Programación en paralelo con *Threads*. Ejemplo y compilación

Introducción a *Threads*

¿Qué es un hilo?

De la misma manera que un Sistema Operativo puede ejecutar varios procesos al mismo tiempo, bien sea por concurrencia o paralelismo, dentro de un proceso pueden haber varios hilos de ejecución. Por tanto, un hilo puede definirse como cada secuencia de control dentro de un proceso que ejecuta sus instrucciones de forma independiente.

Sobre el hardware subyacente (una o varias CPU's) se sitúa el Sistema Operativo->Sobre éste se sitúan los procesos (Pi) que pueden ejecutarse concurrentemente->Dentro de éstos se ejecutan los hilos (hj), que también se pueden ejecutar de forma concurrente dentro del proceso.

Es decir, se da una concurrencia a dos niveles: una entre procesos y otra entre hilos de un mismo proceso. Si, por ejemplo, hay dos procesadores, se podrían estar ejecutando al mismo tiempo el hilo 1 del proceso 1 y el hilo 2 del proceso 3. Otra posibilidad podría ser el hilo 1 y el hilo 2 del proceso 1.

Los procesos son entidades pesadas. La estructura del proceso está en la parte del núcleo y, cada vez que el proceso quiere acceder a ella, tiene que hacer algún tipo de llamada al sistema, consumiendo tiempo extra de procesador. Por otra parte, los cambios de contexto entre procesos son costosos en cuanto a tiempo de computación se refiere. Por el contrario, la estructura de los hilos reside en el espacio de usuario, con lo que **un hilo es una entidad ligera.** Los hilos comparten la información del proceso (código, datos, etc.). Si un hilo modifica una variable del proceso, el resto de los hilos verán esa modificación cuando accedan a esa variable. Los cambios de contexto entre hilos consumen poco tiempo de procesador, de ahí su éxito.

¿Qué es un hilo de ejecución?

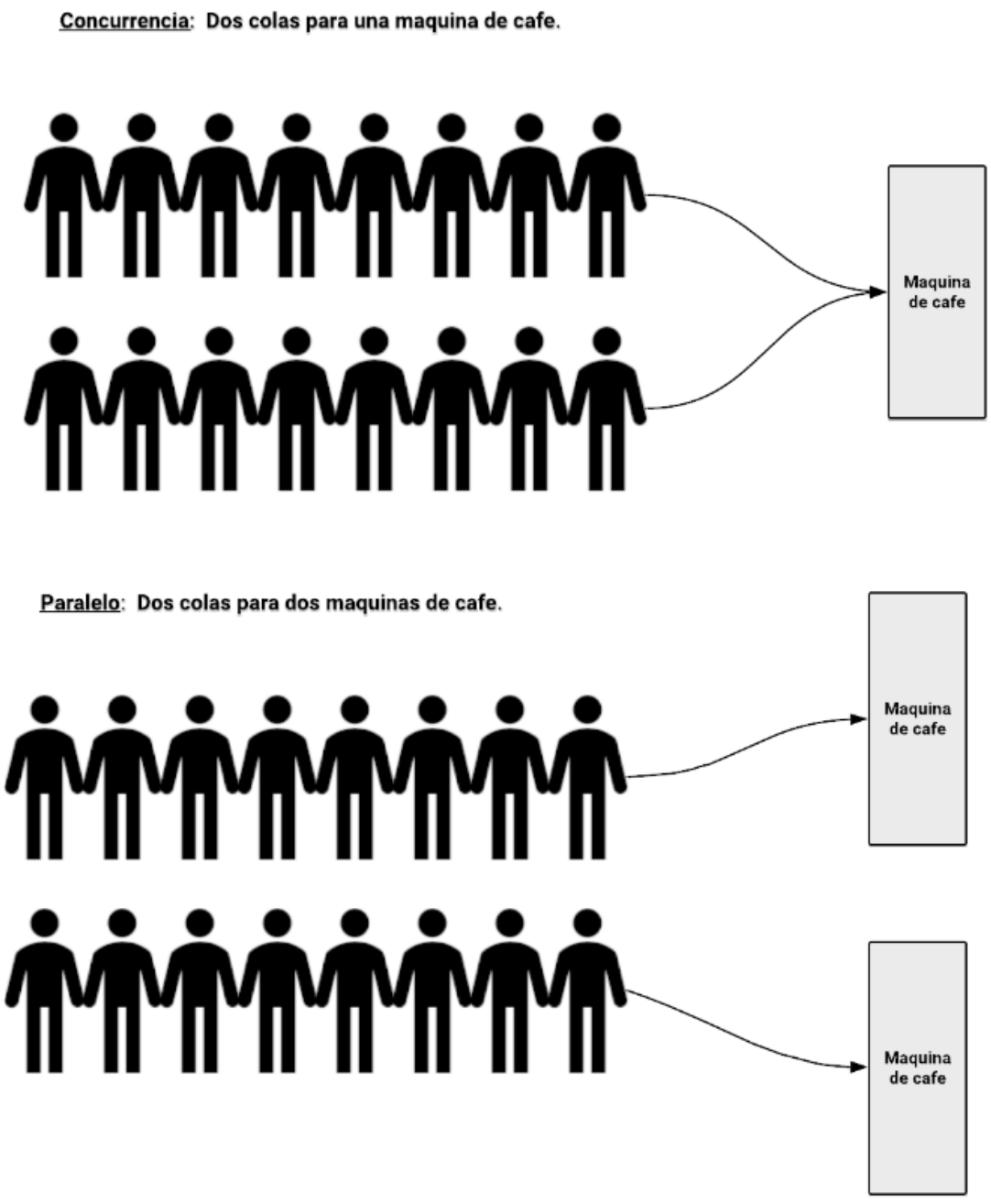
• También llamado hebra, proceso ligero, flujo, sub-proceso o "*thread*".
• Programa en ejecución que comparte la imagen de memoria y otros recursos del proceso con otros hilos.
• Desde el punto de vista de programación: **función cuya ejecución se puede lanzar en paralelo con otras.**
• Un proceso puede contener uno o más hilos.

