SISTEMAS OPERATIVOS: INFORME PROYECTO N°2

INTEGRANTES:

Ceballo Vitale, Pablo Guillermo

Gómez, Tomás Alejandro

Segundo Cuatrimestre de 2018

Fecha y Año: 3/11/2018

Profesor a cargo: Cenci, Karina

NOTAS DESARROLLO EJERCICIO 1

EJERCICIOS CON RESOLUCIÓN DE SECUENCIAS POR PIPES

Para la resolución de ejercicios de secuencias con pipes se utilizó dos variables char globales usadas como buffer para los mensajes, junto con la cantidad de pipes necesarios para realizar la secuencia. En cada uno de los 3 casos se tiene que el proceso principal comienza inicializando cada pipe con pipe(PipeX) (Estos pipes son arreglos de enteros). Luego entra en un ciclo for en el cual crea 5 procesos hijos. Estos hijos serán los encargados de imprimir por pantalla una letra cada uno. Al crear un hijo, el mismo irá a la función dispatch(I) (siendo I el valor actual del ciclo del for, de 1 a 5). En el dispatch, el proceso hijo será enviado a una de las 5 funciones disponibles, es decir, el primer hijo creado (con I=1) será enviado a A(), mientras que el hijo 5 será enviado a E().
Hecho esto, el padre luego se encarga de enviar un mensaje para comenzar el ciclo, tal que envía un mensaje al pipe que lee la primer letra de la secuencia.
Para los pipes se considera que PipeX[0] es para leer los mensajes utilizando "readMessage"y PipeX[1] se usa "escribir" para enviar los mensajes contenidos en writeMessage.
Luego de escribir o leer, cada proceso hace "fflush(NULL)" el cual elimina los datos innecesarios en el buffer (esto se hace para evitar generar errores al momento de escribir).
Al iniciar, cada proceso se encarga de cerrar los lados de los pipes que no necesita (por ejemplo, el proceso X solo necesita leer del PipeX y escribir a PipeY, por lo que cierra todos los pipes con "close", excepto por PipeX[0] y PipeY[1]).
Cada proceso hijo estará dentro de un while true para que se imprima la secuencia indefinidamente.
En algunos caso es necesario que un proceso imprima una letra dos veces en un mismo ciclo while para poder respetar la secuencia.
Todos los mensajes escritos o leídos son de tamaño 1, y lo que contienen no es importante para la secuencia, por eso se usa un caracter X.

EJERCICIOS CON RESOLUCIÓN DE SECUENCIAS POR COLAS DE MENSAJES

Para resolver las secuencias usando colas de mensajes, fue necesario usar un “key_t” para definir la clave de la cola, junto con una estructura que contiene el Tipo y Texto del mensaje. Además se usó “define” para definir los distintos tipos de mensajes, tal que sea más fácil poder cambiarlos.
Además de esto se definieron 3 funciones para simplificar la lectura de los procesos:

-getQueue(): Se encarga de conectar un proceso a la cola de forma tal que usa “msgget” con la Key de la cola y retorna su ID.
-send(int,int): Se encarga de enviar un mensaje de un cierto tipo a una cierta cola. Esto es tal que se le asigna al mensaje el tipo pasado por parámetro junto con el texto X y lo envía a la cola de ID pasada por parámetro usando “msgsnd” con buffer de tamaño 1.
-receive(int,int): Se encarga de recibir un mensaje de la cola cuya ID y tipo están especificados como parámetros. Esto se hace a través de “msgrcv”. Lo que contiene el mensaje no es importante.
Tanto en “send” como en “receive” se usa “fflush(NULL)” para garantizar que los caracteres se impriman correctamente vaciando el buffer.

El proceso principal se encarga de generar la clave usando “ftock” y luego crea la cola de mensajes usando esa misma Key con “msgget” (con el flag IPC_CREAT). Se nota que no se uso “IPC_NOWAIT” por lo que todos los “receive” son bloqueantes.
Luego el proceso principal crea los 5 procesos y los envía a sus respectivas funciones usando “dispatch” de igual manera que con los pipes.
Finalmente el padre de todos estos procesos se encarga de iniciar la secuencia. Esto lo logra enviando un mensaje a donde se ubica la primer letra de la secuencia.
Luego realiza un “wait(NULL)” esperando a que sus 5 hijos terminen.

Cada hijo, al llegar a su proceso, comienza llamando a “getQueue()” para poder conectarse a la cola de mensajes creada por el padre. Luego entra en un ciclo while true en el cual se encarga de llamar a “send” y “receive” para poder replicar la secuencia. El orden de enviar y recibir mensajes para la sincronización es similar al orden de escritura y lectura utilizado en pipes.

EJERCICIO DE RESOLUCIÓN DEL PROBLEMA POR COLAS DE MENSAJES

Para resolver el problema de la miel con colas de mensajes se utilizaron 4 tipos de mensajes. En primer lugar se tienen las constantes N y M que indican la cantidad de abejas y espacios libres en el tarro respectivamente. También se tiene una estructura al igual que antes que contiene el Tipo del mensaje y el Texto. De forma similar a antes también se usaron las funciones “getQueue”, “send” y “receive” para facilitar la conexión con la cola (la única diferencia es que “receive” retorna un entero).
En cuanto a los tipos de mensajes se tienen:

-Tipo 1 (simbolizado por “**tMiel**”): Cuando una Abeja recibe este mensaje significa que ocupa un espacio vacío del tarro y aún quedan espacios libres. Cuando el Oso envía este tipo de mensaje significa que consume una miel producida. Nunca habrá más de M-1 mensajes de este tipo.
-Tipo 2 (simbolizado por “**tUlt**”): La Abeja en recibir esto sabe que ocupó el último espacio del tarro y debe avisar al Oso. El Oso se encarga de enviar este mensaje una vez enviados los M-1 mensajes de tipo 1 (es decir, este es el mensaje número M, el último espacio libre).
-Tipo 4 (simbolizado por tProd): Solo hay 1 de este tipo de mensaje en todo momento. La abeja que lo recibe puede entrar dentro para producir miel. Este tipo de mensaje es enviado por una Abeja luego de recibir un mensaje de tipo 1 o por el Oso luego de enviar M-1 mensajes de tipo 1 y uno de tipo 2.
-Tipo 6 (simbolizado por “**tOso**”): El Oso va a estar en espera de este mensaje, el cual le indica que debe consumir toda la miel en el tarro. Es enviado por una Abeja una vez que recibe un mensaje de tipo 2 (es decir, cuando no hay más espacios libres).

El proceso principal se encarga de crear los N procesos encargados de hacer la función de la Abeja y el proceso único encargado de hacer el trabajo del Oso. Una vez hecho esto, el padre es el que se encarga de iniciar el ciclo, enviando M-1 mensajes de tipo 1 a la cola, un mensaje de tipo 2 y un mensaje de tipo 4. Esto le permite a una Abeja poder entrar a producir miel. Luego espera a que los N+1 procesos terminen.

Otra consideración que se tomó fue al momento de producir. Para producir una Abeja debe recibir un mensaje de tipo -2, es decir puede aceptar mensajes de tipo 1 o 2 (pero 1 toma prioridad sobre 2), esto es tal que, una vez que una abeja recibe 2, se encarga de enviar un mensaje 6, sino envía un mensaje 4. De esta forma, una abeja nunca se quedará esperando indefinidamente para ocupar un espacio vacío. Y con la espera de un mensaje de tipo 4, se asegura de que sólo una única abeja entrará en la sección crítica para producir miel a la vez.

EJERCICIO DE RESOLUCIÓN DEL PROBLEMA POR MEMORIA COMPARTIDA

La estructura de este ejercicio es similar a cuando se usa cola de mensajes. Se separó el problema en 4 archivos distintos:

-common.h se encarga de inicializar cosas como la cantidad de abejas y espacios totales en constantes, junto con definir una estructura que contiene los semáforos para poder ser accedidos por otros procesos. Por último hay una función generateKey() que se encarga de llamar a ftok para generar una clave para crear/asociarse al espacio de memoria.

-init.c es el proceso padre. Este se encarga de crear el espacio de memoria usando generateKey() y shmget. Luego se asocia a dicho espacio de memoria con “shmat”. A continuación se inicializan los 3 semáforos. Estos son: un semáforo que muestra la cantidad de espacios disponibles en el tarro (comienza con M, es decir, hay M espacios vacíos en el tarro), el hacer “wait” de este semáforo significa que se produjo miel y se ocupó uno de estos espacios; Luego está “Producir”, el cual funciona como un mutex tal que se utiliza para que una sola abeja sea capaz de entrar a producir a la vez (de otra forma, no sería posible saber cual abeja fue la última en producir), este semáforo comienza en 1; Por último está el semáforo denominado “Dormir”, el cual se usa para avisarle al Oso cuando se tiene que encargar de vaciar el tarro.
Una vez hecho esto, el proceso padre se encarga de crear los procesos hijos, es decir, crea un proceso encargado de hacer las actividades del oso, y N procesos encargados de hacer las actividades de la abeja. Al crear los procesos, el padre se encarga de ejecutarlos, tal que al ejecutar init también se activen los ejecutables “oso” y “abeja”.
Luego de esto, el padre espera a que los procesos terminen y por último se desasocia del espacio de memoria.

