SO2 - Pràctica 1

Autor: Taras Yarema

Respostes

Pregunta 1

(a)

Donat un `N`, la mida del fitxer hauria de ser `N*sizeof(int)` bytes. En el cas del meu ordinador, `sizeof(int)=4` bytes. Per tant, si executem

```
gcc write_int -o write_int
./write_int data 12
```

obtindrem que la mida de `data` és de 48 bytes.

En quan a la mida del fitxer abans de tancar, podem afegir les següents línies abasn de tancar-lo per a obtenir-ne la mida:

```
off_t fsize;
fsize = lseek(fd, 0, SEEK_END);
printf("Size of file descriptor: %ld\n", fsize);
```

en el cas anterior amb `N=12` obtenim per pantalla:

```
Size of file descriptor: 48
```

Per tant, podem deduïr que que la mida no variar abans de tancar el fitxer.

(b)

El programa `read_int.c` llegeix correctament les dades generades per el programa anterior. Seguint en el mateix cas, si executem

```
gcc read_int.c -o read_int
./read_int data
```

retorna

```
0
1
2
3
4
5
6
7
8
```

```
9
10
11
```

que és el output esperat.

(c)

Donat un valor `N`, la funció executa `N` cops el següent bloc

```
write(fd, a, strlen(a));     // línia 1
write(fd, &i, sizeof(int)); // línia 2
```

Donat que `a` és un `char *` definit com `"so2"` tenim que `strlen(a)=3` i com `sizeof(char)=1`, la primera línia escriu `3` bytes i la segona `4` bytes, en el meu cas. Sumant, l'arxiu de sortida pesarà `4*N` bytes en total.

Efectivament, si executem

```
gcc write_char_int -o write_char_int
./write_char_int data 12
```

obtenim que la mida de `data` és de `84=12*4` bytes.

En quan a les mides dels fitxes abans i després de tancar-lo són les mateixes, realitzan la mateixa comprovació que en l'apartat (a).

(d)

El programa `okteta`, pel que he vist, és equivalent a executar `hexdump data`. Es pot veure la *dump* en format hexadecimal del fitxer.

(e)

En el meu cas, el programa no peta, però imprimeix per pantalla números *random*. Obviament l'output **no és correcte**, ja que el programa `read_int.c` va llegint blocs de `sizeof(int)` bytes del fitxer d'entrada.

Donat que el fitxer d'entrada ha estat generat per `write_char_int.c`, aquest conté `N` blocs de 3 `char` + 1 `int`. En conseqüencia, el programa `read_int.c` llegeix incorrectament.

Pregunta 2

(a)

Si compilem i executem el programa per a `N=100`, havent afegit la següent línia abans de tancar el fitxer

```
fseek(fp, 0L, SEEK_END);
printf("Size of file descriptor: %ld\n", ftell(fp));
```

retorna

```
Size of file descriptor: 400
```

És a dir, la mida del file descriptor és `100*sizeof(int)=400` bytes. La mida final del fitxer és de 400 bytes, comprovant-ho amb la comanda `du -b data`.

Fent canvis, el valor abans i després de tancar el fitxer segueix coincidint. Per tant, no entenc per quina raó em dona la mida igual, tot i que l'enunciat afirma que hauria de donar diferent.

(b)

Donat els resultats obtinguts a l'apartat anterior, **si** es pot saber la mida finalla mida del fitxer és sempre `N*4` bytes.

(c)

No.

He compilat els dos programes a `read_int` i `fread_int`.

```
./read_int data > a
./read_int data > b
```

i executant

```
diff a b
```

no retorna cap resultat, per tant, els fitxers són idèntics. Dedüim d'aquest resultat que les dues funcions llegeixen els mateixos nombres.

Pregunta 3

(a)

Executant

```
gcc fwrite_int.c -o fwrite_int
./fwrite_int data 100
cat data
```

retorna caràcters extranys.

