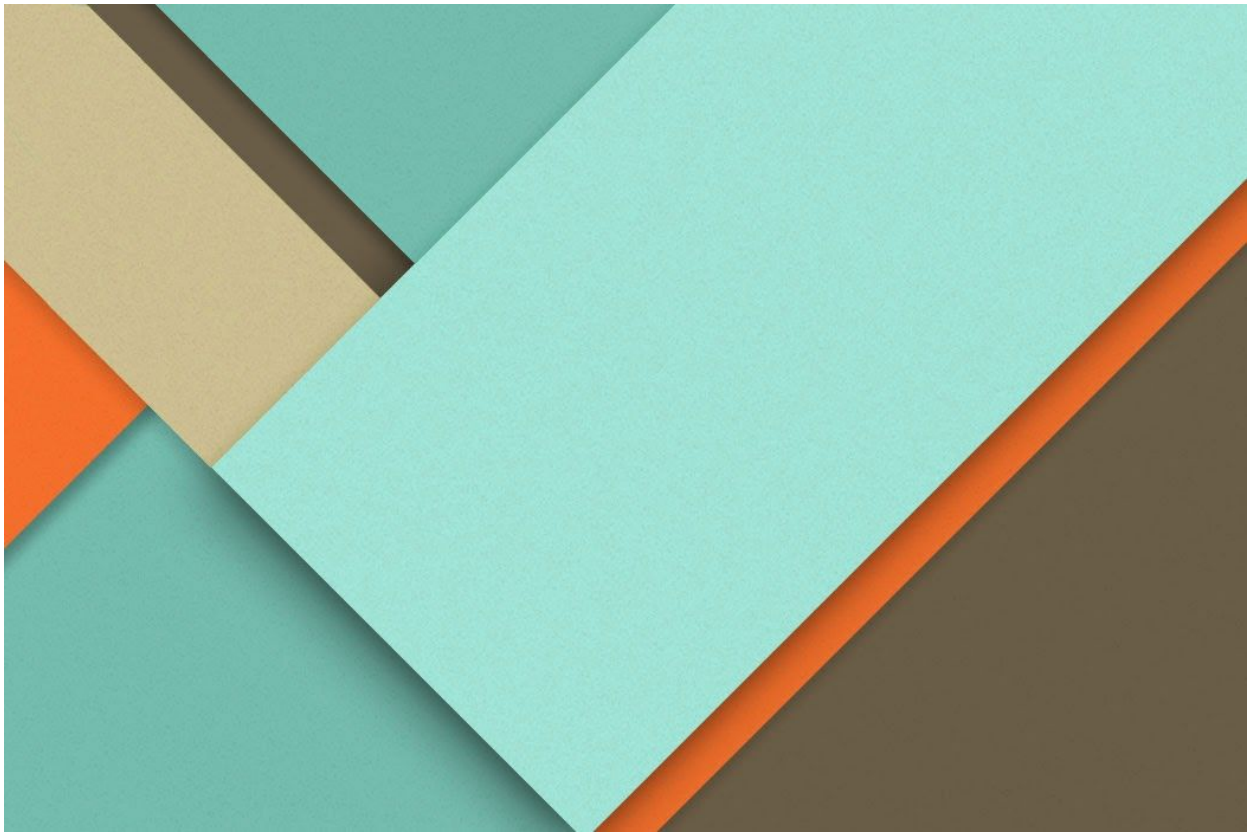


Informe de Proyecto Sistemas Operativos

09/12/2020

Herlein Rodrigo Nicolás 112591

Holzmann Johann Edgardo 117885

Problemas

Procesos, Threads y Comunicación:

Conjunto de Tareas:

El concepto utilizado para realizar este proceso fue:

Un proceso coordinador, en el cual se va a recibir por consola la cantidad de tareas deseada por el usuario. Una vez que se optó por la cantidad de tareas, que pueden ser 4, 5 o 6, se le comunica a sus tres hijos a través de pipes la cantidad de tareas que debe realizar cada subtarea, es decir:

- Si son 4 tareas, se deben ejecutar dos instancias de tareas A y dos instancias de tareas B.
- Si son 5 tareas, se deben ejecutar dos instancias de tareas A, una instancia de tarea B y dos instancias de tareas C.
- Si son 6 tareas, se deben ejecutar dos instancias de tareas A, dos instancias de tareas B y dos instancias de tareas C.