-oso.c se encarga de consumir todo el tarro una vez que está lleno. Esto se logra tal que, una vez asociado al espacio de memoria, el Oso se queda esperando a que alguien levante el semáforo Dormir. Esto le indica que una Abeja ocupó el último espacio del tarro.
Luego el oso va a entrar en un ciclo for, haciendo “signal(Miel)” una M cantidad de veces vaciando todo el tarro. Finalmente hace “signal(Producir)” para avisarle a las abejas que una de ellas ya puede pasar y volver a producir otra vez. Hecho esto, el Oso vuelve a esperar su semáforo “Dormir”.

-abeja.c se encarga de ocupar un espacio de miel a la vez. Luego de asociarse, entra en un ciclo while true, en el cual una de las N abejas podrá activar “signal(Producir)” y ganará entrada a la zona crítica para poder producir. Una vez dentro hará “wait(Miel)” para tomar un espacio de miel, lo cual siempre tendrá éxito. Hecho esto, intentará hacer “trywait(Miel)”. Si el trywait fue exitoso, significa que aún queda miel por producir, por lo que hace “signal(Miel)” para devolver el espacio que tomó y luego signal(Producir) para decirle a otra Abeja que entre.

En el caso de que el trywait no tenga éxito, la Abeja sabe que ocupó el último espacio libre del tarro, por lo que hace “signal(Dormir)” para avisarle al Oso que trabaje y sale de la zona crítica, dejando que el Oso sea el que se encargue de levantar el Producir una vez que terminó.

Para este ejercicio fue necesario trabajar con ipcs e ipcrm -m “id” para buscar el espacio de memoria creado y eliminarlo tras cada ejecución para evitar que se generen errores al intentar tener más de un espacio simultáneamente. Además de que el espacio antiguo ocupa memoria innecesariamente. También fue necesario utilizar -lpthread para la compilación de los archivos.
Como observación, se descubrió que en algunos casos es necesario volver a compilar abeja y oso aun cuando ya tiene ejecutable, junto con init. De lo contrario no ocurre nada.

Otra cosa a tener en cuenta es que, al no especificar nada sobre la velocidad en la que deben aparecer los mensajes, se asumió que no fue necesario usar sleeps para ralentizar el output.

NOTAS DESARROLLO EJERCICIO 2

MODIFICADOR DE PERMISOS

para este problema se asume que:

1) La entrada permitida para usar la función de librería chmod admite **SOLO permisos en Octal,** por ende, un input correcto sería
    a) 757 texto.txt
2) Para el input en Octal, se considera que **NO existe el "Sticky Bit"**, presente en ciertas llamadas a chmod en Octal
3) El input para el programa se hace a través de la consola, por lo cual se asume que los datos a manipular se encontrarán en el arreglo argv y la cantidad de datos en argc respectivamente
    a) argv tendrá 3 parámetros para poder intentar ejecutar chmod: el nombre del programa (agregado por defecto producto de la ejecución por consola), los permisos a establecer y el nombre del archivo al cual se le deben modificar los permisos

La resolución de este problema se basa en aplicar la siguiente secuencia de pasos:

1) Corroborar que la cantidad de argumentos obtenidos sea la correcta
    a) Caso contrario, se interrumpe la ejecución del programa y se lanza un mensaje de error
2) Corroborar que los permisos a asignar en modo Octal sean los permitidos de acuerdo a la especificación de la función chmod
    a) Esto es, los 3 números de los permisos deben ser alguno de los siguientes valores: [7,5,4,2,1,0]
3) Una vez corroborado que los permisos a asignar al archivo son correctos, se procede a ejecutar la función de librería chmod y se captura el código de error a reportar, en caso de que la función devuelva un error

SIMULACIÓN DE TRADUCCIÓN DE DIRECCIONES LÓGICAS A DIRECCIONES FÍSICAS

Para este problema se asume que:

1) Las direcciones lógicas a traducir son en primera instancia números enteros pertenecientes al rango [0, 65536] en Base decimal
2) El reemplazo de valores en el TLB cuando este no disponga de más espacio se lleva a cabo siguiendo una política FIFO, donde la entrada más vieja coincide con el último par (Dirección Lógica, Numero de Frame) presente en la última posición de la matriz
3) La Tabla de Páginas se modela a través de un arreglo de 256 componentes, la cual contendrá números de frame en el rango [0,2^8] aleatorios SIN repetición
4) Las direcciones Lógicas a traducir se encuentran en un archivo de texto denominado “memoria.txt” , el cual reside junto al archivo que contiene el código fuente y en donde las direcciones a traducir siguen el siguiente formato de ejemplo:

```
7472
4315
25315
60825
18529
13841
41488
58857
9425
29249
1745
29749
2508
13264
21003
39589
525
```

5) Los resultados de las direcciones Físicas traducidas se devuelven en Base Decimal

La resolución de este problema puede resumirse en la siguiente secuencia de acciones:

1) Leer del archivo de entrada las direcciones y guardar los valores en un arreglo auxiliar que persistirá a lo largo de la ejecución del programa
2) para cada dirección lógica presente en el arreglo
    a) Transformar el valor entero a Binario y guardarlo en una variable entera de 16 bits
    b) separar el número de 16 Bits en dos números de 8 bits, donde (leyendo el número binario de izquierda a derecha) los primeros 8 bits representarán el Número de Página y los 8 bits restantes representarán el Offset
    c) Teniendo el Número de Página, acceder al TLB en búsqueda del Número de Frame y encontrado el número de Frame, guardarlo en otra variable auxiliar
        i) En caso de que la búsqueda produzca un TLB_MISS, se irá a buscar el Número de Frame a la Tabla de Páginas y encontrado el Número de Frame allí, se procede a agregar el par (Nro. de página, Nro. de frame) al TLB
            (1) En caso que el TLB NO disponga de más entradas libres, se procederá a reemplazar una de las entradas del TLB por medio de un algoritmo FIFO para poder agregar la nueva entrada
    d) Concatenar el Número de Frame y el Offset previamente obtenido para obtener la Dirección Física final y devolver dicha dirección como resultado

CÁLCULO DE ARCHIVOS

Para la resolución de los incisos se asume que:

1) Los tamaño de bloque necesarios para cada tipo de archivo son los siguientes:
    a) 1 bloque para el 80% de los archivos
    b) 2 bloques para el 10% de los archivos
    c) 3 bloques para el 5% de los archivos
    d) 10 bloques para el 5% restante de los archivos
2) En un mismo bloque se pueden almacenar más de un File Descriptor

Procedamos a presentar la solución a los problemas planteados:

para el inciso “a)”, se debe obtener la cantidad de archivos que se pueden alocar en el disco con los datos provistos por el problema, la incógnita a encontrar, entonces, es el número de archivos que se pueden alocar, la cual explicitaré con la variable “X”.

con los datos dados, podemos inferir que:

1) el 80% del total de archivos ocupa un bloque
2) el 10% del total de archivos ocupan 2 bloques
3) el 5% del total de archivos ocupan 3 bloques
4) el 5% restante del total de archivos ocupan 10 bloques

sabemos que un File Descriptor ocupa 64 Bytes y que un bloque tiene una capacidad de 512 Bytes, por ende, en un bloque pueden entrar:

512/64 = 8 File Descriptors por bloque, por ende, ⅛ del total de archivos debe ser reservado para los File Descriptors

Con todos estos datos, es posible escribir la siguiente ecuación para obtener el valor de X:

X*(0,8 * 1) + X* (0,1 * 2) + X * (0,05 * 3) + X * (0,05 * 10) + ⅛ * X = 10000 [Situación Inicial]

0,8X + 0,2X + 0,15X + 0,5X + 0,125X = 10000 [Resuelvo las multiplicaciones]

1,775X = 10000 [Sumo todos los valores asociados a la Incógnita]

X = 10000/1,775 [Paso la multiplicación para el lado de la derecha en forma de división]

X = 5633 Archivos [Resultado Final, máximo de archivos que se pueden alocar en Disco]

Ahora, sabiendo el Número total de archivos que se pueden alocar en Disco, es posible calcular cuantos File Descriptors son necesarios:

