Pregunta 1 (40%)

En el cuadro de la derecha se define la función *f*. Considere que las funciones *p*, *g* y *h* pueden tomar mucho tiempo de ejecución. El caso peor para el tiempo de cálculo de *f* es el tiempo que toma calcular *p* más el máximo que toma calcular *g* o *h*.

```
int p(double x);
double g(double x);
double h(double x);
double f(double x) {
  if (p(x))
    return g(x);
  else
    return h(x);
}
```

Reprograme *f* de manera que se calcule en paralelo *p*, *g* y *h* al ejecutar en un computador con 3 cores. De esta forma el caso peor mejora, puesto que será el máximo tiempo que toma calcular *p*, *g* o *h*. Para lograrlo Ud. debe crear 2 procesos hijos usando *fork*. Los hijos deben entregar sus resultados al padre por medio de pipes. El padre calcula *p*.

Pregunta 2 (60%)

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 toma tiempo $O(n \cdot 2^n)$.

```
typedef unsigned long long Set;
Set buscar(int a[], int n) {
  Set comb= ((Set)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 *buscar* 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.

Se necesita un sistema cliente/servidor para paralelizar la búsqueda de todas las soluciones del problema de la suma de subconjuntos.

El servidor se ejecuta con el comando *coordinador*, corre en anakena, usa el puerto 3000 y emplea threads para conversar con los clientes. Recibe como parámetros los elementos del conjunto *a*. El servidor envía *n* y el conjunto *a en formato binario* a cada cliente que se conecte. El intervalo de búsqueda se descompone en subintervalos enumerados de 1 a $\lceil comb/10000 \rceil$ para repartirlos a los clientes. El *i*-ésimo subintervalo es [*1+(i-1)*10000*, *i*10000*], excepto el último que llega hasta *comb*. Los clientes envían al servidor un caracter con la letra 'p' para pedir un subintervalo en donde buscar. El thread que atiende a ese cliente en el servidor responde con el número del subintervalo *i* asignado. Si se agotaron los subintervalos, responde con un 0 y el thread termina.

El cliente se ejecuta con el comando *sumador*. No se sabe a priori cuántos serán los clientes. Después de contactar al servidor, el cliente recibe a través del socket el valor *n* y los elementos del conjunto *a*. Luego pide al servidor un subintervalo *i* en donde buscar (recuerde: enviando al servidor un caracter con la letra 'p'). Cuando encuentra un subconjunto de *a* que suma 0 lo despliega en pantalla y continúa. Al concluir la búsqueda en ese subintervalo pide un nuevo subintervalo i´. Esto se repite hasta recibir un subintervalo con identificación 0 indicando al cliente que debe terminar.

El siguiente es un ejemplo de uso de este sistema:

servidor	***cliente 1***	***cliente 2***	***cliente 3***
$ **coordinador -7** ***-3 -2 5*** **8 11 21 34 14 56 23 89 93 75 93 95**			
65535 comb. 7 subintervalos enviando 1 enviando 2 enviando 3 enviando 4 enviando 5 enviando 6 enviando 7 fin *(termina con control-C)*	$ **sumador** pedir buscando en 1 **R: -3 -2 5** pedir buscando en 5 pedir $	 $ **sumador** pedir buscando en 2 pedir buscando en 4 pedir buscando en 7 pedir $	 $ **sumador** pedir buscando en 3 pedir buscando en 6 pedir $

Requerimientos: Programe el cliente y el servidor. En la función *main* del servidor use "…" para el código que coincide con el ejemplo del diccionario distribuido o la tarea 4 (manejo de sockets y threads). Pero deberá escribir el código que inicializa las variables globales que necesita. Por ejemplo *n*, el arreglo *a*, el subintervalo *i* que se enviará al próximo cliente que lo pida, etc. En la función de servicio *serv* requerirá un mutex para evitar dataraces al acceder a *i*, pero no necesita una condición.