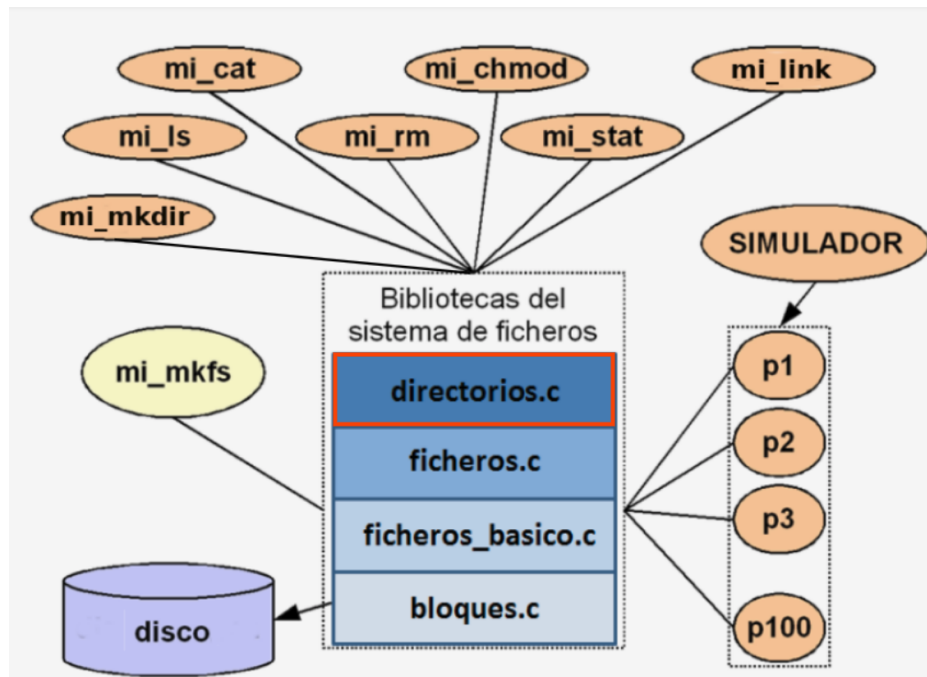


Nivel 12: simulacion.c

En este nivel se trata de crear un programa **simulador** ([simulacion.c](#) y su correspondiente cabecera [simulacion.h](#)), encargado de crear unos **procesos** de prueba que accedan de forma **concurrente** al sistema de ficheros, de modo que se pueda comprobar el correcto funcionamiento de nuestra biblioteca de funciones para más de un proceso en ejecución.

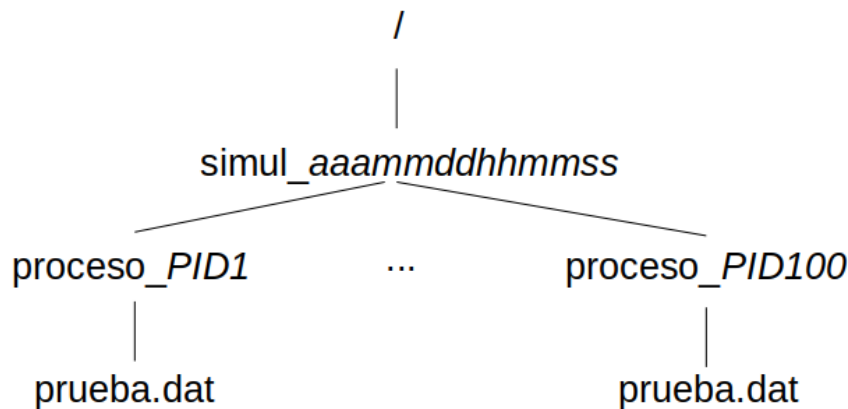


Primeramente se creará el directorio llamado [simul_aaaammddhhmmss](#) (en el directorio raíz) donde [aaaa](#) es el año, [mm](#) es el mes, [dd](#) es el día, [hh](#) es la hora, [mm](#) es el minuto y [ss](#) es el segundo de creación.

Se han de generar **100 procesos** de prueba¹ cada **0,15 segundos**². Cada proceso creará un directorio llamado [proceso_n](#) dentro del directorio [simul_aaaammddhhmmss](#), donde [n](#) es el **PID** del proceso. Además, dentro del directorio "[proceso_n](#)", cada proceso creará un fichero denominado "[prueba.dat](#)".

