Pràctiques de Sistemes Operatius II

Lluís Garrido i Òscar Amorós

Grau d'Enginyeria Informàtica

Informació general

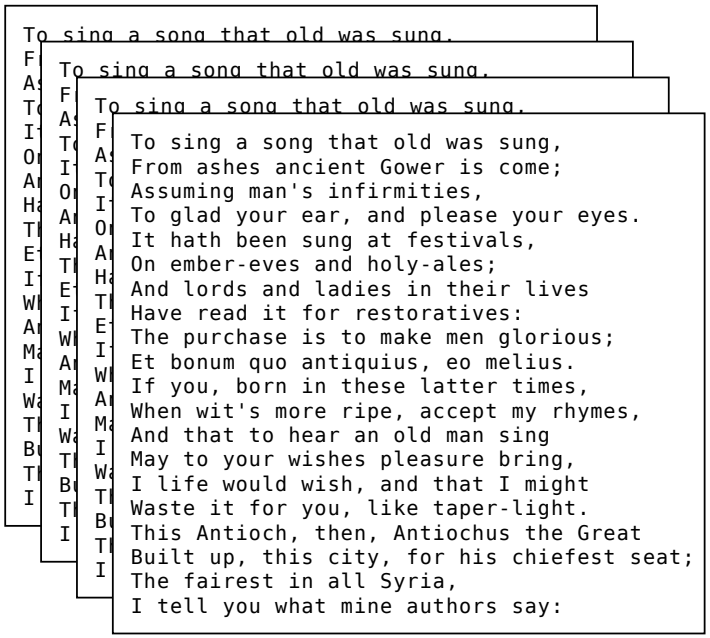
- Laboratori: dijous de 17h a 19h (aula IF i IA) i de 19h a 21h (aula IA)
  - En parelles de dues persones
  - Programació en llenguatge C
  - Una únic projecte pràctic
  - Aprofiteu les sessions!!!
- Professorat
  - Lluís Garrido (teoria i pràctiques, lluis.garrido@ub.edu)
  - Òscar Amorós (pràctiques, oscaramoros@ub.edu)
- La setmana que ve es passarà un full per recollir informació de les parelles.

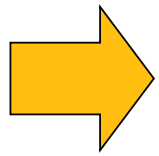
Informació general

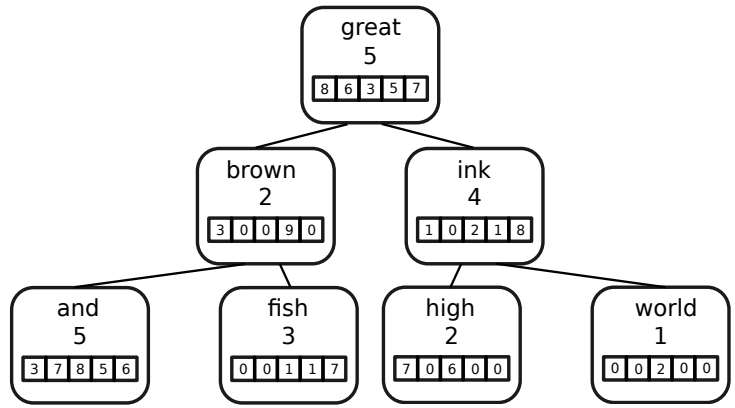
- Quatre entregues de laboratori (LP). Entrega via campus.
  - Pràctica 1 (diumenge 5 d'octubre)
  - Pràctica 2 (diumenge 26 d'octubre)
  - Pràctica 3 (diumenge 16 de novembre)
  - Pràctica 4 (diumenge 21 de desembre)
- Activitats Presencials (AP). Proves de seguiment al laboratori.
  - Petites proves test (màxim 5 preguntes) sobre la pràctica realitzada.
  - Es realitza el dijous següent a l'entrega de la pràctica (a excepció de la pràctica 4, per a la qual es realitza el dijous abans).

Projecte pràctic

- Fil conductor de les quatre pràctiques: aplicació que indexi i processi les paraules de múltiples fitxers de text.
  - Pràctica 1: manipulació de cadenes de caràcters i lectura de dades de fitxers.
  - Pràctica 2: punters i estructures de dades.
  - Pràctica 3: comunicació interprocés mitjançant canonades.
  - Pràctica 4: programació multifil.







Pràctica 1

- Objectius de la pràctica 1:
  1. Aprendre les diverses tècniques amb què es poden llegir dades d'un fitxer (de text pla). Es pretén que analitzeu les avantatges i desavantatges de cadascuna de les tècniques per tal d'escollir la que millor us convingui per a la implementació del segon punt.
  2. Implementar un algorisme que permeti extreure les paraules que hi ha en un únic fitxer de text pla (el problema d'analitzar múltiples fitxers de text es deixarà per a la segona pràctica).

Pràctica 1: lectura de dades d'un fitxer

Un fitxer (de text pla) es pot llegir

- Byte a byte amb la funció *fgetc*.
- "Element a element" amb *fscanf*.
- Línia a línia amb la funció *fgets*.
- Tot de cop amb la funció *fread*.

Teniu tota la informació, juntament amb codis d'exemple, a l'enunciat.

Pràctica 1: extracció de paraules

Cal extreure les paraules del fitxer de text pla.

```
To sing a song that old was sung,
From ashes ancient Gower is come;
Assuming man's infirmities,
To glad your ear, and please your eyes.
It hath been sung at festivals,
On ember-eves and holy-ales;
And lords and ladies in their lives
Have read it for restoratives:
```

A l'exemple indicat a dalt, a la 3a línia cal extraure "Assuming", "man's" i "infirmities". Tots els detalls del que es considera una paraula són a l'enunciat.

Pràctica 1: execució

- L'aplicació ha de funcionar per línia de comandes

  ```
  $ practica1 fitxer.txt
  ```

  on `fitxer.txt` és el fitxer de text pla de la qual s'han d'extreure les paraules.

- L'aplicació ha d'imprimir per pantalla les paraules vàlides i no vàlides que troba a mesura que llegeix el fitxer. Per exemple,

  ```
  Paraula valida:  assuming
  Paraula valida:  man's
  Paraula valida:  infirmities
  ```

Pràctica 1: l'enunciat i les fitxes

- L'enunciat conté tots els detalls necessaris per realitzar la pràctica
  - Les diverses formes amb què es pot llegir un fitxer
  - Com s'han de separar les paraules.
  - I més...
- Les fitxes
  - Són una guia de programació en C. Estan disponibles al campus. Aprofiteu-los!
  - A la primera fitxa es repassen els conceptes bàsics de programació en llenguatge C, en particular l'ús de punters per a vectors i matrius, així com exemples de manipulació de cadenes de caràcters.
  - A la segona fitxa es detallen les funcions que es poden fer sevir per llegir dades d'un fitxer.

Pràctica 1: entrega

- Què s'ha d'entregar ?
  - Document que comenti, entre altres, les proves demanades.
  - Codi font amb les funcions comentades.
- Data límit: diumenge 5 d'octubre, amb prova presencial el dijous següent.
- Proves
  - Execució normal redirigint la sortida a un fitxer.
  - Execució amb aplicació *valgrind*. (Que és això ?)

Aplicació *valgrind: motivació*

- Què passarà amb el següent exemple (codi `exemple1.c`).
  - Compilarà ?
  - Si compila, quins dels missatges imprimirà per pantalla en executar ?

