Crides a Sistema

Les crides a sistema, són els serveis mínims que ofereix tot sistema operatiu als programadors per poder utilitzar els recursos del computador de forma eficient i controlada.

Per poder accedir als recursos del computador, disc, memòria, etc, el codi s'ha d'executar en mode privilegiat o mode sistema. El codi d'una crida a sistema s'executa en mode privilegiat. El codi d'un programa s'executa amb mode usuari o mode no privilegiat, de forma que si un programa vol utilitzar els recursos del sistema, ha d'utilitzar les crides a sistema per poder passar a mode privilegiat.

Quan fem una crida a sistema, hi ha un mecanisme anomenat *trap*, que permet fer un canvi de mode d'execució, de mode usuari a mode sistema, tan sols les crides a sistema activen el *trap* I tan sols així podem accedir als recursos del computador. Aquest mecanisme és una forma de tenir controlat l'accés als recursos, ja que tan sols ho pots amb els serveis programats pel sistema operatiu.

Estructura crides a sistema linux/unix

En linux/unix totes les crides a sistema tenen una estructura equivalent:

Retorn ***nom_crida*** *(parametres)*

El que és important és el retorn de la crida a sistema, aquest segueix el següent format:

- si la crida s'executa correctament, el valor retornat és un nombre positiu més gran o igual a zero.
- si la crida s'executa incorrectament, retorna -1, que indica que ha retornat error. En aquest cas els motius pels quals s'ha generat l'error es poden consultar mitjançant la variable *errno*. Tots els possibles errors que pot retornar una crida es poden consultar en el manual de linux/unix. Per exemple:

```
if ((con = socket(PF_INET, SOCK_STREAM, 0))<0)
/* ens retorna -1*/
   if (errno == EMFILE)
    write(2,"Error opening socket: process file table overflow\n",49);
   if (eerno==ENOMEM)
    write(2,"Error socket: insufficient memory available\n",42);
   exit(-1);
}
```

En el codi d'exemple, quan la crida *socket* retorna un -1, llavors mirem d'esbrinar quin error s'ha generat consultant la variable *errno*, comparant-la amb els possibles errors de la crida (executar la comanda `man socket` per veure el llistat de possibles errors).

Crides a sistema com a serveis mínims

Unix/*linux* és un sistema operatiu que en el seu disseny s'ha potenciat la simplicitat i amb aquest objectiu el sistema ofereix dos crides a sistema per fer operacions d'entrada i sortida -*read* i *write* respectivament-, crides amb les que és possible accedir a qualsevol dispositiu de l'ordinador: disc, pantalla, sockets, teclat,... Aquestes crides gestionen la informació de forma molt simplificada, i tan sols permeten llegir/escriure bytes, sense cap tipus de format -char, integer, float, double, etc.-.

Les crides a sistema són els serveis mínims que ofereix els sistema operatiu al programador i per tant són una mica més complexes d'utilitzar si ho comparem amb les crides a la llibreria del llenguatge, per exemple, de la llibreria del llenguatge de programació "C".

Si volem escriure per la pantalla un missatge amb la crida a sistema *write*, hem d'indicar el canal lògic que té assignat el dispositiu pantalla que per defecte en tot procés és el canal 1, a continuació s'ha d'indicar el missatge que es vol escriure i per últim el nombre de bytes del missatge que volem escriure:

```
write(1, "Resultats del programa \n",24);
```

Aquesta forma és molt més complexa si ho comparem amb la utilització d'una crida a llibreria del llenguatge "C" que tothom coneix i que simplifica el procés de forma notable, fent un simple *printf* aconseguim el mateix resultat que amb la crida *write*:

```
printf("Resultats del programa \n");
```

En aquest cas, la crida al llenguatge *printf*, preprocessa la informació si és necessari –la prepara per a que sigui correcta quan s'escrigui per pantalla- i acaba fent la crida a sistema *write*. Des del punt de vista de l'usuari està clar que és molt més fàcil i còmode d'utilitzar crides a la llibreria del llenguatge ja que no s'ha d'indicar ni el canal lògic del dispositiu a utilitzar, *printf* sempre escriu pel canal 1, ni el nombre de bytes.

La complexitat d'utilitzar les crides a sistema creix quan volem utilitzar en els nostres programes tipus de dades que no són text, és a dir: caràcter *ascii*, com per exemple: els enters, reals, etc. Mentre que amb el *printf* és molt senzill mostrar per pantalla qualsevol tipus de dades, doncs la mateixa crida fa la conversió tipus (enter, float...) a codi ascii de forma automàtica abans de executar la crida a sistema *write*:

```
cont = 10;
printf("El valor del comptador es %d\n",cont);
```

Si volem utilizar la crida a sistema write per escriure enters per pantalla, abans hem de transformar l'enter a *ascii* de forma esplícita i després escriure'l per pantalla:

```
char buff[30];
int cont= 10;
sprintf(buff, "El valor del comptador es %d\n",cont);
write(1,buff,strlen(buff));
```