5633 * 64 = 360512 bytes de espacio necesarios para los File Descriptors

ANEXO: CÓDIGOS FUENTE

TraductorDireccionesMemoria.c

```
#define __STDC_FORMAT_MACROS //Defino las Macros de Impresion para enteros especiales
#include<stdlib.h>
#include<math.h>
#include<inttypes.h>
#include<stdint.h>
#include<stdio.h>
#include<time.h>
#include<stdbool.h>

#define NELEMS(x)  (sizeof(x) / sizeof((x)[0])) //Macro que calcula la cantidad de elementos presentes en un arreglo cualquiera

#define TLB_MISS -1 //Codigo especial para los TLB Miss que se puedan provocar

/*PASOS A SEGUIR:

	0) Leer el archivo y obtener todas las direcciones lógicas a traducir
	1) Obtener la Dirección Lógica
		Lo que debo hacer es, dado el numero entero, Pasarlo a Base 2
		y de ahí usar Enmascarado de Bits para separar el numero de 16 bits resultante en dos secuencias de 8 bits
		que serán el Numero de Página y el Offset
	2) Acceder con el Número de página al TLB
		En primera Instancia, una vez que se el numero de página, accedo al TLB (el cual es una matriz de 16x3, donde La tercer columna del TLB los uso como si fuesen "Dirty Bits", al inciar, los pongo en 0 y con estos puedo determinar si una entrada del TLB
				es o inicial y puede ser reemplazada o no) y busco el Frame correspondiente, en caso de NO encontrar el numero de página en el TLB (Osea, se produjo un TLB_MISS),
				debo bajar a la Tabla de Páginas, encontrar el numero de frame y el par (NroPágina,NroFrame) agregarlo al TLB, si NO lo puedo agregar al TLB porque el mismo NO
				dispone de espacio, debo reemplazar una entrada en el mismo, esto se lleva a cabo por medio de un algoritmo de reemplazo FIFO, donde la entrada más vieja coincide con el último
				par (NroPágina,NroFrame) de la Matriz
	3) Obtener la Dirección Física
```

Obtenido el Nro de Frame, lo concateno con el Offset y devuelvo el resultado de la concatenación EN DECIMAL, el cual es la dirección Física final

Sobre uint_x:

0x.. -> el número lo represento en Hexadecimal

0b.. -> el número lo represento en Binario

Sin literal despues del primer cero lo estoy trabajando en Decimal

También puedo representar el numero completo y poner el literal de representación al final

1111b es un número binario de 4 bits, por ejemplo

Sobre Enmascarado:

Yo al enmascarar, me quedo con los bits que me "Importan" al aplicar una operación AND entre el

Numero original en binario y un número binario arbitrario cuyas posiciones en 1 denotan QUE posiciones de bits dejaré como están

asumamos que el numero es 101011010111111 y la máscara es 1111111100000000

al hacer 101011010111111 & 1111111100000000 obtengo 1010110100000000

Sobre Shifting:

la operación >> o << denota un shifting de los bits del número a derecha o a izquierda respectivamente

se permite usar potencias de dos para hacer shifting de varias posiciones de una.

LOS BITS QUE SE CAEN DEL NÚMERO SE PIERDEN

si mi numero es 010110, hacer 010110 >> resulta en 001011

*/

```
/*
 * Dado un numero en decimal, pasarlo a binario, uso método de la División
*/
uint16_t DecimalABinario(int n){
	uint16_t ret = 0x00; //000000000000000 en binario usando representación en Hexa
	int aux;
	int i = 1; //para ir sumando 1, 10, 100, etc
	while(n > 0){
		aux = n % 2;
		n = n / 2;
		ret = ret + (aux * i);
		i = i * 10;
	}

	return ret;
```

```
}

/*
 *
*/
uint8_t DecimalABinario8Bits(int n){
	uint8_t ret = 0x00; //000000000000000 en binario usando representación en Hexa
	int aux;
	int i = 1; //para ir sumando 1, 10, 100, etc
	while(n > 0){
		aux = n % 2;
		n = n / 2;
		ret = ret + (aux * i);
		i = i * 10;
	}

	return ret;

}

/*
 * Dado un Numero binario, lo paso a Decimal, uso método del producto
*/
int BinarioADecimal(uint16_t n){
	int rem,decimal,base = 1;
	decimal = 0;
	rem = 0;
	while(n > 0){
		rem = n % 10;
		decimal = decimal + rem * base;
		n = n / 10;
		base = base * 2;
	}

	return decimal;
}


int BinarioADecimal8bits(uint8_t n){
	int rem,decimal,base = 1;
	decimal = 0;
	rem = 0;
	while(n > 0){
		rem = n % 10;
		decimal = decimal + rem * base;
```

```
            n = n / 10;
            base = base * 2;
        }

        return decimal;
}


//estas funciones de conversión entre numeros en bits ESTAN PROBADAS Y ANDAN
/*
 * Función para transformar dos enteros de 8 Bits en un entero de 16 Bits
*/
uint16_t pasarDe8A16(uint8_t dataFirst, uint8_t dataSecond) {
   uint16_t dataBoth = 0x0000;

   dataBoth = dataFirst;
   dataBoth = dataBoth << 8; //Muevo el primer valor a la parte alta del numero de  16 bits
   dataBoth |= dataSecond; //Hago OR entre mi valor actual y el segundo valor introducido a
la función, el OR hará la suma en binario entre los bits
   return dataBoth;
}

/*
* función para separar un entero de 16 bits en dos enteros de 8 bits
* los resultados se devuelven en un arreglo de dos componentes
*/
uint8_t *pasarDe16A8(uint16_t dataAll) {
   static uint8_t arrayData[2] = {0x00,0x00};

   arrayData[0] = (dataAll >> 8) & 0x00FF; //Obtengo la parte alta del numero de 16 bits y la
mando a la primer componente del arreglo
   arrayData[1] = dataAll & 0x00FF; //Obtengo la parte baja del numero de 16 bits
   return arrayData;
}


/*
 * Dado un numero de página, buscar el numero de frame correspondiente en la Tabla de
Páginas
*/
uint8_t BusquedaTabla(uint8_t PN, uint8_t TP[]){
        int Nro = BinarioADecimal8bits(PN);
        if(Nro >= 0 & Nro < 256)
                return TP[Nro];
        else
                return 0x00; //En caso que NO Encentre la direccion, devuelvo 0 por default
```

```
}

/*
 * Dado un número de Página, lo intento buscar en el TLB, en caso de no encontrarlo, debo bajar a la Tabla de Páginas
 * Lo que devuelvo es la POSICIÓN en donde encontré el numero de página o -1 en caso de TLB Miss
*/
int BusquedaTLB(uint8_t PN, uint8_t TLB [][3]){
	int C;
	int ret = 0;
	//Buscar valor aquí
	for(C = 0; C <= 15 && ret == 0; C++){
		if(TLB[C][0] == PN)
			ret = 1;
	}
	if(ret == 0)
		C = TLB_MISS;

	return C;
}

/*
	Método que despues se encargará de hacer un reemplazo cuando el TLB este lleno
	Usa un ALgoritmo de reemplazo FIFO donde la página más vieja es aquella que esta al final del arreglo
*/
void Reemplazar(uint8_t FP, uint8_t PN, uint8_t TLB[][3]){
	int i;

	//Lo que debo hacer es correr TODAS las posiciones un lugar hacia abajo, dejando el primer espacio libre
	//de tal forma que la primer componente de la matriz quede libre para el nuevo valor
	TLB[15][0] = 0;
	TLB[15][1] = 0; //Borro la ultima componente (la más vieja) del TLB, dejando un espacio libre
	for(i = 15; i >= 0; i--){
		TLB[i][0] = TLB[i-1][0]; //Muevo los Page Number existentes hacia abajo
		TLB[i][1] = TLB [i -1][1]; //Muevo los FP existentes hacia abajo
		TLB[i][2] = TLB [i-1][2]; //Muevo los Dirty Bits existentes un lugar para abajo
	}

	//Hecho el reemplazo, agrego las nuevas componentes
	TLB[0][0] = PN;
	TLB[0][1] = FP;
	TLB[0][2] = 1; //marco que es una nueva entrada que NO puede ser reemplazada
```

```
}


//EL código de lectura de archivos hay que probarlo por separado en algún otro lado, hay que probar que esa cosa ande si la vamos a agarrar del Proyecto 1
void LeerArchivo(int Dirs[]){
   int i = 0;
   int num = 0;
   FILE *fp = fopen("memoria.txt", "r"); //Abro el archivo en modo lectura
   if(fp == NULL){
      printf("Error al abrir el archivo para lectura!");
      exit(1);
   }
   else{
      while(!feof(fp)){
         fscanf(fp,"%d", &num);
         Dirs[i] = num;
         i++;
      }
   }
   fclose(fp); //cierro el archivo
}


/*
 * Función para determinar si estoy intentando insertar un valor repetido en la Tabla de Paginas
*/
bool Repetido(uint8_t Pages[], uint8_t num){
        bool ret = false;
        int i;
        for(i = 0; i < NELEMS(Pages) && !ret; i++){
                ret = Pages[i] != 0x00 && Pages[i] == num;  //0 es un caso especial ya que PUEDE que alguna página tenga Nro de Frame = 0

        }

        return ret;
}

/*
 * funcion para inicializar los arreglos de direcciones input y Tabla de Paginas
*/
void inicializar(int DirIni[], uint8_t PageT[], uint8_t TLB[][3]){
        int i;
```

```
	time_t t;
	srand((unsigned) time(&t)); //Seteo el "seed" para la aleatoriedad de rand() usando la hora actual del equipo
	//Inicializo arreglo de direcciones logicas
	for(i = 0; i <= 65536; i++){
		DirIni[i] = -1; //Leno el arreglo de direcciones con "-1" para luego poder hacer Peek en caso que NO me llenen el arreglo enteramente
	}
	//Inicializo Tabla de Páginas
	for(i = 0; i <= 255; i++){
		int r = rand() % 255; //numero random para llenar la tabla de páginas
		uint8_t r8 = DecimalABinario8Bits(r);
		if(!Repetido(PageT, r8)) //Si el numero generado NO esta repetido dentro del arreglo, lo pongo como numero de frame
			PageT[i] = r8;
		else
			i--; //Fuerzo al ciclo a repetir la acción para que genere un nuevo numero en esa posición
	}
	//INicializo TLB
	for(i = 0; i < 16; i++){
		TLB[i][0] = 0;
		TLB[i][1] = 1;
		TLB[i][2] = 0; //"Dirty Bit"
	}

}

/*
 * Función que intenta agregar una nueva entrada al TLB cuando se produjo un TLB_MISS
*/
void AgregarTLB(uint8_t PN, uint8_t FN, uint8_t TLB[][3]){
	int i;
	for(i = 0; i < 16; i++){
		if(TLB[i][0] == 0 & TLB[i][1] == 1 & TLB[i][2] == 0){ //Si encuentro una "Entrada Inicial"
			TLB[i][0] = PN;
			TLB[i][1] = FN;
			TLB[i][2] = 1; //Marco que esta entrada fue modificada
		}
		else{
			Reemplazar(FN,PN,TLB); //Si no, debo reemplazar una entrada para agregar la nueva, delego en el algoritmo de reemplazo
		}
	}
```

```
}

int main(){

	int Direcciones [65536]; //Arreglo donde guardaremos las direcciones
	uint8_t TablaPaginas [256]; //Tabla de Páginas
	uint8_t framePag; //Numero de frame resultante
	uint16_t DirFis; //Dirección física Resultante
	uint8_t *Cortes8Bit; //arreglo para los cortes de 8 bits
	uint8_t TLB [16][3]; //TLB
	inicializar(Direcciones, TablaPaginas, TLB); //función para inicializar los arreglos
	uint8_t PageNum = 0x00;
	uint8_t Offset = 0x00;
	LeerArchivo(Direcciones);
	//Asumo que el archivo se leyó y tengo todas las direcciones en el arreglo
"Direcciones"
	int i;
	int corte = 0;
	uint16_t num; //variable que mantendrá el valor original en 16 bits
	for(i = 0; i < 65537 && corte == 0; i++){ //Para cada dirección lógica en el arreglo
		if(Direcciones[i] != -1){ //Hago este chequeo por las dudas
			int OG = Direcciones[i];
			num = DecimalABinario(Direcciones[i]); //Obtengo el número de 16
bits en binario
			uint16_t numCopy = num;
			Cortes8Bit= pasarDe16A8(numCopy);
			PageNum = Cortes8Bit[0];
			Offset = Cortes8Bit[1];
			int BT = BusquedaTLB(PageNum, TLB); //En primera instancia, busco
si el Page Number esta en el TLB
			if(BT == TLB_MISS){ //Si NO se encontró el numero de Frame en el
TLB, debo bajar a la Tabla de Páginas y luego agregar la entrada al TLB
				framePag = BusquedaTabla(PageNum, TablaPaginas);
				AgregarTLB(PageNum, framePag, TLB);
			}
			else{ //Si lo encontré en el TLB, solo obtengo el numero de frame y
devuelvo los datos, en BT tengo la Fila donde encontré la entrada
				framePag = TLB[BT][1];
			}
			DirFis = pasarDe8A16(framePag,Offset); //Obtengo la Dirección Física
final
			printf("Direccion Logica = %d, Direccion Fisica asociada = %d \n",
OG,BinarioADecimal(DirFis)); //Hecha la traducción, imprimo, preguntar si es correcto
		}
		if(Direcciones[i+1] == -1) //Como NO es probable que me llenen el arreglo,
hago un "peek" para ver si debo seguir traduciendo
```

```
                        corte = -1;
        }

        printf("Se ha terminado de traducir las direcciones logicas presentadas en el
archivo\n");
        return 0;

}
```

