CRIDES AL SISTEMA OPERATIU DES DE C

OBJECTIUS

- Conèixer les crides de baix nivell al Sistema Operatiu.
- Desenvolupar un programa utilitzant el llenguatge de programació C que permeti fer crides de baix nivell d'Entrada/sortida al SO.

MATERIAL

- Sistema Operatiu Linux
- Compilador GCC

Important!

- La realització de les pràctiques es farà en grups formats per dues persones.
- La pràctica consta de 2 sessions, i el termini límit de lliurament serà 4 dies després d'haver realitzat la segona sessió de les pràctiques.
- A més dels fitxers de codi desenvolupats, caldrà fer un informe de pràctiques, en el qual es detalli el treball realitzat durant les 2 sessions de pràctiques.
- En cas de detectar còpies, l'alumne tindrà automàticament un 0 a la nota de pràctiques

1. INTRODUCCIÓ

El sistema operatiu (SO) és el programa que permet gestionar els recursos del maquinari d'un sistema informàtic com són la memòria, els dispositius d'emmagatzematge de dades, els terminals, etc.

Linux com a sistema operatiu ofereix serveis de baix nivell per gestionar la memòria, la planificació dels processos, la comunicació entre aquests, i les operacions d'entrada i sortida (E/S). Els processos o programes poden sol·licitar aquests serveis directament al KERNEL (crides al sistema), o indirectament a través de rutines de llibreria.

El llenguatge de programació C permet fer qualsevol crida al sistema a través de les biblioteques de funcions de què es disposa.

Les crides al sistema es poden fer servir per mitjà de funcions escrites en C, i totes elles tenen un format comú, tant en la documentació com en la seqüència de crida. Una crida al sistema s'invoca mitjançant una funció retornant sempre un valor amb informació sobre el servei que proporciona o de l'error que s'ha produït en la seva execució. Encara que aquest retorn pot ser ignorat, és recomanable sempre testejar-ho. Tornen un valor de -1 com a indicació que s'ha produït un error en la seva execució. A més, hi ha una variable externa *errno*, on s'indica un codi d'informació sobre el tipus d'error que s'ha produït.

Amb l'objectiu d'entendre i saber utilitzar aquestes funcions, durant el transcurs d'aquesta pràctica es desenvoluparà alguns programes en C, que fent ús de les funcions de crides al SO d'E/S, permetran llegir i processar informació obtinguda des d'un arxiu.

2. CONCEPTES PREVIS

Abans de fer aquesta pràctica cal que estudieu bé les explicacions que es donen en aquesta secció.

Comencem amb un parell de conceptes:

Biblioteques de funcions: qualsevol llenguatge de programació ve acompanyat per múltiples biblioteques de funcions que implementen de forma eficient i confiable operacions sobre un domini determinat. En aquest curs heu fet servir aquestes biblioteques quan heu programat amb C (cada vegada que feu un #include) i també amb Python (cada vegada que feu un import). Algunes de les biblioteques de Python que fareu servir abastament al llarg dels vostres estudis són TensorFlow, NumPy, SciPy, Pandas, Matplotlib, Keras, SciKit-Learn o PyTorch. Algunes de les que heu fet servir en C són *stdio* i *stdlib*, però n'hi ha moltes més com *math*, *time*, *ctype*, *string* o *signal*.

Fent servir aquestes biblioteques el desenvolupador s'estalvia molta feina al no haver de reescriure un cop i un altre funcions d'us freqüent o d'alta complexitat.

Molts cops, una funció de biblioteca simplement realitza un càlcul i retorna un resultat com per exemple, la funció sqrt inclosa en la biblioteca math de C, que calcula l'arrel quadrada d'un número. En aquests casos, des del nostre programa es fa un call, s'executa el codi de la funció i es fa un retorn al nostre codi, just com si la funció l'haguéssim programat nosaltres.

Ara bé, molts altres cops, la funció de biblioteca es fa servir per demanar o utilitzar un recurs del sistema (per exemple un arxiu). En aquests casos, com a mínim una part de la funció no pot ser executada pel nostre programa (que s'executa en mode usuari i no té accés directe a cap recurs del sistema) i ha de ser servida pel SO a través d'alguna de les crides al sistema que aquest ofereix.

