Sistemes Operatius II - Pràctica 1

Setembre del 2017

Índex

1 Introducció

A l'assignatura de Sistemes Operatius II es realitzarà un únic projecte pràctic al llarg del curs. L'objectiu general del projecte pràctic és desenvolupar una aplicació (sense interfície gràfica) que permeti extreure i indexar les paraules contingudes en múltiples fitxers PDF. L'aplicació haurà de llegir cadascun d'aquests fitxers PDF, extreure i indexar totes les paraules que contenen aquests (pràctica 2 i 3). Per fer l'aplicació més ràpida s'aprofitaran els múltiples processadors que ens ofereixen les màquines del laboratori per tal de paral·lelitzar determinades tasques (pràctica 4 i 5).

Les pràctiques es realitzaran en llenguatge C i estaran centrades en els següents aspectes del llenguatge:

- Pràctica 1: punters a dades fent servir l'aplicació Quicksort
- Pràctica 2: extracció de les paraules dels fitxers PDF; inserció d'aquestes a un arbre binari
- Pràctica 3: emmagatzematge i lectura de l'arbre de disc
- Pràctica 4: implementació multifil de l'aplicació
- Pràctica 5: implementació multifil de l'aplicació fent servir un esquema productor-consumidor

Les pràctiques estan encavalcades entre sí. És a dir, per a realitzar una pràctica és necessari que la pràctica anterior funcioni correctament. Assegureu-vos doncs que les pràctiques estan dissenyades de forma que puguin ser ampliades de forma fàcil. A cada pràctica es donaran les pautes per aconseguir-ho.

Les pràctiques es realitzen en parella i podeu discutir amb els vostres companys la solució que implementeu. En tot cas, cada parella ha d'implementar el seu propi codi.

2 La pràctica

L'objectiu de la primera pràctica és aprendre a manipular i fer servir els punters en C. Per aconseguir aquest objectiu es proposa utilitzar la funció qsort, una funció de la llibreria estàndard que implementa el QuickSort, per tal d'ordenar diferents tipus de dades (nombres sencers, flotants, cadenes de caràcters, etc.)

Les dues subseccions següents s'han extret de la fitxa "Punters, tipus agregats i cadenes de càracters."

2.1 Els punters a funcions

La memòria RAM es pot interpretar com un vector de bytes on es guarden dades i codi executable. Els punters es fan servir moltes vegades per apuntar a dades (seners, flotants, cadenes de caràcters, etc). Però els punters també es poden fer apuntar al punt d'entrada d'una funció.

Vegem-ne un exemple, veure figura 1 (codi `punter_funcio.c`). En aquest programa declarem una variable *myfunc* que és un punter a una funció. Aquesta declaració diu que la funció admet com a paràmetre un sencer i retorna un *double*. La instrucció *myfunc = func1* fa que el la variable *myfunc* apunti a la funció *func1*. En el moment de fer *ret = myfunc(5)* cridem a la funció a la que apunta *myfunc*, que és *func1*. És a dir, que fer *myfunc(5)* és equivalent, en aquest programa, a fer *ret = func1(5)*.

Per a compilar el programa feu servir la instrucció

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

double func1(int a)
{
  return sqrt((double) a);
}

double func2(int b)
{
  return log((double) b);
}

int main(void)
{
  double ret;
  int selecciona = 1;
  double (*myfunc)(int);

  if (selecciona == 1)
    myfunc = func1;
  else
    myfunc = func2;

  ret = myfunc(5);

  printf("Resultat: %e", ret);

  return 0;
}
```

Figura 1: Punters a funcions, veure codi `punter_funcio.c`.

```
$ gcc test.c -o test -lm
```

2.2 La funció *qsort*

La llibreria estàndard de C conté, a més de les funcions típiques (*malloc*, *free*, *printf*, etc.), la funció *qsort* que permet aplicar l'algorisme de quick sort a qualsevol tipus de dades d'entrada. Si feu

```
$ man qsort
```

obtindreu ajut respecte els paràmetres que s'han de passar en aquesta funció. La pàgina del manual us indica que aquests són

```
void qsort(void *base, size_t nmemb, size_t size,
           int(*compar)(const void *, const void *));
```

El primer argument (*base*) és un punter cap a l'inici del vector que conté les dades a ordenar. Amb el segon argument (*nmemb*) indiquem el nombre d'elements que té el vector, mentre que amb el tercer (*size*) indiquem la mida de cadascun dels elements del vector. Finalment, el quart argument és un punter cap a la funció que permet comparar dos elements del vector. Aquest quart argument és una funció que l'usuari (nosaltres) ha d'escriure. En resum, la llibreria del sistema

proporciona l'algorisme de QuickSort però necessita que nosaltres li proporcionem la funció que permeti comparar dos elements qualssevol del vector. Els elements poden ser sencers, cadenes de caràcters, o inclús una estructura amb múltiples membres.

Si llegiu la pàgina del manual, us indica que "The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted vector is undefined."

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

int compara(const void *p1, const void *p2)
{
  int *num1, *num2;

  num1 = (int *) p1;
  num2 = (int *) p2;

  if (*num1 < *num2)
    return -1;
  if (*num1 > *num2)
    return 1;
  return 0;
}

int main(void)
{
  int i;
  int vector[8] = {8, 4, 2, 3, 5, 6, 8, 5};

  qsort(vector, 8, sizeof(int), compara);

  printf("El vector ordenat és ");

  for(i = 0; i < 8; i++)
    printf("%d ", vector[i]);

  printf("\n");

  return 0;
}

