Informática Industrial

4º GIERM, curso 2016-17

Práctica 3

Mutex y variables de condición

1. Introducción

El trabajo consiste en completar una aplicación en C utilizando las llamadas POSIX para controlar el sistema de la Figura 1. La comunicación entre los hilos de la aplicación se realizará mediante variables compartidas, y la sincronización mediante **mutex y variables de condición utilizando un monitor**.

Para el desarrollo de la práctica se proporcionará a los alumnos el código C de los diferentes módulos de la aplicación, aunque con funcionalidad errónea o limitada. Es responsabilidad del alumno modificar y/o completar el código fuente y compilarlo para obtener un comportamiento satisfactorio. La práctica puede realizarse sobre QNX 6.3, Kubuntu 12.04 o Ubuntu 14.04 utilizando los ficheros adecuados.

Figura 1: Representación gráfica

2. Funcionamiento del sistema

En el sistema de la Figura 1 existen dos tipos de almacenes: entrada y salida. Sólo hay un almacén de salida, pero existen **N_ALM_IN** almacenes de entrada. Los almacenes de entrada tienen capacidad para **una sola pieza**, mientras que el de salida tiene capacidad para varias, hasta un máximo de **MAX_PIEZAS_OUT**. Para evitar conflictos en el acceso a las piezas de la entrada y a los lugares para piezas de la salida existirá un procedimiento de reserva para ambos. Existen también

N_VEH vehículos que permanecerán en su aparcamiento mientras no puedan realizar una operación de transporte:

- Si el vehículo está vacío y en el aparcamiento, espera hasta reservar una pieza en alguno de los almacenes de entrada.
- Si el vehículo contiene una pieza y está en el aparcamiento, espera hasta reservar un lugar en el almacén de salida.

Por simplificar a la primera se le llamará "**carga**" y a las segunda "**descarga**":

- En una operación de carga el vehículo reserva una pieza disponible, se dirige al almacén de entrada correspondiente, carga y luego toma el camino de vuelta (ver Figura 1). Cuando llega a la bifurcación determina si es posible descargar la pieza en el almacén de salida; si lo es, reserva un lugar, avanza por el camino del almacén de salida, descarga y vuelve al aparcamiento vacío. Si no lo es sigue su camino y vuelve al aparcamiento con su carga.
- En una operación de descarga el vehículo sigue el mismo camino que en una operación de carga, pero ahora no utiliza ningún almacén de entrada, y se dirige directamente al de salida, donde ya ha reservado un lugar para descargar. Luego vuelve vacío a su aparcamiento.

Los pulsadores **IN0** a **IN3** permiten aportar piezas a los almacenes de entrada (nunca más de una), mientras que el pulsador **OUT** permite retirar piezas del almacén de entrada.

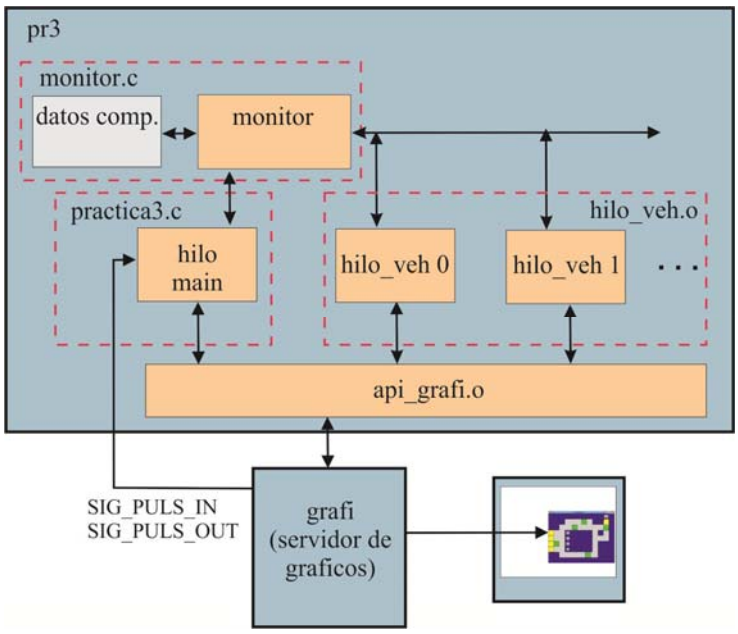

Figura 2: Procesos e hilos del sistema

3. Arquitectura software

La aplicación está formada por

- El proceso **pr3**, cuyo ejecutable se compone de los módulos **practica3.c**, **monitor.c**, **hilo_veh.o** y **api_grafi.o** (interfaz de gráficos). En el proceso **pr3** existen lo siguientes

