Pregunta 1 (40%)

El archivo de texto *dicc.txt* almacena un diccionario en el formato que muestra la columna de la izquierda de la siguiente tabla:

archivo *dicc.txt* original	***dicc.txt* después de compactar**
perro:guau gato:miau vaca:muuu lobo:auuu	perro:guau gato:miau vaca:muuu lobo:auuu

Cada línea almacena una asociación en el diccionario. Por ejemplo la primera línea asocia la llave *perro* con el valor *guau*. Todas las líneas tienen exactamente 19 caracteres más el *\n*. Algunas de las líneas aparecen en blanco porque la asociación fue borrada. Escriba un programa que compacte el diccionario, lo que significa que todas las líneas en blanco deben quedar al final del archivo. La columna de la derecha de la tabla de arriba muestra el resultado de compactar *dicc.txt*. El tamaño del archivo no cambia. Debe escribir las líneas en blanco al final del archivo.

***Restricción*:** El archivo completo no cabe en la memoria. Solo puede almacenar en memoria unas pocas líneas. Lea una línea con *fread* y reescríbala en el lugar adecuado del archivo usando *fseek* y *fwrite*.

Ayuda: API para manejar archivos y otras funciones de utilidad:

```
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);
    // fread retorna el número efectivo de items leídos (no la cantidad de bytes)
size_t  fwrite(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);
int strncmp(char *s, char *r, size_t n); // compara a los más n bytes de s y r
char *strncpy(char *dest, char *src, size_t n); // copia a los más n bytes de src
                                                // en dest
```

Pregunta 2 (60%)

Parte a.- En el cuadro de arriba a la izquierda 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 nuevos threads: uno para calcular *g* y el otro para calcular *h*. Calcule *p* en el mismo thread que invocó *f*.

Parte b.- La función *imprimir* sirve para imprimir documentos desde múltiples threads. Evita que 2 threads accedan a la impresora simultáneamente. Recibe el documento que se debe imprimir y un entero entre 0 y 9 que representa la prioridad. La siguiente es una implementación incompleta de esta función porque no considera la prioridad.

```
pthread_mutex_t m= PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond= PTHREAD_COND_INITIALIZER;
int ocup= 0;
void imprimir(Doc *doc, int pri) {
  lock(&m);
  while (ocup)
    wait(&cond, &m);
  ocup= 1;
  unlock(&m);

  doPrint(doc); // imprime de verdad

  lock(&m);
  ocup= 0;
  broadcast(&cond);
  unlock(&m);
}
```

Complete esta implementación de modo que una impresión con prioridad *p* no pueda comenzar mientras exista una impresión pendiente con prioridad $q > p$.

Ayuda: Necesitará usar un arreglo global que contabilice las impresiones pendientes para cada prioridad. Programe una función que dada una prioridad *p* retorne verdadero si existen impresiones pendientes con mayor prioridad, o falso en caso contrario. Impresiones con la misma prioridad pueden realizarse en cualquier orden.