2018

Pràctica 1

SISTEMES OPERATIUS II

DAVID DE LA PAZ / ORIOL RABASSEDA

Contenido

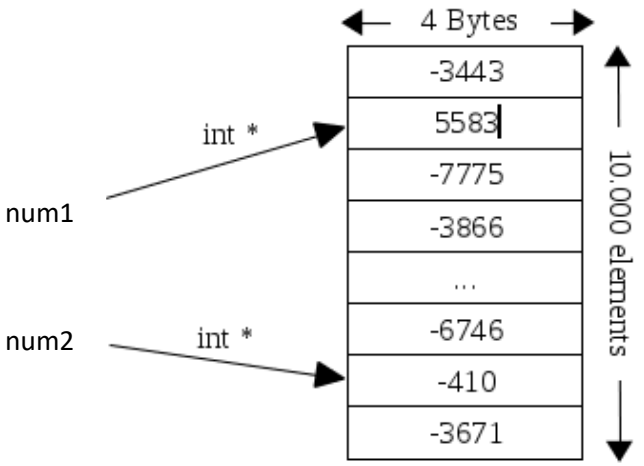
Observacions del codi

En aquest apartat s'expliquen els detalls de les implementacions del codi que no estan explicades en el fitxer de la pràctica i que tampoc s'expliquen en cap altre apartat

- En els casos on s'usa la funció *atoi* o *atof* per tal de convertir una cadena de caràcters en un nombre enter o decimal respectivament, no es necessària l'addició del caràcter '\0' abans d'aplicar la funció perquè la funció *fget* agafa el caràcter '\n' com un caràcter més, i les funcions de conversió ja el contemplen com a final de cadena.
- El nombre màxim de caràcters que poden haver-hi en una fila és fix, de manera que està inicialitzat com a constant estàtica.
- Al final del codi s'allibera tota la memòria usada amb la funció *free.*
- La funció *strcpy* s'usa per copiar el contingut d'una zona de memòria en una altre apuntada per un punter donat.
- En el casos de la ordenació per cadenes de caràcters, es necessari canviar el caràcter '\n' que indica canvi de línia (i de cadena) pel caràcter '\0' que només indica final de cadena. Si no es fa aquesta modificació, la impressió dels resultats surt defectuosa.
- La ruta cap als fitxers d'on extreuen les dades està de forma relativa, de manera que la carpeta *Dades* ha d'estar dins de la carpeta *src.*
- S'usa un bucle *while* en comptes d'un bucle *for* perquè si en algun moment de l'execució del programa falla la lectura en alguna línia s'aturi la lectura. A més, també tractem el cas que el fitxer de dades donat sigui defectuós i no hi hagi el número donat d'entrades.
- La funció *strcmp* compara dues cadenes de caràcters i retorna un enter de la manera que el necessita la funció *compara* que s'ha de passar a la funció *qsort*, encara que no siguin 1, -1 i 0.

Qüestions

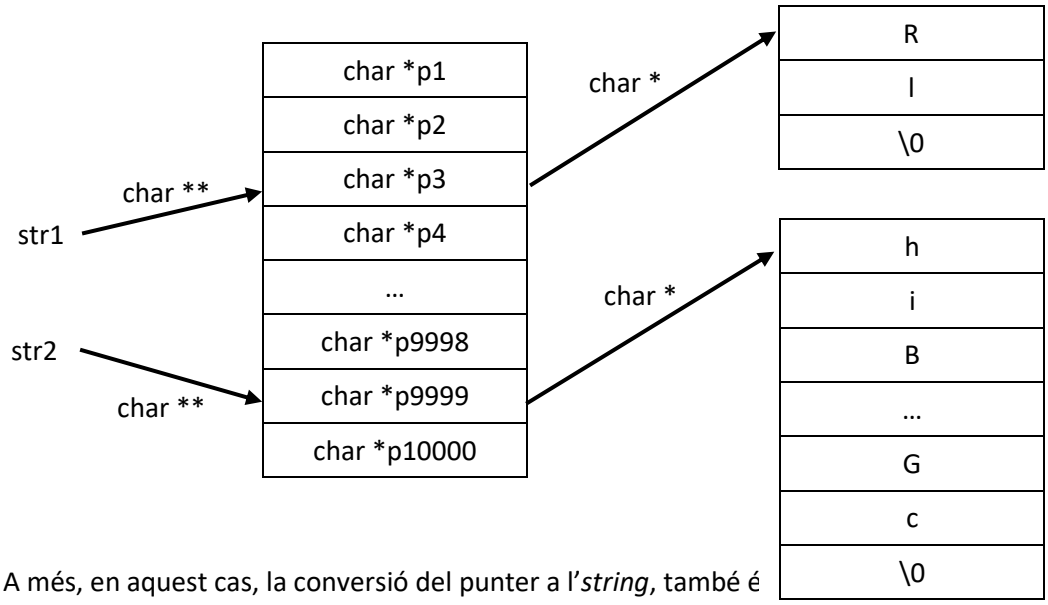
La organització de les dades a memòria és l'ocupació d'espais de memòria de 4 Bytes reservats de forma dinàmica amb un *malloc*, de manera que tenim un punter a l'inici de l'espai de memòria a usar (on hi guardarem les dades que necessitem). En el següent dibuix es veu això:



És important destacar que la funció de comparació que usa l'algorisme *qsort* ha de tenir una capçalera predefinida, de manera que sense importar quin tipus de dades siguin els subjectes de la ordenació, la capçalera és la mateixa. De manera que es necessari fer un cast pel programador que sap quin tipus de dades són, d'aquesta manera es poden comparar els enters per nombre i els *strings* per llargada (al revés no es podria).

