CC3301 Programación de Software de Sistemas

Control 3 – Semestre Otoño 2013

Prof.: Luis Mateu

Pregunta 1

La siguiente función resuelve el problema de comparación de secuencias DNA. Esta función recibe como parámetros un arreglo de secuencias DNA llamado *secuencias*, su tamaño *n* y 2 punteros *pi* y *pj* con la dirección en donde dejar la posición de las 2 secuencias que más se parecen:

```
void mayorSimilitud(Dna *secuencias, int n, int *pi, int *pj) {
  int i, j;
  int min= MAXINT;
  for (i= 1; i<n; i++)
    for (j= 0; j<i; j++) {
      int similitud= compDna(secuencias[i], secuencias[j]);
      if (similitud<min) {
        min= similitud;
        *pi= i; *pj= j;
      }
    }
  }
}
```

En esta pregunta la función *compDna* es dada y entrega un coeficiente de similitud. Considerando que realizar todas estas comparaciones toma mucho tiempo, se necesita modificar la funcion mayorSimilitud de manera que envíe a la salida estándar un texto indicando *cada un minuto* el total de comparaciones realizadas hasta el momento. Un ejemplo de la salida esperada es el siguiente:

```
7 comparaciones
9 comparaciones
25 comparaciones
... etc ...
```

Observe que ciertos pares de secuencias son muy rápidos de comparar mientras que otros pares son lentos de comparar.

Parte a.- Resuelva este problema usando señales.

Parte b.- Resuelva el mismo problema usando threads, mutex y condiciones. Para ello use la función:

```
int pthread_cond_timedwait(pthread_cond_t *cond, pthread_mutex_t *mutex, int secs);
```

Esta función hace lo mismo que la función *phread_cond_wait*, excepto que si al cabo de *secs* segundos el thread no ha recibido un *broadcast* entonces se despierta automáticamente, retornando el valor *ETIMEDOUT*. No olvide enterrar adecuadamente el thread que va a crear.

Nota: el tipo del tercer parámetro de *pthread_cond_timedwait* es *struct timespec* pero por simplicidad en este control considere que el tipo es int.

Pregunta 2

Se dispone de una máquina con 8 cores y se le pide a Ud. modificar la función *mayorSimilitud* de la pregunta anterior de modo que se usen de la mejor forma estos 8 cores para reducir el tiempo de ejecución.

Para ello Ud. *debe* emplear la siguiente metodología. Al inicio de la función cree 8 threads adicionales que se usarán para realizar las comparaciones y que llamaremos los *threads comparadores*. Al finalizar preocúpese de enterrarlos apropiadamente. El thread principal (el que ejecuta *mayorSimilitud*) se usa para alimentar a los threads comparadores con pares (i, j) de secuencias que se deben comparar. Cada thread comparador espera hasta obtener del thread principal un par (i, j) de secuencias, las compara, revisa si se trata del par más similar del momento y luego espera a recibir un nuevo par. Por otra parte el thread principal se dedica a generar un par (i, j), espera a que se desocupe algún thread comparador, se lo asigna y luego genera un nuevo par, espera un thread comparador, se lo asigna, …, etc. Para realizar la sincronización use un mutex y una condición. En un thread comparador *no olvide devolver el mutex* antes de llamar a *compDna*, si no, no habrá paralelismo y su solución no reducirá el tiempo de ejecución.

Observaciones:

- Este es un caso particular del problema del productor/consumidor con 1 productor, 8 consumidores y un buffer de tamaño 1.
- Ud. puede usar variables globales y *no* necesita desplegar cada un minuto la cantidad de secuencias comparadas.