Descriptor de un proceso y de un hilo

• Todos los hilos de un proceso comparten el mismo entorno de ejecución (variables globales, espacio de direcciones, ficheros abiertos, etc.).
• Cada hilo tiene su propio juego de registros de CPU, pila, variables locales, etc.
• No existe protección entre hilos: un error en un hilo puede estropear la pila de otro.
• Para ordenar la forma en la que los hilos acceden a datos comunes **hay que emplear mecanismos de sincronización.**

Diferencia entre concurrencia y paralelismo

Hay que aclarar que la concurrencia no sólo es paralelismo, ya que, cuando hay más procesos o hilos, el *scheduler* del sistema operativo interviene y divide la ejecución de los procesos, característica que la programación paralela **no** tiene (cada proceso tiene que ser ejecutado exclusivamente en una unidad de procesamiento). En la imagen se pude apreciar mejor la diferencia:



En la imagen de ejemplo, la máquina de café representa al recurso, y la cola representa a los procesos o hilos que intentan acceder al recurso.

¿Proceso o hilo?

Primero, los procesos son más pesados de crear y, por lo tanto, utilizan más capacidades de computadora que los hilos. Esto permite entonces hacer tareas más pesadas.
En segundo lugar, los procesos son totalmente independientes entre ellos. No comparten ningún dato entre ellos; mientras que los hilos pueden compartir información fácilmente.
Desafortunadamente, esto crea un problema: si dos hilos trabajan con los mismos datos, se pueden crear errores como, por ejemplo, el valor cambia sin entender por qué. De hecho, cuando se realizan dos tareas al mismo tiempo, si se modifican los datos en un hilo y se hace lo mismo en la otra, será imposible predecir qué hilo modificará primero la variable.
Para esto, hay un sistema para "bloquear" los datos en un hilo. Una vez que este hilo desbloquee los datos, el otro hilo recibe una señal de que puede modificarlo.
Para resumir, cuando se trabaja en los mismos datos con dos subprocesos diferentes, es necesario bloquear los datos en un hilo en el momento de modificarlo, y luego desbloquearlo.

