CC3301 Programación de software de sistemas – Tarea 7 – Otoño 2024 – Profesores Mateu/Ibarra/Urrea

El problema de la suma de subconjuntos es este: dado un conjunto *a* de *n* enteros, ¿existe algún subconjunto de *a* no vacío cuya suma sea exactamente cero? Por ejemplo, dado el conjunto { −7, −3, −2, 5, 8}, la respuesta es sí, porque el subconjunto { −3, −2, 5} suma cero. Considere $1 \le n \le 64$. La siguiente solución secuencial toma tiempo $O(n \cdot 2^n)$.

```
typedef unsigned long long Set;
Set buscarSeq(int a[], int n) {
  Set comb= (1<<(n-1)<<1)-1; // 2^n-1: nº de combinaciones
  for (Set k= 1; k<=comb; k++) {
    // k es el mapa de bits para el subconjunto { a[i] | bit k_i de k es 1 }
    long long sum= 0;
    for (int i= 0; i<n; i++) {
      if ( k & ((Set)1<<i) ) // si bit k_i de k es 1
        sum+= a[i];
    }
    if (sum==0) {   // éxito: el subconjunto suma 0
      return k;     // y el mapa de bits para el subconjunto es k
  } }
  return 0;         // no existe subconjunto que sume 0
}
```

En la función *buscarSeq* los subconjuntos de *a* se representan mediante una máscara de bits $k = k_{n-1} \ldots k_1 k_0$. El elemento *a[i]* está en el subconjunto si k_i es 1. Esta función recorre los 2^n-1 subconjuntos posibles (se descarta el subconjunto vacío) hasta encontrar un subconjunto que sume 0, en cuyo caso se retorna su mapa de bits. Si ningún subconjunto suma 0, se retorna 0.

Programe la siguiente función:

```
Set buscar(int a[], int n, int p)
```

Esta función resuelve el mismo problema que *buscarSeq* pero en paralelo. Para ello Ud. debe invocar *p* veces *fork* para crear *p* procesos pesados. Utilice un *pipe* por cada proceso hijo para que este le entregue al padre el mapa de bits del subconjunto encontrado. El padre no busca subconjuntos, solo espera el término de los hijos. Si hay múltiples subconjuntos que suman 0, entregue cualquiera de ellos.

Instrucciones

Baje *t7.zip* de U-cursos y descomprímalo. El directorio *T7* contiene los archivos *test-suma.c*, *Makefile*, *suma.h* (con los encabezados requeridos) y otros archivos. Ud. debe crear el archivo *suma.c* y programar ahí la función *buscar*.

Pruebe su tarea bajo Debian 12. Ejecute el comando *make* sin parámetros. Le mostrará las opciones que tiene para compilar su tarea. Estos son los requerimientos para aprobar su tarea:

- *make run* debe felicitarlo por aprobar este modo de ejecución. El *speed up* reportado debe ser de al menos 1.5.
- *make run-g* debe felicitarlo.

Cuando pruebe su tarea con *make run* en su notebook asegúrese de que posea al menos 2 cores, que esté configurado en modo alto rendimiento y que no estén corriendo otros procesos intensivos en uso de CPU al mismo tiempo. De otro modo podría no lograr el *speed up* solicitado.

Invoque el comando *make zip* para ejecutar todos los tests y generar un archivo *suma.zip* que contiene *suma.c*, con su solución, y *resultados.txt*, con la salida de *make run*, *make run-g* y make *run-san*.

Entrega

Ud. solo debe entregar por medio de U-cursos el archivo *suma.zip* generado por el comando *make zip*. **A continuación es muy importante que descargue de U-cursos el mismo archivo que subió, luego descargue nuevamente los archivos adjuntos y vuelva a probar la tarea tal cual como la entregó**. Esto es para evitar que Ud. reciba un 1.0 en su tarea porque entregó los archivos equivocados. Créame, sucede a menudo por ahorrarse esta verificación. Se descontará medio punto por día de atraso. No se consideran los días de receso, sábados, domingos o festivos.