Por otro lado, se encuentran las tareas A, B y C, donde sus funciones son:

- Crean los hilos para ejecutar instancias de cada tarea.
- Una vez que se inicia el ciclo, a través de sus respectivos semáforos, le comunican a las instancias de tareas que fueron creadas por un hilo que comiencen la ejecución de cada instancia. Cuando la instancia termina, le comunica a la tarea que lo invoco, a través de un semáforo, que ya terminó su ejecución y puede continuar.
- Cada tarea, una vez finalizada la ejecución de sus instancias, le comunica al padre a través de un pipe que ya terminó de ejecutar la parte correspondiente a su tarea.

Cuando las tareas terminan de ejecutarse, se lo comunican al padre (Coordinador de Procesos) a través del pipe. Una vez finalizada todas las tareas, el coordinador comienza un nuevo ciclo, pidiendo nuevamente ingresa el número de tareas deseado.

Mini Shell

- En cada ciclo, se crea un proceso hijo en el cual va a representar el comando que se quiera realizar.
- Cada comando se separa en archivos con sus respectivos parámetros de entrada, permitiendo así la fácil extensión de la mini shell.
- Una vez que se eligió una opción correcta, se realiza la ejecución de ese comando a través de la función execvp, enviando como parámetros el nombre del archivo que representa ese comando, y los argumentos que necesita el comando.
- Cada archivo computa la función y devuelve un resultado, finalizando así la ejecución de cada ciclo.

Sincronización

Demasiadas botellas de leche

- Cuando hay una o más leches en la heladera, consumo y termino mi ciclo.
- Cuando voy a la heladera y no hay más leches, lo que hago es fijarme si hay alguien comprando. Si ya hay alguien que decidió ir a comprar, entonces opto por esperar a que vuelva con las leches para recién ahí consumir. Si nadie fue a comprar, entonces decidí ir a comprar y comunico a través del semáforo que ya estoy haciéndolo.

a) Resuelva el problema utilizando hilos (threads) y semáforos para su sincronización.
    - Un semáforo llamado leches, de longitud n, que va a representar la cantidad de leches que hay en la heladera en un momento dado.
    - Un semáforo llamado compradores, utilizado de forma binaria, donde su valor en 1 representa que no hay nadie realizando la compra de las leches, y su valor en 0 representa que está yendo al supermercado o ya está en el comprando.

b) Considere que tiene n compañeros de alojamiento y las mismas propiedades se deben garantizar. Resuelva el problema utilizando procesos y colas de mensajes para la comunicación.

Para este caso, se utiliza el mismo concepto de algoritmo para resolver el problema, pero ahora utilizando colas de mensajes para la comunicación, es decir:

- Tendré una cola de mensajes para representar la cantidad de leches actualmente en la heladera.
- Tendré una cola de mensajes para representar si hay alguien o no que decidió ir a comprar al supermercado.
- Ahora cada compañero es representado por un proceso, donde cada uno se crea a partir de que otro proceso invoca a la función fork. La función que realiza el proceso “padre” es crear las colas, llenar o no la heladera e indicar que inicialmente no hay nadie comprando, y crear los compañeros.

Comida Rápida

Se utilizaron varios semáforos para realizar la sincronización y la comunicación entre los distintos recursos humanos.

- El cocinero comienza a cocinar cuando haya lugar en la cola de comidas, es decir, cuando el semáforo que representa la cantidad de lugares que quedan disponibles en la cola de comida permita realizar un wait.
- Cuando el cliente llega, se sentará en la mesa si esta está disponible. Una vez que se sentó, realiza el pedido a un camarero. El camarero, cuando tenga un pedido, se acercara a la cola de comidas y se fijará si hay alguna comida. En el caso de que no haya comida, esperara. Cuando hay comida, la toma, comunica a través del semáforo que hay un nuevo lugar disponible en la cola de comidas, y le comunica al cliente que ya está el pedido para que comience a comer.
- Cuando el cliente termina de comer, le comunica al limpiador que se irá y que puede comenzar a limpiar la mesa.
- El limpiador limpiará la mesa, y una vez terminado, comunicará a través del semáforo que la mesa está nuevamente disponible.

II) La solución no presentó ningún inconveniente.

III) Para la solución mediante cola de mensajes, es de igual manera que la pensada para semáforos. Ahora en las colas, se envían mensajes donde se distinguen por su tipo.