Crida al sistema: funció que el Sistema Operatiu ofereix per tal de permetre l'ús d'algun dels recursos del sistema que no poden ser gestionats en mode usuari. Només pot ser invocada a través d'una interrupció voluntària (molts cops anomenada *trap*) que cedeixi el control al SO.

Així, quan una **funció de biblioteca** es fa servir per demanar o utilitzar un recurs del sistema, necessàriament invocarà una **crida al sistema** i causarà una interrupció per passar el control al SO.

En aquesta pràctica experimentareu amb aquests tipus de funcions a través de les funcions per la gestió d'arxius a baix nivell, incloses a les biblioteques *fcntl* i *unistd* (caldrà que feu #include <fcntl.h> i #include <uistd.h>), que es descriuen a continuació:

- **Open**.
  Aquesta funció indica al sistema operatiu que volem utilitzar (accedir per fer lectures i/o escriptures) un arxiu en el nostre arbre de carpetes. En si mateixa no implica cap operació d'E/S, simplement permet que el SO comprovi que el nostre programa pugui utilitzar l'arxiu i inicialitzi un conjunt d'estructures de dades internes, tant a nivell del procés que fa la petició, com del propi SO.

  La capçalera de la funció open és:
  int open(const char *pathname, int flags);

  On:
  - *pathname* és una cadena de caràcters amb el nom d'un arxiu en el nostre arbre de carpetes (pot incloure un camí relatiu o absolut). Per exemple: titles.cvs, /home/eduardo/titles.csv, ../titles.csv
  - *flags* és una constant que indica com volem utilitzar l'arxiu: O_RDONLY si només volem llegir els seus continguts, O_WRONLY si només volem escriure en l'arxiu, o O_RDWR si volem llegir i escriure.

  Si l'arxiu indicat existeix i el nostre programa té els privilegis necessaris per accedir-hi, la funció retornarà un nombre enter positiu que anomenarem *descriptor d'arxiu* (fd en les seves inicials en anglès). Aquest número serà l'identificador que el nostre programa utilitzarà en totes les operacions que realitzi sobre l'arxiu amb posterioritat a l'open.
  En cas de que es produeixi un error, la funció retornarà -1.

- **Close**.
  Aquesta funció indica al sistema operatiu que volem deixar d'utilitzar un arxiu. Al igual que la funció open, no implica cap operació d'E/S, simplement permet que el SO alliberi les estructures internes que va inicialitzar amb l'operació open corresponent.

  La capçalera de la funció és:
  int close(int fd);

  On *fd* és el descriptor d'arxiu que ens va retornar l'operació open amb la qual vam obrir l'arxiu.
  Si el descriptor proporcionat és vàlid, l'operació close retorna 0 (ha pogut tancar l'arxiu), si no ho és, retorna -1.

- **Read**.
  Aquesta funció permet llegir un cert nombre de bytes (1 o més) des d'un arxiu a partir de la darrera posició llegida (o des del començament si es tracte de la 1a lectura). Des del punt de vista del SO, un arxiu és una seqüència de bytes, en conseqüència, la forma "natural" d'accedir als continguts de l'arxiu serà llegir primer el primer byte de la seqüència, després el segon, etc.

  La capçalera de la funció és:
  ssize_t read(int fd, void *buf, size_t count);

On:

- *fd* és el descriptor de l'arxiu (retornat per la funció open) del que volem llegir.
- *buf* és un array on emmagatzemarem els continguts llegits de l'arxiu (és responsabilitat del programador que la mida d'aquest array sigui prou gran). Si només volem llegir un element de l'arxiu, no cal que buf sigui un array, pot ser una variable escalar (char, int, float...) sempre que passem la seva adreça com a paràmetre (&var).
- *count* és el nombre de bytes que volem llegir de l'arxiu.

Els bytes llegits des de l'arxiu quedaran emmagatzemats a *buf* i la funció retornarà el nombre de bytes que s'han llegit. Si s'intenta llegir al final de l'arxiu, la funció retornarà 0. En cas d'error, retornarà -1.

