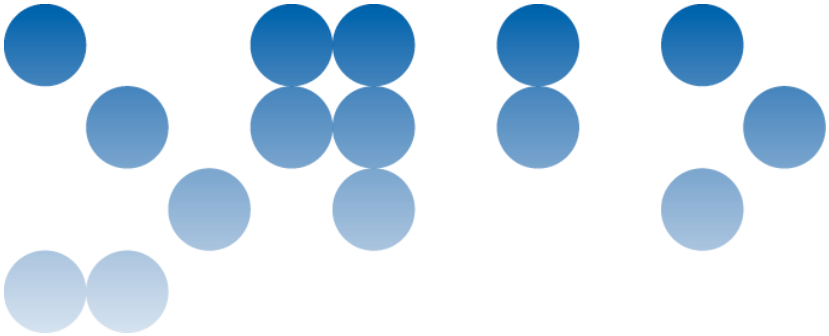


PRA1
Introducció a la programació en Linux

Sistemes Operatius

Programa

2019-01

Estudis d'Informàtica, Multimèdia i Telecomunicació

Presentació

Aquesta pràctica planteja un seguit d'activitats amb l'objectiu que l'estudiant pugui aplicar sobre un sistema Unix alguns dels conceptes introduïts als primers mòduls de l'assignatura.

L'estudiant haurà de realitzar un seguit d'experiments i respondre les preguntes plantejades. També haurà d'escriure un petit programa en llenguatge C.

La pràctica es pot desenvolupar sobre qualsevol sistema Unix (la UOC us facilita la distribució Ubuntu 14.04). S'aconsella que mentre feu els experiments no hi hagi altres usuaris treballant al sistema perquè el resultat d'alguns experiments pot dependre de la càrrega del sistema.

Com a referència, a cada pregunta us aconsellem una possible temporització per tal de poder acabar la practica abans de la data límit. També s'indica el pes de cada pregunta a l'avaluació final de la pràctica.

El pes d'aquesta pràctica sobre la nota final de pràctiques és del 40%.

Competències

Transversals:

- Capacitat per a adaptar-se a les tecnologies i als futurs entorns actualitzant les competències professionals

Especifiques:

- Capacitat per a analitzar un problema en el nivell d'abstracció adequat a cada situació i aplicar les habilitats i coneixements adquirits per a abordar-lo i resoldre'l
- Capacitat per a dissenyar i construir aplicacions informàtiques mitjançant tècniques de desenvolupament, integració i reutilització

Competències pròpies:

- L'ús i aplicació de les TIC en l'àmbit acadèmic i professional

Objectius

Aquesta pràctica té com a objectiu que l'alumne apliqui els conceptes introduïts als 4 primers mòduls de l'assignatura sobre un sistema Unix i que s'introdueixi de forma progressiva en la programació, depuració i execució d'aplicacions en els sistemes Unix.

Restriccions en l'ús de funcions de C

Per realitzar la pràctica heu d'utilitzar les crides a sistema d'Unix. **No es podran utilitzar funcions de C d'entrada/sortida** (getc, scanf, printf, ...), en el seu lloc s'utilitzaran les crides a sistema read i write. Sí que podeu utilitzar funcions de C per al formatatge de cadenes (sprintf, sscanf, ...).

Enunciat

A aquesta pràctica haureu de realitzar un seguit d'experiments i respondre justificadament les preguntes plantejades. Podeu fer els experiments sobre qualsevol sistema Unix al que tingueu accés. Ara be, s'aconsella que en aquell moment no hi hagi altres usuaris treballant al sistema.

Us facilitem el fitxer pr1so.zip amb una sèrie de fitxers que us seran útils per fer la pràctica. Descompacteu-lo executant la comanda unzip pr1so.zip. Per compilar un fitxer, per exemple prog.c, cal executar la comanda: gcc -o prog prog.c i per a executar el programa cal fer ./prog

1: Mòdul 2 [Del 4 al 10 d'octubre] (5%+10%+10%+10%=35%)

En aquest primer exercici anem a analitzar el comportament durant la seva execució d'un programa en un sistema Unix . Per a això, us incloem una sèrie de programes en c que haureu de compilar i executar seguint les instruccions de l'exercici. En concret, realitzarem les proves amb un petit programa que implementa la multiplicació de dues matrius.

Per analitzar el comportament d'aquests programes, utilitzarem la comanda ***/usr/bin/time*** que mostra diverses estadístiques d'utilització de recursos per part de l'aplicació i el temps requerit per la seva execució. A la figura 1, teniu un exemple de la utilització d'aquesta comanda i de les estadístiques que mostra. En aquest exemple l'aplicació que es monitoritza és la comanda de ***sleep***, que únicament realitza un retard del nombre de segons especificats com a argument. Fixeu-vos que a la comanda ***/usr/bin/time*** se li passa l'opció -v per mostrar totes les estadístiques i que s'ha d'especificar el camí complet de la comanda.

