 30 indicando que hay 30 mesas disponibles, clienteSeLevanta, pedidoParaCamarero, limpiar, comida y plato se inicializan en 0.

Cuando un cliente consigue sentarse en una mesa, esto es, el semáforo mesas tiene un valor entero mayor a cero, hace un pedido al camarero incrementando pedidoParaCamarero después esperan por plato. Cuando recibe su plato de comida come y se levanta avisando al limpiador que limpie la mesa incrementando clienteSeLevanta.

Los cocineros siempre están esperando al semáforo cola para poder cocinar, cuando terminan de hacer un plato le avisan al camarero que tiene un plato listo para llevar.

El camarero siempre está esperando por nuevos pedidos de los clientes, cuando recibe uno lo atiende y espera que los cocineros le preparen el plato, una vez que el plato está listo el camarero lo toma y agrega un lugar en la cola para nuevos platos, después le lleva el plato al cliente.

El limpiador siempre está esperando a que un cliente le avise que se levanta para poder limpiar la mesa. Cuando recibe la orden, limpia la mesa y la libera para futuros clientes.

1.2.2.c

Se utilizan 3 colas de mensajes una para los clientes, otra para las comidas y la tercera para el camarero. También se usan 3 hilos para cocineros y 50 para clientes. Se envían 30 mensajes de Tipo_Mesas a la cola de clientes para indicar que hay 30 mesas disponibles, 5 mensajes de Tipo_carne_Cocinar y 5 de Tipo_Vegetal_Cocinar a la cola de comidas para que los cocineros preparen 5 platos de carne y 5 platos vegetarianos.

Los clientes siempre esperan por un mensaje de Tipo_Mesas para poder sentarse en una mesa. Después eligen aleatoriamente entre un plato de carne o uno vegetariano, envían un mensaje a la cola del camarero indicando el tipo de plato que desean Tipo_Vegetal_Pedido o Tipo_Carne_Pedido y esperan por su plato. El plato les llega en la cola cliente y puede ser de Tipo_Vegetal_Listo o Tipo_Carne_Listo según lo que ordenaron. Una vez que tienen el plato, comen y se levantan enviando un mensaje de Tipo_Limpiador a la cola cliente para que limpien la mesa.

Los cocineros siempre esperan mensajes en la cola comidas, y preparan un plato de carne si el mensaje es de Tipo_Vegetal_Cocinar o vegetariano si el mensaje es de Tipo_Vegetal_Cocinar. Cuando terminan de preparar el plato, envían un mensaje a la cola cliente de Tipo_Vegetal o Tipo_Carne según el pedido.

El camarero espera por pedidos del cliente en la cola camarero y estos pueden ser de Tipo_Vegetal_Pedido o Tipo_Carne_Pedido, cuando los recibe se fija si en la cola cliente hay mensajes del tipo de plato que el cliente ordenó. Cuando obtiene el plato se lo entrega al cliente por medio de la cola cliente y ordena que se haga otro plato del mismo tipo que entregó usando la cola comidas.

El limpiador siempre espera que le den un mensaje de Tipo_Limpiador para que limpie la mesa. Cuando le llega el mensaje, limpia la mesa y la habilita mandando un mensaje a la cola cliente de Tipo_Mesas.

2. Problemas

2.1. Lectura

2.1.1. Android Operating System Architecture

a)

Al hablar de la arquitectura de Android podemos dividir dicho tema en 4 capas principales:

1. ***La capa de kernel***
2. ***La capa de librerias nativas***
3. ***La capa de los frameworks de aplicaciones***
4. ***La capa de aplicaciones***

La capa de kernel provee funcionalidades básicas del sistema como el manejo de procesos, el manejo de memoria, el manejo de dispositivos (incluyendo la cámara, la pantalla, el teclado,etc). El kernel utilizado para en Android OS es un kernel Linux 2.6 ya que Linux es muy bueno en operaciones básicas pero igualmente Google mejoró dicho kernel para manejar de mejor manera las necesidades de los móviles, como el manejo de memoria,el de procesos, el manejo de energía, un mecanismo especial para la intercomunicación de procesos en ejecución y un mejor uso de los recursos limitados del sistema. Algunas de las mayores mejoras son las siguientes:

- **Alarm Driver:** El kernel de android lo utiliza para realizar ciertas operaciones y planificar adecuadamente cuando “despertar” una aplicación dependiendo de los eventos que tomen lugar.