- **Write**.
  Aquesta funció permet escriure un cert nombre de bytes (1 o més) a un arxiu a partir de la darrera posició escrita o llegida (o des del començament si es tracte de la 1a escriptura).

  La capçalera de la funció és:

  ssize_t write(int fd, void *buf, size_t count);

  On:

  - *fd* és el descriptor de l'arxiu (retornat per la funció open) en el que volem escriure.
  - *buf* és un array on tenim emmagatzemats els continguts que volem escriure a l'arxiu. Si només volem escriure un element de l'arxiu, no cal que buf sigui un array, pot ser una variable escalar (char, int, float...) sempre que passem la seva adreça com a paràmetre (&var).
  - *count* és el nombre de bytes que volem escriure a l'arxiu.

  El contingut de l'array *buf* quedaran emmagatzemats a l'arxiu i la funció retornarà el nombre de bytes que s'han escrit. En cas d'error, retornarà -1.

3. EXERCICIS PRÀCTICS

En aquesta pràctica us demanem desenvolupar dos programes en els que implementarem operacions senzilles sobre arxius.

El primer programa consistirà en desenvolupar el programa ***my_tee***, el qual rebrà una cadena de caràcter per línia d'ordres.

- El programa intentarà crear un arxiu amb el nom rebut i permisos de lectura i escriptura pel propietari i només de lectura pel grup i els altres usuaris. En cas de que no es pugui crear l'arxiu, el programa donarà un missatge d'error amb la funció *perror* i acabarà (*exit*) retornant el valor -1.

- Després, el programa llegirà caràcters (d'un amb un) de l'entrada estàndard (*read*) i els anirà afegint (*write*) a l'arxiu creat, això vol dir que els caràcters apareixeran per pantalla (sortida estàndard) però també s'emmagatzemaran en l'arxiu. Aquest procés es repetirà fins que s'acabi l'entrada (^*D* si es fa des de teclat)o el *write* produeixi un error (en aquest cas es donarà el missatge d'error corresponent amb la funció *perror* i acabarà (*exit*) retornant el valor -1).

Per demostrar que el vostre programa funciona cal que la sortida que produeixi coincideixi amb la que produiria l'ordre *tee <cadena-caràcters>*. Podeu buscar què fa aquesta ordre amb l'ordre *man* o a la web.

El segon programa consistirà en desenvolupar el programa ***my_cmp***, el qual rebrà per línia d'ordres els noms dels dos arxius a comparar.

- El programa intentarà obrir (*open*) els dos arxius en mode de només lectura. Si algun dels opens falla, es donarà el missatge d'error corresponent amb la funció *perror* i acabarà (*exit*) retornant el valor -1.
- Després, el programa comparará byte a byte els dos arxius. Si ambdós són iguals, no es produirà cap sortida, però si no ho són, el programa indicarà la posició del byte i la línia en la que s'ha trobat la 1a diferència. La numeració dels bytes i les línies comença en 1.

Per demostrar que el vostre programa funciona cal que la sortida que produeixi coincideixi amb la que produeix l'ordre *cmp <nom1> <nom2>*.

En aquesta pràctica us demanem que els programes en C rebin arguments des de la línia d'ordres, això s'aconsegueix declarant dos paràmetres en la funció main de la següent manera:

```
int main ( int argc, char *argv[] )
```

El primer paràmetre (*argc*) és un enter que indica el nombre d'argument rebuts pel programa (el mínim nombre d'arguments rebuts per un programa és, sorprenentment, 1, ja que en C tot programa rep com a 1r argument el seu propi nom). Això vol dir que si volem comprovar que un programa rep com a mínim un argument, farem *if (argc < 2)* ...

El segon paràmetre (argv) és un array de cadenes de caràcters (strings) amb els arguments que l'usuari ha posat en la línia d'ordres.

Per exemple, si tenim un programa prog i l'usuari el crida fent *$ ./prog 100 hola*, tindríem que:

- argc = 3
- argv[0] = "prog"
- argv[1] = "100"
- argv[3] = "hola"

Com vosaltres treballareu només amb strings, amb aquesta informació hauríeu de tenir suficient.