Carlos Buckle
carlos.buckle@gmail.com

Entrada y Salida Estándar

El siguiente programa se llama rw.c. ¿Qué hace? ¿Para qué puede servir?

Tip: Lee de la entrada estándar y escribe en la salida estándar

```c
//
// rw.c - Read Write
// Autor: C. Buckle  2023
//

#include <stdio.h>
main() {

  char ch;

  while ((ch=getchar()) != EOF)
       putchar(ch);

  return 0;

}
```

Ejecutando desde la línea de comandos y redireccionando entrada y/o salida estándar:

Puede servir para crear archivos redireccionando la salida, por ejemplo al archivo “nota.txt”

```
$ rw > nota.txt
```

Puede servir para copiar archivos. Por ejemplo, redireccionando la entrada desde “nota.txt” y redireccionando la salida al nuevo archivo “nota.bak”

```
$ rw < nota.txt > nota.bak
```

Puede servir para imprimir archivos. Por ejemplo, redireccionando la entrada desde “nota.txt” y redireccionando la salida al archivo especial de impresora del sistema

```
$ rw < nota.txt > /dev/pr1
```

Puede servir para enviar un mensaje a otra terminal. Redireccionando la salida a la terminal en cuestión

```
$ rw > /dev/pts/1
```

Si nuestro programa utiliza entrada y salida estandar (STDIN/STDOUT) será un programa “Independiente del dispositivo” y podremos conectarlo a los dispositivos/archivos de entrada/salida que necesitemos

Carlos Buckle
carlos.buckle@gmail.com

Programas Concurrentes

El sistema operativo es quien hace posible que en un mismo momento se estén ejecutando varios programas simultáneamente, en forma "concurrente"

Un programa convencional sólo tiene un "hilo" (thread) de ejecución. Pero podemos escribir programas multi-hilos (multi-thread)

Ejemplo:

```c
//
// collage.c
// Autor: C. Buckle  2023
//
// Ejemplo de concurrencia
// Se disparan 3 threads concurrentes. Cada uno imprime 600 veces una letra
//
// Ejecutar varias veces para ver que no siempre
// se ejecutan en el mismo orden ni con la misma dedicación de tiempo

#include <stdio.h>
#include <pthread.h>

void* imprimir(char* letra) {
    int i;
    for (i = 0; i < 600; i++) {
      putchar(letra);
    }
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t hilo1, hilo2, hilo3;

    // Crea 3 threads
    pthread_create(&hilo1, NULL, imprimir,'A');
    pthread_create(&hilo2, NULL, imprimir,'B');
    pthread_create(&hilo3, NULL, imprimir,'C');

    // Espera que finalicen los threads
    pthread_join(hilo1, NULL);
    pthread_join(hilo2, NULL);
    pthread_join(hilo3, NULL);

    return 0;
}
```

(linkear con librería **pthread**)

Carlos Buckle
carlos.buckle@gmail.com

Problemas de concurrencia

En la ejecución concurrente de programas pueden darse problemas si dichos programas utilizan un recurso compartido (variable, archivo, dispositivo, etc.)

En este ejemplo dos threads comparten la variable “contador”. Cada thread incrementa el valor de esa variable, pero para ello requiere leer el valor actual, sumar 1 a dicho valor y luego asignar el nuevo valor a la variable contador. Cuando los threads intercalan estas instrucciones pueden darse problemas.

Probar el caso con parámetro 10, 1000 y luego 100000 (veremos que en este último caso generalmente pueden darse problemas)

```c
//
// contador.c
// Autor: C. Buckle  2023
//
// Ejemplo de concurrencia y sus problemas
// El programa recibe desde la línea de comandos un parámetro
// El parámetro indica la cantidad de veces que se incrementa el contador
// Se disparan 2 threads concurrentes
// Haga 3 pruebas, pasando parámetro 10, 1000 y 100000

#include <stdio.h>
#include <pthread.h>

int contador = 0;

void* incrementar(int* cant) {
    int i;

    for (i = 0; i < cant; i++) {
        contador = contador + 1;  //contador++
    }
    return NULL;
}


int main(int argc, char *argv[]) {
    pthread_t hilo1, hilo2;

    int vueltas = atoi(argv[1]);

    pthread_create(&hilo1, NULL, incrementar,vueltas);
    pthread_create(&hilo2, NULL, incrementar,vueltas);

    pthread_join(hilo1, NULL);
    pthread_join(hilo2, NULL);

    printf ("La suma final es %i\n",contador);
    return 0;
}
```

(linkear con librería **pthread**)