UNIVERSIDAD DE MÁLAGA

Dpto. Lenguajes y Ciencias de la Computación

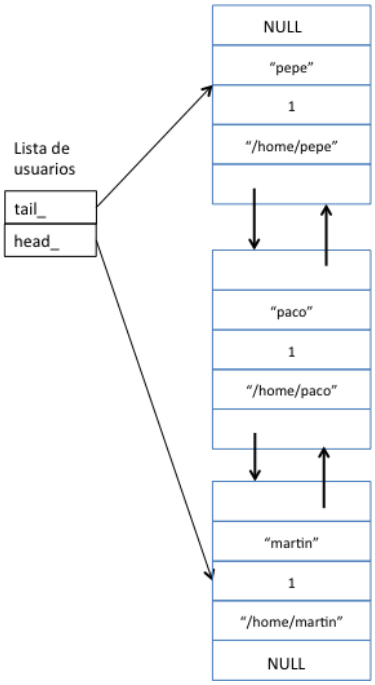
Programación de Sistemas y Concurrencia
Junio 20/6/2013

APELLIDOS_____NOMBRE_____

DNI _____ ORDENADOR _____GRUPO/TITULACIÓN_____

Ejercicio de C (Bloque 1)

Un sistema operativo usa una lista doblemente encadenada de tipo T_userList para gestionar los usuarios del sistema. Un usuario tiene un nombre, un identificador (UID) y un directorio home, de acuerdo a la siguiente definición:




```
typedef struct user {
   int   uid_          ;
   char *userName_     ;
   char *homeDirectory_ ;

   struct user * nextUser_ ;
   struct user * previousUser_ ;
} T_user ;
```

Definir el tipo T_userList con al menos los siguientes campos: head_ y tail_ (se pueden incluir alguno más si se estima oporturno). Las operaciones de la lista son :

T_user * createUser(char *name, int uid, char *dir);
Crea un usuario a partir de su nombre, UID y directorio home.

T_userList createUserList() ;
Devuelve una lista de usuarios vacía

int addUser(T_userList *, T_user*) ;
Añade un usuario en la cabeza de la lista previa comprobación que ni el nombre ni el UID ya están en la lista. Si se añade con éxito se devuelve 1 y 0 en caso contrario.

int getUid(T_userList list, char *userName) ; Dado el nombre de un usuario, busca su UID en la lista y lo devuelve si lo encuentra; en caso contrario, devuelve -1.

int deleteUser(T_userList *list, char* userName) ;
Borra a un usuario de la lista. Devuelve 0 si la operación tiene éxito y -1 si el usuario no existe.

void printUserList(T_userList list, int reverse) ;
Imprime la lista de cabeza a cola o viceversa dependiendo si el segundo argumento es 0 o 1.

Nota.- Se recomienda utilizar las funciones strcpy y strcmp de manejo de cadena de caracteres:

```
char *strcpy(char *s1, const char *s2);
```

Copia la cadena apuntada por **s2** (incluyendo el carácter nulo) a la cadena apuntada por **s1**

```
int strcmp(const char *s1, const char *s2);
```

Compara la cadena apuntada por **s1** con la cadena apuntada por **s2. Y**

Mayor que cero si s1>s2 (orden lexicográfico)

Cero si s1==s2 (orden lexicográfico)

Menos que cero si s1<s2 (orden lexicográfico)

A partir del programa principal que se suministra (main.c) se pide implementar los ficheros userList.h y userList.c. La salida por pantalla del programa principal debe ser idéntica a la mostrada en el fichero salida.txt, que también se suministra.

Ejercicio de semáforos (Bloque 2)

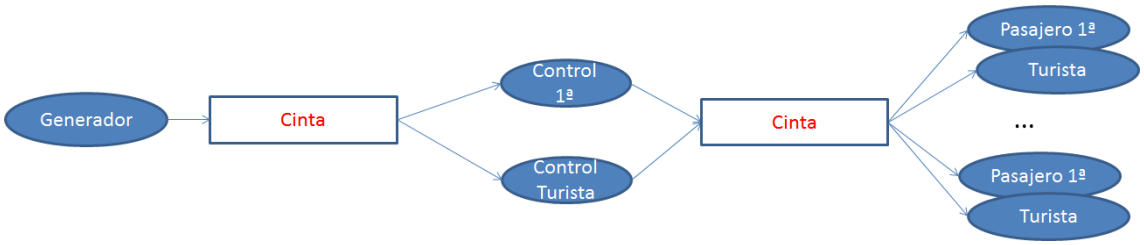
Considera un sistema con dos productores y un consumidor, en el que el consumidor es más rápido que los dos productores. Los tres procesos se comunican a través de un buffer compartido pero, para acelerar la producción, un productor utiliza las componentes pares del buffer, y el otro las impares. Resuelve este problema utilizando sólo SEMÁFOROS BINARIOS, de manera que:
- el buffer se utiliza en exclusión mutua;
- el consumidor lee los datos del buffer de forma ordenada, es decir, primero el dato que está en la posición 0, luego el que está en la posición 1, etc;
- el buffer se utiliza de forma circular;
- el productor "par" no puede escribir sobre el buffer si todas las componentes pares están ocupadas;
- el productor "impar" no puede escribir sobre el buffer si todas las componentes impares están ocupadas;
- el consumidor no puede leer el dato i-ésimo, si el correspondiente productor (par o impar) no lo ha almacenado todavía.

