El objetivo de esta actividad es comprobar lo aprendido hasta ahora sobre Llamadas al Sistema, la Tabla de Canales, y sobre las LlS que manejan la información de los inodos.

Recuerda que tienes la descripción de las llamadas al sistema en el tomo 2 de la ayuda `man`, escribiendo en la línea de comandos `man <llamada>` o buscando en https://www.kernel.org/doc/man-pages/

## 4.1. Concepto de Llamada al Sistema: `strace` y `ltrace`

Prueba este programa con las instrucciones entre las líneas 4 y 9, y comprueba qué funciones de librería usa y qué llamadas al sistema hace.

```
1  #include <stdio.h>
   #include <sys/types.h>
   #include <unistd.h>
4  /**
   $ gcc -no-pie -o nc cuenta.c
   $ ./nc
7  $ echo $?
   $ ltrace ./nc
   $ strace ./nc
10  */
   int main(){
     int numcuenta;
13     numcuenta=getuid();
       printf("\nSoy la cuenta %d\n", numcuenta);
     return (numcuenta-745); // Para verlo, echo $? en la lÃnea de comandos
16     // Solamente se puede devolver un valor menor que 256
   }
```

Fichero 4.1: `cuenta.c`

Prueba el siguiente programa con estas instrucciones.

```
1    gcc -no-pie -o compara compara.c
     ./compara
     strace ./compara
4    strace -e trace=open,read,write ./compara
     ltrace ./compara
```

```
1  #include <stdio.h>
   #include <string.h>
   int main() {
4      printf("\nVamos a comparar dos cadenas\n");
       char a[]="hOLA";
       char b[]="hola";
7      int n=strcmp(a,b);
       printf("\nLa diferencia es %d \n",n);
   }
```

Fichero 4.2: `compara.c`

Para saber el efecto *puro* del programa, elimina `ltrace`. Comprueba la diferencia con la anterior, ésta da las llamadas a librerías mientras que `strace` informa de las llamadas al sistema.

Observa los siguientes programas:

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
int main(){
  FILE * f; // fichro
  f=fopen("hola.txt", "w");
  fprintf(f, "Hola... C\n");
  fclose(f);
  return 0; // Para verlo, echo $? en la lÃnea de comandos
}
```

Fichero 4.3: `c-abre.c`

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
int main(){
  int fd; // entrada en la tabla de canales
  fd=open("hola.txt", O_WRONLY|O_CREAT, 0666);
  write(fd, "Hola... S\n",8);
  close(fd);
  return 0; // Para verlo, echo $? en la lÃnea de comandos
}
```

Fichero 4.4: `s-abre.c`

Pruébalos con las siguientes instrucciones. Analiza si dan igual resultado o diferente:

```
gcc -no-pie -o c c-abre.c
./c
strace ./c
ltrace ./c
```

```
gcc -no-pie -o s s-abre.c
./s
strace ./s
ltrace ./s
```

Prueba el programa `compara.c` (el primero de esta sección) con estas instrucciones y justifica por qué no se hace ninguna llamada al sistema entre los dos `printf` pero sí que aparece `strcmp` con `ltrace`.

## 4.2. Enlaces duros y enlaces simbólicos

Observa la información de las instrucciones `stat` y lo que se ve con `ls` al hacer enlaces duros y simbólicos. Observa el tamaño de los ficheros, el número de inodo y el número de enlaces duros. ¿Qué ocurre cuando se borra el fichero `hola.txt` y se crea otro?

```
vi hola.txt
ls -l
```

```
ls -li *.txt # cada cambio que hagamos lo comprobamos con esta instrucción
ln hola.txt kaixo.txt
stat hola.txt
stat kaixo.txt
ls -li *.txt # revisamos los inodos
cat kaixo.txt
cat hola.txt
vi kaixo.txt  # cambiamos el fichero
cat hola.txt  # y se ven los cambios en los dos nombres de fichero
ln kaixo.txt hi.txt # tres enlaces duros
vi hi.txt
ln -s kaixo.txt es1.txt # hacemos enlaces simbólicos
ln -s hola.txt es2.txt
ls -li *.txt
rm hola.txt # un enlace duro menos
vi hola.txt
ls -li \
*.txt   # en dos líneas separadas por un salto (intro justo después de la \ )
history | tail -25 # los últimos 25 comandos
history | tail -25 >e2.txt # los guardamos en un fichero
```

## 4.3. Llamadas al Sistema sobre ficheros

En esta parte vamos a comprobar el comportamiento de estas Llamadas al Sistema.