Modificadorpermisos.c

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/stat.h>
#include <string.h>
#include <errno.h>


char ALLOWED_OCTAL[6] = {'7','5','4','2','1','0'}; //Arreglo Global para saber los valores
permitidos en octal para chmod
#define NELEMS(x)  (sizeof(x) / sizeof((x)[0])) //Macro que calcula la cantidad de elementos
presentes en un arreglo cualquiera
//DEFINO TIPOS DE ERRORES QUE PUEDEN PRODUCIRSE AL PARSEAR LOS DATOS
#define ERR_EXTRA_PARAMS -1
#define ERR_INSUFFICIENT_PARAMS -2
#define ERR_WRONG_PERM -3


/*NOTAS:
        Se asume que la llamada a chmod se hace en Octal

        en las llamadas que yo uso de ejemplos NO considero el "Sticky Bit", el cual es un
cuarto valor en modo octal

                ejemplo -> chmod 1777 myfile

*/

/*
 * Función para explicitar los mensajes de error propios asociados a la ejecución de este
programa
*/
char* MensError(int n){

        char *ret;
```

```
	switch(n){
		case -1:
			ret = "Los parametros presentados sobran\n";
			break;
		case -2:
			ret = "NO hay parametros suficientes para la llamada a chmod\n";
			break;
		case -3:
			ret = "Codigo de permisos en sistema de representacion Octal
ilegal\n";
			break;
	}

	return ret;
}

/*
 * Método para determinar que un número pertenezca a los valores permitidos en Octal de
chmod
*/
int PerteneceNums(char n){
	int i;
	int ret = 0;
	for(i = 0; i < NELEMS(ALLOWED_OCTAL) && ret == 0; i++){
		if(n == ALLOWED_OCTAL[i]){
			ret = 1;  //El numero obtenido es un octal legal para chmod
		}
	}
	if(ret != 1)
		ret = ERR_WRONG_PERM; //Codigo de error propio, "Código de permisos
Ilegal"

	return ret;
}

/*
 * Método con el cual intento corroborar que la llamada EN OCTAL de chmod sea correcta
*/
int LeerOctal(char *Perms){
	int ret = 1;
	int i;
	if (sizeof(Perms) > 4)
		ret = ERR_EXTRA_PARAMS; //Código de error propio, como hay más
parámetros de los permitidos
	else{
```

```
		if(sizeof(Perms) < 4)
			ret = ERR_INSUFFICIENT_PARAMS; //Código de error propio "Cantidad de permisos a setear insuficiente, debe ser del tipo ugo"
		else{
			for(i = 0; i < (sizeof(Perms) - 1) && ret == 1; i++){
				ret = PerteneceNums(Perms[i]);
			}
		}
	}
	return ret;
}



int main(int argc, char * argv[]){

	char *Permisos;
	char *FILE;
	int errno; //entero para códigos de error, sean propios o producto de la llamada a chmod

	if(argc != 3){ //ERROR, me pasaron una cantidad insuficiente o de más de argumentos
		printf("ERROR: NO se han especificado la cantidad de argumentos correctos para la llamada a chmod\n");
		printf("Por favor, ingrese una linea de la forma <permisos> <Nombre_Arcivo> de acuerdo a los parametros de la funcion chmod\n");
		exit(1);
	}
	else{
		Permisos = argv[1];
		FILE = argv[2];
		if(PerteneceNums(Permisos[0]))
				errno = LeerOctal(Permisos); //Chequear resto de la sintaxis de la entrada en octal
		else{ //Si no entra, me mandaron cualquier cosa, abortar y reportar
			fprintf(stderr, "La llamada que se esta intentando hacer NO coincide con la llamada a chmod");
			exit(1);
		}

		if(errno == 1){
			int p = strtol(Permisos,0,8); //Hago la llamada a Sting to Long() para transformar la linea de los permisos en un número que pueda leer chmod()
			//Si llegué hasta aquí, el control fue correcto, puedo invocar a chmod
			errno = (int) chmod (FILE,p);
```

```
                    if (errno < 0){ //
                fprintf(stderr, "%s: error en la llamada chmod(%s, %s) - %d (%s)\n",
                        argv[0], FILE, Permisos, errno, strerror(errno));
                exit(1);
                }
                else{
                        printf("Se ha podido ejecutar chmod con exito\n");
                }
                return(0);
        }
        else{
                fprintf(stderr, "error en el analisis de los datos suministrados con codigo %d,
lo que significa: %s \n", errno, MensError(errno));
                exit(1);
        }
        }
}
```

AbejaYOsoColasDeMensajes.c

```
#include <stdio.h>
#include <strings.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdbool.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/ipc.h>
#include <sys/wait.h>

/*

        El ciclo comienza con el Padre enviando M-1 mensajes de tipo 1, uno de tipo 2 y otro de tipo 4;

