Práctica de Arquitectura de Ordenadores.
Balanceo de carga en un multiprocesador bajo Solaris.
UAX. 4º Curso
Arquitectura de Ordenadores

Izaskun Puerta Castaño NP:20.137
Fernando Muñoz Rosales NP:26.843

Práctica 2: Balanceo de carga en un multiprocesador bajo Solaris

Esta práctica tiene como objetivo la utilización de los servicios existentes en Solaris para poder asignar tareas a procesadores en un multiprocesador.

Se trata de los siguientes servicios (para mayor información consultar páginas del manual):

- **`processor_bind()`**: permite asignar procesos de peso ligero (LWP's) a un procesador de forma explícita.
- **`pset_bind()`**: permite asignar procesos de peso ligero (LWP's) a conjuntos de procesadores de forma explícita. Para ello hay que crear previamente un conjunto de procesadores con los servicios `pset_create()`, para crear un conjunto de procesadores vacío y `pset_assign()` para incluir un procesador concreto en un conjunto de procesadores previamente creado.

La utilización de estos servicios ofrece un control total al programador a la hora de poder gestionar los recursos dentro de un multiprocesador (e.g. para distribuir recursos de cómputo a una o varias tareas en la aplicación mediante su asignación a uno o varios procesadores).

Durante la práctica se utilizará un servidor Ultra Enterprise 250 con 2 procesadores. Para obtener los identificadores de procesador puede utilizarse el comando `psrinfo –v`.

La práctica 2 se basará en el código de test realizado en la práctica 1. Se trata de modificar el código de test de la práctica 1, para que la mitad de los *threads* vayan a un procesador y la otra mitad a otro y comprobar que efectivamente se reduce el tiempo de cómputo al utilizarse ambos procesadores. Para realizar esta asignación es preciso comprobar que los *threads* utilizados de la práctica 1 sean LWP's (y no ULT's), si no es así, modificar el código para ello.

En cada iteración se medirá el tiempo que tarda en realizarse la misma y se imprimirá dicho tiempo en pantalla. Para medida de tiempos utilizar el servicio `gethrtime()`.

Durante la práctica se realizarán dos versiones, una con `processor_bind()` y otra con `pset_bind()`, en este último caso se crearán dos conjuntos de procesadores que incluirán cada uno de ellos un único procesador.

Para comprobar cómo se balancea la carga pueden realizarse diversos ensayos:

- Ejecutar la aplicación en el multiprocesador sin asignación explícita de carga, es decir, dejando que Solaris balancee la carga de forma transparente al programador.
- Ejecutar la aplicación utilizando los servicios mencionados anteriormente, y analizar las diferencias.
- Balancear la carga con otra proporción distinta al 50% entre un procesador y otro (e.g. 80% de las tareas a un procesador y 20% al restante). Analizar los resultados.

Izaskun Puerta Castaño NP:20.137
Fernando Muñoz Rosales NP:26.843

BALANCEO DE CARGA EN MULTIPROCESADOR DE SOLARIS

OBJETIVO

Basándonos en la barrera de sincronización creada en la practica 0, vamos a realizar una serie de cálculos para el control de balanceo de carga de threads en un sistema multiprocesador en Solaris.

A continuación veremos una breve explicación de las librerías utilizadas para realizar el balanceo, después hablaremos los tipos de threads utilizados en las practicas y de los objetivos ha realizar.

LIBRERIA

La librería utilizada es la creada en la practica 0, que es un barrera de sincronización controlada mediante un semáforo y una variable de control. Nos basamos en una estructura del tipo que se encuentra en el archivo barrier.h:

```
typedef struct {
        int contadorTh;
        int threadsIniciales;
        int tipo;
        mutex_t semaforo;
        cond_t condicion;
}barrier_t;
```

En la que como se puede apreciar, tenemos un contador de Threads, otro contador auxiliar que nos sirve para no perder el valor de los Threads iniciales, en el caso de que llamemos en varias ocasiones a la barrera de sincronización.