Crides a Sistema Sockets

A continuació es pot trobar la llista de crides a sistema que ofereix Linux per poder programar un client i un servidor que utilitzi sockets TCP i UDP. Per cada crida s'indica la llista de llibreries que s'han d'incloure per utilitzar-la i la capçalera de la crida, que indica el tipus de dades de les variables que s'han de passar a la crida a sistema. Per més informació de qualsevol crida recordeu que teniu on-line el manual de linux que s'utilitza mitjançant la comanda *man*, per exemple si volem més informació de la crida *listen* executarem la comanda:
$> man listen

Socket(TCP/UDP):

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

Bind(TCP/UDP):

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

Listen(TCP):

```
#include <sys/types.h>
#include <sys/socket.h>

int listen(int sockfd, int backlog);
```

Accept (TCP)–servidor-:

```
#include <sys/types.h>
#include <sys/socket.h>

int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

Connect (TCP)– client-:

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

Send(TCP):
SendTo (UDP):

```
#include <sys/types.h>
#include <sys/socket.h>

 ssize_t send(int sockfd, const void *buf, size_t len, int flags);

ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
      const struct sockaddr *dest_addr, socklen_t addrlen);
```

Recv(TCP):
ReceiveFrom(UDP):

```
#include <sys/types.h>
 #include <sys/socket.h>

ssize_t recv(int sockfd, void *buf, size_t len, int flags);

ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
        struct sockaddr *src_addr, socklen_t *addrlen);
```

Close:

```
#include <unistd.h>

int close(int fd);
```

A continuació podeu trobar el codi client/servidor que teniu en el document "sockets C#", però reescrit amb crides a sistema Linux-C.

Teniu en compte que C# és un llenguatge de programació orientat a objectes i per tant el que es fa és crear un objecte socket i totes les operacions (bind, listen, accept, connect, send, receive, close, sendto, recvfrom) que es poden realitzar amb l'objecte socket es defineixen com a métodes de l'objecte.

En C, que és un llenguate procedural, tant per crear un socket com per realitzar qualsevol operació amb el socket s'utilitzen crides a sistema. Els métodes de C# i les crides a sistema C tenen un comportament similar sino igual (poden existir petites modificacions de com es passen els paràmetres).

Finalment, en els codis amb sockets TCP codificats amb llenguatge C, s'ha utilitzat les crides a sistema read i write per enviar i rebre dades del socket, en lloc de send i recv.

Servidor TCP:

```
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <stdio.h>


int main(int argc, char *argv[])
{
   int sock_conn, sock_listen, ret;
   struct sockaddr_in serv_adr;
   char buff[512];

      if (argc != 2)
          error("\nParemetres erronis\nUs correcte: server num_port\n\n");

      // INICIALITZACIONS
          // Obrim el socket
      if ((sock_listen = socket(AF_INET, SOCK_STREAM, 0)) < 0)
          error("Error creant socket");

          // Fem el bind al port
      memset(&serv_adr, 0, sizeof(serv_adr));// inicialitza a zero serv_addr
      serv_adr.sin_family = AF_INET;
      serv_adr.sin_addr.s_addr = htonl(INADDR_ANY); /* El fica IP local */
      serv_adr.sin_port = htons(port);
      if (bind(sock_listen, (struct sockaddr *) &serv_adr, sizeof(serv_adr)) < 0)
          error("Error al bind");

          // Limitem el nombre de connexions pendents
          if (listen(sock_listen, 10) < 0)
                    error("listen");
     for(;;){
          // acceptem connexió d'un client
          sock_conn = accept(sock_listen, NULL, NULL);

          //Servei
          ret=read(sock_conn,buff, sizeof(buff));
          buff[ret]='\0';
          write(1, "Missatge rebut: ",16);
          write(1,buff,strlen(buff));

           close(sock_conn);  /* Necessari per a que el client detecti EOF */
      }
```

Client TCP:

```
#include <netdb.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <stdio.h>


int main(int argc, char *argv[])
{
   int sock, n, i ;
   int sock;
   struct hostent *host;
   struct sockaddr_in serv_adr;

          //Inicialitzacions
   host = gethostbyname(nom_host);
   if (host == NULL)
      error("No s'ha trobat info del host");

           // preparem l'adreça del servidor
    memset(&serv_adr, 0, sizeof(serv_adr));// inicialitza a zero serv_addr
   serv_adr.sin_family = AF_INET;
   memcpy(&serv_adr.sin_addr, host->h_addr, host->h_length);
   serv_adr.sin_port = htons(port);

   if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
      error("creant socket");

   if (connect(sock, (struct sockaddr *) &serv_adr, sizeof(serv_adr)) < 0)
      error("connect");

          //Servei
  write(1,"Enter Message for Server <Enter to Stop>: ", 42);
  ret=read(1,buff, sizeof(buff));
  buff[ret]='\0';
  write(sock,buff,strlen(buff));

 close(sock);

 exit(0);
}
```