        Oso:
                Recibe Mensaje Tipo 6;
                Envia M-1 Mensajes Tipo 1;
                Envia Mensaje Tipo 2;
                Envia Mensaje Tipo 4;

        Abeja:
                Recibe Mensaje Tipo 4;
                Recibe Mensaje Tipo -2:
                Si Recibe Mensaje Tipo 1;
                        Envia Mensaje Tipo 4;
                Si Recibe Mensaje Tipo 2;
                        Envia Mensaje Tipo 6;

*/

int const N = 6; //Tengo N Abejas
int const M = 14; //Tengo M espacios para la miel

//Tipos de mensajes
#define tMiel 1
#define tUlt 2
#define tProd 4
#define tOso 6

//Tipo 1: Abeja produce miel (recibe un mensaje) y Oso consume miel (envia M-1 mensajes)
//Tipo 2: Ultimo espacio libre de miel (lo recibe la ultima Abeja, la cual se encarga de llamar al Oso) (El Oso envia este mensaje despues de M-1 mensajes de tipo 1)
```

```
//Tipo 4: Abeja entra a producir miel (Recibido por las Abejas y es enviado por otra Abeja o
por un Oso cuando termina de consumir la miel)
//Tipo 6: Despertar Oso (Abeja envia cuando recibe un mensaje de tipo 2 y Oso recibe)

key_t Key;

struct Buffer_M {
   long Tipo;
   char Texto;
};

typedef struct Buffer_M Msg;

int getQueue () { //Usado para conectar a la cola de mensajes (ya creada)
          int ID = msgget (Key, 0666);
          if (ID == -1){
                    perror ("Error - msgget: ");
          }
          return ID;
}

void send (int qid, int tipo) { //Usado para enviar mensajes
   Msg M;
   M.Tipo = tipo;
   M.Texto = 'X';
          fflush (NULL);
          int Result = msgsnd (qid,&M,1,0666);
          fflush (NULL);
   if (Result == -1) {
      perror ("Error - msgsnd: ");
      exit(1);
   }
}

int receive (int qid, int tipo) { //Usado para recibir mensajes
   Msg M;
          fflush (NULL);
          int Result = msgrcv (qid,&M,1,tipo,0666);
          fflush (NULL);
   if (Result == -1){
      perror ("Error - msgrcv: ");
      exit(1);
   }
          return M.Tipo;
}
```

```
void Oso () {
	int MsgID = getQueue ();
	int I;
	while (true) {
		//El Oso recibe un mensaje de una Abeja cuando se lleno el tarro
		receive (MsgID,tOso);

		for (I = 0; I < M-1; I++) {
			//El Oso enviando un mensaje de tipo 1 significa que consumio un
espacio del tarro
			send (MsgID,tMiel);
		}

		//El Oso envia un mensaje 2 el cual simboliza el ultimo espacio libre del tarro,
ubicado al final de la cola
		send (MsgID,tUlt);

		printf ("El Oso consumio toda la miel.\n");
		//El Oso vacio el tarro, envia un mensaje a una Abeja para que pueda entrar
a producir
		send (MsgID,tProd);
	}
}

void Abeja () {
	int MsgID = getQueue ();
	int Tipo;
	while (true) {
		//La Abeja recibe un mensaje de el Oso u otra Abeja para entrar a producir
		receive (MsgID,tProd);

		//La Abeja recibe un mensaje que puede o no ser del ultimo espacio libre en
el trarro
		Tipo = receive (MsgID,-tUlt);

		if (Tipo == tMiel) {
			printf ("Se produjo miel y quedan espacios, pasa la siguiente
Abeja.\n");
			//Si no era el ultimo espacio, envia un mensaje a otra Abeja para que
entre a producir
			send (MsgID,tProd);
		}

		else {
			if (Tipo == tUlt) {
```

```
                                printf ("Se produjo miel y se lleno el tarro, se despierta al
Oso.\n");
                                //Si era el ultimo espacio disponible y ahora esta lleno, envia
un mensaje al Oso para despertarlo
                                send (MsgID,tOso);
                        }
                }
        }
}

int main () {

        Key = ftok (".",10);

        //Se crea la cola de mensajes
   int MsgID = msgget(Key, 0666 | IPC_CREAT);
        if (MsgID == -1) {
      perror ("Padre.mssget - Error: ");
      exit (1);
        }

        pid_t pid = NULL;
        int I;

        //Se crea 1 proceso Oso
        pid = fork ();
        if (pid == -1) {
                fprintf (stderr,"Error al crear el Proceso Oso");
                exit (1);
        }
        if (pid == 0) {
                Oso ();
                exit (0);
        }

        if (pid > 0) {
                //Se crean los N procesos Abejas
                for (I = 0; I < N; I++){
                        pid = fork ();
                        if (pid == -1) {
                                fprintf (stderr,"Error al crear el Proceso Abeja");
                                exit (1);
                        }
                        if (pid == 0) {
                                Abeja ();
                                exit (0);
```

```
                }
            }
        }

        //Se inicializan los M espacios para producir miel
        for (I = 0; I < M-1; I++) {
            send (MsgID,tMiel);
        }
        send (MsgID,tUlt);
        //Se envia un mensaje para que la primera Abeja pueda entrar
        send (MsgID,tProd);

        //El proceso Padre espera por las N Abejas y 1 Oso
        for (I = 1; I <= N+1; I++) {
            wait(NULL);
        }

        return 0;
}
```

Secuencia1Pipes.c

```
#include <stdio.h>
#include <strings.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdbool.h>
#include <sys/types.h>
#include <sys/wait.h>

//ABCDE ABCDE ABCDE ABCDE...

/*

	A Hace: Recibe Mensaje PipeA; Imprime A; Envia Mensaje PipeB;

	B Hace: Recibe Mensaje PipeB; Imprime B; Envia Mensaje PipeC;

	C Hace: Recibe Mensaje PipeC; Imprime C; Envia Mensaje PipeD;

	D Hace: Recibe Mensaje PipeD; Imprime D; Envia Mensaje PipeE;

	E Hace: Recibe Mensaje PipeE; Imprime E; Envia Mensaje PipeA;

*/

//0 es para leer y 1 es para escribir

char writeMessage;
char readMessage;

int PipeA[2];
int PipeB[2];
int PipeC[2];
int PipeD[2];
int PipeE[2];

void A () {
	close (PipeA[1]);
	close (PipeB[0]);
	close (PipeC[0]);
	close (PipeC[1]);
	close (PipeD[0]);
	close (PipeD[1]);
	close (PipeE[0]);
	close (PipeE[1]);
```

```
        while (true) {
                //A recibe un mensaje de E
                read(PipeA[0],&readMessage,1);
                printf("A");
                fflush(NULL);

                //A envia un mensaje a B
    writeMessage = 'X';
                write(PipeB[1],&writeMessage,1);
                fflush(NULL);
        }
}

void B () {
        close (PipeA[0]);
        close (PipeA[1]);
        close (PipeB[1]);
        close (PipeC[0]);
        close (PipeD[0]);
        close (PipeD[1]);
        close (PipeE[0]);
        close (PipeE[1]);

        while (true) {
                //B recibe un mensaje de A
                read(PipeB[0],&readMessage,1);
                printf("B");
                fflush(NULL);

                //B envia un mensaje a C
    writeMessage = 'X';
                write(PipeC[1],&writeMessage,1);
                fflush(NULL);
        }
}

void C () {
        close (PipeA[0]);
        close (PipeA[1]);
        close (PipeB[0]);
        close (PipeB[1]);
        close (PipeC[1]);
        close (PipeD[0]);
        close (PipeE[0]);
        close (PipeE[1]);
```

```
while (true) {
        //C recibe un mensaje de B
        read(PipeC[0],&readMessage,1);
        printf("C");
        fflush(NULL);
```

```
                //C envia un mensaje a D
        writeMessage = 'X';
                write(PipeD[1],&writeMessage,1);
                fflush(NULL);
        }
}

void D () {
        close (PipeA[0]);
        close (PipeA[1]);
        close (PipeB[0]);
        close (PipeB[1]);
        close (PipeC[0]);
        close (PipeC[1]);
        close (PipeD[1]);
        close (PipeE[0]);

        while (true) {
                //D recibe un mensaje de C
                read(PipeD[0],&readMessage,1);
                printf("D");
                fflush(NULL);

                //D envia un mensaje a E
        writeMessage = 'X';
                write(PipeE[1],&writeMessage,1);
                fflush(NULL);
        }
}

void E () {
        close (PipeA[0]);
        close (PipeB[0]);
        close (PipeB[1]);
        close (PipeC[0]);
        close (PipeC[1]);
        close (PipeD[0]);
        close (PipeD[1]);
        close (PipeE[1]);

        while (true) {
                //E recibe un mensaje de D
                read(PipeE[0],&readMessage,1);
                printf("E");
                fflush(NULL);
```