También tenemos un variable de tipo, que utilizamos para asignar el tipo de Threads que estamos utilizando (Más a delante hablaremos del tipo de Threads que se pueden utilizar).

La variable *mutex_t semáforo*; se utilizará en la función *mutex_lock*, como variable de control para el semáforo.

La variable *cond_t condición*; se utilizará en la función *cond_wait*, función utilizada para dormir los Threads.

Izaskun Puerta Castaño NP:20.137
Fernando Muñoz Rosales NP:26.843

En la implementación de esta librería se han creado los métodos *barrier_wait*, en el que se duermen los threads según las variables de control, el método *barrier_destroy* en el que se destruye la barrera de sincronización y el método *barrier_init*, en el que se inicializan los parámetros de la barrera.

Como prueba de esta barrera se creo un fichero iteraciones.c en el que se creaban los threads y llamaban a la barrera de sincronización en un sistema monoprocesador bajo Solaris. El código que ejecutan los threads es una función *comienzo*, en la que se suman los valores de una matriz en una única variable. La matriz es rellenada por una función *random*.

Izaskun Puerta Castaño NP:20.137
Fernando Muñoz Rosales NP:26.843

TIPOS DE THREADS

Los tipos de threads que se pueden crear son dos :

ULT (User Level Thread):

Este tipo de hilos son los llamados procesos de usuario, por que no pueden ser asignados a diferentes tipos de procesadores, ya que al ser un proceso de usuario el sistema operativo trabaja con el como si de un LWP se tratara. En este caso, para nuestro estudio de balanceo no nos son de gran utilidad.

LWP (Light Weight Process):

Este tipo de hilo son los llamados procesos ligeros, pueden ser asignados a un procesador en concreto, de forma que trabajan en paralelo. Este tipo de hilo nos van a ser de gran ayuda en nuestro estudio de balanceo de carga entre dos procesadores.

En la creación de los Threads para indicar que tipo de hilo que estamos creando se indica en el flag correspondiente. Si queremos crear el Thread del tipo ULT, lo indicamos poniendo el flag a Null, en el caso de que queramos crear el Thread del tipo LWP, el flag se pone THR_BOUND.

Para realizar la función de asignación a un procesador especifico utilizamos la función **processor_bind()**, para asignar a procesadores específicos, para ello en el equipo Perseo, tuvimos que preguntar el identificador de cada uno de los procesadores con el comando "psrinfo", lo cual nos indicó que los procesadores eran respectivamente 0 y 2.

Izaskun Puerta Castaño NP:20.137
Fernando Muñoz Rosales NP:26.843

MODIFICACIONES

Para la practica actual de balanceo hemos tenido que realizar una modificación sobre la practica anterior que probaba la barrera, se ha tenido que modificar la inicialización de la matriz, ya que era *random* y en nuestro caso para el estudio del balanceo necesitas que los valores de la matriz sean fijos, para poder observar la mejora realizada en el balanceo a la hora de la toma de tiempos.

Código anterior que ha sido modificado;

```
for(i=0;i<NumThreads;i++){
        for(j=0;j<NumIterac;j++){
                DatosComputo[i][j].N=rand()%100;
                DatosComputo[i][j].N=rand()%100;
        }
}
```

Ahora la matriz es rellenada de la siguiente forma;

```
DatosComputo[i][j].N=i*50;
DatosComputo[i][j].M=j*50;
```

EJECUCION

En cuanto a la compilación la hemos realizado mediante un fichero *makefile*, en el que cuando hacemos un *make* nos recompila el código de cada uno de los ficheros que hayan sido modificados desde la ultima compilación.

En nuestro caso como tenemos dividimos los ficheros que nos realizan los diferentes balanceos tenemos que hacer un *makefile* para cada fichero. Un ejemplo de fichero *makefile* seria el siguiente:

```
Balanceando25050: Balanceando25050.o barrier.o
        gcc -o Balanceando25050 Balanceando25050.o barrier.o -lthread
Balanceando25050.o: Balanceando25050.c barrier.h
        gcc -c Balanceando25050.c
barrier.o:barrier.c barrier.h
        gcc -c barrier.c
clean:
        rm *.o pruebabarrera
```


Izaskun Puerta Castaño NP:20.137
Fernando Muñoz Rosales NP:26.843

BALANCEO DE CARGA EN MULTIPROCESADOR DE SOLARIS

La maquina Perseo, donde hemos realizado las pruebas tiene dos procesadores de 50 MHz, y en el momento de realizar las pruebas solo la estábamos usando nosotros, ya que si no fuera así estas carecerían de valor, ya que si alguien esta usando los procesadores al tiempo que nosotros si su carga de proceso fuera de un 50-50 solo nos haría retrasar los tiempos de ejecución de nuestro código, pero en cambio si su uso de los procesadores no fuera balanceado perfecto su tiempo de utilización de un procesador respecto al otro quitarían validez a nuestras pruebas.

Hemos realizado varios tipos de balanceo uno al 50 por ciento a un procesador y el otro 50 al segundo procesador, el otro tipo de balanceo es a 80 por ciento a un procesador y el 20 por ciento restante a segundo procesador.

Se han obtenido los siguientes tiempos para los hilos del tipo LWP:

50%-50%	80%-20%
825	943
883	970
822	963
835	946
845	998
842	934

Para realizar el balanceo de carga de los hilos entre los procesadores del 50% a un procesador y el otro 50% restante al segundo procesador, hemos asignado los hilos pares al procesador 0 y los hilos impares al procesador 2.

```
if(NumeroT%2= =0){
        error=processor_bind(P_LWPID,P_MYID,0,NULL);
        printf("\nThread %d asignado a procesador 0",NumeroT);

}else{
        error=processor_bind(P_LWPID,P_MYID,2,NULL);
        printf("\nThread %d asignado a procesador 2",NumeroT);

}
```


Izaskun Puerta Castaño NP:20.137
Fernando Muñoz Rosales NP:26.843

En el caso del balanceo de 80% a un procesador y el 20% a otro procesador el código se ha modificado de la siguiente forma;

```
if(NumeroT<(NumThreads*0.8)){
        error=processor_bind(P_LWPID,P_MYID,0,NULL);
        printf("\nThread %d asignado a procesador 0",NumeroT);

}else{
        error=processor_bind(P_LWPID,P_MYID,2,NULL);
        printf("\nThread %d asignado a procesador 2",NumeroT);

}
```

También realizamos una comprobación de los tiempos de ejecución de los hilos, especificando nosotros el tipo de threads ha crear, con las mismas condiciones en cuanto a la matriz y dejando que el sistema operativo los balanceara;

ULT	LWP
1470	**1491**
1469	**1501**
1482	**1492**
1469	**1495**
1468	**1501**

Izaskun Puerta Castaño NP:20.137
Fernando Muñoz Rosales NP:26.843

CONCLUSIONES

Se puede observar que el tiempo de ejecución de los threads de tipo ULT y los de tipo LWP sin realizar ningún tipo de balanceo, son prácticamente iguales. Esto es debido a que no se realiza ningún tipo de mejora, ambos tipos de hilos se ejecutan en un mismo procesador.

También podemos observar que asignando un balanceo de 50-50 es mejor que el balanceo de 80-20, por que de esta forma no se sobrecarga ningún procesador y el tiempo de finalización será prácticamente parecido, sin embargo si los asignamos con un balanceo de 80- 20 el procesador que tiene asignado el 80% de la carga de los hilos tardará más que el otro procesador.

Izaskun Puerta Castaño NP:20.137
Fernando Muñoz Rosales NP:26.843

CODIGO

BARRIER.H

```
typedef struct {
        int contadorTh;
        int threadsIniciales;
        int tipo;
        mutex_t semaforo;
        cond_t condicion;

}barrier_t;
int barrier_init(barrier_t *barrier, int count, int type, void *arg);
int barrier_wait(barrier_t *barrier);
int barrier_destroy(barrier_t *barrier);
```

