CONCURRENCIA

Flores, Facundo Gabriel

Paradigmas y Lenguajes

15-08-2014 - UNSA

Outline

Introducción

- ¿Qué es concurrencia?
- Cosas que aparentar suceder al mismo tiempo, pero las cuales podrían ocurrir serialmente.
- Describe el comportamiento de threads y procesos en un sistema uni-procesador.

Definición POSIX

Las funciones que suspenden la ejecución de un thread no deberían causar que la ejecución de otros threads sea suspendida infinitamente.

POSIX

POSIX

"Portable Operating System Interface Es una familia de estándares IEEE para mantener la compatibilidad entre sistemas operativos.

POSIX Threads

- Una API para C/C++ con un conjunto de estándares para el manejo de threads.
- `pthreads`
- Nos permite "engendrar" flujos concurrentes
- *Es más efectivo en sistemas multiprocesador ya que se puede ganar velocidad a través del procesamiento paralelo o distribuído*
- El propósito de utilizar una librería como tal es la de ejecutar software de manera más rápida.

Threads vs Procesos

Procesos	Threads
Diferentes espacios de memoria	Comparten un mismo espacio de memoria
Una instancia de un programa en ejecución	Un flujo de control en un programa con un único punto de ejecución
Son unidades de asignación	Unidades de ejecución
Recursos, privilegios, etc.	PC, SP, etc.
Cada proceso tiene uno o más threads	Cada thread pertenece a un único proceso

Operaciones sobre Threads

- Creación
- Finalización
- Sincronización - `join` `blocking`
- Scheduling
- Manejo de datos

Threads en un mismo proceso

- Instrucciones del proceso
- Datos
- Archivos abiertos
- Señales
- Privilegios

¿Qué contiene cada Thread?

- Thread ID
- Conjunto de registros, Stack Pointer
- Stack de variables locales

POSIX API

- Crear un thread

```
int pthread_create(pthread_t *thread,
                   pthread_attr_t *attr,
                   void *(*start_routine)(void *),
                   void *arg);
```

POSIX API

- Terminar un thread

```
void pthread_exit(void *value_ptr);
```

- Además deja disponible el puntero `*valueptr` para cualquier llamada a `pthread_join`

POSIX API

- Sincronización

```
int pthread_join(pthread_t thread,
                 void **value_ptr);
```

- Esta llamada hace que el hilo se duerma hasta que el otro hilo termine. Si el otro hilo ya había terminado, la función `pthread_join` sale inmediatamente.

POSIX API

- Para mayor información acerca de las funciones consultar: `http://linux.die.net/`

Ejemplo 1: Creación, destrucción, sincronización

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define NUM_THREADS 2

/* Una estructura para demostrar el manejo de threads */
typedef struct _PACKAGE_THREAD_T {
    int threadID;
    float data;
} PACKAGE_THREAD_T;

/* La funcion que ejecutaremos en forma concurrente */
void *worker(void *arg) {
    PACKAGE_THREAD_T *pkg = (PACKAGE_THREAD_T *)arg;
    fprintf(stdout, "Este es el thread con id: %d\n", pkg->threadID);
    fprintf(stdout, "El thread contiene la sig info: %f\n", pkg->data);
    pthread_exit(NULL);
}
```

Ejemplo 1: Creación, destrucción, sincronización

```
int main(int argc, char **argv) {
    /* Un array con threads */
    pthread_t threads[NUM_THREADS];
    /* Auxiliares */
    int i, _error;
    /* Creamos un array PACKAGE_THREAD_T */
    PACKAGE_THREAD_T threads_pkg[NUM_THREADS];

    /* Creamos los threads */
    for (i = 0; i < NUM_THREADS; ++i) {
        threads_pkg[i].threadID = i;
        threads_pkg[i].data = i / 2.0f;
        //Intentamos la creacion de un thread
        _error = pthread_create(&threads[i], NULL, worker, &threads_pkg[i]);
       if (_error){
          fprintf(stderr, "error: pthread_create, rc: %d\n", _error);
          return EXIT_FAILURE;
          }
    }

    /* Bloqueamos la ejecucion hasta que todos los threads hayan terminado */
    for (i = 0; i < NUM_THREADS; ++i)
        pthread_join(threads[i], NULL);

    return EXIT_SUCCESS;
}
```

Ejemplo 2: Multiplicación Matriz-Vector - Serial

```
void _math_vec_prod(void)
{
    int i,j;
    for(i = 0; i < M; i++)
    {
        y_serial[i] = 0.0f;
        for(j = 0; j < N; j++)
        y_serial[i] += A[i * N + j] * x[j];
    }
}
```

Ejemplo 2: Multiplicación Matriz-Vector - Paralelo

```
void *_math_vec_prod(void* rank) {
    int whoami = *(int *) rank;
    int from = whoami * M / NUM_OF_THREADS;
    int to = from + (M / NUM_OF_THREADS) - 1;
    int i, j;
    for (i = from; i <= to; i++) {
        y[i] = 0.0f;
        for (j = 0; j < M; j++)
            y[i] += A[i*M+j]*x[j];
    }
    return NULL;
}
```

Algunos Resultados

```
$: ./a.out 512 512 4
TEST OK
SIZE: 512 x 512 AND 4 THREADS
TIEMPO CONC.:3 mseg
TIEMPO SERIAL:7 mseg

$: ./a.out 2048 2048 4
TEST OK
SIZE: 2048 x 2048 AND 4 THREADS
TIEMPO CONC.:26 mseg
TIEMPO SERIAL:35 mseg

$: ./a.out 4096 4096 4
TEST OK
SIZE: 4096 x 4096 AND 4 THREADS
TIEMPO CONC.:55 mseg
TIEMPO SERIAL:105 mseg

$: ./a.out 8192 8192 16
TEST OK
SIZE: 8192 x 8192 AND 16 THREADS
TIEMPO CONC.:120 mseg
TIEMPO SERIAL:379 mseg

$./a.out 16384 16384 16
TEST OK
SIZE: 16384 x 16384 AND 16 THREADS
TIEMPO CONC.:457 mseg
TIEMPO SERIAL:1507 mseg
```