SCD2020EXAMENP3.pdf

danielsp10

Sistemas Concurrentes y Distribuidos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

SISTEMAS CONCURRENTES Y DISTRIBUIDOS

EXAMEN PRÁCTICAS 2020/21 - P3 | [GENÉRICO]

Autor: DanielsP

En el siguiente documento se encuentran los enunciados del examen asociado a PRÁCTICA 3 realizado en el Doble Grado de Ingeniería Informática y Matemáticas (DGIIM) y Doble Grado de Ingeniería Informática y Administración y Dirección de Empresas (DGIIADE). Las soluciones se plantearán en otro documento diferente.

Nota: Este examen no contiene todos los modelos de examen que se hicieron en su momento. Sólo se encuentran aquellos que de los que se tiene constancia. En cualquier caso, lo que se pedía hacer en cualquier modelo era exactamente lo mismo, cambiando las hebras/funciones sobre la que se implementan las nuevas funcionalidades.

EJERCICIOS SOBRE SISTEMA DE PASO DE MENSAJES (MPI)

1. **Sobre el problema de los FILÓSOFOS con camarero: Crea un nuevo archivo llamado `p3e2_mpi.cpp`. Crea un nuevo proceso que ejecuta un bucle infinito. En cada iteración recibe un mensaje de uno cualquiera de los filósofos o del camarero.**
    - **Los filósofos (cada vez que terminan de pensar) y el camarero (cada vez que ha aceptado una solicitud de levantarse), envían un mensaje (asíncrono seguro) al nuevo proceso.**
    - **Cada vez que el nuevo proceso recibe un mensaje, imprime un texto en pantalla, ese texto incluye la cuenta de mensajes recibidos de los filósofos y la cuenta de mensajes recibidos del camarero (contados todos desde el inicio).**
2. **Sobre el problema de los FILÓSOFOS con camarero: Crea un nuevo archivo llamado `p3e3_mpi.cpp` con una copia de `p3e2_mpi.cpp`. Extiende o modifica el programa para cumplir estos requerimientos adicionales a los del problema anterior:**
    - **El nuevo proceso, cuando va a recibir un mensaje, si ha recibido hasta ese momento, en total, menos mensajes de filósofos que del camarero, solo acepta un mensaje de un filósofo. Si ha recibido menos mensajes del camarero que de filósofos, solo acepta un mensaje del camarero. Si ha recibido el mismo número de mensajes de cada tipo de proceso, acepta un mensaje de cualquiera proceso, ya sea filósofo o el camarero.**
    - **El nuevo proceso, al inicio de su bucle, queda esperando, bloqueado, hasta que haya al menos un mensaje pendiente de recibir de uno de los posibles emisores. En ese momento se desbloquea e imprime un texto con el número de proceso emisor del mensaje (el texto será como este: Se va a recibir un mensaje del proceso número N). Después, el nuevo proceso recibe ese mensaje, igual que en el ejercicio anterior.**
3. **Sobre el problema de los FILÓSOFOS con camarero: Crea un nuevo archivo llamado `p3e2_mpi.cpp` y realiza las siguientes modificaciones:**
    - **Los fiósofos antes de sentarse a la mesa deberán darle una pequeña propina al camarero por los servicios ofrecidos. Esta propina se contará por unidades, pudiendo variar una propina de un filósofo entre 1 y 5 unidades (1 unidad = 1 INT).**
    - **Además, cuando el camarero reciba 10 unidades de propina deberá sacar un mensaje por pantalla indicándolo, antes de reiniciar la cuenta.**