BARRIER.C

```
#include <thread.h>
#include <synch.h>
#include "barrier.h"

int barrier_init(barrier_t *barrier, int count, int type, void *arg){
        barrier->contadorTh=count;
        barrier->threadsIniciales=count;
        barrier->tipo=type;
        mutex_init(&barrier->semaforo,type,NULL);
        cond_init(&barrier->condicion,type,NULL);


return 0;
}
int barrier_wait(barrier_t *barrier){
        mutex_lock(&barrier->semaforo);
        barrier->contadorTh--;
        if(barrier->contadorTh==0){
                cond_broadcast(&barrier->condicion);
                printf("\nEl thread %d despierta al resto el que mas tardo",thr_self()-2);
                barrier->contadorTh=barrier->threadsIniciales;
        }else{

                printf("\nSe duerme este %d",thr_self()-2);
                cond_wait(&barrier->condicion,&barrier->semaforo);
        }
        mutex_unlock(&barrier->semaforo);
        return 0;
}

int barrier_destroy(barrier_t *barrier){

        mutex_destroy(&barrier->semaforo);
        cond_destroy(&barrier->condicion);
        return 0;
```


Izaskun Puerta Castaño NP:20.137
Fernando Muñoz Rosales NP:26.843

BALANCEANDO 50 50.C

```
#define _REENTRANT
#include <stdlib.h>
#include <thread.h>
#include <synch.h>
#include <sys/types.h>
#include <sys/processor.h>
#include <sys/procset.h>
#include "errno.h"
#include "barrier.h"
#define NumThreads 10
#define NumIterac 10

int idthreads[NumThreads];
void *comienzo(void *args);
barrier_t mibarrera;
mutex_t variable_exclusion;
int i;

struct {
        int N;
        int M;
}DatosComputo[NumThreads][NumIterac];
float res[NumIterac];
hrtime_t start, end;
int tiempo;
main(int argc, char *argv[]){
    int i=0;
    int j=0;
    barrier_init(&mibarrera,NumThreads,NULL,NULL);
        for(i=0;i<NumThreads;i++){
                for(j=0;j<NumIterac;j++){
                        DatosComputo[i][j].N=i*50;
                        DatosComputo[i][j].M=j*50;
                }
        }

        for(i=0;i<NumIterac;i++){
                res[i]=0;
        }
        printf("\nComienzo a contar el tiempo");
        start = gethrtime();
        for(i=0;i<NumThreads;i++){
                thr_create(NULL, NULL,comienzo,(int *)i,THR_BOUND,&idthreads[i]);
                //printf("\nThread creado con numero %d",&idthreads[i]);
        }
         for ( i = 0; i < NumThreads; i++){
                thr_join(idthreads[i], NULL, NULL);
        }
        end = gethrtime();
        printf("\nDejo de medir el tiempo");
        tiempo=(end - start) / 1000000;
        printf("\nEl tiempo es:  time = %d msg\n", tiempo);
        for(i = 0; i< NumIterac; i++){
```


Izaskun Puerta Castaño NP:20.137
Fernando Muñoz Rosales NP:26.843

```
                printf("Calculo de la %d iteracion es: %lf\n",i+1,res[i]);
        }
        return 0;
}
void *comienzo(void *args){
        int i,j,z;
        int x=0;
        int error=0;
        int num;
        int NumeroT=0;
        NumeroT=(int)args;
        printf("\nComienza el thread %d",NumeroT);
        if(NumeroT%2==0){
                error=processor_bind(P_LWPID,P_MYID,0,NULL);
                printf("\nThread %d asignado a procesador 0",NumeroT);
        }else{
                error=processor_bind(P_LWPID,P_MYID,2,NULL);
                printf("\nThread %d asignado a procesador 2",NumeroT);

        }
        switch(error){
                case EFAULT:
                        printf("\nError EFAULT");
                        break;

                case EINVAL:
                        printf("\nError EINVAL");
                        break;

                case EPERM:
                        printf("\nError EPERM");
                        break;

                case ESRCH:
                        printf("\nError ESRCH");
                        break;
        }
        for(z=0;z<NumIterac;z++){
                if(thr_self() > NumThreads)
                        num = NumThreads;
                else
                        num = thr_self();
                for(i=0;i<DatosComputo[num][z].N;i++){
                        for(j=0;j<DatosComputo[num][z].M;j++){
                                x++;
                        }
                }
                barrier_wait(&mibarrera);
                mutex_lock(&variable_exclusion);
                res[z]+=x;
                //printf("\nSe despierta el threads %d",NumeroT);
                //printf("\nEl resultado es %g y el total %g con los valores de N %d y M
                //%d",x,res[z],(int)args,z);
                mutex_unlock(&variable_exclusion);

        }
```