hilos, que se comunican y sincronizan entre sí por medio del monitor contenido en **monitor.c**:

- **main**: Arranca la interfaz de gráficos de la ventana principal, inicializa el monitor, y crea a los demás hilos.
- **N_VEH** hilos de control de vehículo con función de arranque **hilo_veh**, que debe recibir como argumento una estructura **arg_veh** con los datos adecuados (ver cabecera **practica3.h**). Las diferencias en su funcionamiento proceden de los diferentes argumentos que reciben; cada hilo controla un vehículo distinto, se sincroniza con el resto utilizando las llamadas del monitor, sigue el camino descrito en el apartado 2 del enunciado e intenta evitar colisiones. Toda esta funcionalidad está ya resuelta, pero depende de que las funciones del monitor se implementen adecuadamente.

- El proceso **grafi**, que actúa como servidor de gráficos. Las funciones de **grafi** se utilizan por medio de **api_grafi.o**.

El alumno debe completar los ficheros **practica3.c** y **monitor.c** para que la funcionalidad sea la descrita y se corresponda con el ejecutable de referencia del que también se dispone (**ej_referencia/pr3**). El código de los hilos de control de vehículo no se suministra, sólo el módulo **hilo_veh.o** que ha de enlazarse con el resto del programa.

Señales de pulsadores:

Los pulsadores para incrementar o decrementar los almacenes de entrada y salida envían a **pr3** señales **SIG_PULS_IN** y **SIG_PULS_OUT** respectivamente cuando se pulsan. Las señales **SIG_PULS_IN** van acompañadas de un índice de pulsador entre **0** y **N_ALM_IN-1**.

Algunos detalles sobre el monitor:

Para realizar la sincronización y comunicación entre los hilos de **pr3** se utilizan un monitor que gestiona internamente las peticiones de carga y descarga con mutex y variables de condición. Las funciones de entrada al monitor se definen en la cabecera **monitor.h**, y el código está en **monitor.c**. En los comentarios de ambos ficheros se explica qué argumentos reciben estas funciones y qué deben hacer.

Arranque del sistema de soporte gráfico:

Como no se crean procesos aparte de **pr3**, no es necesario utilizar ningún procedimiento especial. El sistema de gráficos se crea en **main**, y **cualquier hilo puede utilizarlo directamente**. De hecho actualmente **no es posible** abrir varias veces simultáneamente la misma pantalla gráfica desde el mismo proceso en diferentes hilos; sólo puede hacerse desde uno, pero el identificador obtenido lo puede utilizar el resto. Se deben inicializar los gráficos antes que el monitor, porque éste los utiliza.

4. Trabajo del alumno

El trabajo a realizar consta de dos partes:

- **Parte obligatoria**: A realizar en el laboratorio en horario de prácticas. Consiste en completar y corregir el código que se proporciona para conseguir la funcionalidad descrita en los puntos anteriores. En particular es necesario que ningún hilo se quede

realizando continuamente operaciones durante un tiempo prolongado de modo que lleve el procesador al 100%.

