Ángel Barrilao Bensrhir – Grupo 2º A(1)

Explicación Filósofos

Aspectos mas desatacados

Las partes mas significativas para la solución de este problema, es realmente tener bien claro como debe de funcionar el programa con la secuencia de sucesos que deben de ir aconteciendo:

1º El filosofo envía petición a tenedor
2º Tenedor recibe petición

3º Filosofo envía petición para tomar otro tenedor
4º Tenedor recibe otra petición

5º Come …...

6º Filosofo termina de usar los tenedores y envía una señal a cada uno de ellos.

Aparte de lo arriba mencionado , hay que tener cuidado para que no se produzca interbloqueo. En este caso se produciría interbloqueo si cada filosofo tomase un tenedor a la vez , por lo que se quedarían esperando a que alguno de ellos cediese algún tenedor.

Para evitar este fallo tenemos que cambiar el orden en que toma el tenedor un proceso de ellos. Se ha elegido el proceso(filosofo) cero . Lo que hace es que tomara el tenedor contrario con respecto a sus compañeros filósofos.

Según especifica el problema los filósofos toman el tenedor primero de su izquierda , exceptuamos al filosofo cero que le forzamos a que tome primero el de la derecha.

```
void Filosofo( int id, int nprocesos )
{
   int izq = (id+1) % nprocesos;
   int der = ((id+nprocesos)-1) % nprocesos;


   while(1)
   {

    //El primer filosofo tiene que coger los tenedores al reves para que no se produzca interbloqueo
    if(id == 0)
    {
        //Solicita tenedor derecho
        cout << GREEN << "Filosofo "<<id<< " coge tenedor der ..." << der << BLACK << endl << flush;
        MPI_Ssend(NULL, 0, MPI_INT, der, coger, MPI_COMM_WORLD);

        //solicita tenedor izquierdo
        cout<< RED <<"Filosofo "<<id<< " solicita tenedor izq ..." << izq << BLACK << endl << flush;
        MPI_Ssend(NULL, 0, MPI_INT, izq, coger, MPI_COMM_WORLD);
    }else//resto de filosofos
    {
        //solicita tenedor izquierdo
        cout << RED <<"Filosofo "<<id<< " solicita tenedor izq ..." << izq << BLACK << endl << flush;
        MPI_Ssend(NULL, 0, MPI_INT, izq, coger, MPI_COMM_WORLD);

        //Solicita tenedor derecho
        cout<< GREEN << "Filosofo "<<id<< " coge tenedor der ..." << der << BLACK << endl << flush;
        MPI_Ssend(NULL, 0, MPI_INT, der, coger, MPI_COMM_WORLD);
    }
```

Nótese que el filosofo solo hace llamadas (para ver el código completo revisar el archivo adj.) , no tiene que esperar la respuesta de ningún otro proceso.

En donde si se reciben llamadas es en el proceso Tenedor , en este proceso se esperan dos llamadas, una para tomar el tenedor y otra para soltarlo.

Otro punto importante a tener en cuenta , es definir los tags apropiados , para poder saber que acción nos corresponde a la hora de enviar y recibir un mensaje.

```
//Atributos de control
#define soltar  0
#define coger   1
```

Otro cosa a tener en cuenta es en la llamada . Hay que indicarle bien que tenedor quiere llamar , según la posición del filosofo (o lo que es lo mismo su ID) , sabemos que los tenedores que tiene que coger en todo momento son los que tiene contiguos a su misma ID.

Aquí podemos ver una linea de la función Filosofo que realiza una llamada.

```
MPI_Ssend(NULL, 0, MPI_INT, der, coger, MPI_COMM_WORLD);
```

Analicemos:

NULL : Como no vamos a enviar ningún tipo de información , lo dejamos a null. No hace falta pasarle ninguna variable.

0 : Numero de elementos a enviar; Como no enviamos ninguno pues lo ponemos a cero

MPI_INT : tipo de dato va a ser entero

der: Significa ‘derecho’ y es un valor entero, este valor corresponde al ID de un proceso Tenedor. Ponemos este campo para llamar al proceso Tenedor correspondiente.

coger: Es el tag que hemos definido para saber como tratar y ubicar esta llamada.

MPI_COMM_WORLD : Comunicador

Captura de ejecución del programa: