Pregunta 1

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≤n≤64. La siguiente solución toma tiempo $O(n \cdot 2^n)$.

```
typedef unsigned long long Set;
Set buscar(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 *buscar* los subconjuntos de *a* se representan mediante una máscara de bits $k= k_{n-1}... k_1 k_0$. 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.

Reprograme completamente la función *buscar* de modo que la búsqueda se haga paralelamente en 8 cores. Para ello use *fork* para crear 8 procesos pesados. Utilice un *pipe* por cada proceso hijo para que este le entregue al padre el mapa de bits del subconjunto encontrado. Si hay múltiples subconjuntos que suman 0, entregue cualquiera de ellos.

Pregunta 2

Se dispone de 3 ayudantes, Pedro, Juan y Diego, para realizar tareas diversas. Se requiere programar el servidor y el cliente de un sistema que administre la disponibilidad de los ayudantes. El servidor se llama *admin*, corre siempre en *localhost* y escucha a los clientes en el puerto 3000. No recibe parámetros.

El cliente se llama *ayuda* y recibe 1 o 2 parámetros. El primer parámetro es la operación que puede ser S para solicitar un ayudante o L para notificar que se liberó un ayudante. Si la operación es S se despliega el nombre del ayudante otorgado. Un ayudante previamente otorgado no puede otorgarse nuevamente hasta que sea liberado. Si no hay ayudantes disponibles, se espera hasta que se libere uno. Si la operación es L el segundo parámetro es el nombre del ayudante que se libera. El siguiente es un ejemplo de uso del comando *ayuda*:

shell 1	***shell 2***	***shell 3***	***shell 4***
$ **ayuda S** pedro $			
	$ **ayuda S** juan $		
		$ **ayuda S** diego $	
			$ **ayuda S** *(espera)*
	$ **ayuda L juan** $		juan $

Note que el *prompt* $ indica cuando un comando termina o si debe esperar. El tiempo avanza hacia abajo. En **negritas** aparece lo que escribió un usuario.

Programe el cliente y el servidor de este sistema. Use threads en el servidor para conversar con los clientes. No programe la función *main* del servidor, solo necesita programar la función de servicio (*serv*) más otras funciones que le hagan falta. Declare las variables globales que requiera señalando cómo deben ser inicializadas. No necesita preocuparse por el término del servidor. Para el cliente sí necesita programar la función *main*. El cliente debe reproducir exactamente la salida que se muestra en el ejemplo, exceptuando el texto que dice *(espera)*.