Código

```
#include <pthread.h> //Libreria Hilos para C
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define CANTIDAD 10     //Se recomienda colocar en def.h

//Variable global para semaforo(Mutex)
pthread_mutex_t mutex;

//La funcion para hilo siempre tiene que
//ser del tipo void* y recibir únicamente
// un solo parametro del tipo void*
void *funcionThread (void *parametro)
{
      //logica o tarea del hilo.
      int i;
      printf ("Hijo\n");
      for(i=0; i<CANTIDAD; i++)
      {
           pthread_mutex_lock (&mutex);
           printf("Soy el hijo y tengo el mutex\n");

          sleep (1);
          pthread_mutex_unlock (&mutex);
          sleep(1);
      }

      printf("Soy el hijo y espero 10seg\n");
      sleep (10);
      printf ("Hijo : Termino\n");
      pthread_exit ((void *)"Listo");

}
```

```
int main(int argc,char *argv[])
{
	//Los tipos de datos están definidos en
	//la librería pthread.
	pthread_t idHilo;
	pthread_attr_t atributos;
	int error;
	int i;
	char* valorDevuelto = NULL;

	//Inicializo el semáforo (mutex)
	pthread_mutex_init (&mutex, NULL);

	//Inicializo loe atributos del hilo.
	pthread_attr_init (&atributos);

	//Seteo el estado inicial del hilo.
	//PTHREAD_CREATE_JOINABLE --> indica que voy a esperar la
	//finalización
	//del hilo. Esto me permite usar la funcion pthread_join() que esta
	//más abajo.
	pthread_attr_setdetachstate (&atributos, PTHREAD_CREATE_JOINABLE);

	//pthread_create lanza el hilo. Le paso como parámetro:
	//idHilo ---> se carga cuando se crea el hilo.
	//atributos ---> los atributos del hilo.
	//funcionThread ---> es el nombre de la función que se va a
	//ejecutar
	// al crear el hilo.
	//NULL ---> iría la dirección de memoria que recibe como parámetro
	//la función del hilo (void *parametro).
	if (pthread_create (&idHilo, &atributos, funcionThread, NULL)!= 0)
	{
		perror ("No puedo crear thread");
		exit (-1);
	}
	//Mientras se ejecutan los hilos continua ejecutandose el main.

	for(i=0; i<CANTIDAD; i++)
	{
		pthread_mutex_lock (&mutex);
		printf("Soy el padre y tengo el mutex\n");
		sleep (1);
		pthread_mutex_unlock (&mutex);
		sleep(1);
	}

	printf ("Padre : Espero al thread\n");

	//pthread_join es una funcion bloqueante que evita que
	//finalice el main(). Se desbloquea cuando van finalizando
	//los hilos. Le paso el idHilo que tiene que esperar.
	//Cada idHilo es unico para cada hilo.
	pthread_join (idHilo, (void **)&valorDevuelto);
```

```
	printf ("Padre : Ya ha terminado el thread\n");
	printf ("Padre : devuelve \"%s\"\n", valorDevuelto);
	return 0;
}
```

El código está comentado para una mejor comprensión. **Intentar compilar y ejecutar el código.**
Importante: al compilar el código agregar la opción "-phtread" en el comando "gcc" para que no de error en la compilación (si compila con el script MAKEFILE también se debe agregar la opción).
El mutex es un semáforo más liviano. Porque no se necesita pedir recursos al SO, está incluido en la librerías de hilos. Su declaración de realiza a nivel global para que sea conocida por todos los hilos, es la única variable que se permite declarar globalmente.

Ejemplo de resolución de ejercicio

Consigna

Implementar un programa que reciba a través de argumentos 1 o más números enteros. Cada uno de estos números deberá ser utilizado para determinar los N primeros números primos. Se deberán lanzar tantos *thread* como números ingresados. Cada *thread* deberá devolver la sumatoria de los N números primos calculados. Al finalizar, se deberá realizar una sumatoria total del resultado de cada *thread.*
Implementar la solución con *thread* y calcular el tiempo empleado en el cálculo total, comparar este tiempo con una solución estándar sin *thread*, es decir, realizando cada cálculo en forma secuencial (un sólo hilo). Los tiempos deberán tener una precisión del orden de los milisegundos.