A continuación se proporciona una plantilla de código para la realización del examen:

```
#include <mpi.h>
#include <thread> // this_thread::sleep_for
#include <random> // dispositivos, generadores y distribuciones aleatorias
#include <chrono> // duraciones (duration), unidades de tiempo
#include <iostream>

using namespace std;
using namespace std::this_thread;
using namespace std::chrono;

//========================================================================
// VARIABLES CONSTANTES
//========================================================================

const int num_filosofos = 5,
          num_procesos  = 2 * num_filosofos +1;

const int id_camarero = num_procesos -1;

const int etiq_reservarTenedor = 1,
          etiq_liberarTenedor = 2,
          etiq_sentarseMesa = 3,
          etiq_levantarseMesa = 4;

const int sentadosPermitidos = 3;

//========================================================================
// FUNCIONES COMUNES
//========================================================================


/**
 * @brief Genera un entero aleatorio uniformemente distribuido entre dos
 * valores enteros, ambos incluidos
 * @param<T,U> : Números enteros min y max, respectivamente
 * @return un numero aleatorio generado entre min,max
 */
template< int min, int max > int aleatorio(){
    static default_random_engine generador((random_device())());
    static uniform_int_distribution<int> distribucion_uniforme(min, max);
    return distribucion_uniforme(generador);
}

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

//========================================================================
// FUNCIONES DE PROCESOS
//========================================================================

/**
 * @brief Funcion que simula un filsofo
 * @param id : id de proceso
 */
void funcion_filosofos(int id){
```