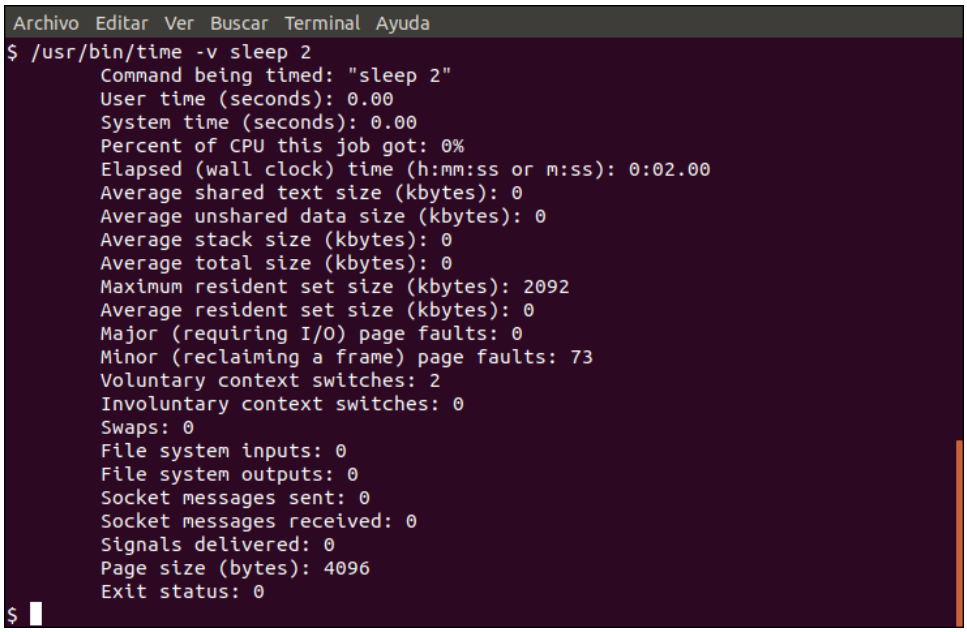
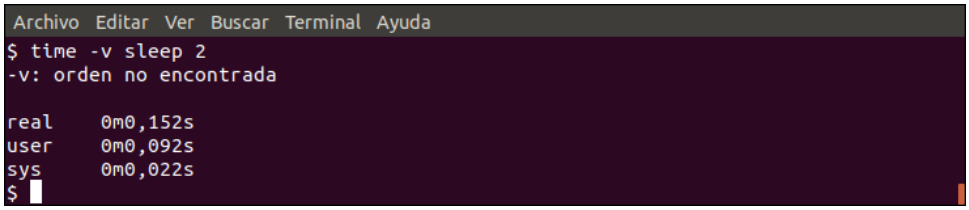


Figura 1. Exemple utilització comanda /usr/bin/time

1.1. Executar la comanda "***time***" sense especificar el camí complet. Annexar la captura de pantalla (screen-shot) del resultat obtingut. S'obté el mateix resultat que el mostrat en la figura 1? En cas de resposta negativa, indicar a que és degut l'obtenció d'un resultat diferent.Consulteu la secció "Shell Builtin Commands" de l'ajuda de l'intèrpret d'ordres bash per respondre a aquesta pregunta (*man bash*).



S'obté un resultat diferent perquè s'està executant una comanda diferent. Existeix una comanda interna del bash (com les comandes ***pwd***, ***history***, ***cd***, ...) que té el mateix nom i que mostra el temps d'execució d'una aplicació. Per tant, si no s'específica la ruta absoluta, l'interpret de comandes entén que volem invocar a la comanda interna ***time***.

1.2. Us facilitem el programa sum1.c, per calcular el sumatori d'un nombre constant N (per defecte, 100.000.000). El codi utilitzat no és representatiu del càlcul eficient d'aquest sumatori (segur que coneixeu mètodes més eficients per calcular la suma dels termes d'una progressió aritmètica), però ens permetrà realitzar diferents proves i anàlisis de la seva execució.

Executeu aquest programa, juntament amb la comanda ***/usr/bin/time –v*** (en la resta de subapartats també caldrà executar els diferents programes utilitzant aquesta comanda). Annexeu una captura de pantalla de les estadístiques d'execució obtingudes. Interpreteu el significat de les següents estadístiques, relacionant-les amb els conceptes estudiats a l'assignatura: "Command being timed", "User time", "System time", "Percent of CPU this job got", "Elapsed (wall clock) time", "Maximum resident set size", "Average resident set size", "Major (requiring I/O) page faults", "Minor (Reclaiming a frame) page faults", "Page size", i "Exit status".

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
$ /usr/bin/time -v ./sum1
5000000050000000
        Command being timed: "./sum1"
        User time (seconds): 0.27
        System time (seconds): 0.68
        Percent of CPU this job got: 93%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:01.02
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 782856
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 195373
        Voluntary context switches: 10
        Involuntary context switches: 64
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
$
```

La comanda ***/usr/bin/time*** mostra la següent informació referent a l'execució del procés :

- **Command being timed** (Comando executat): Indica el programa que s'ha executat, amb els arguments que li han passat.
- ***User time*** (Temps d'execució del procés en mode usuari): Nombre total de segons de CPU que el procés ha consumit en mode usuari.
- ***System time*** (Temps d'execució del procés en mode sistema): Nombre total de segons de CPU que el procés ha consumit en mode sistema.
- ***Percent of CPU this job got*** (Percentatge de CPU consumit pel procés): Es calcula com (Temps CPU mode Usuari + Temps CPU mode Sistema) / Temps Real Transcorregut.

- ***Elapsed (wall clock) time*** : Temps real d'execució, en hores, minuts i segons.
- ***Maximum resident set size***: Mida màxima de memòria resident del procés durant el seu temps de vida, en Kbytes.
- ***Average resident set size***: Mida mitjana de memòria resident del procés durant el seu temps de vida, en Kbytes.
- ***Major (requiring I/O) page faults***: Nombre de fallades principals de pàgina, que van ocórrer mentre el procés s'executava. Aquestes fallades de pàgines requereixen que la pàgina sigui llegida del disc i portada a memòria.
- ***Minor (Reclaiming a frame) page faults***: Nombre de fallades de pàgina menors o recuperables, que van ocórrer mentre el procés s'executava. Aquests errors fan referència a pàgines no vàlides però que encara no han estat reclamades per altres pàgines virtuals (no cal portar-les del disc, ja que les dades a la pàgina són encara vàlides).
- ***Page size*** (bytes): Mida de pàgina del sistema, en Kbytes. És una constant del sistema, però pot variar entre un sistema i un altre o d'una arquitectura a una altra.
- ***Exit status***: Codi de finalització del procés.

A partir de la informació mostrada, respondre a les següents preguntes:

a) Quin sistema de gestió de memòria dels utilitzats en teoria s'està utilitzant?

Paginació sota demanda, com queda constància en el nombre de fallades de pàgina.

b) Quin percentatge del temps està el programa executant crides al sistema?

En aquest cas el percentatge és mínim, ja que el temps d'execució en mode sistema (per processar les crides al sistema) és de 0.00 segons.

c) Quines crides al sistema invoca el programa durant la seva execució?

El programa invoca a la crida al sistema ***write*** per escriure el resultat per pantalla i a la crida al sistema ***exit*** per finalitzar el programa.

d) Quin és el resultat mostrat per pantalla pel programa?

El resultat retornat és 5000000050000000, la resta són estadístiques generades per la comanda time.

e) Què codi d'acabament retorna el programa?

El codi de retorn del programa és un 0, indicant que tot ha funcionat correctament.

1.3. Utilitzeu ara el programa *sum2.c*, que és una variant de l'anterior que permet calcular i emmagatzemar en memòria el sumatori de la progressió de tots els nombres d'1 a N. En comparar els dos codis podreu constatar que l'única diferència està en que els resultats intermedis s'emmagatzemen en memòria. No obstant això, en executar aquesta versió i comparar les estadístiques d'execució, podeu constatar que el temps d'execució varia considerablement.

Annexeu una captura de pantalla de les estadístiques d'execució de sum2. Analitzant la resta d'estadístiques de rendiment, justificar la raó d'aquesta diferència en el temps d'execució.

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
$ /usr/bin/time -v ./sum2
5000000050000000
        Command being timed: "./sum2"
        User time (seconds): 0.32
        System time (seconds): 0.21
        Percent of CPU this job got: 92%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.58
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 782900
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 195374
        Voluntary context switches: 10
        Involuntary context switches: 40
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
$
```

Com es pot constatar en comparar les estadístiques dels dos programes, el nombre de fallades de pàgines realitzats per la segona versió (195.373) és molt més gran que els errors comesos en la primera versió (59). Això és degut a que per emmagatzemar els resultats intermedis necessitem accedir i escriure en un vector de memòria de 100 milions de double (8 bytes), per al que es requereix un total 762 MBytes de memòria. Com la mida de memòria assignada al procés està limitat pel SO, a mesura que es va recorrent el vector, es van accedint a noves pàgines de memòria, la qual cosa genera considerables fallades de pàgina.

1.4. Modificar ara el programa *sum2.c*, perquè el vector amb els resultats intermedis, en lloc de definir-se com una variable global al programa, es

defineixi com una variable local a dins de la funció *main()*. Incloure una còpia del programa resultant. Executeu la nova versió i annexar una captura de pantalla de les estadístiques de la seva execució. Es s'executa correctament l'aplicació? A que és degut? Justifiqueu la resposta.