```
                //E envia un mensaje a A
        writeMessage = 'X';
                write(PipeA[1],&writeMessage,1);
                fflush(NULL);
          }
}

void dispatch (int I) {
          switch (I) {
        case 1: A(); break;
        case 2: B(); break;
        case 3: C(); break;
        case 4: D(); break;
        case 5: E(); break;
    }
}

int main () {

          pipe (PipeA);
          pipe (PipeB);
          pipe (PipeC);
          pipe (PipeD);
          pipe (PipeE);

          pid_t pid = NULL;

          int I;
    int cantP = 5;
          //Se crean los 5 procesos A, B, C, D y E
    for (I = 1; I <= cantP; I++){
        pid = fork ();
        if (pid == -1) {
                        fprintf (stderr,"Error al crear el Proceso");
                }
        if (pid == 0){
            dispatch (I);
            exit (0);
        }
    }

          close (PipeA[0]);
          close (PipeB[0]);
          close (PipeB[1]);
          close (PipeC[0]);
          close (PipeC[1]);
```

```
        close (PipeD[0]);
        close (PipeD[1]);
        close (PipeE[0]);
        close (PipeE[1]);
        //El Padre envia un mensaje a A para empezar el ciclo
        writeMessage = 'X';
        fflush (NULL);
   write(PipeA[1],&writeMessage,1);
        fflush(NULL);

        for (I = 1; I <= cantP; I++) {
      wait (NULL);
   }

        return 0;
}
```

Secuencia2Pipes.c

```
#include <stdio.h>
#include <strings.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdbool.h>
#include <sys/types.h>
#include <sys/wait.h>

//(AoBoC)(AoBoC)DE (AoBoC)(AoBoC)DE (AoBoC)(AoBoC)DE...

/*

        A Hace: Recibe Mensaje PipeABC; Imprime A; Envia Mensaje PipeD;

        B Hace: Recibe Mensaje PipeABC; Imprime B; Envia Mensaje PipeD;

        C Hace: Recibe Mensaje PipeABC; Imprime C; Envia Mensaje PipeD;

        D Hace: Recibe Mensaje PipeDAux; Recibe Mensaje PipeD; Imprime D; Envia
Mensaje PipeE;

        E Hace: Recibe Mensaje PipeD; Envia Mensaje PipeDAux; Envia Mensaje PipeABC;
Recibe Mensaje PipeE; Imprime E; Envia Mensaje PipeABC;

*/

//0 es para leer y 1 es para escribir
```

```
char writeMessage;
char readMessage;

int PipeABC[2];
int PipeD[2];
int PipeE[2];
int PipeDAux[2];

void A () {
	close (PipeABC[1]);
	close (PipeD[0]);
	close (PipeE[0]);
	close (PipeE[1]);
	close (PipeDAux[0]);
	close (PipeDAux[1]);

	while (true) {
		//A recibe un mensaje de E
		read(PipeABC[0],&readMessage,1);
		printf("A");
		fflush(NULL);

		//A envia un mensaje a D
		writeMessage = 'X';
		write(PipeD[1],&writeMessage,1);
		fflush(NULL);
	}
}

void B () {
	close (PipeABC[1]);
	close (PipeD[0]);
	close (PipeE[0]);
	close (PipeE[1]);
	close (PipeDAux[0]);
	close (PipeDAux[1]);

	while (true) {
		//B recibe un mensaje de E
		read(PipeABC[0],&readMessage,1);
		printf("B");
		fflush(NULL);

		//B envia un mensaje a D
		writeMessage = 'X';
```

```
                write(PipeD[1],&writeMessage,1);
                fflush(NULL);
        }
}

void C () {
        close (PipeABC[1]);
        close (PipeD[0]);
        close (PipeE[0]);
        close (PipeE[1]);
        close (PipeDAux[0]);
        close (PipeDAux[1]);

        while (true) {
                //C recibe un mensaje de E
                read(PipeABC[0],&readMessage,1);
                printf("C");
                fflush(NULL);

                //C envia un mensaje a D
                writeMessage = 'X';
                write(PipeD[1],&writeMessage,1);
                fflush(NULL);
        }
}

void D () {
        close (PipeABC[0]);
        close (PipeABC[1]);
        close (PipeD[1]);
        close (PipeE[0]);
        close (PipeDAux[1]);

        while (true) {
                //D recibe un mensaje de E
                read(PipeDAux[0],&readMessage,1);
                fflush(NULL);

                //D recibe un mensaje de (AoBoC)
                read(PipeD[0],&readMessage,1);
                printf("D");
                fflush(NULL);

                //D envia un mensaje a E
                writeMessage = 'X';
                write(PipeE[1],&writeMessage,1);
```

```
                fflush(NULL);
        }
}

void E () {
        close (PipeABC[0]);
        close (PipeD[1]);
        close (PipeE[1]);
        close (PipeDAux[0]);

        while (true) {
                //E intercepta el primer mensaje para D enviado por (AoBoC)
                read(PipeD[0],&readMessage,1);
                fflush(NULL);

                //E le avisa a D que puede recibir el proximo mensaje de (AoBoC)
                writeMessage = 'X';
                write(PipeDAux[1],&writeMessage,1);
                fflush(NULL);

                //E envia un mensaje a (AoBoC)
                writeMessage = 'X';
                write(PipeABC[1],&writeMessage,1);
                fflush(NULL);

                //E recibe un mensaje de D
                read(PipeE[0],&readMessage,1);
                printf("E");
                fflush(NULL);

                //E envia un mensaje a (AoBoC)
                writeMessage = 'X';
                write(PipeABC[1],&writeMessage,1);
                fflush(NULL);
        }
}

void dispatch (int I) {
        switch (I) {
      case 1: A(); break;
      case 2: B(); break;
      case 3: C(); break;
      case 4: D(); break;
      case 5: E(); break;
   }
}
```

```
int main () {

        pipe (PipeABC);
        pipe (PipeD);
        pipe (PipeE);
        pipe (PipeDAux);

        pid_t pid = NULL;

        int I;
   int cantP = 5;
        //Se crean los 5 procesos A, B, C, D y E
   for (I = 1; I <= cantP; I++){
      pid = fork ();
      if (pid == -1) {
                        fprintf (stderr,"Error al crear el Proceso");
                }
      if (pid == 0){
         dispatch (I);
         exit (0);
      }
   }

        close (PipeABC[0]);
        close (PipeD[0]);
        close (PipeD[1]);
        close (PipeE[1]);
        close (PipeE[0]);
        close (PipeDAux[0]);
        close (PipeDAux[1]);
        //El Padre envia un mensaje a (AoBoC) para empezar el ciclo
        writeMessage = 'X';
        fflush (NULL);
        write(PipeABC[1],&writeMessage,1);
        fflush(NULL);

        for (I = 1; I <= cantP; I++) {
      wait (NULL);
   }

        return 0;
}
```

Secuencia3Pipes.c

```
#include <stdio.h>
#include <strings.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdbool.h>
#include <sys/types.h>
#include <sys/wait.h>

//ACDE BCDE ABCDE  ACDE BCDE ABCDE...

/*

	A Hace: Recibe Mensaje PipeA; Imprime A; Envia Mensaje PipeC; Recibe Mensaje PipeA; Imprime A; Envia Mensaje PipeB;

	B Hace: Recibe Mensaje PipeB; Imprime B; Envia Mensaje PipeC;

	C Hace: Recibe Mensaje PipeC; Imprime C; Envia Mensaje PipeD;

	D Hace: Recibe Mensaje PipeD; Imprime D; Envia Mensaje PipeE;

	E Hace: Recibe Mensaje PipeE; Imprime E; Envia Mensaje PipeB; Recibe Mensaje PipeE; Imprime E; Envia Mensaje PipeA; Recibe Mensaje PipeE; Imprime E; Envia Mensaje PipeA;

*/

//0 es para leer y 1 es para escribir

char writeMessage;
char readMessage;

int PipeA[2];
int PipeB[2];
int PipeC[2];
int PipeD[2];
int PipeE[2];

void A () {
	close (PipeA[1]);
	close (PipeB[0]);
	close (PipeC[0]);
	close (PipeD[0]);
	close (PipeD[1]);
```

```
        close (PipeE[0]);
        close (PipeE[1]);

        while (true) {
                //A recibe un mensaje de E
                read(PipeA[0],&readMessage,1);
                printf("A");
                fflush(NULL);

                //A envia un mensaje a C
    writeMessage = 'X';
                write(PipeC[1],&writeMessage,1);
                fflush(NULL);

                //A recibe un mensaje de E
                read(PipeA[0],&readMessage,1);
                printf("A");
                fflush(NULL);

                //A envia un mensaje a B
    writeMessage = 'X';
                write(PipeB[1],&writeMessage,1);
                fflush(NULL);
        }
}

void B () {
        close (PipeA[0]);
        close (PipeA[1]);
        close (PipeB[1]);
        close (PipeC[0]);
        close (PipeD[0]);
        close (PipeD[1]);
        close (PipeE[0]);
        close (PipeE[1]);

        while (true) {
                //B recibe un mensaje de A o E
                read(PipeB[0],&readMessage,1);
                printf("B");
                fflush(NULL);