A més, la capçalera d'aquesta funció té com a paràmetres punters i no les dades en si, de manera que cal assignar aquests punters a punters d'un tipus concret (no *void*) i no a les dades en si. Un cop fet això, per comparar els elements cal passar del punter a la dada en concret.

En el cas dels *strings*, l'emmagatzemament de dades es més complex, perquè la funció *malloc* ha de reservar espai per a un vector de punters a *char* (recordem que un *string* es un punter a un inici d'una cadena de caràcters de *char*). Per tant, cada posició del vector reservat dinàmicament serà un punter a un altre inici de vector reservat dinàmicament que contindrà una cadena de caràcters. Això es mostra en el dibuix següent:



A més, en aquest cas, la conversió del punter a l'*string*, també é nt que es passa com a paràmetre a la funció és un punter a un punter de *void,* de manera que s'ha de fer el cast a un *char ***, posteriorment, s'ha de passar a l'element en concret, un *char **.

Valgrind

En aquest apartat es realitzen les tasques demanades amb el depurador de memòria Valgrind.

- Si no alliberem memòria el Valgrind dona el següent missatge (en aquest cas amb el fitxer de *strings*):

```
==4045== LEAK SUMMARY:
==4045==    definitely lost: 80,000 bytes in 1 blocks
==4045==    indirectly lost: 108,822 bytes in 10,000 blocks
==4045==      possibly lost: 0 bytes in 0 blocks
==4045==    still reachable: 0 bytes in 0 blocks
==4045==         suppressed: 0 bytes in 0 blocks
==4045== Rerun with --leak-check=full to see details of leaked memory
==4045==
==4045== For counts of detected and suppressed errors, rerun with: -v
==4045== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

  Podem veure que hi ha 80.000 bytes en 1 vector (que és el vector principal - 10.000*8). A més, està l'esplai gastat i no alliberat de cada cadena en concret.

- Per tal d'aconseguir veure les línies del codi on s'ha guardat l'esplai que no ha estat alliberat, s'ha d'executar la comanda.

```
valgrind –leak-check=full –show-leak-kinds=all ./test
```

  Amb aquestes dues instruccions es mostra tota la informació possible sobre la no alliberació de memòria. D'aquesta manera, el resultat és:

```
==4490== HEAP SUMMARY:
==4490==     in use at exit: 188,822 bytes in 10,001 blocks
==4490==   total heap usage: 10,004 allocs, 3 frees, 269,958 bytes allocated
==4490==
==4490== 108,822 bytes in 10,000 blocks are indirectly lost in loss record 1 of 2
==4490==    at 0x4C28C20: malloc (vg_replace_malloc.c:296)
==4490==    by 0x40098F: main (qsort_strcmp.c:44)
==4490==
==4490== 188,822 (80,000 direct, 108,822 indirect) bytes in 1 blocks are definitely lost in loss record
==4490==    at 0x4C28C20: malloc (vg_replace_malloc.c:296)
==4490==    by 0x400948: main (qsort_strcmp.c:38)
==4490==
==4490== LEAK SUMMARY:
==4490==    definitely lost: 80,000 bytes in 1 blocks
==4490==    indirectly lost: 108,822 bytes in 10,000 blocks
==4490==      possibly lost: 0 bytes in 0 blocks
==4490==    still reachable: 0 bytes in 0 blocks
==4490==         suppressed: 0 bytes in 0 blocks
==4490==
==4490== For counts of detected and suppressed errors, rerun with: -v
==4490== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

  De manera que l'assignació de memòria no alliberada està en les línies 44 i 38 del codi.

- El Valgrind no és capaç de donar missatges d'error quan es tracta de memòria estàtica, però quan es tracta de memòria dinàmica, es possible que doni missatges d'error en accedir a les dades tant per llegir com per escriure. De manera que fer un *malloc* d'una mida i després intentar llegir o accedir a posicions de memòria més grans donarà els següents errors:

```
==4386== Invalid write of size 8
==4386==    at 0x400A9F: main (qsort_strcmp.c:59)
==4386==  Address 0x51f3b40 is 0 bytes after a block of size 80,000 alloc'd
==4386==    at 0x4C28C20: malloc (vg_replace_malloc.c:296)
==4386==    by 0x400988: main (qsort_strcmp.c:38)
==4386==
```

```
==4453== Invalid read of size 8
==4453==    at 0x400A74: main (qsort_strcmp.c:56)
==4453==  Address 0x51f3b40 is 0 bytes after a block of size 80,000 alloc'd
==4453==    at 0x4C28C20: malloc (vg_replace_malloc.c:296)
==4453==    by 0x400988: main (qsort_strcmp.c:38)
==4453==
```