Per respondre aquesta pregunta us pot ajudar la comanda "***ulimit –a***" que mostra informació sobre els limitis d'utilització de recursos definits pel sistema per a l'execució dels processos.

El codi font de la versió modificada s'adjunta en el fitxer sum2bOk.c

El resultat de l'execució és el següent:

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
$ /usr/bin/time -v ./sum2bOk
Command terminated by signal 11
        Command being timed: "./sum2bOk"
        User time (seconds): 0.00
        System time (seconds): 0.00
        Percent of CPU this job got: 0%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.19
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 1052
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 50
        Voluntary context switches: 14
        Involuntary context switches: 0
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
$
```

Com es pot comprovar l'aplicació no s'executa correctament ja que genera una excepció SIGSEV de referència invàlida a memòria. Això és degut a que la variable local sum, requereix més de 760 MBytes de memòria. En declarar la variable sum com a variable local, l'estem situant a la pila del procés. Per defecte, la pila té una mida de 8 MBytes (depenent de la versió Linux), tal com podem veure amb la comanda *ulimit -a*:

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
$ ulimit -a
core file size          (blocks, -c) unlimited
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 23798
max locked memory       (kbytes, -l) 16384
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 23798
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
$
```

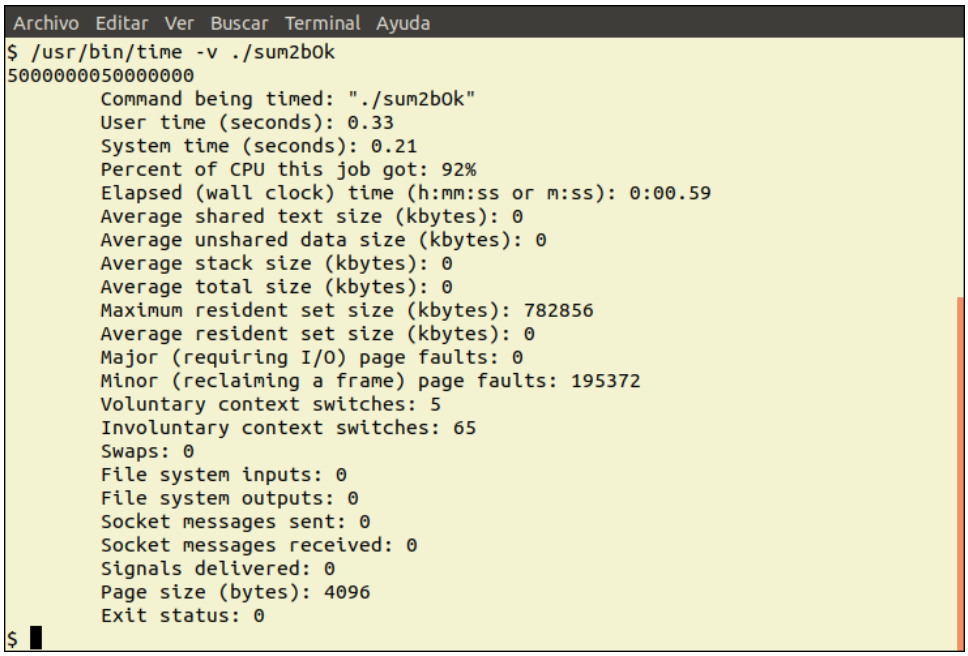
Per tant, no cap tota la variable a memòria i quan intentem accedir a la posició del vector fora de l'espai disponible es genera una excepció.

Utilitzeu la comanda ***ulimit*** per permetre l'execució d'aquesta segona versió del programa sum2. Què paràmetre s'ha modificat i que valor li heu assignat? Per què? Annexeu una captura de pantalla de l'execució.

Per aconseguir que aquesta última versió del programa *sum2* es pugui executar, hem d'ampliar la mida de la pila dels processos. Això es pot fer mitjançant la següent ordre: ulimit -s 1000000 que permet una mida de pila de fins a 1GB.

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
$ ulimit -s 1000000
$ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 23798
max locked memory       (kbytes, -l) 16384
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 1000000
cpu time               (seconds, -t) unlimited
max user processes              (-u) 23798
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
$
```

El que permet l'execució de la nova versió sense cap problema:



Observacions:

- Per compilar el programa *sum1*.c, cal executar la comanda:

  > *gcc sum1.c -o sum1*

  i per executar cal utilitzar les següents ordres:

  > */usr/bin/time -v ./sum1*

  > *./sum1*

  en funció de si volem o no mostrar les estadístiques de la seva execució. La resta de programes d'aquest enunciat es compila i s'executen de forma anàloga.

2: Mòdul 3 [Del 11 al 23 d'octubre] (45%, els tres apartats tenen el mateix pes)

2.1. Modifiqueu el programa sum2.c perquè utilitzi memòria dinàmica per al vector sum. Expliqueu els principals canvis introduïts a nivell de codi per suporta la memòria dinàmica. Executeu la nova versió i annexar una captura de pantalla de la seva execució. En què varia la nova execució?

El codi font de la versió modificada s'adjunta en el fitxer sum3bOk.c

Els canvis introduïts estan relacionats amb la reserva de memòria dinàmica per guardar els resultats dels càlculs realitzats.

S'ha modificat el tipus de la variable sum, ja que ara ha de ser de tipus apuntador a double per poder guardar l'apuntador a l'inici a la memòria dinàmica reservada :