                //B envia un mensaje a C
    writeMessage = 'X';
                write(PipeC[1],&writeMessage,1);
                fflush(NULL);
```

```
        }
}
```

```
void C () {
        close (PipeA[0]);
        close (PipeA[1]);
        close (PipeB[0]);
        close (PipeB[1]);
        close (PipeC[1]);
        close (PipeD[0]);
        close (PipeE[0]);
        close (PipeE[1]);

        while (true) {
                //C recibe un mensaje de A o B
                read(PipeC[0],&readMessage,1);
                printf("C");
                fflush(NULL);

                //C envia un mensaje a D
    writeMessage = 'X';
                write(PipeD[1],&writeMessage,1);
                fflush(NULL);
        }
}

void D () {
        close (PipeA[0]);
        close (PipeA[1]);
        close (PipeB[0]);
        close (PipeB[1]);
        close (PipeC[0]);
        close (PipeC[1]);
        close (PipeD[1]);
        close (PipeE[0]);

        while (true) {
                //D recibe un mensajde de C
                read(PipeD[0],&readMessage,1);
                printf("D");
                fflush(NULL);

                //D envia un mensaje a E
    writeMessage = 'X';
                write(PipeE[1],&writeMessage,1);
                fflush(NULL);
        }
}
```

```
void E () {
        close (PipeA[0]);
        close (PipeB[0]);
        close (PipeC[0]);
        close (PipeC[1]);
        close (PipeD[0]);
        close (PipeD[1]);
        close (PipeE[1]);

        while (true) {
                //E recibe un mensaje de D
                read(PipeE[0],&readMessage,1);
                printf("E");
                fflush(NULL);

                //E envia un mensaje a B para imprimir BCDE
    writeMessage = 'X';
                write(PipeB[1],&writeMessage,1);
                fflush(NULL);

                //E recibe un mensaje de D
                read(PipeE[0],&readMessage,1);
                printf("E");
                fflush(NULL);

                //E envia un mensaje a A para imprimir ABCDE
    writeMessage = 'X';
                write(PipeA[1],&writeMessage,1);
                fflush(NULL);

                //E recibe un mensaje de D
                read(PipeE[0],&readMessage,1);
                printf("E");
                fflush(NULL);

                //E envia un mensaje a A para imprimir ACDE
    writeMessage = 'X';
                write(PipeA[1],&writeMessage,1);
                fflush(NULL);
        }
}
```

```
void dispatch (int I) {
        switch (I) {
      case 1: A(); break;
      case 2: B(); break;
      case 3: C(); break;
      case 4: D(); break;
      case 5: E(); break;
   }
}

int main () {

        pipe (PipeA);
        pipe (PipeB);
        pipe (PipeC);
        pipe (PipeD);
        pipe (PipeE);
        pid_t pid = NULL;
        int I;
   int cantP = 5;
        //Se crean los 5 procesos A, B, C, D y E
   for (I = 1; I <= cantP; I++){
      pid = fork ();
      if (pid == -1) {
                        fprintf (stderr,"Error al crear el Proceso");
                }
      if (pid == 0){
         dispatch (I);
         exit (0);
      }
   }
        close (PipeA[0]);
        close (PipeB[0]);
        close (PipeB[1]);
        close (PipeC[0]);
        close (PipeC[1]);
        close (PipeD[0]);
        close (PipeD[1]);
        close (PipeE[0]);
        close (PipeE[1]);
        //El Padre envia un mensaje a A para empezar el ciclo
        writeMessage = 'X';
        fflush (NULL);
   write(PipeA[1],&writeMessage,1);
        fflush(NULL);
        for (I = 1; I <= cantP; I++) {
```

```
        wait (NULL);
    }
        return 0;
}
```

Secuencia2ColasMensajes.c

```
#include <stdio.h>
#include <strings.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdbool.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/ipc.h>
#include <sys/wait.h>

//(AoBoC)(AoBoC)DE (AoBoC)(AoBoC)DE (AoBoC)(AoBoC)DE...

/*

        El Padre comienza enviando un mensaje 1;

        A Hace: Recibe Mensaje 1; Imprime A; Envia Mensaje 4;

        B Hace: Recibe Mensaje 1; Imprime B; Envia Mensaje 4;

        C Hace: Recibe Mensaje 1; Imprime C; Envia Mensaje 4;

        D Hace: Recibe Mensaje 2; Recibe Mensaje 4; Imprime D; Envia Mensaje 3;

        E Hace: Recibe Mensaje 4; Envia Mensaje 2; Envia Mensaje 1; Recibe Mensaje 3;
Imprime E; Envia Mensaje 1;

*/

//Tipos de mensajes
#define tABC 1
#define tD 2
#define tE 3
#define tDE 4

key_t Key;

struct Buffer_M {
   long Tipo;
   char Texto;
};
```

```
typedef struct Buffer_M Msg;

int getQueue () { //Usado para conectar a la cola de mensajes (ya creada)
        int ID = msgget (Key, 0666);
        if (ID == -1){
                perror ("Error - msgget: ");
        }
        return ID;
}

void send (int qid, int tipo) { //Usado para enviar mensajes
   Msg M;
   M.Tipo = tipo;
   M.Texto = 'X';
        fflush (NULL);
        int Result = msgsnd (qid,&M,1,0666);
        fflush (NULL);
   if (Result == -1) {
      perror ("Error - msgsnd: ");
      exit(1);
   }
}

void receive (int qid, int tipo) { //Usado para recibir mensajes
   Msg M;
        fflush (NULL);
        int Result = msgrcv (qid,&M,1,tipo,0666);
        fflush (NULL);
   if (Result == -1){
      perror ("Error - msgrcv: ");
      exit(1);
   }
}

void A () {
        int MsgID = getQueue ();
        while (true) {
                //A recibe un mensaje de E
                receive (MsgID,tABC);
                printf ("A");

                //A envia un mensaje a D o E
                send (MsgID,tDE);
        }
}
```

```
void B () {
	int MsgID = getQueue ();
	while (true) {
		//B recibe un mensaje de E
		receive (MsgID,tABC);
		printf ("B");

		//B envia un mensaje a D o E
		send (MsgID,tDE);
	}
}

void C () {
	int MsgID = getQueue ();
	while (true) {
		//C recibe un mensaje de E
		receive (MsgID,tABC);
		printf ("C");

		//C envia un mensaje a D o E
		send (MsgID,tDE);
	}
}

void D () {
	int MsgID = getQueue ();
	while (true) {
		//D espera a que E le avise que puede recibir el mensaje de (AoBoC)
		receive (MsgID,tD);

		//D espera el mensaje del segundo (AoBoC)
		receive (MsgID,tDE);
		printf ("D");

		//D envia un mensaje a E
		send (MsgID,tE);
	}
}

void E () {
	int MsgID = getQueue ();
	while (true) {
		//E es notificado que ABC termino por pimera vez
		receive (MsgID,tDE);
```

```
                //Le avisa a D para que en el siguiente ciclo reciba el mensaje
                send (MsgID,tD);

                //E envia un mensaje a (AoBoC)
                send (MsgID,tABC);

                //E recibe un mensaje de D e imprime
                receive (MsgID,tE);
                printf ("E");

                //E envia un mensaje a (AoBoC)
                send (MsgID,tABC);
        }
}

void dispatch (int I) {
        switch (I) {
      case 1: A(); break;
      case 2: B(); break;
      case 3: C(); break;
      case 4: D(); break;
      case 5: E(); break;
   }
}

int main () {

        Key = ftok (".",10);

        //Se crea la cola de mensajes
        int MsgID = msgget(Key, 0666 | IPC_CREAT);
        if (MsgID == -1) {
      perror ("Padre.mssget - Error: ");
      exit (1);
        }

   pid_t pid = NULL;

   int I;
   int cantP = 5;
        //Se crean los 5 procesos A, B, C, D y E
   for (I = 1; I <= cantP; I++){
      pid = fork ();
      if (pid == -1) {
                        fprintf (stderr,"Error al crear el Proceso");
```

```
                }
        if (pid == 0){
            dispatch (I);
            exit (0);
        }
    }

            //El padre envia un mensaje a (AoBoC) para empezar el ciclo
            send (MsgID,tABC);

            for (I = 1; I <= cantP; I++) {
        wait (NULL);
    }

            return 0;
}
```

Secuencia3ColasMensajes.c

```
#include <stdio.h>
#include <strings.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdbool.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/ipc.h>
#include <sys/wait.h>

//ACDE BCDE ABCDE  ACDE BCDE ABCDE...