### `unlink, link`

Tenemos las llamadas al sistema `link` y `unlink`. Ver `man 2 unlink` o también http://man7.org/linux/man-pages/man2/unlink.2.html

A partir de ellas hay que hacer unos programas equivalentes a los comandos:

1. `rm` (borrar)
2. `mv` (mover o renombrar, cambia el camino de acceso a un inodo)
3. `ln` (crea un enlace duro)

Para ello prueba el programa visto en C que muestra los argumentos:

```
#include <stdio.h>
void main(int argc, char *argv[]) {
  int i;
  printf("\nPrograma: %s\n",argv[0]);
  for(i=1;i<argc;i++)
       printf("argumento %d: %s\n",i,argv[i]);
}
```

Si lo llamas `prueba.c` lo ejecutas con:

```
gcc -o prueba prueba.c
./prueba uno dos tres
```

A partir de este programa, para borrar un fichero deberías crear el fichero `borra.c` y usar la *llamada al sistema* `unlink(argv[1]);` como única instrucción del programa, además de `#include <unistd.h>` , la librería que la contiene.

### nice

La & ejecuta el proceso en segundo plano, permite escribir más instrucciones.

```
sleep 4
sleep 4 &
```

La segunda instrucción (`sleep 2000`) cuelga de otro proceso.

```
nice sleep 1000 &
bash -c 'sleep 2000' &
pstree | less
```

Observa cómo actúa `nice` para cambiar la prioridad de un proceso. La *N* indica la menor prioridad. Los corchetes se colocan para que `grep` sólo saque la línea de la ejecución de la instrucción `ps aux`

```
nice ps aux|grep a[u]x
ps aux|grep a[u]x
```

Observa la prioridad absoluta (20 es la normal) y el incremento de 10. Ver https://www.tecmint.com/set-linux-process-priority-using-nice-and-renice-commands/

```
sleep 1000 &
nice -n 20 sleep 2000 &
top
sudo nice -n -20 sleep 3000 &
top
history | tail -20 > nice.txt
```

Si se usa `htop` se puede buscar con pulsando / y después `sleep`.

### Más sobre strace

Prueba estas instrucciones. Para saber su efecto *puro*, elimina `strace`.

```
strace echo Hola
strace -c ls -lR /home/ > salida.txt
strace -r sleep 1

strace -e trace=open,read,write echo Hola
strace -e file echo Hola
diff <(strace -e file echo Hola) <(strace echo Hola)

strace mv f1.txt f2.txt
strace rm f2.txt

strace bash -c 'echo Hola'
strace bash -c 'echo Hola >f1.txt'
```

Más sobre las últimas tres líneas
en DTrace: [even better than] strace for OS X
Un ejemplo de la utilidad de este comando:
Debug like a sysadmin: using strace and ltrace
Para profundizar:
http://www.brendangregg.com/blog/2014-05-11/strace-wow-much-syscall.html

## 4.4. Entrada y salida. Canales

Vamos a ver ejemplos sobre la Tabla de Canales.

### Salida por canales estándares

Salida por los canales 1 y 2 con la sintaxis en C:

```
/* Prueba de distintos canales A */
//      gcc canales.c -o c
//      ./c
//      ./c 2>k2

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

int main(){
  printf("\n1. Canal 1\n");          // Salida estandar
  fprintf(stdout,"\n2. Canal 1\n"); // Salida estandar
  fprintf(stderr,"\n3. Canal 2\n"); // Salida de errores
  exit(0);
}
```

Fichero 4.5: `canalesA.c`

Observa el siguiente programa `psaluda.c` y pruébalo con estos comandos:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void main(int argc, char * argv[]){
  char nombre[100];
  if (argc!=2) {
          fprintf(stderr, "\nNÂº incorrecto de argumentos\n");
    exit(1);
  }
                //  scanf("%s", nombre);
  strcpy(nombre, argv[1]);
  if (strcmp(nombre,"")==0) {
                // if (nombre[0]=='\0')
                // if (*nombre==0)
                // if (!*nombre)
                // if (strlen(nombre)==0)
    fprintf(stderr, "\nError, tu nombre...\n");
        exit(2);
    }
  else
    printf("\nTu nombre es %s\n", nombre);
  exit(0);
}
```

Fichero 4.6: `psaluda.c`

```
vi psaluda.c
gcc -o saluda psaluda.c
./saluda Pablo
echo $?
./saluda a b c d
```

```
echo $?
```

### Redirecciones

Prueba también las redirecciones. Tienes una explicación detallada en los apuntes: https://lsi.vc.ehu.eus/pablogn/docencia/ISO/Apuntes.zip

```
./saluda Pablo > c1.txt 2> c2.txt
cat c1.txt
cat c2.txt
./saluda 'Hola, Pablo' >> c1.txt 2> c2.txt
./saluda > c1.txt 2> c2.txt
./saluda Pablo 2>&1 | grep --colour -n -i n
./saluda 2>&1 | grep --colour -n -i n
```

### Tabla de Canales

Vamos a comprobar cuál es la tabla de canales de un programa que busca `666` en el generador de números aleatorios, reenvía la salida a un fichero y los errores a otro:

```
grep < /dev/random 66666 2> errores.txt > salida.txt
```

Pulsamos CTRL-Z para parar el programa, y comprobamos en la información del Sistema Operativo (a través de `/proc`) cuál es la tabla de canales. Para ello averiguamos el PID del proceso con `ps`.