```
double *sum;
```

També cal incloure el codi per reservar la memòria, això ho fem amb la funció de llibreria malloc, en la que especifiquem la mida total de memòria que volem reservar en bytes. Aquesta funció ens retorna l'apuntador a la zona de memòria reservada si hi ha espai suficient i un apuntador a NULL si es produeix un error. Sempre cal comprovar si la crida a malloc ha funcionat i en cas contrari indicar l'error i finalitzar.

```
sum = malloc(sizeof(double)*(N+1));
if (sum==NULL)
    exit(1);
```

La resta del programa no cal modificar-lo, pel fet que utilitzar un apuntador a double és equivalent a un vector de double i permet realitzar el mateix tipus d'accessos indexats .

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
$ /usr/bin/time -v ./sum3Ok
5000000050000000
        Command being timed: "./sum3Ok"
        User time (seconds): 0.37
        System time (seconds): 0.17
        Percent of CPU this job got: 93%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.58
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 782312
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 195372
        Voluntary context switches: 9
        Involuntary context switches: 43
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
$
```

Des del punt de vista del rendiment (temps execució, fallades de pàgina i mida de la memòria) les dues versions (sum2b i sum3) són equivalents.

2.2. En aquest exercici anem a analitzar alguns errors típics de programació relacionats amb la utilització de la memòria i aprendre com evitar-los. Per

realitzar aquest exercici, us adjuntem el programa fib.c que calcula els N primers nombres de la sèrie de Fibonacci de forma seqüencial i recursiva.

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  #include <stdlib.h>
4.  #include <unistd.h>
5.  void rfib(int *fib, int x, int *last);

6.  int main (int argc, char *argv[])
7.  {
8.     int N,x;
9.     int *s;
10.    char *str, *str2;

11.    N = atoi(argv[1]);

12.    s[0]=0;
13.    s[1]=1;

14.   s = malloc(sizeof(int)*N);
15.   if (s=NULL)
16.      exit(1);

17.    // Calculate Fibonacci Serie.
18.    for (x=0;x<=N;x++)
19.         s[x] = s[x-1] + s[x-2];

20.    // Print Fibonacci Serie.
21.    str2=" Iterative Fibonacci Calculation";
22.    for (x=0;x<N;x++)
23.    {
24.            sprintf(str,"%s %d -> %d.\n",str2,x, s[x]);
25.            write(1,str,strlen(str));
26.    }

27.    rfib(s,0,0);

28.    // Print Fibonacci Serie.
29.    strcpy(str2,"Recursive Fibonacci calculation");
30.    for (x=0;x<N;x++)
31.    {
32.            sprintf(str,"%s %d -> %d.\n",str2,x);
33.            write(1,str,strlen(str));
34.    }
35.    exit(0);
36. }

37. void rfib(int *fib, int x, int *last)
38. {
39.    if (x==0)
40.            *fib=0;
41.    else if (x==1)
42.            *fib=1;
43.    else if (x>1)
44.            *fib=fib[-1]+fib[-2];