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

int main(void)
{
  int *a;

  a = (int *) malloc(sizeof(int) * 10);

  printf("Abans assignacio\n");
  a[15] = 15;
  printf("Despres assignacio\n");

  printf("Valor a[15]: %d\n", a[15]);

  free(a);
}
```

Aplicació *valgrind: motivació*

- Proveu-ho vosaltres mateixos:

```
$ gcc exemple1.c -o exemple
$ ./exemple
Abans assignacio
Despres assignacio
Valor a[15]:  15
$
```

- Per què no "peta" el programa ?

Aplicació *valgrind: motivació*

- En el llenguatge C
  - No es comprova, en temps d'execució, si els accessos a vectors són dintre del rang correcte. Python i Java sí que ho fan: això pot alentir l'aplicació.
  - En escriure fora de la memòria reservada potser estem sobreescrivint el valor d'altres variables. No ens fins més "tard" en l'execució que ens adonem que alguna cosa falla.
- Però com detectar fàcilment en quin punt estem fent accessos invàlids ?

Aplicació *valgrind*: exemples d'ús

- Per utilitzar *valgrind* és important compilar el codi amb mode *debug*

  ```
  $ gcc -g exemple1.c -o exemple
  ```

  aquesta opció inclou informació necessària a l'executable.

- Un cop compilat cal executar l'aplicació així

  ```
  $ valgrind ./exemple
  ```

  valgrind comprova, en temps d'execució, si els accessos a memòria es realitzen correctament.

Aplicació *valgrind*: exemples d'ús

```
$ valgrind ./exemple
==3657== Memcheck, a memory error detector
==3657== Copyright (C) 2002 2011, and GNU GPL'd, by Julian Seward et al.
==3657== Using Valgrind 3.7.0 and LibVEX; rerun with  h for copyright info
==3657== Command: ./exemple
==3657==
Abans assignacio
==3657== Invalid write of size 4
==3657==    at 0x400644: main (exemple1.c:11)
==3657==  Address 0x51d607c is not stack'd, malloc'd or (recently) free'd
==3657==
Despres assignacio
==3657== Invalid read of size 4
==3657==    at 0x40065C: main (exemple1.c:14)
==3657==  Address 0x51d607c is not stack'd, malloc'd or (recently) free'd
==3657==
Valor a[15]: 15
==3657==
==3657== HEAP SUMMARY:
==3657==     in use at exit: 0 bytes in 0 blocks
==3657==   total heap usage: 1 allocs, 1 frees, 40 bytes allocated
==3657==
==3657== All heap blocks were freed    no leaks are possible
==3657==
==3657== For counts of detected and suppressed errors, rerun with:  v
==3657== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 2 from 2)
$
```

Aplicació *valgrind*: exemples d'ús

- El resultat anterior ens indica que
  - Hi ha una escriptura no vàlida a la línia 11 del codi.
  - Hi ha una lectura no vàlida a la línia 14 del codi.
- L'aplicació valgrind detecta els accessos no vàlids a memòria dinàmica, però no a la pila! Vegem un exemple...

Aplicació *valgrind*: exemples d'ús

- Observeu aquest exemple (codi `exemple2.c`)...

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

int main(void)
{
  int a[10];

  printf("Abans assignacio\n");
  a[15] = 15;
  printf("Despres assignacio\n");

  printf("Valor a[15]: %d\n", a[15]);
}
```

- Què passa en executar aquesta aplicació normalment ?

Aplicació *valgrind*: exemples d'ús

- Execució de l'aplicació

```
$ gcc exemple2.c -o exemple2
$ ./exemple2
Abans assignacio
Despres assignacio
Valor a[15]:  15
Violació de segment
$
```

  Què està passant ?

- L'execució del main es realitza correctament, però en retornar el main ho fa fent servir la informació guardada a la pila. I hem sobreescrit part de la informació!