16

LA ÚLTIMA

AITANA OCAÑA · MIGUEL BERNARDEAU

· UNA HISTORIA SOBRE EMPEZAR ·

Temporada completa
Ya disponible solo en

Disney+

```
    int id_ten_izq = (id+1)              % (num_procesos-1), //id. tenedor izq.
        id_ten_der = (id+num_procesos-2) % (num_procesos-1); //id. tenedor der.

    //ITERACIONES INFINITAS
    while(true){
        //LOS FILOSOFOS SOLICITAN SENTARSE EN LA MESA
        cout << ">>> Filósofo " <<id << " solicita al camarero sentarse en la
mesa." << endl;
        MPI_Ssend(&id, 1, MPI_INT, id_camarero, etiq_sentarseMesa,
MPI_COMM_WORLD);

        //LOS FILOSOFOS SOLICITAN TENEDORES
        cout << ">>> Filósofo " <<id << " solicita ten. izq: " << id_ten_izq <<
endl;
        // ... solicitar tenedor izquierdo
        MPI_Ssend(&id, 1, MPI_INT, id_ten_izq, etiq_reservarTenedor,
MPI_COMM_WORLD);

        cout << ">>> Filósofo " <<id <<" solicita ten. der: " << id_ten_der <<
endl;
        // ... solicitar tenedor derecho
        MPI_Ssend(&id, 1, MPI_INT, id_ten_der, etiq_reservarTenedor,
MPI_COMM_WORLD);

        //LOS FILOSOFOS EMPIEZAN A COMER
        cout << "<---> Filósofo " << id << " comienza a comer" << endl;
        sleep_for(milliseconds(aleatorio<10,100>()));

        //LIBERAR TENEDORES
        cout << "<<< Filósofo " << id << " suelta ten. izq: " << id_ten_izq
<<endl;
        // ... soltar el tenedor izquierdo
        MPI_Ssend(&id, 1, MPI_INT, id_ten_izq, etiq_liberarTenedor,
MPI_COMM_WORLD);

        cout << "<<< Filósofo " <<id <<" suelta ten. der: " << id_ten_der <<endl;
        // ... soltar el tenedor derecho
        MPI_Ssend(&id, 1, MPI_INT, id_ten_der, etiq_liberarTenedor,
MPI_COMM_WORLD);

        //LOS FILOSOFOS SE LEVANTAN DE LA MESA
        cout << ">>> Filósofo " <<id << " avisa al camarero que se levanta de la
mesa." << endl;
        MPI_Ssend(&id, 1, MPI_INT, id_camarero, etiq_levantarseMesa,
MPI_COMM_WORLD);

        //LOS FILOSOFOS PIENSAN
        cout << "<---> Filosofo " << id << " comienza a pensar " << endl;
        sleep_for(milliseconds(aleatorio<10,100>()));
    }
}

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

/**
 * @brief Funcion que simula un tenedor
 * @param id : id de proceso
 */
```

```
void funcion_tenedores(int id){
    int valor, id_filosofo;  // valor recibido, identificador del filósofo
    MPI_Status estado;       // metadatos de las dos recepciones

    while (true){
        // ..... recibir petición de cualquier filósofo (completar)
        // ..... guardar en 'id_filosofo' el id. del emisor (completar)
        MPI_Recv(&valor, 1, MPI_INT, MPI_ANY_SOURCE, etiq_reservarTenedor,
MPI_COMM_WORLD, &estado);
        id_filosofo = estado.MPI_SOURCE;

        cout << "Ten. " <<id <<" ha sido cogido por filo. " <<id_filosofo <<endl;

        // ..... recibir liberación de filósofo 'id_filosofo' (completar)
        MPI_Recv(&valor, 1, MPI_INT, id_filosofo, etiq_liberarTenedor,
MPI_COMM_WORLD, &estado);

        cout << "Ten. "<< id << " ha sido liberado por filo. " <<id_filosofo
<<endl;
    }
}

/**
 * @brief
 */
void funcion_camarero(){
    int valor, etiq_aceptable,
        sentados = 0;

    MPI_Status estado;

    while(true){
        //PASO 1: ATENDER PETICIONES DE SENTARSE A LA MESA

        //COMPROBAMOS SI SE PUEDEN SENTAR A LA MESA
        if(sentados == 0)
            etiq_aceptable = etiq_sentarseMesa;
        else if (sentados == sentadosPermitidos)
            etiq_aceptable = etiq_levantarseMesa;
        else
            etiq_aceptable = MPI_ANY_TAG;

        //RECIBIMOS EL MENSAJE

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

        //MOSTRAMOS POR PANTALLA LA PETICION INDICADA
        if(estado.MPI_TAG == etiq_sentarseMesa){
            sentados++;
            cout << "------> Camarero acepta la peticion de sentarse a filosofo:
" << estado.MPI_SOURCE << endl;
        }
        if(estado.MPI_TAG == etiq_levantarseMesa){
            sentados--;
            cout << "<------ Camarero acepta la peticion de levantarse a
filosofo: " << estado.MPI_SOURCE << endl;
        }
    }
```

```
}

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

//========================================================================
// FUNCION PRINCIPAL
//========================================================================

/**
 * @brief Funcion principal
 * @param argc : Numero de argumentos
 * @param argv : Total de argumentos
 * @return 0 [EX COMPLETADA]
 */
int main(int argc, char** argv){
    //ID PROPIO : IDENTIFICADOR DE PROCESO
    //NUM_PROCESOS_ACTUAL : TOTAL DE PROCESOS EJECUTADOS
    int id_propio, num_procesos_actual;

    //PASO 1: INICIALIZAR MPI, LEER ID_PROPIO Y NUM_PROCESOS_ACTUAL

    //MPI INIT --> INICIO DEL PROCEDMIENTO MPI
    //-> int*         : argc = NUMERO DE ARGUMENTOS
    //-> char***      : argv = TOTAL DE ARGUMENTOS
    MPI_Init(&argc, &argv);

    //MPI_COMM_RANK --> OBTIENE LOS VALORES DE CADA PROCESO
    //-> MPI_Comm  : MPI_COMM_WORLD = COMUNICADOR MPI
    //-> int*      : id_propio      = ALMACENA EL IDENTIFICADOR DEL PROCESO
[AUTO-INVOKE]
    MPI_Comm_rank(MPI_COMM_WORLD, &id_propio);

    //MPI_COMM_SIZE --> OBTIENE EL TOTAL DE PROCESOS
    //-> MPI_Comm  : MPI_COMM_WORLD      = COMUNICADOR MPI
    //-> int*      : num_procesos_actual = ALMACENA EL TOTAL DE PROCESOS ACTUALES
    MPI_Comm_size(MPI_COMM_WORLD, &num_procesos_actual);


    if(num_procesos == num_procesos_actual){
        // ejecutar la función correspondiente a 'id_propio'
        if (id_propio == (id_camarero))
            funcion_camarero();
        else if (id_propio % 2 == 0)          // si es par
            funcion_filosofos(id_propio); //   es un filósofo
        else                                   // si es impar
            funcion_tenedores(id_propio); //   es un tenedor
    }
    else{
        // solo el primero escribe error, indep. del rol
        if(id_propio == 0){
            cout << "el número de procesos esperados es:    " << num_procesos <<
endl
            << "el número de procesos en ejecución es: " << num_procesos_actual
<< endl
            << "(programa abortado)" << endl ;
        }
    }
```

```
    MPI_Finalize();

    return 0;
}

// ----------------------------------------------------------------------
```