45.    if (fib+1!=last)
46.            rfib(fib++,x++,last);
47. }
```

El programa fib.c conté almenys 10 errors relacionats amb la memòria, que es solen cometre molt sovint en implementar programes en els sistemes Linux. Aquests errors poden provocar errades, en forma d'excepcions, al llarg de l'execució del programa. Se us demana que identifiqueu aquests errors, quina fallada d'execució poden produir i si aquest errors es poden identificar o no en temps de compilació.

a) Identifiqueu els errors comesos al programa, proporcionant la següent informació: les línies de codi que contenen els errors, en què consisteix l'error i quin efecte pot produir l'error en l'execució del programa. Tingueu en compte, que una línia de codi pot generar més d'un error d'execució.

   Com a exemple, us donem la solució dels dos primers errors.

   1) En la línia 11, s'assigna l'argument d'entrada de l'usuari a la variable N, sense controlar primer si existeix aquest argument. Aquest error podria generar una excepció "segmentation fault" durant l'execució del programa i finalitzar el programa .

   2) En les línies 12-13, s'assignen valors a les dues primeres posicions d'un vector que encara no tenen memòria assignada. Aquest error també podria generar una excepció "segmentation fault".

   3) En la línia 14, si l'usuari no introdueix almenys un paràmetre i pel fet que no s'assigna un valor per defecte a la variable N, pot passar que la mida de la memòria a reservar (*sizeof(int)*N*) no estigui definida. Això pot provocar que no es reservi suficient memòria per al vector o bé que la memòria que es demani sigui tan gran que el sistema no la pugui concedir (tornant un NULL). En ambdós casos, quan s'accedeixi al vector es pot generar una excepció.

   4) En la línia 14, es controla de forma incorrecta si la reserva de memòria ha funcionat bé. En comptes de fer una comparació (s==NULL) s'ha fet una assignació (s=NULL). Per tant, independentment si el ***malloc*** falla com si no, la variable *s* tindria el valor NULL i quan s'accedeixi a qualsevol posició del vector es generarà una excepció.

   5) En la línia 19, es podria generar un excepció en accedir al vector s, sempre que la reserva de memòria de la línia 14 falli o es reservi prou memòria.

   6) En les primeres dues iteracions del bucle de la línia 18, generarà un error en el codi de la línia 19, degut a que s'accedeix a índexs negatius del vector s (s[-1], s[-2]). Aquest error pot generar una excepció durant l'execució del programa.

   7) En la línia 19 , encara que la reserva de memòria de la línia 14 es realitzi correctament, es pot generar una fallada pel fet que només es reserva memòria per a N posicions del vector (des del l'índex 0 al N-1), mentre que el bucle de la línia 18 arriba fins l'índex N. Per tant, en l'última iteració del bucle es

generarà un error en accedir a la posició s[x=N] , que no té memòria assignada.

8) A la línies 24 i 32 es generarà una excepció de "segmentation fault" degut a que la cadena *str* no té memòria assignada. Per tant quan el *sprintf* provi de copiar la cadena resultant en la cadena *str*, es produirà l'error.

9) La línia 27 té un error en no definir de manera correcta el final de la recursivitat (tercer paràmetre). Això pot generar una recursivitat molt gran, omplir la pila amb els valors de retorn de les diferents invocacions recursives i finalment un error quan la pila s'ompli.

10) La línia 29 conté un error, pel fet que *str2* té una mida limitada definit per la cadena constant "*Sequential Calculation*" que s'assigno en la línia de codi 21. Quan provi de copiar una cadena més gran, en la línia 29, es produirà un desbordament que pot generar una excepció.

11) El *sprintf* de la línia 32 conté un error, ja que en la cadena de format es defineixen 3 paràmetres (una cadena i dos enters) i posteriorment només se li passa dos (la cadena str2 i el sencer x).

b) Indiqueu per a quins dels errors que heu trobat en l'apartat anterior el compilador us ha avisat mitjançant un *warning* o error de compilació. Expliqueu el significat de l'error.

Perquè el compilador generi tots els avisos/*warnings*, podeu compilar el programa amb l'opció –*Wall*.

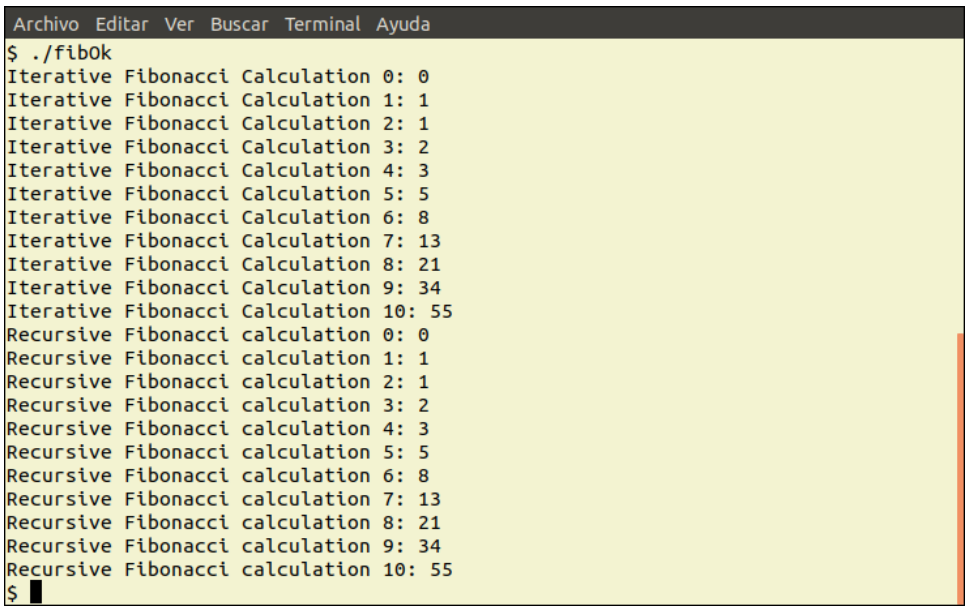
Per a l'error de les línies 12-13, el compilador genera el següent avís: s'utilitza 's' sense inicialitzar en aquesta funció [-Wuninitialized]. Informa que la variable 's' s'està utilitzant sense haver-se inicialitzat prèviament

En l'error de la línia 14, el compilador genera el següent avís: es suggereixen parèntesi al voltant de l'assignació usada com a valor verdader [-Wparentheses]. Amb aquest error està indicant que hem inclòs un assignació dins d'una comparació i que per evitar problemes de precedència caldria posar-la entre parèntesi.

En l'error de les línies 24 i 32, el compilador genera el següent avís: pot ser que s'utilitzi 'str' sense inicialitzar en aquesta funció [-Wuninitialized]. En indica que la variable 'str' s'està utilitzant sense inicialitzar.

En l'error de la línia 32, el compilador genera el següent avís: format '%d ' espera un argument 'int' coincident [-Wformat]. El compilador ens està avisant que en formatar la cadena hem indicat que hi hauria un enter, però en especificar els valors no ho hem posat

2.3. *Modifiqueu* el programa*, per solucionar els errors comesos i que funcioni correctament. Lliurar* el *codi font modificat i una captura de pantalla amb l'execució correcta del programa.*



Observacions:

- Per detectar i corregir els errors, podeu utilitzar el depurador ***gdb***, amb els fitxers de *core* (bolcat de la memòria del procés) que genera un programa quan és finalitzat per una excepció.

  Tal com es pot veure en la següent captura de pantalla, per depurar aquests errors, primer s'ha d'incloure informació de depuració en l'executable (opció -g del compilador), s'ha d'activar la generació dels fitxers core *("ulimit –c unlimited"*) i posteriorment executar el programa que falla.

  En les noves versions de Linux, la gestió dels fitxers core es delega a tercers programes. En el cas de la distribució Ubuntu 18.04 l'aplicació que s'encarrega de la gestió dels cores és *apport*, pel que és necessari instal·lar-la (sudo apt-get install *apport*) i crear un fitxer de configuració en el directori ~/.config/*apport*/settings amb el següent contingut:

