

Sistemes Operatius

Curs 2016–2017

© Xavi Canaleta, 2016

CONTINGUTS

1. File Descriptors.
2. Signals en C.

1. File descriptors

File Descriptor: element bàsic per a crides al sistema d'entrada sortida.
→ a baix nivell tracta a tothom igual, ja que "escriure" a un txt, per pantalla, etc, ho fem igual

- File descriptor 0: entrada (habitualment teclat).
- File descriptor 1: sortida (habitualment pantalla).
- File descriptor 2: error (habitualment pantalla).

Entrada sortida bàsica

- Ja no usarem `gets()`, `scanf()`, `printf()` etc.
- Usarem només `read()` i `write()`
di 1 → si volem interesting etc ens els haurem de montar
- Exemples:

```
printf("Hola\n"); → write(1, "Hola\n", 5);
```

fileDescriptor nosaltres el que enviem Byte enviem

```
scanf("%s", cadena); → n=read(0, cadena, 100);
```

retorne n'Bytes
- Interessant per sortida: `sprintf()`
fileDescriptor
avar ones guarda la info
Unum caracters per llegir, però si troba un para ja.

1. File descriptors

Treball amb arxius → fitxers NO bin o txt, fitxer de bits

- Ja no usarem fopen(), fclose(), fread, fwrite(), fgets(), fscanf(), fprintf() o feof().
- Usarem open(), close(), read() i write()
- Exemple:

```
FILE *f;
f=fopen("fitxer.dat","rb");
if (f==NULL)...
```

abans

```
int fd; ← fileDescriptor
fd=open("fitxer.dat",O_RDONLY);
If (fd<0)...
```

→ permisos!

ara!

→ ens retorna el fileDescriptor que li pertoca.
→ No hem pogut obrir, ja que ens han assignat un fileDescriptor negatiu

si fem un read del
fitxer i ens retorna
0 → final de fitxer
→ és com fem el
feof()

close(fd);

```
printf("el valor de x = %d\n", x);
```

↓ ara!

```
sprintf(cadena, "el valor x = %d\n", x); → em ha posat dins la variable cadena "el valor  
write(1, cadena, strlen(cadena));
```

x = 34 \n"

1. Signals

- **Signal:** mecanisme del nucli per comunicar events als processors.

- Cada signal té associat un nom i un número (**kill -1**).

- **Funcions que podem usar:**

- *a qui li enviem la interrupció* → `signal(int sig, sig_t funció)`
interrupció que s'envia → *nom de la nova funció que té*
- `kill(int pid, int sig)` → *permet l'enviar interrupcions*
interrupció que volem l'enviar
- `raise(int sig)` → *l'enviar interrupcions a mi mateixa*
interrupció

→ et dóna totes les interrupcions

`kill -9 pid` → mata el programa, NO es pot reprogramar.
`ps` → mostra tots els processos executant-se

té 64 interrupcions

1. Signals

```
int main()
{
    int i;
    for (i = 1; i <= 64; i++)
    {
        signal(i, ksighandler);
    }
    alarm(30);
    while (1)
    {
        printf("Soc el pid %d i espero senyals\n", getpid());
        sleep(2);
    }
}
```

aca hem de reprogramar les 64 interrupcions i els hi ha posat la interrupció ksighandler

→ té a veure amb la interrupció del clock → diem que d'agui 30 seg llensí una interrupció; com que abans hem reprogramat, llencem ksighandler. ← No esperem els 30 seg, anem fent....

1. Signals

↓ Punició que hem posat a les interrupcions

↪ sempre ha de rebre el num d'interrupció

```
void ksighandler(int signum)
{
    switch (signum)
    {
        case SIGINT: ← ctrl C
            kctrlc(signum);
            break;
        case SIGALRM: ← interrupció relotge
            ksigalrm(signum);
            break;
        default:
            kgeneric(signum);
            break;
    }
    signal(signum, ksighandler);
}
```


1. Signals

```
void kctrlc(int signum)
{
    printf("He rebut un ctrl + c (%d).\n", signum);
}

void ksigalrm(int signum)
{
    printf("He rebut un sigalrm (%d).\n", signum);
}

void kgeneric(int signum)
{
    printf("He rebut un altre senyal (%d)\n", signum);
}
```