Resolución

```
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <sys/time.h>

int funcionPrimos(int n)
{
	int c=0,c2=0,res=0,nc=0, total=0;
	for(c=1;c<=n;c++)
	{
	    for(c2=1;c2<=c;c2++)
	    {
	        res=c%c2;
	        if(res==0)
	        nc=nc+1;
	    }
	    if(nc==2)
	    total+=c;
	    nc=0;
	}
	return total;
}

void *funcionPrimosThread (void *parametro)
{
	int c=0,c2=0,res=0,nc=0, total=0;
	int* ptr = (int*) parametro; //recibo el parametro y lo casteo como
```

```
	//int *;
	int n= *ptr; //en n cargo el contenido apuntado por *ptr.
	printf("THREAD: Voy a calcular los %d numeros primos y los
	//sumo\n",n);
	total = funcionPrimos(n);

	printf("THREAD: %d -> El total es %d\n", n, total);
	*ptr = total; //utilizo *ptr para cargar el parametro de entrada el
	//dato de salida.

	pthread_exit ((void*)0); //Esto solo puedo utilizarlo para devolver
	//valores de status, no puedo devolver valores creados en la propia
	//funcion thread.
}

int difMilli(struct timeval tiempo_start, struct timeval tiempo_end)
{
	int sec_start, micro_start;
	int start;
	int sec_end, micro_end;
	int end;

	sec_start = tiempo_start.tv_sec & 0xFFF;
	micro_start = tiempo_start.tv_usec;
	start = (sec_start *1000) + (micro_start/1000);

	sec_end = tiempo_end.tv_sec & 0xFFF;
	micro_end = tiempo_end.tv_usec;
	end = (sec_end *1000) + (micro_end/1000);

	return end - start;
}

int main (int argc, char *argv[])
{
	char con_thread = 's';
	int i, total=0;
	int cantidad = argc -1;
	int* num = (int*) malloc(sizeof(int)*(argc-1));
	struct timeval t_inicio;
	struct timeval t_final;

	pthread_t* idHilo = (pthread_t* )
	malloc(sizeof(pthread_t)*cantidad);
	pthread_attr_t atributos;
	pthread_attr_init (&atributos);

	pthread_attr_setdetachstate (&atributos, PTHREAD_CREATE_JOINABLE);

	printf("PPAL: Cantidad = %d\n", cantidad);
	printf("PPAL: Con thread (s/n)");

	scanf(" %c", &con_thread);
	if (con_thread=='s')
		printf("\nPPAL: CON THREAD\n");
	else
		printf("\nPPAL: SIN THREAD\n");
```

```
      gettimeofday(&t_inicio, 0);

      for(i=0; i<cantidad; i++)
      {
           num[i]=atoi(argv[i+1]);
           if (con_thread=='s')
              pthread_create (&idHilo[i], &atributos, funcionPrimosThread, &num[i]);
           else
              total +=funcionPrimos(num[i]);
      }

      if (con_thread=='s')
      {
            for(i=0; i<cantidad; i++)
            {
                 printf("PPAL: Voy a esperar al thread %d\n", i);
                 pthread_join (idHilo[i], NULL);
                 printf("PPAL: Recibi: %d\n", num[i]);
                 total +=num[i];
            }
      }
      printf("PPAL: El total es %d\n", total);

      gettimeofday(&t_final, 0);

      printf("PPAL: Tiempo empleado %dms\n", difMilli(t_inicio, t_final));
      free(idHilo);

      return 0;
}
```

Nota: de las funciones o comandos de cálculo de tiempos no es necesario conocer el funcionamiento. Concentrarse en cómo se lanza o se crea más de un hilo, y cómo la función “funcionPrimosThread” retorna datos al main(int argc,char *argv[]).

Cómo enviar y recibir datos a un hilo por referencia de punteros

Código comentado(no compilable)

```
void *funcionThread (void *parametro)
{
      int* ptr = (int*) parametro; //recibo el parametro y lo casteo como int*;
      int n= *ptr; //en n cargo el contenido apuntado por *ptr.

      /*Ahora puedo utilizar n en mi funcion.
      --------------------------------
      --------------------------------
      --------------------------------
      -------------------------------- */
      *ptr = dato; //si quiero utilizar el parametro para devolver un dato generado
      //en la funcion.

      pthread_exit ((void*)0); //Esto solo puedo utilizarlo para devolver valores de
      //status, no puedo devolver valores creados en la propia funcion thread.
}
```

```
int main(int argc,char *argv[])
{
      int dato=10;
      ----------------------------------------
      ----------------------------------------
      ----------------------------------------

      //envio el puntero a la variable dato, ojo es por referencia.
      pthread_create (&idHilo, &atributos, funcionPrimosThread, &dato);
      ----------------------------------------
      ----------------------------------------

      pthread_join (idHilo, NULL);
      //ahora la variable dato seguramente tendra el valor modificado en el thread.
}
```