  ```
  [main]
  unpackaged=true
  ```

Posteriorment executar el programa que falla, perquè generi el fitxer de core. En aquesta versió de Linux, el fitxer es genera en el directori /var/crash. Finalment, per a poder utilitzar aquest bolcat amb el gdb, cal desempaquetar amb la comanda *apport*-unpack.

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
$ gcc -g fib.c -o fib
fib.c: In function 'main':
fib.c:42:24: warning: format '%d' expects a matching 'int' argument [-Wformat=]
   sprintf(str,"%s %d: %d\n", str2, x);
                        ~^
$ ulimit -c unlimited
$ sudo apt-get install apport
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
apport ya está en su versión más reciente (2.20.9-0ubuntu7.7).
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 408 no actualizados.
$ cat ~/.config/apport/settings
[main]
unpackaged=true
$ ./fib
Violación de segmento (`core' generado)
$ ls /var/crash/
_media_sf_Dropbox_Docencia_UOC_SemestreOct19-Feb20_Enunciados_PRA1_pr1so_fib.1000.crash
$ apport-unpack /var/crash/_media_sf_Dropbox_Docencia_UOC_SemestreOct19-Feb20_Enunciados_PRA1_p
r1so_fib.1000.crash fib_core
$ gdb ./fib fib_core/CoreDump
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
Para las instrucciones de informe de errores, vea:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Leyendo símbolos desde ./fib...hecho.
[Nuevo LWP 17179]
El núcleo se generó por «./fib».
Program terminated with signal SIGSEGV, Segmentation fault.
#0  __GI_____strtol_l_internal (nptr=0x0, endptr=endptr@entry=0x0, base=base@entry=10,
    group=group@entry=0, loc=0x7f6d774c7560 <_nl_global_locale>)
    at ../stdlib/strtol_l.c:292
292     ../stdlib/strtol_l.c: No existe el archivo o el directorio.
(gdb) bt
#0  __GI_____strtol_l_internal (nptr=0x0, endptr=endptr@entry=0x0, base=base@entry=10,
    group=group@entry=0, loc=0x7f6d774c7560 <_nl_global_locale>)
    at ../stdlib/strtol_l.c:292
#1  0x00007f6d77120122 in __strtol (nptr=<optimized out>, endptr=endptr@entry=0x0,
    base=base@entry=10) at ../stdlib/strtol.c:106
#2  0x00007f6d7711b690 in atoi (nptr=<optimized out>) at atoi.c:27
#3  0x0000559deffd17cc in main (argc=1, argv=0x7ffe63e93f88) at fib.c:15
```

Figura 2. Procediment per la depuració d'excepcions.

Una vegada produït l'error, s'invoca al depurador (gdb), indicant el nom del programa que es vol depurar i el fitxer de *core* generat ("gdb ./fib ./fib_core/CoreDump"). Utilitzant la comanda ***bt*** del depurador, podrem obtenir la pila de crides i veure quin és la línia de codi del nostre programa que ha generat l'excepció. En l'exemple de la figura 1, l'excepció es produeix en executar la instrucció situada en la línia de codi 15 del programa fib.c, en realitzar el ***atoi*** sense que se li hagi passat un argument al programa.