```

Figura 2: Exemple d'ús de la funció *qsort* amb sencers, codi `quicksort_sencers.c`.

Anem a veure aquí com es pot aplicar la funció *qsort* per a ordenar un vector de sencers. Per simplificar, farem servir memòria estàtica (codi `quicksort_sencers.c`), veure figura 2. Analitzeu els paràmetres que se li passen a la funció *qsort*. De tots els paràmetres el darrer és el que interessa aquí: és un punter a una funció que permet comparar dos elements del vector. Cada cop que l'algorisme de *qsort* necessiti comparar dos elements, cridarà a la funció *compara* i li passarà per argument els dos elements a comparar.

Analitzem la funció de comparació *compara*, veure figura 3. La funció obté, per paràmetre, dos punters genèrics, p1 i p2 de tipus *void*, cap als elements que s'han de comparar. Un punter genèric és un punter a una direcció de memòria sense especificar el tipus de dades que s'hi emmagatzema. La

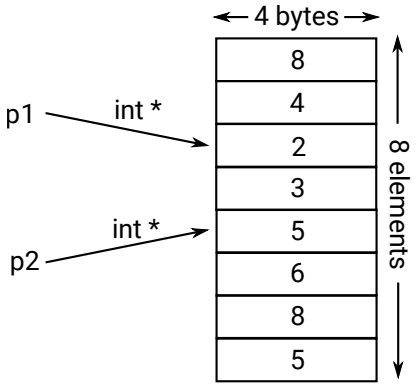

Figura 3: Representació gràfica de les dades en cas que siguin un vector de sencers.

funció *qsort* no sap pas si estem comparant sencers, *double* o algun tipus d'estructura més complexa. El que sap *qsort* és que cada element ocupa size bytes (3r argument) i ens passa dos punters genèrics als elements a comprar. Nosaltres, com a programadors de la funció, sabem que els elements que es comparen són sencers. Sabem que els dos índexos de memòria que obté la funció *compara* són en realitat punters a sencers. És per això que es fa el *casting* a les variables *num1* i *num2*. La funció *compara* ha de retornar un -1,0 o 1 segons si *num1* és inferior, igual o superior, respectivament, a *num2*.

2.3 Feina a realitzar

Es demana implementar, per a cadascun dels següents casos, la funció de comparació equivalent a *compara* perquè permeti ordenar els següents elements

1. Nombres sencers (int) en ordre invers (de major a menor)
2. Nombres flotants (float)
3. Nombres flotants de doble precisió (double)
4. Cadenes de caràcters ordenats per la longitud de la cadena (funció *strlen*)
5. Cadenes de caràcters ordenats pel seu contingut (funció *strcmp*)
6. Cadenes de caràcters ordenats per la darrera lletra de la cadena (cal tenir en compte el detall del '\n' explicat a la secció 3.1)

3 Implementació i planificació

Per tal d'assolir amb èxit aquesta pràctica es recomana revisar abans de tot la Fitxa 1 (recordatori bàsic del llenguatge C) que fa un repàs als punters.

3.1 Lectura de dades de fitxer

A l'exemple 2 les dades a ordenar estan declarades a una variable de forma estàtica. Per a la implementació d'aquesta pràctica les dades a ordenar estan emmagatzemats en fitxers. Es proporciona un fitxer de nombres sencers, un de nombre flotants, un de nombre flotants de doble precisió i una de cadenes de caràcters. Les dades estan en línies diferents i es demana procedir de la següent forma

```c
#include <stdio.h>

int main()
{
   FILE *fp;
   char str[100];

   /* opening file for reading */
   fp = fopen("file.txt" , "r");
   if(fp == NULL)
   {
      perror("Error opening file");
      return(-1);
   }
   if( fgets (str, 100, fp)!=NULL )
   {
      /* writing content to stdout */
      puts(str);
   }
   fclose(fp);

   return(0);
}
```

Figura 4: Exemple d'ús de la funció fgets.