En canvi si executem `okteta data` podem veure la següent informació

```
00000000  00 00 00 00 01 00 00 00  02 00 00 00 03 00 00 00  |................|
00000010  04 00 00 00 05 00 00 00  06 00 00 00 07 00 00 00  |................|
00000020  08 00 00 00 09 00 00 00  0a 00 00 00 0b 00 00 00  |................|
```

```
00000030  0c 00 00 00 0d 00 00 00  0e 00 00 00 0f 00 00 00  |................|
00000040  10 00 00 00 11 00 00 00  12 00 00 00 13 00 00 00  |................|
00000050  14 00 00 00 15 00 00 00  16 00 00 00 17 00 00 00  |................|
00000060  18 00 00 00 19 00 00 00  1a 00 00 00 1b 00 00 00  |................|
00000070  1c 00 00 00 1d 00 00 00  1e 00 00 00 1f 00 00 00  |................|
00000080  20 00 00 00 21 00 00 00  22 00 00 00 23 00 00 00  | ...!..."...#...|
00000090  24 00 00 00 25 00 00 00  26 00 00 00 27 00 00 00  |$...%...&...'...|
000000a0  28 00 00 00 29 00 00 00  2a 00 00 00 2b 00 00 00  |(...)...*...+...|
000000b0  2c 00 00 00 2d 00 00 00  2e 00 00 00 2f 00 00 00  |,...-......./...|
000000c0  30 00 00 00 31 00 00 00  32 00 00 00 33 00 00 00  |0...1...2...3...|
000000d0  34 00 00 00 35 00 00 00  36 00 00 00 37 00 00 00  |4...5...6...7...|
000000e0  38 00 00 00 39 00 00 00  3a 00 00 00 3b 00 00 00  |8...9...:...;...|
000000f0  3c 00 00 00 3d 00 00 00  3e 00 00 00 3f 00 00 00  |<...=...>...?...|
00000100  40 00 00 00 41 00 00 00  42 00 00 00 43 00 00 00  |@...A...B...C...|
00000110  44 00 00 00 45 00 00 00  46 00 00 00 47 00 00 00  |D...E...F...G...|
00000120  48 00 00 00 49 00 00 00  4a 00 00 00 4b 00 00 00  |H...I...J...K...|
00000130  4c 00 00 00 4d 00 00 00  4e 00 00 00 4f 00 00 00  |L...M...N...O...|
00000140  50 00 00 00 51 00 00 00  52 00 00 00 53 00 00 00  |P...Q...R...S...|
00000150  54 00 00 00 55 00 00 00  56 00 00 00 57 00 00 00  |T...U...V...W...|
00000160  58 00 00 00 59 00 00 00  5a 00 00 00 5b 00 00 00  |X...Y...Z...[...|
00000170  5c 00 00 00 5d 00 00 00  5e 00 00 00 5f 00 00 00  |\...]...^..._...|
00000180  60 00 00 00 61 00 00 00  62 00 00 00 63 00 00 00  |`...a...b...c...|
00000190
```

Podem extreure d'aquí que les files contenen els nombres escrits en hexadecimal.
Per exemple, la segona columna

```
00000010  04 00 00 00 05 00 00 00  06 00 00 00 07 00 00 00  |................|
```

conté els nombres

```
4 5 6 7
```

(b)

