Threads y cola de mensaje

Al utilizar cola de mensajes en un proceso que contenga hilos, es necesario crear la cola de mensajes desde el hilo principal del proceso (función *main*).

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


    id_cola_mensajes    = creo_id_cola_mensajes(CLAVE_BASE);

}
```

Luego de este paso, ya se estaría en condiciones de utilizar la cola de mensajes. Uno de los requerimientos de la cola de mensajes es poder comunicar datos o mensajes entre distintos procesos, y cada proceso tiene que tener un ID único e irrepetible.

Por ejemplo: *"Una estación de peaje está representada por un proceso PEAJE con varios threads (uno por vía))*, y un proceso AUTOS"; suponer que cada vía tiene que poder enviar y recibir mensajes. Aquí surge un *inconveniente*: las vías son hilos. ¿Cómo se le asigna un ID único e irrepetible a varias vías? Y encima no se conoce la cantidad de vías que se van a utilizar.

¿Cómo se puede solucionar?

En principio, tener en cuenta que se tendrá que pasar por parámetro al *thread* el *id_cola_mensajes* (para usar la cola de mensajes); esto se realiza agrupando datos con un *struct* (recordar que la función del *thread* sólo recibe por parámetro una dirección de memoria del tipo genérico -void). El *struct* sirve para pasarle más de un parámetro a la función.

Ejemplo struct:

```
typedef struct tipo_peaje tpeaje;
struct tipo_peaje
{
    int nro_via;
    int id_colamensaje;

};
```

Luego, para crear y lanzar el hilo, se usa:

```
    tpeaje *datos_thread;

pthread_t *idHilo = (pthread_t *)malloc(sizeof(pthread_t));
    pthread_attr_t atributos;
    pthread_attr_init(&atributos);
    pthread_attr_setdetachstate(&atributos, PTHREAD_CREATE_JOINABLE);

    datos_thread = (tpeaje *)malloc(sizeof(tpeaje));


    datos_thread.nro_via = 1;
    datos_thread.id_colamensaje = id_cola_mensajes;


    pthread_create(&idHilo, &atributos, ThreadVia, &datos_thread);
```

Viendo el código anterior, se lanza un hilo, pero hay algo que no se tuvo en cuenta: ¿cómo se lanzan y crean *Nhilos*?

Suponer que la cantidad de vías se la recibe por parámetro del proceso:

```
int main(int argc, char *argv[])
{
    int id_cola_mensajes;
    int i, ctl = 0;
    int cantidad=1;

    tpeaje *datos_thread;

    srand(time(NULL));

    if (argc>1)
        cantidad = atoi(argv[1]);

    printf("%d\n", cantidad);

    id_cola_mensajes    = creo_id_cola_mensajes(CLAVE_BASE);

    pthread_t* idHilo = (pthread_t* ) malloc(sizeof(pthread_t)*cantidad);
    pthread_attr_t  atributos;
    pthread_attr_init (&atributos);
    pthread_attr_setdetachstate (&atributos, PTHREAD_CREATE_JOINABLE);

    datos_thread = (tpeaje*) malloc(sizeof(tpeaje)*cantidad);
```

```
for(i=0; i<cantidad; i++)
    {
       datos_thread[i].nro_via = i;
       datos_thread[i].id_colamensaje = id_cola_mensajes;

       pthread_create (&idHilo[i], &atributos, ThreadVia, &datos_thread[i]);
    }


    return 0;
}
```

Observando el código, se puede observar que cada hilo tiene su propio id de hilo (no es el mismo de la cola de mensajes) y, al lanzarlo con la funcionpthread_create, cada vía contiene su propios datos que recibe por parámetro; para que esto sea más visible, cada hilo tiene el número de vía que varía en cada vuelta del comando “for” (bucle).

Hasta acá se pudo resolver cómo lanzar más de un hilo; queda ver cómo se logra que cada vía tenga un ID único para poder enviar y recibir mensajes. Para esto, se debe utilizar el número de vía. Mirar en el archivo definiciones.h (que es donde se van a identificar los IDs) y buscar el “*enum*”:

```
typedef enum
{
    MSG_NADIE, // MSG_NADIE = 0
    MSG_AUTOS, // MSG_AUTOS = 1
    MSG_VIAS,  // MSG_VIAS = 2
} Destinos;
```

Importante: para no repetir los valores, es fundamental el orden que se va a utilizar en el “enum”; en este caso, se ubica a “MSG_VIAS” al final del “enum”, ya que es el proceso que va a utilizar N hilos.

Ahora, observar el código de la función ThreadVia (la función del hilo):

```
void *ThreadVia (void *parametro)
{
    int             nro_via;
    int             id_cola_mensajes;
    int             done=0;
```

```
    mensaje     msg;

    tpeaje *datos_thread = (tpeaje*) parametro;
    nro_via = datos_thread->nro_via;
    id_cola_mensajes = datos_thread->id_colamensaje;


    printf("Soy la via %d\n", nro_via+1);


    while(done==0)
    {
     printf("\nVIA:%d DESTINO:%d\n", nro_via, MSG_VIAS+nro_via);

     recibir_mensaje(id_cola_mensajes, MSG_VIAS+nro_via, &msg);  //bloqueate

     //Logica para los eventos.

    };
    pthread_exit ((void *)"Listo");
}
```

Lo primero que hay que hacer es convertir la dirección de memoria genérica, que se recibe por parámetro, en una dirección de memoria válida con el tipo de dato que corresponde. Para esto se usa el casteo:

```
    tpeaje *datos_thread = (tpeaje*) parametro;
```

Luego, se usan las variables locales para que tomen los valores recibidos por parámetro:

```
    nro_via = datos_thread->nro_via;
    id_cola_mensajes = datos_thread->id_colamensaje;
```

Ya es posible utilizar el número de vía y la cola de mensajes; como ejemplo se utiliza la función recibir_mensaje:

```
recibir_mensaje(id_cola_mensajes, MSG_VIAS+nro_via, &msg);
```

Recordar que, en la función recibir mensaje, se debe utilizar el ID del proceso en el que se está ubicado; en este caso, es el MSG_VIAS, y se le suma a este valor el nro_via para que sea un valor único.

Hay que comprobar que sea un valor único:

- MSG_VIAS = 2, en el **primer** hilo nro_via vale “0”, porque la variable “i” arranca de “0” en el “for” que se lanza el hilo y es lo que se le asigna a cada hilo. Entonces 2+0 es igual a 2 → primer ID.
- MSG_VIAS = 2, en el **segundo** hilo nro_via vale “1”, porque la variable “i” se incrementa de a uno y arranca de “0” en el “for” que se lanza el hilo y es lo que se le asigna a cada hilo. Entonces 2+1 es igual a 3 → segundo ID.
- MSG_VIAS = 2, en el **tercer** hilo nro_via vale “2”, porque la variable “i” se incrementa de a uno y arranca de “0” en el “for” que se lanza el hilo y es lo que se le asigna a cada hilo. Entonces 2+2 es igual a 4 → tercer ID.
- MSG_VIAS = 2, en el **N** hilo nro_via vale “i”, porque la variable “i” se incrementa de a uno y arranca de “0” en el “for” que se lanza el hilo y es lo que se le asigna a cada hilo. Entonces 2+i → N ID.

Si no se respeta el orden en el “enum” se podría perder el valor único que tiene que tener el ID (remitente / destinatario) para comunicarse con la cola de mensajes. Es posible verificarlo invirtiendo el orden entre MSG_AUTO y MSG_VIAS en el “enum”; se puede ver que MSG_AUTO, al estar al final del “enum”, puede tener un valor igual que algún hilo (MSG_VIAS + nro_via).

Hay que respetar el orden en el enum.