/*

        El Padre comienza enviando un mensaje 1;

        A Hace: Recibe Mensaje 1; Imprime A; Envia Mensaje 3; Recibe Mensaje 1; Imprime
A; Envia Mensaje 2;

        B Hace: Recibe Mensaje 2; Imprime B; Envia Mensaje 3;

        C Hace: Recibe Mensaje 3; Imprime C; Envia Mensaje 4;

        D Hace: Recibe Mensaje 4; Imprime D; Envia Mensaje 5;
```

```
        E Hace: Recibe Mensaje 5; Imprime E; Envia Mensaje 2; Recibe Mensaje 5; Imprime
E; Envia Mensaje 1; Recibe Mensaje 5; Imprime E; Envia Mensaje 1;

*/

//Tipos de mensajes
#define tA 1
#define tB 2
#define tC 3
#define tD 4
#define tE 5

key_t Key;

struct Buffer_M {
   long Tipo;
   char Texto;
};

typedef struct Buffer_M Msg;

int getQueue () { //Usado para conectar a la cola de mensajes (ya creada)
        int ID = msgget (Key, 0666);
        if (ID == -1){
                perror ("Error - msgget: ");
        }
        return ID;
}

void send (int qid, int tipo) { //Usado para enviar mensajes
   Msg M;
   M.Tipo = tipo;
   M.Texto = 'X';
        fflush (NULL);
        int Result = msgsnd (qid,&M,1,0666);
        fflush (NULL);
   if (Result == -1) {
      perror ("Error - msgsnd: ");
      exit(1);
   }
}

void receive (int qid, int tipo) { //Usado para recibir mensajes
   Msg M;
        fflush (NULL);
        int Result = msgrcv (qid,&M,1,tipo,0666);
```

```
            fflush (NULL);
    if (Result == -1){
        perror ("Error - msgrcv: ");
        exit(1);
    }
}

void A () {
        int MsgID = getQueue ();
        while (true) {
                //A recibe un mensaje de E
                receive (MsgID,tA);
                printf ("A");

                //A envia un mensaje a C
                send (MsgID,tC);

                //A recibe un mensaje de E
                receive (MsgID,tA);
                printf ("A");

                //A envia un mensaje a B
                send (MsgID,tB);
        }
}

void B () {
        int MsgID = getQueue ();
        while (true) {
                //B recibe un mensaje de A o E
                receive (MsgID,tB);
                printf ("B");

                //B envia un mensaje a C
                send (MsgID,tC);
        }
}

void C () {
        int MsgID = getQueue ();
        while (true) {
                //C recibe un mensaje de A o B
                receive (MsgID,tC);
                printf ("C");

                //C envia un mensaje a D
```

```
                send (MsgID,tD);
        }
}

void D () {
        int MsgID = getQueue ();
        while (true) {
                //D recibe un mensaje de C
                receive (MsgID,tD);
                printf ("D");

                //D envia un mensaje a E
                send (MsgID,tE);
        }
}

void E () {
        int MsgID = getQueue ();
        while (true) {
                //E recibe un mensaje de D
                receive (MsgID,tE);
                printf ("E");

                //E le envia un mensaje a B para imprimir la parte BCDE
                send (MsgID,tB);

                //E recibe un mensaje de D
                receive (MsgID,tE);
                printf ("E");

                //E le envia un mensaje a A para imprimir la parte ABCDE
                send (MsgID,tA);

                //E recibe un mensaje de D
                receive (MsgID,tE);
                printf ("E");

                //E le envia un mensaje a A para imprimir la parte ACDE
      send (MsgID,tA);
        }
}

void dispatch (int I) {
        switch (I) {
      case 1: A(); break;
      case 2: B(); break;
```

```
        case 3: C(); break;
        case 4: D(); break;
        case 5: E(); break;
    }
}

int main () {

        Key = ftok (".",10);

        //Se crea la cola de mensajes
    int MsgID = msgget(Key, 0666 | IPC_CREAT);
        if (MsgID == -1) {
        perror ("Padre.mssget - Error: ");
        exit (1);
        }

    pid_t pid = NULL;

    int I;
    int cantP = 5;
        //Se crean los 5 procesos A, B, C, D y E
    for (I = 1; I <= cantP; I++){
        pid = fork ();
        if (pid == -1) {
                        fprintf (stderr,"Error al crear el Proceso");
                }
        if (pid == 0){
            dispatch (I);
            exit (0);
        }
    }

        //El padre envia un mensaje a A para empezar el ciclo
        send (MsgID,tA);

    for (I = 1; I <= cantP; I++) {
        wait (NULL);
    }

        return 0;
}
```

EJERCICIO DE MEMORIA COMPARTIDA: ABEJAS Y EL OSO

Oso.c

```
#include "common.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <stdbool.h>
#include <sys/sem.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <semaphore.h>

int main () {

	Memoria* MemoriaCompartida;

	int I;

	int shmid = shmget(generateKey(), sizeof(Memoria), 0660);
	if (shmid == -1) {
		perror("Failed GET!");
		exit(1);
	}

	void* P = shmat(shmid, NULL, 0);
	if (P == (void*)-1) {
		perror("Failed ATTACH!");
		exit(1);
	}
	MemoriaCompartida = (Memoria*) P;

	while (true) {
		sem_wait(&MemoriaCompartida->Dormir);
		for (I = 0; I < M; I++) {
			sem_post(&MemoriaCompartida->Miel);
		}
		printf("El Oso consumio toda la miel.\n");
		sem_post(&MemoriaCompartida->Producir);
	}

	return 0;
}
```

abeja.c

```
#include "common.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <stdbool.h>
#include <sys/sem.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <semaphore.h>

int main () {

	Memoria* MemoriaCompartida;

	int T;

	int shmid = shmget(generateKey(), sizeof(Memoria), 0660);
	if (shmid == -1) {
		perror("Failed GET!");
		exit(1);
	}

	void* P = shmat(shmid, NULL, 0);
	if (P == (void*)-1) {
		perror("Failed ATTACH!");
		exit(1);
	}
	MemoriaCompartida = (Memoria*) P;

	while (true) {
		sem_wait(&MemoriaCompartida->Producir);
		sem_wait(&MemoriaCompartida->Miel);
		T = sem_trywait(&MemoriaCompartida->Miel);
		if (T == 0) {
			sem_post(&MemoriaCompartida->Miel);
			printf("Se produjo miel y quedan espacios, pasa la siguiente
Abeja.\n");
			sem_post(&MemoriaCompartida->Producir);
		}
		else {
			printf("Se produjo miel y se lleno el tarro, se despierta al Oso.\n");
			sem_post(&MemoriaCompartida->Dormir);
```

```
                }
        }

        return 0;
}
```

common.h: header que comparten los códigos fuente de la solución

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <unistd.h>
#include <stdio.h>
#include <semaphore.h>

int const N = 6; //Tengo N Abejas
int const M = 14; //Tengo M espacios para la miel

struct MemoryStruct {
        sem_t Miel; //Si las Abejas pueden hacer wait de este semaforo, es que aun hay
espacios vacios
        sem_t Producir; //Mutex que bloquea a las Abejas para que solo pasen de a una
        sem_t Dormir; //Mutex que bloquea al Oso hasta que se llene el tarro
};
typedef struct MemoryStruct Memoria;

key_t generateKey() {
        key_t Key = ftok("File",10);
        return Key;
}
```

init.c

```
#include "common.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <semaphore.h>

//ipcs para listar
//ipcrm -m id para eliminar el segmento

/*

   Mutex Bear = 0;
```

```
   Semaphore Bees = 0;
   Semaphore Empty = M;
   Mutex Mutex = 1;

   Oso {
      wait(Dormir)
      Repetir M Veces {
         signal(Miel)
      }
      signal(Producir)
   }

   Abeja {
      wait(Producir)
      wait(Miel)
      try_wait(Miel) {
         signal(Miel)
                        signal(Producir)
               }
      else {
         signal(Dormir)
      }
         }

*/

int main(int argc, char* argv[]) {

         Memoria* MemoriaCompartida;

         int pid;

         int I;

         key_t Key = generateKey();

         int shmid = shmget(Key, sizeof(Memoria), 0660 | IPC_CREAT);
         if (shmid == -1) {
                  perror("Failed GET!");
                  exit(1);
         }

         void* P = shmat(shmid, NULL, 0);
         if (P == (void*)-1) {
                  perror("Failed ATTACH!");
                  exit(1);
```

```
        }
        MemoriaCompartida = (Memoria*) P;

        sem_init(&MemoriaCompartida->Miel,1,M);
        sem_init(&MemoriaCompartida->Producir,1,1);
        sem_init(&MemoriaCompartida->Dormir,1,0);

        pid = fork();

        if (pid == -1) {
                fprintf (stderr,"Error al crear el Proceso Oso");
                exit(1);
        }
        if (pid == 0) {
                char *args[] = {"./oso",NULL};
                execvp(args[0],args);
        }
        if (pid > 0) {
                for (I = 0; I < N; I++){
                        pid = fork();
                        if (pid == -1) {
                                fprintf (stderr,"Error al crear el Proceso Abeja");
                                exit(1);
                        }
                        if (pid == 0) {
                                char *args[] = {"./abeja",NULL};
                                execvp(args[0],args);
                        }
                }
        }

        if (pid > 0) {
                for (I = 1; I <= N+1; I++) {
                        wait(NULL);
                }
        }

        if (shmdt(P) != 0) {
                perror("Failed DETTACH!");
                exit(1);
        }

        return 0;
}
```