- Utilitzar la funció *fgets* per llegir les dades de disc. No es pot fer servir la funció *fscanf* atès que, en general, és complicat detectar si es produeix algun problema a l'hora de llegir la dada de disc. La declaració de la funció *fgets* és

  `char *fgets(char *string, int n, FILE *fp)`

  El primer argument (*string*) és la cadena on s'emmagtzemarà les dades llegides del fitxer especificat al tercer argument (*fp*). El segon argument (*n*) especifica el nombre màxim de bytes a llegir. En concret, el manual especifica que es llegeixen un màxim de *n-1* bytes atès que cal assegurar que hi ha el byte 0 al final de la cadena, veure fitxa 1. A la figura 4 en mostra un exemple d'ús. Per a la realització d'aquesta pràctica, podeu suposar que a cada línia hi ha menys de 100 caràcters.

- Abans de cridar a la funció *qsort*, caldrà transformar les dades i emmagatzemar-les en un vector del tipus corresponent (sencers, flotant, doble precisió o cadena). La mida del vector s'especifica a la primera línia del fitxer a llegir, veure fitxers proporcionats. El vector s'haurà de reservar amb la funció *malloc*. No es pot reservar el vector de forma estàtica tal com es fa l'exemple 2.

- Podeu utilitzar les funcions *atoi* i *atof* per convertir una cadena de caràcter a sencer o flotant (precisió simple o doble).

- Per trebaballar amb les cadenes de caràcters es demana tenir en compte que a) La funció *fgets* inclou a la variable string el caràcter de final de línia, '\n'. Es demana eliminar aquest caràcter de la cadena, b) Emmagatzemar cadascuna de les cadenes al vector amb el nombre mínim necessari de bytes per fer-ho – la funció *strlen* permet saber la longitud de la cadena–, c) La funció *strcmp* permet comparar dues cadenes. Aquesta funció és la que permetrà implementar la ordenació de cadenes pel seu contingut.

3.2 Implementació

Es proposa aquí una planificació per a realitzar la implementació dels algorismes d'ordenació. La idea es basa en implementar la funcionalitat assegurant que cadascuna de les parts funciona correctament.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4
 5 int compara(const void *p1, const void *p2)
 6 {
 7   int value;
 8   char *str1, *str2;
 9
10   str1 = *((char **) p1);
11   str2 = *((char **) p2);
12
13   value = // el que sigui
14
15   return value;
16 }
17
18 int main(void)
19 {
20   int i;
21   char *vector[8] = {"hola", "sistemes", "operatius", "dos", "cadena",
22                      "quicksort", "canonada", "compara"};
23
24   qsort(vector, 8, sizeof(char *), compara);
25
26   printf("El vector ordenat és\n");
27
28   for(i = 0; i < 8; i++)
29     printf("%s\n", vector[i]);
30
31   printf("\n");
32
33   return 0;
34 }
35
```

Figura 5: Exemple d'ús de la funció qsort fent servir cadenes de caràcters.

1. Començar per una implementació de tots els algorismes fent servir dades estàtiques, tal com es mostra a la figura 2. A la figura 5 es mostra l'ús de la funció *qsort* amb cadenes de caràcters. Observar les línies 10 i 11 del codi que accedeixen a les dades a comparar.

2. Un cop finalitzat el punt 1, ampliar l'aplicació perquè les dades d'entrada siguin de fitxer, veure secció 3.1.

4 Entrega

El fitxer que entregueu s'ha d'anomenar `P1_NomCognom1NomCognom2.tar.gz` (o `.zip`, o `.rar`, etc), on `NomCognom1` és el nom i cognom del primer component de la parella i `NomCognom2` és el cognom del segon component de la parella de pràctiques. El fitxer pot estar comprimit amb qualsevol dels formats usuals (`tar.gz`, `zip`, `rar`, etc). Dintre d'aquest fitxer hi haurà d'haver dues carpetes: `src`, que contindrà el codi font, i `doc`, que contindrà la documentació addicional en PDF. Aquí hi ha els detalls del contingut des cada directori:

- La carpeta `src` contindrà els 6 codi font demanats a 2.3. S'haurà d'incloure un makefile o un script que permeti generar els executables corresponents.
- El directori `doc` ha de contenir un document – tres o quatre pàgines, màxim cinc pàgines, en format PDF, sense incloure la portada – explicant els següents aspectes a) Explicar, ajudant-se d'un dibuix, de la organització de la informació pel cas de les dades de tipus sencers, veure figura 3, i per què cal accedir a les dades tal com s'indica a les línies 8 i 9 del codi 2, b) Explicar, ajudant-se d'un dibuix, de la organització de la informació pel cas de les cadenes de caràcters i per què cal accedir a les dades tal com s'indica a les línies 10 i 11 del codi 5, c) Suposem que les dades són de tipus sencer, codi 2. Què passa en cas que el *casting* no es faci de forma correcta? És a dir, què passa si fem un *cast* a tipus flotant o doble precisió? Explica la teva experiència.

La data límit d'entrega està indicat al document de planificació. El codi té un pes d'un **50%** i el document un pes del **50%**.