Aplicació *valgrind*: exemples d'ús

```
$ valgrind ./exemple2
==4244== Memcheck, a memory error detector
==4244== Copyright (C) 2002 2011, and GNU GPL'd, by Julian Seward et al.
==4244== Using Valgrind 3.7.0 and LibVEX; rerun with  h for copyright info
==4244== Command: ./exemple2
==4244==
Abans assignacio
Despres assignacio
Valor a[15]: 15
==4244== Jump to the invalid address stated on the next line
==4244==    at 0xF04E52455: ???
==4244==  Address 0xf04e52455 is not stack'd, malloc'd or (recently) free'd
==4244==
==4244==
==4244== Process terminating with default action of signal 11 (SIGSEGV)
==4244==  Bad permissions for mapped region at address 0xF04E52455
==4244==    at 0xF04E52455: ???
==4244==
==4244== HEAP SUMMARY:
==4244==     in use at exit: 0 bytes in 0 blocks
==4244==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==4244==
==4244== All heap blocks were freed    no leaks are possible
==4244==
==4244== For counts of detected and suppressed errors, rerun with:  v
==4244== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 2 from 2)
$
```

Aplicació *valgrind*: exemples d'ús

- Observeu que *valgrind* no detecta l'accés invàlid a la pila.
- És recomanable utilitzar la memòria dinàmica (*malloc* i free) perquè valgrind pugui detectar els problemes d'accessos invàlids a memòria.

- Quins altres problemes pot detectar *valgrind* ? La falta d'alliberaments de memòria!

Aplicació *valgrind*: exemples d'ús

- Exemple on no s'allibera la memòria (exemple3.c).

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

int main(void)
{
  int *a, i;

  a = (int *) malloc(sizeof(int) * 10);

  for(i = 0; i < 10; i++)
    a[i] = 0;

  // No alliberem la memoria
}
```

- És important alliberar la memòria ?
  - En sortir d'un procés tota la memòria ubicada s'allibera automàticament pel sistema operatiu.
  - En un algorisme iteratiu, però, si ubiquem però no alliberem podem alentir l'ordinador.

Aplicació *valgrind*: exemples d'ús

```
$ valgrind ./exemple3
==4971== Memcheck, a memory error detector
==4971== Copyright (C) 2002 2011, and GNU GPL'd, by Julian Seward et al.
==4971== Using Valgrind 3.7.0 and LibVEX; rerun with  h for copyright info
==4971== Command: ./exemple3
==4971==
==4971==
==4971== HEAP SUMMARY:
==4971==     in use at exit: 40 bytes in 1 blocks
==4971==   total heap usage: 1 allocs, 0 frees, 40 bytes allocated
==4971==
==4971== LEAK SUMMARY:
==4971==    definitely lost: 40 bytes in 1 blocks
==4971==    indirectly lost: 0 bytes in 0 blocks
==4971==      possibly lost: 0 bytes in 0 blocks
==4971==    still reachable: 0 bytes in 0 blocks
==4971==         suppressed: 0 bytes in 0 blocks
==4971== Rerun with  --leak-check=full to see details of leaked memory
==4971==
==4971== For counts of detected and suppressed errors, rerun with:  v
==4971== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 2 from 2)
$
```

Aplicació *valgrind*: exemples d'ús

