CC3301 Programación de software de sistemas

- buffering
- perror
- fseek
- búsqueda binaria en un archivo

Buffering

- Leer de archivos es lento
- Comparado con leer la memoria del computador
- Leer 1 byte toma el mismo tiempo que leer 8 KB
- La estructura FILE maneja un *buffer* de 8 KB
- Buffer: memoria para almacenar datos temporalmente
- Aunque se lean pocos bytes, internamente se lee el buffer completo
- Los próximos bytes no se leen del archivo, si no que del buffer
- Cuando se acaba el buffer, se vuelve a leer el buffer completo

Buffering

- Experimento: compile el programa fprintf.c con:

  make fprintf

- Ejecute el programa con ddd fprintf hasta ver que printf muestra esta línea:

  14475267-1 20 pedro

- Luego muestre el contenido de datos.txt: ¡esa línea todavía no aparece!

```c
#include <stdio.h>
#include <string.h>

int main() {
  char lin[82];
  FILE *in= fopen("pers.txt", "r");
  FILE *out= fopen("datos.txt", "w");
  for (;;) {
    if (fgets(lin, 82, in)==NULL)
      break;
    fprintf(out, "%s", lin);
    printf("%s", lin);
  }
  return 0;
}
```

Fflush

- Experimento 2: compile el programa fprintf2.c con:

  make fprintf2

- Ejecute el programa con ddd fprintf2 hasta ver que printf muestra esta línea:

  14475267-1 20 pedro

- Luego muestre el contenido de datos.txt: ¡ahora la línea sí aparece!

```c
#include <stdio.h>
#include <string.h>

int main() {
  char lin[82];
  FILE *in= fopen("pers.txt", "r");
  FILE *out= fopen("datos.txt", "w");
  for (;;) {
    if (fgets(lin, 82, in)==NULL)
      break;
    fprintf(out, "%s", lin);
    fflush(out);
    printf("%s", lin);
  }
  return 0;
}
```

La función fflush escribe el contenido del buffer en el archivo, dejando el buffer vacío

Buffering

- Escribir archivos es lento
- Cuando se escriben pocos bytes, se escriben en el buffer
- Solo se escribe en disco
  - cuando se llena el buffer
  - cuando se cierra el archivo con:

    fclose(stream);
  - Cuando se invoca:

    fflush(stream);

Manejo de errores

- Prácticamente todas las funciones estándares tienen una manera de indicar un error
- Fopen retorna NULL
- Fread retorna número negativo
- La siguiente función permite explicar el error al usuario

  void perror(const char *s);

- Ejemplo:

```
FILE *in= fopen(argv[1], "r");
if (in==NULL) {
  perror(argv[1]);
  exit(1);
}
```

Acceso directo: fseek

- En la estructura FILE se mantiene un cursor hacia la próxima posición en el archivo que se debe leer o escribir
- Normalmente los archivos se leen y escriben secuencialmente
- La función fseek permite leer o escribir directamente cualquier parte del archivo sin tener que haber leído o escrito lo que venía antes
- Ejemplo: búsqueda binaria en un archivo
- Podemos ver ese archivo como un arreglo, pero todas las líneas deben tener el mismo tamaño
- Definir estructura de una línea:

```
typedef struct {  char llave[20], val[40], newline;  } Linea;
```

Cuidado: ¡No son strings!

- Leer k-ésima línea:

```
fseek(fileDicc, k*sizeof(Linea), SEEK_SET);
```

- Calcular tamaño del archivo en bytes:

```
fseek(fileDicc, 0, SEEK_END);
int tamArch= ftell(fileDicc);
```

Ejemplo: búsqueda en diccionario

- Archivo: dicc.txt
- Ordenado por la llave
- Líneas de 60 caracteres más cambio de línea
- Compilación:

  make buscar

- Ejemplos de uso

```
gato                mamifero que dice miau
loro                ave parlanchina
oveja               mamifero que dice be-e-e-e-e-e
perro               mamifero que dice guau
pollo               ave que dice pio pio
raton               mamifero despreciable
vaca                mamifero que dice muuuuuuu
zorro               mamifero astuto
```

```
$ ./buscar dicc.txt loro
Valor(loro)=ave parlanchina
pss@debian11mate:~/CC3301/Slides/Fseek
$ ./buscar dicc.txt zorro
Valor(zorro)=mamifero astuto
pss@debian11mate:~/CC3301/Slides/Fseek
$ ./buscar dicc.txt canario
Llave canario no encontrada
```

Programa: página 1

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

typedef struct {
  char llave[20];
  char val[40];
  char newline;
} Linea;

int main(int argc, char *argv[]) {
  if (argc!=3) {
    fprintf(stderr, "Uso: %s <arch-dicc> <llave>\n", argv[0]);
    exit(1);
  }
  char *nomArch= argv[1];
  char *llave= argv[2];
  int tamLlave= strlen(llave);

  FILE *fileDicc= fopen(nomArch, "r");
  if (fileDicc==NULL) {
    perror(nomArch);
    exit(2);
  }
  fseek(fileDicc, 0, SEEK_END);
  int tamArch= ftell(fileDicc);
  int numDef= tamArch / sizeof(Linea);
  int i= 0, j= numDef-1;
```

Programa: página 2

```
  int i= 0, j= numDef-1;
  Linea lin;
  while (i<=j) {
    int k= (i+j+1)/2;
    if (fseek(fileDicc, k*sizeof(Linea), SEEK_SET)!=0) {
      perror(nomArch);
      exit(3);
    }
    if (fread(&lin, sizeof(Linea), 1, fileDicc)!=1) {
      perror(nomArch);
      exit(4);
    }
    int rc= strncmp(llave, lin.llave, tamLlave);
    if (rc==0)
      break;
    else if (rc<0)
      j= k-1;
    else
      i= k+1;
  }
  if (i>j)
    printf("Llave %s no encontrada\n", llave);
  else {
    lin.newline= 0;
    printf("Valor(%s)=%s\n", llave, lin.val);
  }
  return 0;
}
```