SISTEMAS CONCURRENTES Y DISTRIBUIDOS

PRACTICA 1

Jose Antonio Padial Molina
josepadial@correo.ugr.es

Contenido

PRODUCTOR-CONSUMIDOR

Como variables globales hemos creado los semáforos, un entero para controlar el buffer, un vector de enteros para usarlo de buffer y un mutex para usarlo de cerrojo.

```
Semaphore puede_leer = 0, puede_escribir = tam_vec;
int primera_libre = 0;
int buffer[tam_vec];
mutex m;
```

El semáforo puede_leer se inicializa a '0' para que no pueda leer sin antes haber producido un valor. En el caso del semáforo de puede_escribir se inicializa al tamaño del vector ya que inicialmente está vacío el buffer y lo puede llenar entero.

La función hebra productora va a producir 40 elementos usando un bucle for desde 0 hasta 40. Donde se una la función producir_dato() para obtener un entero aleatorio. Hacemos por seguridad un sem_wait de puede_escribir y bloqueamos el cerrojo para que no se interrumpa el proceso de producción. Al finalizar el proceso libreamos el cerrojo e indicamos que ya hay un elemento para consumir con sem_signal a puede leer. La función consumir sigue la misma estructura solo que hace los semáforos de forma inversa.

```
void  funcion_hebra_productora(  )
{
   for( unsigned i = 0 ; i < num_items ; i++ )
   {
      int dato = producir_dato() ;
      sem_wait(puede_escribir);
      m.lock();
      buffer[primera_libre] = dato;
      cout << "Se ha producido el valor: " << dato << endl;
      primera_libre++;
      m.unlock();
      sem_signal(puede_leer);
   }
}

//----------------------------------------------------------------------

void funcion_hebra_consumidora(  )
{
   for( unsigned i = 0 ; i < num_items ; i++ )
   {
      int dato ;
      sem_wait(puede_leer);
      m.lock();
      primera_libre--;
      dato = buffer[primera_libre];
      cout << "Se ha extraido el valor: " << dato << endl;
      m.unlock();
      sem_signal(puede_escribir);
      consumir_dato( dato ) ;
    }
}
```

FUMADORES-PLANTILLA

Como variables globales tenemos el número de fumadores y los semáforos que van a controlar el mostrador y los ingredientes disponibles.

```
const int NUM_FUMADORES = 3;

Semaphore mostr_vacio=1, ingr_disp[NUM_FUMADORES]={0,0,0};
```

La función hebra productora contiene un while que no contiene fin el cual va a producir un recurso, en la primera vez va a indicar que ya no está el mostrador vacío con un sem_wait e indicamos que tenemos un igrediente disponible con sem_signal. La función hebra fumador es idéntica a la anterior solo que se hace el sem_wait a los ingredientes disponibles y el sem_siganl al mostrador vacio.

```
void funcion_hebra_estanquero(  )
{
    while(true){
        int recurso_producido;

        recurso_producido = producir();

        sem_wait(mostr_vacio);

        string s1[] = {"El estanquero produce tabaco","El estanquero produce papel","El estanquero produce cerillas"};

        cout << s1[recurso_producido] << endl;
        sem_signal(ingr_disp[recurso_producido]);
    }

}
```

```
void  funcion_hebra_fumador( int num_fumador )
{
   while( true )
   {
        sem_wait(ingr_disp[num_fumador]);
        string s1[] = { "Retirado el ingrediente tabaco ", "Retirado el ingrediente papel ", "Retirado el ingrediente cerillas "} ;
        cout << "El fumador: " << num_fumador << s1[num_fumador] << endl;
        sem_signal(mostr_vacio);
        fumar(num_fumador);
   }
}
```

En el main se crean las hebras de los fumadores con un bucle for y la hebra del estanquero.

```
int main()
{
   thread hebra_estanquero;
   thread fumador[NUM_FUMADORES];

   hebra_estanquero = thread(funcion_hebra_estanquero);
   for(int i=0; i<NUM_FUMADORES; i++){
        fumador[i] = thread(funcion_hebra_fumador,i);
   }

   hebra_estanquero.join();
   for(int i=0; i<NUM_FUMADORES; i++){
        fumador[i].join();
   }
}
```