```
$ valgrind --leak-check=full ./exemple3
==5179== Memcheck, a memory error detector
==5179== Copyright (C) 2002 2011, and GNU GPL'd, by Julian Seward et al.
==5179== Using Valgrind 3.7.0 and LibVEX; rerun with  h for copyright info
==5179== Command: ./exemple3
==5179==
==5179==
==5179== HEAP SUMMARY:
==5179==     in use at exit: 40 bytes in 1 blocks
==5179==   total heap usage: 1 allocs, 0 frees, 40 bytes allocated
==5179==
==5179== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==5179==    at 0x4C2ABED: malloc (in /.../vgpreload_memcheck amd64 linux.so)
==5179==    by 0x40054D: main (exemple3.c:8)
==5179==
==5179== LEAK SUMMARY:
==5179==    definitely lost: 40 bytes in 1 blocks
==5179==    indirectly lost: 0 bytes in 0 blocks
==5179==      possibly lost: 0 bytes in 0 blocks
==5179==    still reachable: 0 bytes in 0 blocks
==5179==         suppressed: 0 bytes in 0 blocks
==5179==
==5179== For counts of detected and suppressed errors, rerun with:  v
==5179== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 2 from 2)
$
```

Aplicació *valgrind*: conclusions

- *Valgrind* ens informa de:
  - Accessos no vàlids (lectura o escriptura) en memòria dinàmica
  - Operacions amb valors no inicialitzats a memòria dinàmica
  - Memòria no alliberada
- Heu de provar la vostra aplicació amb *valgrind*: nosaltres ho farem!

Pràctica 1: manipulació de cadenes

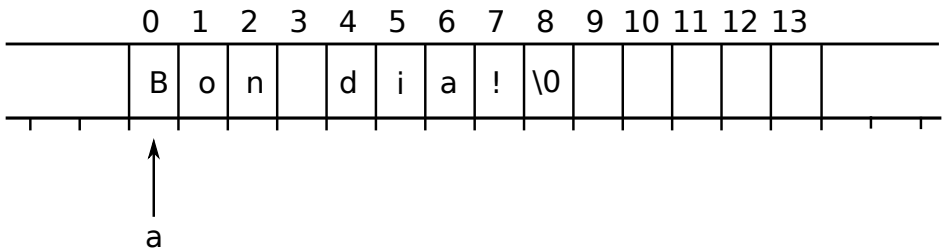
- La pràctica requereix manipular cadenes de caràcters.
- En C les cadenes de caràcters funcionen de forma diferent a les cadenes del llenguatges com Python o Java.
- En particular
  - Tota cadena ha d'estar finalitzada amb el byte zero (caràcter '\0'). Moltes funcions de la llibreria C ho assumeixen per funcionar correctament.
  - Una cadena no és un "objecte" i cal anar amb compte amb certs tipus de manipulacions (exemples a les següents transparències).

```c
#include <stdio.h>

int main(void)
{
  char a[14] = "Bon dia!";

  printf("%s", a);

  return 0;
}
```



Pràctica 1: manipulació de cadenes

- Aquest codi no funcionarà (però compila!). Per què ?

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

char *genera_cadena()
{
  char str[100];

  // Processa

  return str;
}


int main(void)
{
  char *cadena;

  cadena = genera_cadena();

  // Processa
}
```

- En retornar de `genera_cadena` la cadena `str` es "destrueix".

Pràctica 1: manipulació de cadenes

- La forma correcta d'implementar la funcionalitat anterior és

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

char *genera_cadena()
{
  char *str = malloc(100 * sizeof(char));

  // Processa

  return str;
}


int main(void)
{
  char *cadena;

  cadena = genera_cadena();

  free(cadena);
}
```

- Observar que la cadena s'ubica i s'allibera en dues funcions diferents.

Pràctica 1: manipulació de cadenes

- Altres formes d'implementar-ho són aquestes

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

void genera_cadena(char *str)
{
  // Processa

}


int main(void)
{
  char *cadena;

  cadena = malloc(100 * sizeof(char));

  genera_cadena(cadena);

  free(cadena);
}
```

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

void genera_cadena(char *str)
{
  // Processa

}


int main(void)
{
  char cadena[100];


  genera_cadena(cadena);


}
```

Pràctica 1: manipulacions de cadenes

- Com s'inicialitza de forma correcta una cadena ?

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

int main(void)
{
  char *cadena;

  cadena = malloc(100 * sizeof(char));

  // No es la forma correcta d'inicialitzar!
  cadena = "";

  // La forma correcta d'inicialitzar es
  strcpy(cadena, "");

  // O be tambe
  cadena[0] = '\0';

}
```

Pràctica 1: ara vosaltres!

- En el temps que queda avui
  - Proveu els diversos exemples per llegir un fitxer (codi disponible a l'enunciat).
- Per la setmana que ve
  - Llegiu la fitxa 1 i fitxa 2.
  - Llegiu l'enunciat.