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 demasiado tiempo: es $O(2^n)$. La función *llenarMaleta* de abajo es una heurística: entrega una buena solución para este problema y en un tiempo razonable, pero no es la solución óptima. Para lograrlo genera aleatoriamente *k* subconjuntos de artículos y elige el de mayor valor que no exceda *maxW*. Al retornar, **la solución del problema se entrega en el arreglo *z***: 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.

```
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;
}
```

Reprograme la función *llenarMaleta* usando la misma heurística pero de modo que la elección se haga en paralelo para una máquina con 8 cores. Para ello lance 8 nuevos procesos invocando 8 veces *fork*. Cada proceso evalúa *k/8* subconjuntos aleatorios. Use el proceso padre solo para elegir la mejor solución entre las mejores encontradas por los procesos hijos. Deberá crear 8 pipes para que cada proceso hijo envíe su solución al proceso padre.

Ud. encontrará el siguiente problema: todos los procesos hijo generarán exactamente los mismos subconjuntos llegando todos a la misma solución porque la función *random0or1* entregará la misma secuencia de números aleatorios en los 8 procesos hijos. Para lograr que cada proceso genere secuencias distintas cambie la semilla para la función *random* en cada proceso hijo, antes de generar los subconjuntos aleatorios con:

*srandom(getUSecsOfDay()*getpid());*

Para evitar el warning de encabezado indefinido para *srandom* agregue esta primera línea en *t8.c* (como está hecho en *test-t8.c*):

#define _XOPEN_SOURCE 500

Se requiere que el incremento de velocidad (*speed up*) sea al menos un factor 1.5x. Cuando pruebe su tarea en su notebook asegúrese que posea al menos 2 cores y que esté configurado en modo máximo rendimiento. Si está configurado para ahorro de batería podría no lograr el *speed up* solicitado.

Instrucciones

Descargue *t8.zip* de U-cursos y descomprímalo. Ejecute el comando *make* sin parámetros en el directorio *T8* para recibir instrucciones acerca del archivo en donde debe programar su solución (*maleta.c*), cómo compilar y probar su solución y los requisitos que debe cumplir para aprobar la tarea.

Entrega

Ud. solo debe entregar por medio de U-cursos el archivo *maleta.zip* generado por el comando *make zip*. Contiene los archivos *maleta.c* y *resultados.txt*. 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, sábado, domingo o festivos.