- **Parte voluntaria**: El alumno podrá realizar mejoras y desarrollos adicionales de la práctica, que influirán positivamente en la calificación final. Como ejemplo se proponen las siguientes modificaciones, realizándolas con los medios de comunicación y sincronización utilizados en esta práctica:

    - Añadir una condición de sobretiempo que permita cargar piezas en los almacenes cuando están vacíos y retirar piezas del almacén de salida cuando está lleno de manera automática, si tales situaciones se prolongan demasiado tiempo.
    - Añadir un pulsador para realizar una parada ordenada del sistema de modo que todos los vehículos acaben la operación en curso y terminen aparcados en su aparcamiento vacíos antes de cerrar los gráficos y terminar la aplicación.
    - Modificar la aplicación para que se admita más de una pieza en los almacenes de entrada, sin dar lugar a colisiones entre vehículos.
    - Modificar la gestión del acceso a pasillos para minimizar las paradas de los vehículos sin dar lugar a colisiones
    - Realizar una implementación de la función de arranque **hilo_veh**.

Deberá entregarse la siguiente documentación:

- Memoria del trabajo realizado, incluyendo
    - Explicación detallada del funcionamiento general del programa. **No se trata de repetir el enunciado**.
    - Explicación de los errores detectados y las acciones realizadas para corregirlos.
- Ficheros fuentes, de manera que pueda comprobarse el funcionamiento de la aplicación.

5. Ficheros disponibles

- **practica3.c**: Fichero fuente para **main** del proceso **pr3** (a completar y corregir).
- **monitor.c**: Fichero fuente para el monitor (a completar y corregir).
- **practica3.h**: Cabecera común para los programas de la práctica 3.
- **monitor.h**: Cabecera para utilizar el monitor (no hay que cambiarlo para la parte obligatoria).
- **hilo_veh.o**: Fichero objeto para los hilos de control de vehículos.
- **api_grafi.o**: Fichero objeto con las funciones de interfaz para el servidor de gráficos.
- **api_grafi.h**: Cabecera para las funciones de **api_grafi.o**.
- **grafi**: Ejecutable del servidor de gráficos.
- **ej_referencia/pr3**: Ejecutable de **pr3** sin fallos, para utilizarlo como referencia.

Para compilar el ejecutable **pr3** pueden utilizarse los siguientes comandos en QNX 6.3.2:

gcc –o pr3 practica3.c monitor.c hilo_veh.o api_grafi.o

Si en el directorio sólo están los ficheros de la práctica basta con hacer: **gcc –o pr3 *.c *.o**

En Ubuntu el comando es algo distinto:

gcc –o pr3 practica3.c monitor.c hilo_veh.o api_grafi.o –lrt -lpthread

Del mismo modo, si en el directorio sólo están los ficheros de la práctica este comando podría abreviarse a **gcc –o pr3 *.c *.o –lrt -lpthread**

6. Cabeceras

6.1. Cabecera practica3.h

```
/* Cabecera para practica 3 de INFI (11/2016)  */
/* Copyright (C) Joaquin Ferruz Melero 2016 */

#ifndef    __PRACTICA3_H__
#define    __PRACTICA3_H__

/* Acceso a interfaz grafica */

#include "api_grafi.h"

/* Hilo de vehiculo */

struct arg_veh {
    int fila;          /* Fila inicial */
    int col;           /* Columna inicial */
    int id;            /* Identificador entre 0 y N_VEH-1 */
    id_grafi_t idgr;   /* Identif. de los graficos para que pueda dibujar */
};

void *hilo_veh(void *p); /* Funcion de arranque del hilo de vehiculo */

/* Numero de elementos */

#define N_VEH          5     /* Numero de vehiculos */
#define N_ALM_IN       4     /* Numero de posiciones de almacenamiento */
#define N_PASILLOS     3     /* Numero de pasillos */
#define MAX_PIEZAS_OUT 3     /* Max. numero de piezas en almacen de salida */

/* Pasillos (ver figura) */

enum tipo_pasillo {APARCAMIENTO, ENTRADA, SALIDA};

/* Senales de pulsadores */

#define SIG_PULS_IN    SIGRTMIN   /* Senal para los pulsadores de entrada */
#define SIG_PULS_OUT   SIGRTMIN+1 /* Senal para los pulsadores de salida */

/* Dimensiones del dibujo */

#define ANCHO          12
#define ALTO           9
#define ESCALA_DIB     40

/* Filas y columnas */

#define FILA_PULS_IN   3.25  /* Fila de los pulsadores de entrada */
#define COL_PULS_IN    3.25  /* Columna del primer pulsador de entrada */
#define FILA_PULS_OUT  0.25  /* Fila del pulsador de salida */
```

```
#define COL_PULS_OUT  10.15 /* Columna del pulsador de salida */

#define FILA_APARCAMIENTO   1     /* Fila de los aparcamientos */
#define COL_APARCAMIENTO    3     /* Columna del aparcamiento 0 */
#define FILA_ALM_IN         3     /* Fila del primer almacen de entrada */
#define COL_ALM_IN          0     /* Col. de los almacenes de entrada */
#define FILA_ALM_OUT        1     /* Fila del almacen de salida */
#define COL_ALM_OUT         10    /* Col. del almacen de salida */
#define FILA_DESV_OUT       4     /* Fila de la desviacion para ir a la
                                     salida */
#define COL_DESV_OUT   ANCHO-2    /* Col. de la desviacion para ir a la
                                     salida */
#define FILA_VUELTA    ALTO-2     /* Fila del pasillo inferior para volver al
                                     aparcamiento */
#define COL_VUELTA     ANCHO-4    /* Col. del pasillo de la derecha para
                                     volver al aparcamiento */
#endif
```