Izaskun Puerta Castaño NP:20.137
Fernando Muñoz Rosales NP:26.843

}

BALANCEANDO 80 20

```
#define _REENTRANT
#include <stdlib.h>
#include <thread.h>
#include <synch.h>
#include <sys/types.h>
#include <sys/processor.h>
#include <sys/procset.h>
#include "errno.h"
#include "barrier.h"
#define NumThreads 10
#define NumIterac 10

int idthreads[NumThreads];
void *comienzo(void *args);
barrier_t mibarrera;
mutex_t variable_exclusion;
int i;
struct {
        int N;
        int M;
}DatosComputo[NumThreads][NumIterac];
float res[NumIterac];
hrtime_t start, end;
int tiempo;

main(int argc, char *argv[]){
        int i=0;
        int j=0;
        barrier_init(&mibarrera,NumThreads,NULL,NULL);
        for(i=0;i<NumThreads;i++){
                for(j=0;j<NumIterac;j++){
                        DatosComputo[i][j].N=i*50;
                        DatosComputo[i][j].M=j*50;
                }
        }
        for(i=0;i<NumIterac;i++){
                res[i]=0;
        }
        printf("\nComienzo a contar el tiempo");
        start = gethrtime();
        for(i=0;i<NumThreads;i++){
                thr_create(NULL, NULL,comienzo,(int *)i,THR_BOUND,&idthreads[i]);
                //printf("\nThread creado con numero %d",&idthreads[i]);
        }
         for ( i = 0; i < NumThreads; i++){
                thr_join(idthreads[i], NULL, NULL);
        }
        end = gethrtime();
        printf("\nDejo de medir el tiempo");
        tiempo=(end - start) / 1000000;
        printf("\nEl tiempo es:  time = %d msg\n", tiempo);
        for(i = 0; i< NumIterac; i++){
                printf("Calculo de la %d iteracion es: %lf\n",i+1,res[i]);
```


Izaskun Puerta Castaño NP:20.137
Fernando Muñoz Rosales NP:26.843

```
	}


	return 0;
}
void *comienzo(void *args){
	int i,j,z,error;
	int num;
	int x=0;
	int NumeroT=0;
	NumeroT=(int)args;
	printf("\nComienza el thread %d",NumeroT);
	if(NumeroT<(NumThreads*0.8)){
		error=processor_bind(P_LWPID,P_MYID,0,NULL);
		printf("\nThread %d asignado a procesador 0",NumeroT);

	}else{
		error=processor_bind(P_LWPID,P_MYID,2,NULL);
		printf("\nThread %d asignado a procesador 2",NumeroT);

	}

	switch(error){
		case EFAULT:
				printf("\nError del tipo EFAULT");
				break;
		case EINVAL:
				printf("\nError  del tipo EINVAL");
				break;
		case EPERM:
				printf("\nError del tipo EPERM");
				break;
		case ESRCH:
				printf("\nError del tipo ESRCH");
				break;
	}
	for(z=0;z<NumIterac;z++){
		 if(thr_self() > NumThreads)
			num = NumThreads;
		else
			num = thr_self();
		for(i=0;i<DatosComputo[num][z].N;i++){
			for(j=0;j<DatosComputo[num][z].M;j++){
				x++;
			}
		}
		barrier_wait(&mibarrera);
		mutex_lock(&variable_exclusion);
		res[z]+=x;
		mutex_unlock(&variable_exclusion);
	}
}
```


Izaskun Puerta Castaño NP:20.137
Fernando Muñoz Rosales NP:26.843