- **Binder:** Se implementó el binder para manejar de manera más eficiente la intercomunicación de procesos ya que consumen muchos recursos. Además, hace que las llamadas de mensajes entre procesos sean más rápidas y eficientes al utilizar memoria compartida. Otro motivo para su implementación fue lo inseguro que era el IPC tradicional de linux.

- **Power Managment**
- **Low Memory Killer:** Esta mejora es manejada por el manejador Out of memory (OOM). Cuando un dispositivo se está quedando sin memoria, este elige un proceso y lo elimina, liberando la memoria utilizada. El proceso a eliminar es seleccionado por el kernel quien elimina los procesos en segundo plano que se estén ejecutando pero que no se estén utilizando.

- **Kernel Debbuger**: Tanto android como linux son de código abierto y cualquiera puede realizar cambios en ellos. Por este motivo, para mantenerse al tanto de que los cambios funcionen correctamente se provee con este debbuger.
- **Logger:** Carga en el sistema grandes mensajes con el propósito de generar problemas incluyendo el logger buffer principal, el logger buffer de eventos, el de radio y el buffer del sistema.

- **Ashmen:** Se trata de la memoria compartida de android, la cual fue agregada en el kernel para facilitar compartir memoria y la conservación, la cual provee mejor soporte para dispositivos de poca memoria.

La capa de librerías nativas provee librerías que le permite a los dispositivos manejar diferentes tipos de información específica del hardware. Esta capa está dividida en dos partes, una es Android Library y la otra es Android Runtime. Las librerías en Android Library están escritas en C++ y se encargan de realizar todas las tareas pesadas para proveer mucho poder a la plataforma android.
Algunas de las librerías más importantes son:

- **Libc:** Android implementa su propia versión de la librería bionic libc la cual es pequeña en comparación con la librería GNU libs.
- **SQLite:** Es una base de datos relacional utilizada para guardar información que utilizan las aplicaciones según la necesiten.
- **Media Framework:** Provee grabación y reproducción de una gran cantidad de formatos de audio, video e imágenes.
- **Surface Flinger**: Provee un manejador de renderizado que combina superficies 2D con superficies 3D.
- **Webkit:** Es un motor de búsqueda que se utiliza para mostrar contenido HTML.

La segunda parte, Android Runtime, consiste en la Máquina Virtual Dalvik (DVM) y librerías java centrales. Todas las aplicaciones de java y la mayoría de sus frameworks están escritos en Java. En vez de utilizar la Máquina Virtual de Java (JVM), android utiliza su propia DVM que está diseñada para sistemas pequeños que proveen poca memoria RAM y un procesador lento. DVM tiene su propio código en formato byte que se ajusta a las necesidades de los dispositivos con android. Este código está más comprimido que el típico código en formato byte de Java. Además, ejecuta archivos ejecutables Dalvik (.dex) que está optimizado para un uso mínimo de memoria. DVM provee portabilidad, consistencia en ejecución y permite que cada aplicación corra su propio proceso con su propia instancia de DVM.

La capa de los frameworks de aplicaciones provee una gran cantidad de interfaces de programación de aplicaciones (API's) y mejores servicios en forma de clases Java. Estas API's están disponibles para todo el mundo que quiera crear aplicaciones android. Hay diferentes tipos de componentes de aplicaciones y cada uno tiene un ciclo de vida distinto y un propósito distinto. Alguno de estos componentes son:

- **Activity:** Representa una pantalla única con interfaz de usuario. En las aplicaciones, las actividades trabajan en conjunto para mejorar la experiencia del usuario. Siempre hay una actividad principal que se le muestra al usuario cuando corre la aplicación por primera vez y para realizar otras acciones, está actividad principal puede iniciar otras actividades. Cuando una nueva actividad comienza, la anterior se detiene y se guarda en una pila con su estado actual, para algún uso posterior.

- **Servicies:** Un servicio es un componente que corre en segundo plano y no provee una interfaz. Los servicios realizan operaciones largas en ejecución y trabajan para procesos remotos. Además, los servicios corren en el hilo principal de la aplicación por defecto. Una actividad puede conectarse a un servicio en ejecución para comunicarse con otros servicios

- **Content Providers**: Maneja cómo acceder y compartir información desde otras aplicaciones como podría ser la base de datos SQLite, el archivo de sistema o cualquier otra locación de guardado de información persistente.

- **Package Manager:** Android utiliza una extensión especial para los paquetes llamada APK(Android Package). Este componente mantiene información de todos los paquetes disponibles en el sistema y los servicios que los mismos ofrecen.