6.2.Cabecera monitor.h

```
/* Cabecera para el monitor de la practica 3 de INFI 11/16 */
/* Copyright (C) Joaquin Ferruz Melero 2016 */

#ifndef    __MONITOR_H__
#define    __MONITOR_H__

/* Funciones de acceso al monitor */

/* Inicializacion del monitor */
/* Hay que utilizarla antes de invocar cualquier otra */

void ini_monitor(id_grafi_t id);

/* Pedir pasillo */

void pedir_pasillo(int npasillo);

/* Liberar pasillo  */

void liberar_pasillo(int npasillo);

/* Incrementar contenido del almacen de entrada */
/* Debe utilizarse para ejecutar la pulsacion de teclado correspondiente */
/*  nalm: Numero del almacen que ha de incrementarse */
/*  El contenido del almacen no puede superar 1 */

void inc_alm_in(int nalm);

/* Decrementar contenido del unico almacen de salida */
/* Debe utilizarse para ejecutar la pulsacion de teclado correspondiente */
/* El contenido no puede ser menor que cero */

void dec_alm_out(void);


/* Reservar un lugar para la pieza si es posible */
/* NO ESPERA a que haya lugar disponible */
/* nveh: Vehiculo que intenta descargar */
```

```
/* Devuelve 0 si no ha sido posible reservar, y 1 si se ha podido
    realizar la reserva y por tanto se puede descargar */

int reservar_salida(int nveh);

/* Esperar hasta haber reservado un lugar en el almacen de salida */
/* Utilizada por el vehiculo cuando llega cargado al aparcamiento */
/* nveh: Vehiculo que ha de esperar */
/* Debe devolver 0 (lugar disponible)  */
/* Si devolviese -1 el vehiculo desaparece y su hilo acaba */
/* Si devuelve cualquier otra cosa, el vehiculo se bloquea */

int esperar_salida_disponible(int nveh);

/* Esperar a que exista una pieza no reservada en algun almacen de entrada */
/* Utilizada por el vehiculo cuando esta vacio en el aparcamiento */
/*  nveh: Vehiculo que ha de esperar */
/* Debe devolver un numero valido de almacen en el que el vehiculo deberia
cargar */
/* Si devuelve -1 el vehiculo desaparece y su hilo acaba */
/* Si devuelve cualquier otra cosa, el vehiculo se bloquea */

int esperar_entrada_disponible(int nveh);

/* Cargar de almacen de entrada */
/* Utilizada por el vehiculo cuando esta ante el almacen de entrada */
/* nveh: Numero de vehiculo que carga */

void cargar(int nalm, int nveh);

/* Descargar de almacen de entrada */
/* Utilizada por el vehiculo cuando esta ante el almacen de salida */
/* nveh: Numero de vehiculo que carga */

void descargar(int nveh);

#endif
```