Pregunta 1 (30%)

El cuadro de la izquierda es un ejemplo del contenido de un archivo en el que se ha almacenado una lista enlazada *ordenada* alfabéticamente. Cada línea tiene exactamente 40 caracteres. Los primeros 10 contienen el número de la siguiente línea en la lista enlazada (0 para el final de la lista), luego viene el texto en 29 caracteres y por último el newline \n. La línea 0 es especial porque solo contiene el número de la primera línea de la lista enlazada.

Programe la función *void mostrar(char *arch)* que muestra en la salida estándar el texto contenido en *arch* ordenadamente. El cuadro de al lado muestra el formato de la salida para el archivo del ejemplo.

```
alimento
casa
celular
gato
radio
```

Restricción: No puede leer todo el archivo en memoria. Use *fseek* para recorrer ordenadamente el archivo.

Ayuda: API para manejar archivos.

```
FILE  *fopen(char *nom_arch, char *modo);   // modo es "r+" para lect/escr
size_t  fread(char *buf, size_t ancho_item, int n_item, FILE *file);
int  fseek(FILE *file, long despl, int w);   // w==SEEK_SET
int  fclose(FILE *file);
```

Pregunta 2 (40%)

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 arriba a la derecha 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.

Reprograme la función *llenarMaleta* de modo que la elección se haga en paralelo para una máquina con 8 cores. Para ello lance 8 nuevos threads. Cada uno evalúa *k/8* subconjuntos aleatorios. Use el thread principal solo para elegir la mejor solución encontrada por cada uno de los threads y retornar esa solución.

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

```
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;
}
```

Pregunta 3 (30%)

Considere un productor y múltiples consumidores. La función *get* de al lado es usada por los consumidores para extraer un ítem del buffer. El problema de esta solución es que si hay múltiples consumidores en espera, cuando el productor deposita un ítem, cualquiera de las llamadas pendientes de *get* podría extraer ese ítem.

```
Item *get(Buffer *b) {
  lock(&b->m);
  while (b->cnt==0)
    wait(&b->cond, &b->m);
  Item *item=
          b->array[b->out];
  b->out= (b->out+1) %
           b->size;
  b->cnt--;
  broadcast(&b->cond);
  unlock(&b->m);
  return item;
}
```

Reprograme la función *get* para que haga que el consumidor que logra extraer el ítem recién depositado es aquel que lleva más tiempo esperando. Señale los campos que necesita agregar a la estructura *Buffer* y qué valor inicial deben tener.

Ayuda: Use un protocolo similar al que usan farmacias, isapres, bancos, etc. Al llegar, un cliente toma un número. Un visor anuncia el número del próximo cliente que se atenderá. Cuando se termina de atender a ese cliente, el visor se incrementa en 1. Un cliente sabe que es el próximo cuando el número del visor coincide con su propio número.