Para cada cocinero, se comienza esperando de la cola “cola_comidas_disponibles” que haya lugar en la cola de comidas, una vez que hay lugar, comienza a cocinar. Para saber qué pedido preparar, se utilizó la convención de realizar inicialmente equitativamente el tipo de menú, es decir, en una iteración realizará un menú vegano y en la siguiente iteración realizará un menú de carne. Luego, cuando un camarero retira un pedido, el cocinero cocina un menú del mismo tipo; esto se realiza para que siempre se mantengan pedidos de los dos tipos de menú en la cola de comidas listas y que un camarero no se quede esperando infinitamente por un tipo de menú que podría no estar preparado si los cocineros cocinarian aleatoriamente. Por último, el cocinero agrega un menú a la cola “cola_comida” y comienza el ciclo nuevamente, aguardando por un lugar en la cola de comidas.

El camarero, comienza esperando que algún cliente realice un pedido en la cola “cola_pedidos”. Una vez que recibió un pedido, identifica el tipo de menú solicitado y se dirige a la cola “cola_comida” a esperar que esté listo un menú del tipo solicitado. Una vez que recoge el pedido, notifica a la cola “cola_comidas_disponibles” que hay

un nuevo lugar para la cola de comidas, y por último le entrega la comida al cliente a través de la cola “cola_pedidos”.

En el caso del cliente, comienza aguardando por una mesa disponible. Una vez que se libera alguna mesa, se sienta y realiza un pedido de forma random. Esto lo hace depositando un mensaje en la cola “cola_pedidos” de tipo “TIPO_PEDIDO”. En el atributo “dato” del msj que se envía por la cola, se ve reflejado si el cliente solicitó por un menú de carne (dato = 0) o por un menú vegetariano (dato = 1). Luego de esto, el cliente aguarda por su pedido, y una vez que lo recibe, comienza a comer. Cuando termina de comer, notifica que hay una mesa a limpiar a la cola “cola_mesas_limpiar”. El cliente se retira y vuelve más tarde.

Por último, el limpiador, aguarda un mensaje de la cola “cola_mesas_limpiar” para comenzar a limpiar una mesa. Una vez que termina de limpiarla, notifica que hay una mesa limpia a la cola “cola_mesas_disponibles” y finaliza el ciclo.

En el archivo main.c, me encargo de inicializar las colas (si ya estaban creadas, me encargo de borrarlas al inicio). Luego cargo las mesas disponibles en la cola “cola_mesas_disponibles”, cargo en la cola “cola_comidas_disponibles” los tipos de menú a cocinar inicialmente (se cocinan menus equitativamente, 5 menú carne y 5 menú vegetariano). Por último, se realizarán los fork para crear los procesos.

Problemas Conceptuales

Android Operating System Architecture

Se realizó un informe en base al paper publicado por el magíster en redes de computadoras Umer Farooq acerca de la arquitectura del sistema operativo Android. Se profundizó en el tema utilizando conceptos vistos en clase y fuentes diversas. Por último se redactó una conclusión teniendo en cuenta lo aprendido en el desarrollo del ejercicio y durante el cuatrimestre.

Memoria Virtual de Dos Procesos en Ejecución

Conceptos previos al desarrollo del enunciado:

- **copy-on-write**, Permite que los procesos padre-hijo compartan sus páginas asociadas. Estas páginas asociadas se llaman páginas copy-on-write, si uno de los dos procesos quiere escribir en una de estas páginas se realiza una copia de la misma asociada al proceso que busca hacer la escritura, se realizan los cambios en la misma y se guarda en una porción disponible de la memoria principal.

Partiendo de la figura 2 presentada en el enunciado, en el primer inciso del punto 2.2.1 realizamos un esquema del uso de memoria virtual de dos procesos cuando uno de ellos utiliza la función fork() usando la estrategia copy-on-write y acto seguido, el proceso padre realiza un cambio en su página dos. En dicha situación el sistema operativo copia la página dos del proceso padre, realiza los cambios necesarios sobre la copia y la guarda en un espacio vacío de la memoria principal.

En el segundo inciso del punto 2.2.1 se expandió el esquema realizado para la situación planteada en el inciso a para que las tablas de páginas de cada proceso guarden sus páginas asociadas y los bits de válido y escritura asociados a cada frame. El bit de válido permite saber si un frame se encuentra en memoria principal (si tiene el valor uno) o en memoria secundaria (si tiene el valor cero), el bit de escritura permite saber si un frame fue editado (si tiene valor uno) o no (si tiene valor cero) en algún momento luego de llegar a la memoria principal.