- **Window Manage**r: Se encarga de dibujar las diferentes ventanas en pantalla para la interacción mientras que Surface Flinger maneja el output de la pantalla.

- **Hardware Services**: Interactúa con la capa de abstracción de hardware para acceder a los dispositivos.

- **Telephony Services:** Maneja todas las actividades relacionadas a llamadas de teléfono y mensajes de texto.

- **Location Services**: Es utilizado para manejar la ubicación del dispositivo. Utiliza el GPS o las torres de teléfono para brindar la ubicación del dispositivo.

La capa de aplicaciones es la capa más visible, contiene aplicaciones pre-instaladas como el marcador de teléfono, la aplicación de mensajes, algún buscador predefinido, etc. La mayoría de los usuarios interactúan con esta capa, incluso descargan aplicaciones desde la Play Store dependiendo de sus necesidades.

b)

Los elementos más representativos para este tipo de sistemas son:

- Dalvik Virtual Machine (DVM)
- Application Framework
- Optimized Graphics
- Integrated Browser
- SQLite
- GSM Technology
- Edge
- 3G
- 4G
- Media Support
- Camera
- Bluetooth
- WiFi

c)

El artículo en sí nos pareció interesante, ya que nos brindó información sobre muchísimas partes de Android que desconocíamos. En un principio brinda información general sobre el sistema operativo Android desarrollado por la compañía Google, cómo está evolucionando el uso de teléfonos móviles en los últimos años y como se utilizan 3.5 veces más los dispositivos móviles que las PC's. Principalmente Android OS está desarrollado principalmente para teléfonos móviles y tablets. Pensado para dispositivos con una batería que se les agota rápidamente y equipado con hardware de posicionamiento global (GPS),WiFi,UMTS, pantalla táctil, entre otras cosas.

Las aplicaciones Android son, en su mayoría, desarrolladas usando el lenguaje de programación Java utilizando Android Development Kit (SDK) el cual incluye un debbuger, librerías, QEMU, tutoriales y documentación. Si bien hemos programado en Android, principalmente con Android Studio, nunca nos habíamos adentrado en el tema más allá de la parte de programación de aplicaciones móviles.

No sabíamos que se habían realizado tantos cambios en el kernel de Android para poder obtener el sistema operativo que es hoy en día, inclusive lo que más nos sorprendió fue el Binder ya que sin esta mejora de kernel la intercomunicación entre procesos sería muy pesada ya que consumiría muchos recursos y no hubiera sido posible la creación de los primeros smartphones con Android, los cuales tenían poca memoria RAM y un procesador muy lento.

