SISTEMAS CONCURRENTES Y DISTRIBUIDOS

PRACTICA 3

Jose Antonio Padial Molina
josepadial@correo.ugr.es

Contenido

PRODUCTOR- CONSUMIDOR PARA MÚLTIPLES PRODUCTORES Y CONSUMIDORES

En el nuevo programa, en vez de identificar cada proceso con un identificador único 0,1,2 para el productor, bufer y consumidor respectivamente, se identifica al productor con num_prod procesos (4 en este caso), num_cons=5 procesos para consumidores y el buffer con el identificador 4. Como la solución es basada en el uso de etiquetas he creado dos variables constantes enteras para la etiqueda de productor y consumidor.

```
const int
    num_prod=4, num_cons=5,
    num_procesos=num_cons+num_prod+1,
    num_items=100, tam_vector=40,
    item_cada_prod=num_items/num_prod,
    item_cada_cons=num_items/num_cons,
    id_buffer=num_prod;

const int
    etiqueta_productor=0,
    etiqueta_consumidor=1;
```

Como indica el enunciado del problema, la función de los productores y la de los consumidores reciben como parámetro el número de orden del productor o del consumidor, respectivamente (esos números son los números de orden en cada rol, comenzando en 0, no son los identificadores de proceso). Estos números de orden se calculan en el main.

El productor ahora produce num_items/num_prod y el consumidor consume num_items/num_cons. En el caso de la función del productor en el envío de mensaje Ssend se le pasa los siguientes valores:

MPI_Ssend(&valor_prod, 1, MPI_INT, id_buffer,etiqueta_productor, MPI_COMM_WORLD);

```
//Funcion del productor
void funcion_productor(int num_productor){
    for(unsigned i=0; i<item_cada_prod; i++){
        int valor_prod = producir(num_productor);

        cout << "Productor numero: " << num_productor << " va a enviar: " <<
        valor_prod << endl << flush;

        MPI_Ssend(&valor_prod, 1, MPI_INT, id_buffer,etiqueta_productor, MPI_COMM_WORLD);
    }
}
```

Y para los consumidores, en la función consumidor tanto para el envío Ssend para la petición como Recv para recibir el mensaje se le pasan los siguientes valores:

MPI_Ssend(&peticion, 1, MPI_INT, id_buffer, etiqueta_consumidor, MPI_COMM_WORLD);

MPI_Recv(&valor_rec, 1, MPI_INT, id_buffer, etiqueta_productor, MPI_COMM_WORLD, &estado);

```
//Funcion consumidor
void funcion_consumidor(int num_consumidor){
    int peticion, valor_rec;
    MPI_Status estado;

    for(unsigned i=0; i<item_cada_prod; i++){
        //Envia una peticion para consumir un dato
        MPI_Ssend(&peticion, 1, MPI_INT, id_buffer, etiqueta_consumidor, MPI_COMM_WORLD);

        //Recive un dato para consumir
        MPI_Recv(&valor_rec, 1, MPI_INT, id_buffer, etiqueta_productor, MPI_COMM_WORLD, &estado);
        cout << "\t\tEl consumidor numero: " << num_consumidor << " ha recivido el valor: " <<
        valor_rec << endl << flush;

        consumir(valor_rec, num_consumidor);
    }
}
```

En la función buffer, se calcula en vez de la id_emisor_aceptable, como es una solución basada en etiquetas, la etiqueta aceptable según si solo se puede producir o consumir:

```
//Funcion para el buffer
void funcion_buffer(){
    int buffer[tam_vector], valor,
        primera_libre = 0,
        primera_ocupada = 0,
        num_celda_ocupadas = 0,
        etiqueta_aceptable;
    MPI_Status estado;

    for(unsigned i=0; i<num_items*2; i++){
        if(num_celda_ocupadas == 0)
            etiqueta_aceptable = etiqueta_productor;
        else if(num_celda_ocupadas == tam_vector)
            etiqueta_aceptable = etiqueta_consumidor;
        else
            etiqueta_aceptable = MPI_ANY_TAG;

        MPI_Recv(&valor, 1, MPI_INT, MPI_ANY_SOURCE, etiqueta_aceptable, MPI_COMM_WORLD, &estado);

        switch(estado.MPI_TAG){
            case etiqueta_productor:
                buffer[primera_libre] = valor;
                primera_libre = (primera_libre + 1) % tam_vector;
                num_celda_ocupadas++;
                cout << "\tBuffer ha recibido: " << valor << endl;
                break;

            case etiqueta_consumidor:
            valor = buffer[primera_ocupada];
            primera_ocupada = (primera_ocupada + 1) % tam_vector;
            num_celda_ocupadas--;
            cout << "\tBuffer va a enviar: " << valor << endl;
            MPI_Ssend( &valor, 1, MPI_INT, estado.MPI_SOURCE, etiqueta_productor, MPI_COMM_WORLD );
            break;
        }
    }
}
```