Gestión de Archivos y Espacio en Memoria

Se completó la información correspondiente a una tabla **File Allocation Table (FAT)** la cual tiene un comportamiento parecido al de una lista enlazada. Se completó una segunda tabla cuyas entradas contenían la información referida a cada archivo o directorio como su nombre, fecha de creación y el bloque de memoria en el cual comienza dicho archivo o directorio. En dicho bloque se encontraba parte de la información del archivo junto con la dirección del siguiente bloque y así sucesivamente hasta llegar a un bloque que contenga un valor espacial que representa un end of file. Al momento de alocar un nuevo bloque se reemplazó el valor del end of file por la dirección correspondiente al nuevo bloque y en este se escribió el eof. Este esquema de organización necesita sucesivos accesos a memoria y no puede trabajar con archivos de tamaño mayor a 4GB.

Situación final:

Nombre	Tipo	Fecha	Nro Bloque
Actividades.txt	F	29-09-2020	3
Act-Labo1.txt	F	10-10-2020	16
Act-Labo2.txt	F	15-10-2020	23
Prueba	D	15-10-2020	19
Informe	F	18-10-2020	21

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

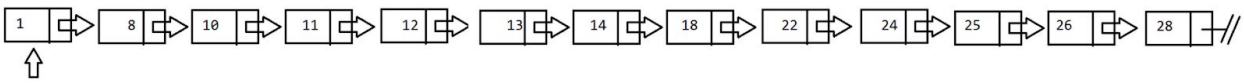
Luego se procedió a gestionar el espacio libre utilizando un **mapa de bits** el cual estaba compuesto por una sucesión de ceros (que representaban espacios libres) y unos (que representaban espacios ocupados), el resultado final fue el siguiente:

Bit map: 011111101000001 110111010001011

De izquierda a derecha el i-ésimo bit representa al i-ésimo espacio en memoria. El bit map ocupaba un total de 30 bits.

Por último se gestionó el espacio libre utilizando una **lista enlazada** la cual "unía" todos los espacios disponibles en la memoria a través de punteros, los cuales ocupan 4 Bytes cada uno. En una lista el puntero a la primera celda de memoria disponible se guarda en una locación especial, luego se unen las siguientes celdas disponibles utilizando punteros, los cuales se guardan dentro de las mismas celdas. En nuestro caso, la celda 1 guardaba el puntero a la celda 8, la cual guardaba el puntero a la celda 10 y así sucesivamente hasta llegar a la celda 28. El espacio en memoria requerido para almacenar la lista fue de 52 bytes pues se necesitaban 13 punteros de 4 bytes.

Lista enlazada:



Si se pierde el puntero a la primera celda disponible (la primera celda de la lista), el mismo puede recuperarse recorriendo la memoria y rearmando la lista con los espacios libres durante el recorrido. Esta estrategia es muy costosa debido a que para recorrer la memoria debo hacerlo de inicio a fin lo cual nos da un Orden de tiempo de ejecución O(n) siendo n la cantidad de espacios disponibles en memoria.

Representació de Información Utilizando Tablas de I-Nodos, Archivos y Descriptores

Definiciones previas:

- **I-Nodo**: Cada I-Nodo corresponde a un archivo abierto y contiene información respecto del mismo, su ubicación, su nombre, tipo de archivo, cantidad de procesos que apuntan hacia el I-Nodo, su tamaño, permisos, etc... Al configurar el sistema operativo se define la cantidad de I-Nodos máxima que se podrá tener en el sistema y cada uno tendrá un número que lo identificará unívocamente.
- **File Table**: Cada entrada de la tabla contiene:
    - cantidad de procesos que apuntan a la entrada
    - Offset que representa en qué parte del archivo se está leyendo o escribiendo
    - La dirección de un I-Nodo
- **Tabla de File Descriptors**: Corresponde a cada proceso y los file descriptors del 0 al 2 están reservados para usos específicos (standard input, standard output y standard error)

En este ejercicio se representó la información correspondiente a la situación planteada en el enunciado. Dos procesos abren un mismo archivo (Actividad.txt) en modo lectura/escritura, lo cual implica que el primer File Descriptor libre que posean (File Descriptor 3 en ambos casos) apuntará a entradas distintas en la tabla de archivos pero estas entradas apuntarán al mismo I-Nodo (correspondiente al archivo que abrieron). Luego uno de los procesos crea un proceso hijo, el cual tendrá una copia de los File Descriptors del padre y por lo tanto apuntará a la misma entrada en la tabla de archivos. Por último el proceso hijo abre un archivo (Informe.txt) en modo lectura/escritura, en consecuencia se crea una nueva entrada en la tabla de archivos, el primer File Descriptor libre de este proceso (el 4) apuntará a dicha entrada y la misma apuntará a un nuevo I-Nodo, distinto al del archivo Actividad.txt.

Situación final: