DESARROLLO DE SOFTWARE PARA SISTEMAS EMPOTRADOS

Mecanismos básicos de sincronización en POSIX

Mario Aldea
Michael González
Héctor Pérez

Mecanismos básicos de sincronización en POSIX

- Sincronización en POSIX: Conceptos básicos
- Mutexes

Sincronización en POSIX: Conceptos básicos

- Los procesos o *threads* pueden:
  - ser *independientes* de los demás
  - *cooperar* entre ellos
  - *competir* por el uso de recursos compartidos
- En los dos últimos casos, los procesos o threads deben sincronizarse para:
  - intercambiar datos o eventos, mediante algún mecanismo de intercambio de mensajes
  - usar recursos compartidos (p.ej., datos o dispositivos) de manera mutuamente exclusiva
    - para asegurar su **congruencia** ante accesos concurrentes
    - además, el acceso al recurso puede ser condicional
      - sólo se puede acceder a él si se encuentra en el estado apropiado

DSSE: M. Aldea, M. González y H. Pérez

Sincronización en POSIX: Conceptos básicos

- Mecanismos de sincronización definidos en POSIX:
  - Sincronización por intercambio de mensajes:
    - señales
    - semáforos contadores
    - colas de mensajes
  - Exclusión mutua:
    - semáforos contadores
    - **mutexes**
    - variables condicionales
- POSIX no proporciona *monitores* ni *regiones críticas*
  - aunque **pueden** construirse utilizando semáforos o mutexes y variables condicionales

Mutexes

- El *mutex* representa un objeto de sincronización
  - por el que múltiples threads/procesos acceden de forma **mutuamente exclusiva** a un recurso
  - en cada instante, cada mutex tiene un único propietario
- Operaciones:
  - tomar o bloquear (***lock***):
    - si el mutex está libre, se toma y el thread se convierte en propietario
    - si el mutex no está libre, se suspende y se añade a una cola
  - liberar (***unlock***):
    - si hay threads esperando en la cola, el de mayor prioridad toma el mutex y se convierte en su nuevo propietario
    - si no, el mutex queda libre
  - sólo el propietario del mutex puede invocar la operación ***unlock***

Mutexes: Atributos de inicialización

- *Atributo pshared:* indica si el mutex es compartido o no entre procesos
  - PTHREAD_PROCESS_SHARED
  - PTHREAD_PROCESS_PRIVATE
- *Atributo protocol:* indica el protocolo de sincronización utilizado por el mutex
  - PTHREAD_PRIO_NONE: sin protocolo
  - PTHREAD_PRIO_INHERIT: herencia de prioridad (BIP)
  - PTHREAD_PRIO_PROTECT: techo de prioridad (HLP)
- *Atributo prioceiling*:
  - indica el techo de prioridad (valor entero) para la política PTHREAD_PRIO_PROTECT
- Estos atributos se almacenan en un objeto de atributos del tipo ***pthread_mutexattr_t***
  - los valores por defecto **no están definidos por el estándar**

Mutexes: Funciones POSIX para los atributos

```
#include <pthread.h>

int pthread_mutexattr_init (pthread_mutexattr_t *attr);
int pthread_mutexattr_destroy (pthread_mutexattr_t *attr);

int pthread_mutexattr_getpshared (const pthread_mutexattr_t *attr,
                                  int *pshared);
int pthread_mutexattr_setpshared (pthread_mutexattr_t *attr,
                                  int pshared);

int pthread_mutexattr_getprotocol (const pthread_mutexattr_t *attr,
                                   int *protocol);
int pthread_mutexattr_setprotocol (pthread_mutexattr_t *attr,
                                   int protocol);


int pthread_mutexattr_getprioceiling (const pthread_mutexattr_t *attr,
                                      int *prioceiling);
int pthread_mutexattr_setprioceiling (pthread_mutexattr_t *attr,
                                      int prioceiling);
```


Mutexes: Funciones POSIX (1/2)

- Inicialización de un mutex

```
int pthread_mutex_init (pthread_mutex_t *mutex,
                        const pthread_mutexattr_t *attr);
```

- Finalización de un mutex

```
int pthread_mutex_destroy (pthread_mutex_t *mutex);
```

- Liberar un mutex

```
int pthread_mutex_unlock (pthread_mutex_t *mutex);
```

  - Si hay threads esperando en la cola, el de mayor prioridad se convierte en el nuevo propietario. En caso contrario, el mutex queda libre
  - Retorna el error *EPERM* si el thread llamante no es el propietario del mutex

Mutexes: Funciones POSIX (2/2)

- Tomar un mutex o suspenderse si no está libre

```
int pthread_mutex_lock (pthread_mutex_t *mutex);
```

  - Retorna el error *EINVAL* si el mutex utiliza el protocolo PTHREAD_PRIO_PROTECT y el thread tiene prioridad mayor que el techo

- Tratar de tomar un mutex sin suspenderse si no está libre

```
int pthread_mutex_trylock (pthread_mutex_t *mutex);
```

  - Retorna el error *EBUSY* si el mutex está tomado

- Tratar de tomar un mutex o suspenderse un tiempo límite

```
int pthread_mutex_timedlock (pthread_mutex_t *mutex,
                             const struct timespec *abstime);
```

  - Retorna el error *ETIMEDOUT* si se alcanza el tiempo límite
  - El tiempo límite está basado en el reloj del sistema o en el *CLOCK_REALTIME*

Uso de mutexes: Ejemplo (1/3)

```
// Dado que se utiliza la librería pthread, debe indicarse en el
// proceso de compilación.
// Ejemplo: gcc main.c -lpthread

#include <pthread.h>
#include <stdio.h>
#include <misc/error_checks.h> // Define la macro no estándar CHK()

struct shared_data {
   pthread_mutex_t mutex;
   int a,b,c,i;
} data;

// Este thread incrementa a, b y c 20 veces
void * incrementer (void *arg)
{
   for (data.i=1; data.i<20; data.i++) {
         CHK( pthread_mutex_lock(&data.mutex) );
         data.a++;
         data.b++;
         data.c++;
         CHK( pthread_mutex_unlock(&data.mutex) );
   }
   return NULL;
}
```

Uso de mutexes: Ejemplo (2/3)

```
// Este thread muestra el valor de a, b y c hasta que i vale 20
void * reporter (void *arg)
{
   do {
         CHK( pthread_mutex_lock(&data.mutex) ); // CHK es una macro no estándar
         printf("a=%d, b=%d, c=%d\n", data.a, data.b, data.c);
         CHK( pthread_mutex_unlock(&data.mutex) );
   } while (data.i<20);
   return NULL;
}


// Programa principal: crea el mutex y los threads
int main()
{
   pthread_mutexattr_t mutexattr;
   pthread_t t1,t2;
   struct sched_param sch_param;
   pthread_attr_t attr;

   data.a = 0; data.b =0; data.c = 0; data.i = 0;
```


Uso de mutexes: Ejemplo (3/3)

```
   // crea el mutex
   CHK( pthread_mutexattr_init (&mutexattr) );
   CHK( pthread_mutexattr_setprotocol (&mutexattr, PTHREAD_PRIO_PROTECT) );
   CHK( pthread_mutexattr_setprioceiling(&mutexattr, sched_get_priority_min(SCHED_RR)) );
   CHK( pthread_mutex_init(&data.mutex, &mutexattr) );

   // Prepara atributos para planificación round-robin
   CHK( pthread_attr_init(&attr) );
   CHK( pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED) );
   CHK( pthread_attr_setschedpolicy(&attr, SCHED_RR) );
   sch_param.sched_priority = sched_get_priority_min(SCHED_RR);
   CHK( pthread_attr_setschedparam(&attr, &sch_param) );

   // crea los threads
   CHK( pthread_create(&t1,&attr,incrementer,NULL) );
   CHK( pthread_create(&t2,&attr,reporter,NULL) );

   // espera a que acaben
   CHK( pthread_join(t1,NULL) );
   CHK( pthread_join(t2,NULL) );
   return 0;
}
```