FILÓSOFOS CON INTERBLOQUEO

Archivo filoso-plantilla.cpp copiado en filósofos-interb.cpp y completado según las directrices del guion de la práctica. Una vez hecho esto, al ejecutarlo compruebo que se da interbloqueo cuando un filósofo solicita un tenedor y este ya lo tiene otro filosofo.

```
void funcion_filosofos(int id){
    int id_tenedor_izq = (id+1) % numero_procesos,
        id_tenedor_dcha = (id+numero_procesos-1) % numero_procesos,
        peticion;

    while(true){
        cout << "Filosofo numero: " << id << " solicita tenedor izquierdo: " <<
            id_tenedor_izq << endl;
        MPI_Ssend(&peticion, 1, MPI_INT, id_tenedor_izq, 0, MPI_COMM_WORLD);

        cout << "Filosofo numero: " << id << " solicita tenedor derecho: " <<
            id_tenedor_dcha << endl;
        MPI_Ssend(&peticion, 1, MPI_INT, id_tenedor_dcha, 0, MPI_COMM_WORLD);

        cout << "Filosofo numero: " << id << " comienza a comer" << endl;
        sleep_for( milliseconds( aleatorio<10,100>() ) );

        cout << "Filosofo numero: " << id << " suelta tenedor izquierdo: " <<
            id_tenedor_izq << endl;
        MPI_Ssend(&peticion, 1, MPI_INT, id_tenedor_izq, 0, MPI_COMM_WORLD);

        cout << "Filosofo numero: " << id << " suelta tenedor derecho: " <<
            id_tenedor_dcha << endl;
        MPI_Ssend(&peticion, 1, MPI_INT, id_tenedor_dcha, 0, MPI_COMM_WORLD);

        cout << "Filosofo numero: " << id << " comienza a pensar" << endl;
        sleep_for( milliseconds( aleatorio<10,100>() ) );
    }
}
```

FILÓSOFOS CON SOLUCIÓN INTERBLOQUEO

La solución ha sido que uno de los filósofos debe comenzar a coger por la derecha, en vez de por la izquierda.

```
void funcion_filosofos(int id){
    int id_tenedor_izq = (id+1) % numero_procesos,
        id_tenedor_dcha = (id+numero_procesos-1) % numero_procesos,
        peticion;

    while(true){
        if(id == 0){
            cout << "Filosofo numero: " << id << " solicita tenedor derecho: " <<
                id_tenedor_dcha << endl;
            MPI_Ssend(&peticion, 1, MPI_INT, id_tenedor_dcha, 0, MPI_COMM_WORLD);

            cout << "Filosofo numero: " << id << " solicita tenedor izquierdo: " <<
                id_tenedor_izq << endl;
            MPI_Ssend(&peticion, 1, MPI_INT, id_tenedor_izq, 0, MPI_COMM_WORLD);
        }
        else{
            cout << "Filosofo numero: " << id << " solicita tenedor izquierdo: " <<
                id_tenedor_izq << endl;
            MPI_Ssend(&peticion, 1, MPI_INT, id_tenedor_izq, 0, MPI_COMM_WORLD);

            cout << "Filosofo numero: " << id << " solicita tenedor derecho: " <<
                id_tenedor_dcha << endl;
            MPI_Ssend(&peticion, 1, MPI_INT, id_tenedor_dcha, 0, MPI_COMM_WORLD);
        }

        cout << "Filosofo numero: " << id << " comienza a comer" << endl;
        sleep_for( milliseconds( aleatorio<10,100>() ) );

        cout << "Filosofo numero: " << id << " suelta tenedor izquierdo: " <<
            id_tenedor_izq << endl;
        MPI_Ssend(&peticion, 1, MPI_INT, id_tenedor_izq, 0, MPI_COMM_WORLD);

        cout << "Filosofo numero: " << id << " suelta tenedor derecho: " <<
            id_tenedor_dcha << endl;
        MPI_Ssend(&peticion, 1, MPI_INT, id_tenedor_dcha, 0, MPI_COMM_WORLD);

        cout << "Filosofo numero: " << id << " comienza a pensar" << endl;
        sleep_for( milliseconds( aleatorio<10,100>() ) );
    }
}
```

FILÓSOFOS CON CAMARERO

La solución para el problema de interbloqueo ha sido introducir dos nuevos pasos a los filósofos, que es sentarse a la mesa y levantarse. El nuevo proceso camarero controlará los filósofos que se sientan a la mesa con un contador (filósofos_sentados) que se incrementara cuando haya menos de 5 filósofos sentadas y decrementará cuando se levante un filósofo.

Para saber si se puede que levantar o sentar un filósofo he creado dos nuevas etiquetas (etiq_sentarse = 0 y etiq_levantarse = 1) que utilizo para pedir permiso con el envío síncrono:

MPI_Ssend(&peticion, 1, MPI_INT, id_camarero, etiq_sentarse, MPI_COMM_WORLD);

MPI_Ssend(&peticion, 1, MPI_INT, id_camarero, etiq_levantarse, MPI_COMM_WORLD);

El camarero siempre aceptará las peticiones de un filósofo para levantarse. Otra modificación ha sido el número de procesos, he creado dos variables, una para los procesos efectivos y otra para los procesos esperados (11). Tambien he añadido el identificador del proceso camarero (id_camarero=num_procesos_efectivos).

```
void funcion_camarero(){
    int num_filosofos_sentados=0,
        peticion, id_filosofo,
        etiq_aceptable;

    MPI_Status estado;

    while(true){
        if(num_filosofos_sentados < num_filosofos - 1)
            etiq_aceptable = MPI_ANY_TAG;
        else
            etiq_aceptable = etiq_levantarse;

        MPI_Recv( &peticion, 1, MPI_INT, MPI_ANY_SOURCE, etiq_aceptable, MPI_COMM_WORLD, &estado );
        id_filosofo = estado.MPI_SOURCE;

        switch(estado.MPI_TAG){
            case etiq_levantarse:
                cout << "\tFilósofo " << id_filosofo << " se levanta de la mesa" << endl;
                num_filosofos_sentados--;
                break;
            case etiq_sentarse:
                cout << "\tFilósofo " << id_filosofo << " se sienta a la mesa" << endl;
                num_filosofos_sentados++;
                break;
        }

        cout << "\t --- Actualmente hay " << num_filosofos_sentados << " filósofos sentados --- " << endl;
    }
}
```