3: Mòdul 4 [Del 24 al 30 d'octubre] (20%, els dos apartats tenen el mateix pes)

3.1. Utilitzant el programa de l'apartat 2 (càlcul sèrie de Fibonacci), les ordres de sistema (grep, cut, wc, tr, entre d'altres) i les redireccions / pipes, implementar les següents comandes:

a) Escriure en dos fitxers separats els nombres de la sèrie calculats de forma iterativa i de forma recursiva.

```
./fibOk | grep Iterative > FactIterative.txt
./fibOk | grep Recursive > FactRecursive.txt
./fibOk | wc -l
```

b) Comptar el nombre de Fibonacci que s'han calculat en total.

```
./fibOk | wc -l
```

c) Comptar els caràcters necessaris per mostrar el resultat dels nombres de Fibonacci.

```
./fibOk | cut -d":" -f2 | tr -d ' ' | wc -c
```

d) Ordeneu els resultats dels nombres de Fibonacci de major a menor.

```
./fibOk | sort -t: -k2 -n -r
```

3.2. Utilitzant exclusivament crides al sistema de Linux, modificar el programa de càlcul dels nombres de Fibonacci perquè escrigui el resultat en un fitxer especificat per l'usuari. Incloure una còpia del programa resultant i comentar el codi afegit.

El codi font de la versió modificada s'adjunta en el fitxer fib3Ok.c

Recursos

Recursos Basics

- Document "Introducció a la programació de Unix" (disponible al campus virtual).
- Document "Intèrpret de comandes UNIX" (disponible a l'aula) o qualsevol altre manual similar
- Qualsevol manual bàsic de llenguatge C.
- L'aula "Laboratori de Sistemes Operatius" (podeu plantejar els vostres dubtes relatius a l'entorn Unix, programació,...).

Recursos Complementaris

- Manual de crides al sistema instal·lat a qualsevol màquina UNIX (comanda man).

Criteris de valoració

Cadascun dels apartats de la pràctica té un pes del 50% sobre la puntuació final. El pes de cada apartat es distribueix de forma equitativa entre tots els subapartats que ho integren.

En la correcció es tindran en compte els següents aspectes:

- Les respostes hauran d'estar articulades a partir dels conceptes estudiats en teoria i en la guies de l'assignatura.
- Es valorarà essencialment la correcta justificació de les respostes.
- S'agrairà la claredat i la capacitat de síntesis en les respostes.
- La capacitat d'anàlisi dels resultats obtinguts i la metodologia seguida per a la seva obtenció.
- El codi lliurat ha de poder executar-se correctament en qualsevol màquina amb Linux. Que el codi s'executi correctament en el vostre ordinador, no implica necessàriament que sigui correcte. Revisar diverses vegades la vostra solució i no ignorar els *warnings* que us pugui estar generant el compilador.

Format i data de lliurament

Es crearà un fitxer zip que contingui un fitxer pdf amb la resposta a totes les preguntes i els un fitxer .c amb el codis fonts de l'exercicis.

El nom del fitxer tindrà el format següent: "Cognom1Cognom2PRA1.zip". Els cognoms s'escriuran sense accents. Per exemple, l'estudiant Marta Vallès i Marfany utilitzarà el nom de fitxer següent: VallesMarfanyPRA1.tar

La data límit per al lliurament de la pràctica és el **dimecres 30 d'octubre de 2019**.

Nota: Propietat intel·lectual

Sovint és inevitable, en produir una obra multimèdia, fer ús de recursos creats per terceres persones. És per tant comprensible fer-ho en el marc d'una pràctica dels estudis del Grau Multimèdia, sempre i això es documenti clarament i no

suposi plagi en la pràctica.

Per tant, en presentar una pràctica que faci ús de recursos aliens, s'ha de presentar juntament amb ella un document en què es detallin tots ells, especificant el nom de cada recurs, el seu autor, el lloc on es va obtenir i el seu estatus legal: si l'obra està protegida pel copyright o s'acull a alguna altra llicència d'ús (Creative Commons, llicència GNU, GPL ...). L'estudiant haurà d'assegurar-se que la llicència que sigui no impedeix específicament seu ús en el marc de la pràctica. En cas de no trobar la informació corresponent haurà d'assumir que l'obra està protegida pel copyright.

Hauran, a més, adjuntar els fitxers originals quan les obres utilitzades siguin digitals, i el seu codi font si correspon.

Un altre punt a considerar és que qualsevol pràctica que faci ús de recursos protegits pel copyright no podrà en cap cas publicar-se en Mosaic, la revista del Graduat en Multimèdia a la UOC, a no ser que els propietaris dels drets intel·lectuals donin la seva autorització explícita.