En aquest cas, `okteta` llegeix

```
00000000  30 0a 31 0a 32 0a 33 0a  34 0a 35 0a 36 0a 37 0a  |0.1.2.3.4.5.6.7.|
00000010  38 0a 39 0a 31 30 0a 31  31 0a 31 32 0a 31 33 0a  |8.9.10.11.12.13.|
00000020  31 34 0a 31 35 0a 31 36  0a 31 37 0a 31 38 0a 31  |14.15.16.17.18.1|
00000030  39 0a 32 30 0a 32 31 0a  32 32 0a 32 33 0a 32 34  |9.20.21.22.23.24|
00000040  0a 32 35 0a 32 36 0a 32  37 0a 32 38 0a 32 39 0a  |.25.26.27.28.29.|
00000050  33 30 0a 33 31 0a 33 32  0a 33 33 0a 33 34 0a 33  |30.31.32.33.34.3|
00000060  35 0a 33 36 0a 33 37 0a  33 38 0a 33 39 0a 34 30  |5.36.37.38.39.40|
00000070  0a 34 31 0a 34 32 0a 34  33 0a 34 34 0a 34 35 0a  |.41.42.43.44.45.|
00000080  34 36 0a 34 37 0a 34 38  0a 34 39 0a 35 30 0a 35  |46.47.48.49.50.5|
00000090  31 0a 35 32 0a 35 33 0a  35 34 0a 35 35 0a 35 36  |1.52.53.54.55.56|
000000a0  0a 35 37 0a 35 38 0a 35  39 0a 36 30 0a 36 31 0a  |.57.58.59.60.61.|
```

```
000000b0  36 32 0a 36 33 0a 36 34  0a 36 35 0a 36 36 0a 36  |62.63.64.65.66.6|
000000c0  37 0a 36 38 0a 36 39 0a  37 30 0a 37 31 0a 37 32  |7.68.69.70.71.72|
000000d0  0a 37 33 0a 37 34 0a 37  35 0a 37 36 0a 37 37 0a  |.73.74.75.76.77.|
000000e0  37 38 0a 37 39 0a 38 30  0a 38 31 0a 38 32 0a 38  |78.79.80.81.82.8|
000000f0  33 0a 38 34 0a 38 35 0a  38 36 0a 38 37 0a 38 38  |3.84.85.86.87.88|
00000100  0a 38 39 0a 39 30 0a 39  31 0a 39 32 0a 39 33 0a  |.89.90.91.92.93.|
00000110  39 34 0a 39 35 0a 39 36  0a 39 37 0a 39 38 0a 39  |94.95.96.97.98.9|
00000120  39 0a                                            |9.|
00000122
```

en aquest cas, com diu l'enunciat, podem veure els nombres escrits al bloc dret, no com en el cas anterior on no hi veiem res de profit.

(c)

Si. No apareixen els nombres que ha escrit `fprintf_int`. Això es degut que la funció `fprintf(fp, "%d\n", i)` no escriu `sizeof(i)` bytes, sinó que escriu els caràcters que formen el enter `i` en format de cadena de caràcters.

(d)

En aquest cas, **si** llegim correctament els nombres. És així donat que la funció `fscanf(fp, "%d", &i)` llegeix cada línia del fitxer `data` com un `int`, tot i ser una série de caràcters.

Pregunta 4

(a)

El fitxer `mmap_read_int` extreu la mida del fitxer amb la funció `stat` i, dividint per `sizeof(int)` calcula `N`. Donat que la funció `fwrite_int` escriu directament els nombres com a `int` i no com a cadena de caràcters, la funció `mmap` llegeix els bytes obtinguts per stat en un vector `int *`. Per aquesta raó, la funció llegeix correctament el fitxer.

(b)

La raó de la diferència és que la funció `mmap_write` escriu blocs de `int` i no cadenes de caracters. Per aquesta raó, `fread_int` llegeix correctament el fitxer i `fscanf_int` no.

(c)

1. `lseek(fd, len-1, SEEK_SET)` mou el punter a la posició `len-1` del file descriptor `fd`.

2. `write(fd, "", 1)` escriu 1 byte en la posició `len` de `fd`.

Aquestes dues línies serveixen per a que després es puguin escriure els `N` nombres amb la funció `munmap(vector, len)`.

Si es comenten les dues linies, el programa deixa de funcionar: `(core dumped)`. Això és donat que s'intenta escriure en una posició de memòria fora de la reservada per `fd`.