```
ps
ls -l /proc/17165/fd
```

Y podemos terminar el proceso con `kill %1` o `fg` para traerlo al primer plano y luego CTRL-C para pararlo.

Más para `fg` y `bg`:
https://websistent.com/how-to-manage-jobs-in-linux-fg-bg-kill-ctrlz/

### Ejemplo de utilidad de las redirecciones

Si entendemos canales y redirecciones, podemos ver qué hace este código:

```
python3 -c \
'import sys,yaml,json; json.dump(yaml.load(sys.stdin),sys.stdout,indent=4)' \
< broken.yaml > fixed.json
```

### Llamadas al Sistema dup y dup2

Para ver cómo duplican las entradas en la Tabla de Canales (TC), vamos a probar los programas siguientes. En concreto, `dup` copia el canal indicado en la primera entrada libre de la TC, es decir, la que tiene el valor más bajo.

```
// Primero gcc canales.c -o c despuÃ©s ./c por Ãºltimo cat f
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```c
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

int main(){
  char cad[100];
  printf("\n1. Canal 1\n");    // Salida estandar
  // imprimir en un fichero con Llamadas al sistema
  getchar(); // CTRL-Z para ver la TC, fg y pulsar tecla
  int fd = creat("f", 0666);
  sprintf(cad,"\n2. Canal %d\n",fd);
  write(fd, cad, strlen(cad));
  getchar(); // CTRL-Z para ver la TC, fg y pulsar tecla
  dup2(fd,1);
  // Ahora canales 1 y fd redireccionados al fichero
  getchar(); // CTRL-Z para ver la TC, fg y pulsar tecla
  printf("\n3. Canal 1\n"); // canal 1 y fichero f1.txt
  getchar(); // CTRL-Z para ver la TC, fg y pulsar tecla
  close(fd);
  getchar(); // CTRL-Z para ver la TC, fg y pulsar tecla
  exit(0);
}
```

Fichero 4.7: `canalesB.c`

```c
/* Prueba de distintos canales */
//      gcc canalesC.c -o c2

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>

int main(){
//        The following example creates the file /tmp/file with read and
   write permissions for the file owner and read permission for group and
   others. The resulting file descriptor is assigned to the fd variable.
    close(1);
    int fd;
    mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
    char *filename = "./fich";
    fd = creat(filename, mode);
    printf("\n1. Canal %d\n",fd);
    fflush(stdout);
    close(fd);
    dup(2);
    fprintf(stdout,"\n2. Canal 1 y 2\n");
    exit(0);
}
```

Fichero 4.8: `canalesC.c`

## 4.5. Llamadas al Sistema e Inodos

Si hacemos `man 3 scandir` nos explica cómo mirar los ficheros y directorios de un directorio dado. Dentro tenemos este ejemplo:

```
/* imprimir ficheros en el directorio actual en orden inverso */
#include <dirent.h>
main(){
  struct dirent **namelist;
  int n;
  n = scandir(".", &namelist, 0, alphasort);
  if (n < 0)
    perror("scandir");
  else {
    while(n--) {
      printf("%s\n", namelist[n]->d_name);
      free(namelist[n]);
    }
    free(namelist);
  }
}
```

Si llamamos al código `cdir.c`, compilaremos con `gcc -o cdir cdir.c` y ejecutaremos con `./cdir` pero necesitarás averiguar qué librerías hay que incluir para poder compilar. Usa `man` para encontrar esa información.

Nuestro programa necesita obtener el tamaño en bytes de los ficheros que contiene un directorio, por lo que a partir del ejemplo anterior vamos a hacer un programa que lo haga. Encontramos en internet la página https://en.wikipedia.org/wiki/Stat_%28system_call%29 que nos sirve de base para conocer qué llamadas al sistema necesitamos.

También puedes hacer `man 2 stat` o ver http://man7.org/linux/man-pages/man2/stat.2.html

Mira esta *chuleta* de la línea de comandos.

Compila y ejecuta el programa anterior.

Escribe la modificación del programa anterior que hace la tarea descrita (imprimir los bytes de todos los ficheros) y pon comentarios en el código que expliquen dichas variaciones.