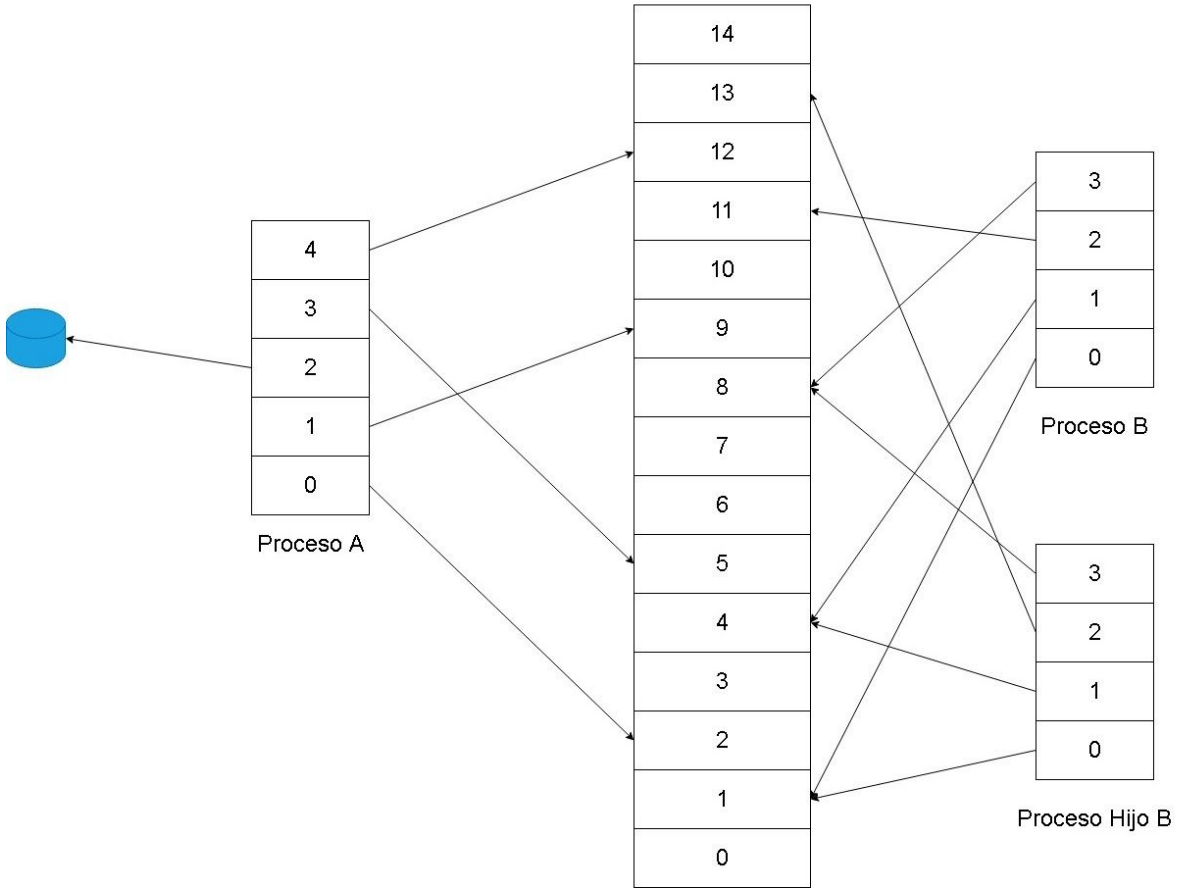
Estábamos al tanto de los problemas de la seguridad de Android ya que al ser un sistema de código abierto que puede ser corrido por múltiples dispositivos es lógico que existan grandes vulnerabilidades, es el típico trade-off de la computación. Una gran vulnerabilidad de Android es que al estar basado en el kernel de Linux implementa una arquitectura monolítica en la que los componentes están interconectados y acoplados en una misma pieza. Muchos desarrolladores de dispositivos android usualmente utilizan drivers personalizados para su hardware y eso puede provocar fallas de seguridad ya que dicho código de los drivers no son controlador y testeados por la comunidad de Linux.

El hecho de rootear un dispositivo móvil también puede comprometer la seguridad del mismo. Algunas aplicaciones o usuarios rootean sus dispositivos, es decir que le/se brinda al usuario permisos de administrador, y esto hace que su dispositivo deje de ser seguro ya que se rompe la barrera de integridad del kernel. Esto causa que el kernel modificado pueda deshabilitar las medidas de seguridad de android y así infectar el dispositivo con algún malware que cause que se filtre información privada. En comparación con otros sistemas operativos móviles como Apple's iOS o Black Berry OS, Android no es seguro. Esto no quiere decir que android sea completamente inseguro, solamente es más vulnerable a fallos en seguridad comparado con otros sistemas operativos. Pero eso también es lo que lo hace un sistema operativo tan versátil, que inclusive uno como programador puede crear su propia ROM, probarla en su propio móvil o descargar una ROM y modificar el sistema operativo a gusto propio.

2.2. Problemas Conceptuales

2.2.1

a)



b)

Tabla de Paginas
Proceso B

	Frame	Bit de Valido	Bit de Escritura
0	1	1	1
1	4	1	1
3	8	1	1
2	11	1	0

Tabla de Paginas
Proceso Hijo B

	Frame	Bit de Valido	Bit de Escritura
0	1	1	1
1	4	1	1
3	8	1	1
2	13	1	0

2.2.2

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	8
Informe	F	18-10-2020	10

FAT

1	*
2	4
3	15
4	5
5	*
6	7
7	*
8	9
9	*
10	11
11	12
12	13
13	14
14	*
15	6

16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	

b)
La sucesión de bits que contendrá el mapa de bits es : 00000000000000011111111111111

El mapa de bits ocuparía un bloque ya que se necesitará 1 bit por bloque en la tabla FAT, es decir un total de 30 bit.

c)
Si se utilizara una lista enlazada para la gestión de espacio libre, ésta requerirá el espacio de un puntero por bloque libre. Como tenemos 15 bloques libres y tenemos punteros de 1 bytes, serían en total 15 bytes.

Sería posible reconstruirla, pero sería muy costosa, ya que se debería recorrer toda la memoria y el SO tendría que ir viendo qué bloques están libres para ir reconstruyendo la lista.

2.2.3