Para resolver el problema debes escribir:
- Una clase Buffer con los métodos void nuevoDatoPar(int), nuevoDatoImpar(int), leerDato() usados por los productores para poner datos en el buffer y por el consumidor para leerlos. Supón que los mensajes son enteros, y que el tamaño del buffer es par.
- Las clases Productor y Consumidor que simulen el comportamiento de los procesos, suponiendo que todos se ejecutan de forma ininterrumpida.
- Una clase Driver que cree un sistema como el descrito, y lo ponga en ejecución.

Ejercicio de monitores (Bloque 2)

Se quiere simular el sistema de gestión de cintas de transporte de equipajes en un aeropuerto. Para ello, disponemos de una hebra generadora que va "generando" maletas de primera clase o de clase turista de manera aleatoria. Las maletas son colocadas en una cinta de transporte intermedia con capacidad finita, y en ella, son retiradas de una en una por dos hebras controladoras, una para retirar maletas de primera clase y otra para retirar maletas de clase turista. Una vez que las hebras controladoras han retirado una maleta, proceden a colocarla en una segunda cinta (también con capacidad finita), que es a la que tienen acceso los pasajeros, teniendo hebras representando a los pasajeros de primera clase y a los turistas. Las hebras de primera clase (controladora y pasajera) tienen prioridad sobre las de clase turista, de manera que cuando se van a retirar maletas, si en ese momento se están retirando maletas de primera clase, las hebras de clase turista tendrán que esperar (al revés no hay espera).

La siguiente figura muestra una representación del sistema a modelar.



El comportamiento de todas las hebras es infinito. La retirada de las maletas de la cinta supone un tiempo máximo de 1 segundo. Además, en el caso de las hebras de pasajeros de primera clase, podemos suponer que retiran maletas cada 10 segundos, mientras que los de clase turista lo hacen una vez por segundo. Pista: Puesto que sólo interesa el número de maletas de cada tipo, no es necesario modelar arrays al estilo productor/consumidor en las cintas de transporte.

Con el fin de facilitar la implementación del sistema, en el campus virtual se encuentran varios ficheros parcialmente implementados. Así, `Generador.java` contiene la implementación de la hebra generadora de maletas. `Control.java` contiene la implementación de las hebras controladoras. `Pasajero.java` contiene el código de las hebras de pasajeros. `Cinta.java` tiene los métodos necesarios para colocar y retirar maletas de la cinta. De esta forma, el método `poner` es utilizado para colocar maletas en la cinta. Los métodos `qRetirarPrimera` y `qRetirarTurista` son utilizados para indicar que se quiere comenzar a retirar maletas de primera clase o de clase turista. Por su parte, los métodos `fRetirarPrimera` y `fRetirarTurista` son invocados cuando se ha finalizado la retirada (en el código de la hebra) de una maleta de primera clase o de clase turista. Finalmente, en `maletas.java` se encuentra el método `main`.

Cada estudiante puede decidir si quiere seguir la estructura sugerida en esta solución, o implementar la solución de otra manera.

Ejercicio de paso de mensajes (Bloque 3)

Se quiere implementar el juego del “tú te llevas el dinero”. Este juego consiste en repartir una cantidad de dinero entre un conjunto de jugadores que sólo puede tener contacto con un único predecesor y un sucesor, salvo el primero (Jugador 0) que sólo tiene sucesor y el último que sólo tiene predecesor. El jugador 0 es la persona encargada de sacar un numero de una bolsa y una cantidad de dinero. Él número extraído indica por cuántas personas tiene que circular el dinero para que llegue al ganador. Este ganador recibirá la cantidad y, además de disfrutar del premio, le comunicará al jugador 0 que él ha sido el vencedor para que pueda publicarlo con un mensaje. Si el número extraído de la bolsa es mayor que el número de personas, es decir, si la última persona que lo puede recibir todavía tiene que reenviarlo, indicará que no hay vencedor para que el jugador 0 lo publique y el dinero se lo quedará él. Realizar una implementación basada en paso de mensajes.