Pregunta 1

Considere que Ud. viaja a Europa y puede llevar una maleta de hasta *maxW* kilos. Dispone de un conjunto de *n* artículos $\{A_0, A_1, ..., A_{n-1}\}$. El artículo A_i pesa *w[i]* kilos y vale *v[i]* euros. No puede llevar todos los artículos porque la suma de sus pesos excede *maxW*. Debe elegir qué artículos llevar maximizando la suma de sus valores. Este problema se conoce como Knapsack 0-1 y está en la categoría NP-difícil. El mejor algoritmo conocido que calcula la solución óptima toma tiempo $O(2^n)$. La siguiente función genera aleatoriamente *k* subconjuntos de artículos y elige el de mayor valor que no exceda *maxW*. No es el óptimo pero es una buena solución y se calcula en un tiempo razonable.

```
double llenarMaleta(double w[], double v[], int z[], int n,
                    double maxW, int k) {
  double best= -1;
  while (k--) {
    int x[n];
    double sumW= 0, sumV= 0;
    for (int i=0; i<n; i++) {
      x[i]= random0or1() && sumW+w[i]<=maxW ? 1 : 0;
      if (x[i]==1) {
        sumW += w[i];
        sumV += v[i];
    } }
    if (sumV>best) {
      best= sumV;
      for(int i=0; i<n; i++) {
        z[i]= x[i];
  } } }
  return best;
}
```

Al retornar, el subconjunto con los artículos elegidos es $\{ A_i \mid tq\ z[i]=1 \}$. La función *random0or1* es dada y entrega aleatoriamente 0 o 1.

Reprograme la función *llenarMaleta* de modo que la elección se haga en paralelo en 8 cores. Para ello use *fork* para lanzar 8 procesos pesados. Cada uno de ellos evalúa *k/8* subconjuntos aleatorios. Use un pipe por cada proceso hijo para que este entregue al padre el mejor subconjunto que encontró. El padre elige el mejor entre los 8 subconjuntos seleccionados por los hijos y entrega el resultado final.

Revise que su solución posea paralelismo. Si no, su nota será muy baja.

Pregunta 2

Esta pregunta consiste en paralelizar el mismo problema de la pregunta 1 recurriendo a un número variable de computadores conectados a Internet, organizados en un esquema cliente/servidor.

El proceso servidor corre en anakena y se invoca de la siguiente manera:

```
$ ./llenar-maleta n max-w k w[0] … w[n-1] v[0] … v[n-1]
```

Acepta conexiones de los clientes a través del puerto 3000 y crea para cada uno de ellos un thread que se encarga de enviar los parámetros del problema al cliente y conversar con él.

Cada cliente (comando *explorar*) realiza múltiples exploraciones de 1 millón de subconjuntos de artículos cada una. Al final de cada exploración envía el subconjunto con mejor valor al servidor y espera una confirmación para proseguir con una nueva exploración. Si la respuesta es negativa, termina.

El siguiente es un ejemplo de uso que muestra el servidor trabajando con 3 clientes. Los clientes pueden llegar en cualquier momento. El despliegue de los comandos se muestra en orden cronológico.

servidor	*cliente 1*	*cliente 2*	*cliente 3*
$ **./llenar-maleta 60 23 1000000000** *pesos en kilos … precios en euros …*			
cliente c1 conectado	$ **./explorar**		
cliente c2 conectado		% **./explorar**	
cliente c3 conectado			% **./explorar**
mejor valor (c1)= 1000	mejor= 1000		
c2 descartado		mejor= 900	
mejor valor (c3)= 1100			mejor= 1100
c1 descartado	mejor= 850		
mejor valor (c2)= 1200		mejor= 1200	
…	…	…	…
Fin! 1000000000 de subconjuntos explorados. Mejor valor= 1300 Articulos= A3 A5 A12 A17 …	$	$	$

Programe el servidor (*llenar-maleta*) y el cliente (*explorar*) de manera que reproduzcan las salidas estándares mostradas en el ejemplo de uso. En el servidor programe toda la función *serv* y solo el inicio de *main* para transferir los parámetros del comando de *argv* a variables globales (no programe desde *j_bind* en adelante). Use *atof(str)* para convertir un número contenido en un string a *double*. No se preocupe por el término del servidor. Tenga cuidado con los datarraces en el servidor: necesitará un mutex. Para el cliente (*explorar*) debe programar toda la función *main* y preocuparse de su término.