¹ Recordad que la función [fork\(\)](#) permite crear un nuevo proceso.

² 0,15 segundos = 150 milisegundos = 150.000 microsegundos = 150.000.000 nanosegundos .



Cada **0,05 segundos**³ y un total de **50 veces**, cada proceso escribirá dentro de su fichero "**prueba.dat**" un registro del siguiente tipo, declarado en **simulacion.h**:

```

struct REGISTRO { //sizeof(struct REGISTRO): 24 bytes
    time_t fecha; //Precisión segundos
    pid_t pid; //PID del proceso que lo ha creado
    int nEscritura; //Entero con el número de escritura, de 1 a 50 (orden por tiempo)
    int nRegistro; //Entero con el número del registro dentro del fichero (orden por posición)
};
  
```

El algoritmo detallado a seguir sería el siguiente:

```

En el main() asociar la señal SIGCHLD al enterrador.4
Comprobar la sintaxis del comando. // uso: ./simulacion <disco>
Montar el dispositivo virtual. //padre
Crear el directorio de simulación: /simul_aaammddhmmss/
Para proceso:=1 hasta proceso<=NUMPROCESOS hacer
    pid:=fork()
    Si es el hijo entonces //pid = 0
        Montar el dispositivo. //hijo5
        Crear el directorio del proceso hijo añadiendo el PID al nombre.
        Crear el fichero prueba.dat dentro del directorio anterior.
  
```

³ 0,05 segundos = 50 milisegundos = 50.000 microsegundos = 50.000.000 nanosegundos .

⁴ Recordemos que cuando un proceso finaliza, toda la memoria y recursos asociados con dicho proceso son liberados, pero la entrada del mismo en la tabla de procesos aún existe, para cuando su padre llame a la función `wait()` devolverle el **PID** (Identificador) y el **estado** (un parámetro) del hijo finalizado. Si el padre no realiza el `wait()`, el proceso finalizado entra en estado **zombie**.

⁵ El proceso principal hace el `bmount()` por lo que abre el fichero con el `open()`. Luego hace varios `fork()`s, y los hijos comparten los datos de los descriptores del fichero del padre. También se comparte el puntero (cambiado por el `lseek()`). Un proceso hace el `lseek()` para leer o escribir, otro hace lo mismo y lo deja cambiado, cuando el primero hace su operación de lectura o escritura accede a posiciones diferentes al `lseek` que hizo previamente. Hacer el `bmount()` nuevamente elimina ese problema de concurrencia.

```

Inicializar la semilla de números aleatorios6: srand(time(NULL) + getpid());
Para nescritura:=1 hasta nescritura<=NUMESCRITURAS hacer //50
    Inicializar el registro:
        registro.fecha = time(NULL)
        registro.pid = getpid()
        registro.nEscritura = nescritura
        registro.nRegistro = rand() % REGMAX7 //[0, 499.999]
    Escribir el registro con mi_write() en registro.nRegistro * sizeof(struct REGISTRO).
    Esperar 0,05 seg para hacer la siguiente escritura.
fpara
    Desmontar el dispositivo. //hijo
    exit(0) //Necesario para que se emita la señal SIGCHLD
fsi
    Esperar 0,15 seg para lanzar siguiente proceso.
fpara

//Permitir que el padre espere por todos los hijos8:
Mientras acabados < NUMPROCESOS hacer
    pause();
fmientras

Desmontar el dispositivo //padre
exit(0) // o return 09

```

Función enterrador:

```

void reaper(){
    pid_t ended;
    signal(SIGCHLD, reaper);
    while ((ended=waitpid(-1, NULL, WNOHANG))>0) {
        acabados++;
    }
}

```

Observaciones:

- Se precisan las siguientes cabeceras:

```

#include <sys/wait.h>
#include <signal.h>

```

⁶ Para que genere números diferentes en cada ejecución para obtener los números de registro.

⁷ Nuestro sistema de ficheros nos permitiría un $REGMAX = (((12+256+256^2+256^3)-1)*BLOCKSIZE)/sizeof(struct registro)$ pero en la simulación lo limitaremos a **500.000 registros** (valor de REGMAX).

⁸ Tendremos una variable global, **acabados**, inicializada a 0 para llevar la cuenta del nº de procesos finalizados, y que el **enterrador** irá incrementado.

⁹ Hay discusiones a favor y en contra de la conveniencia en C de usar exit() o return para finalizar el main():

<https://stackoverflow.com/questions/461449/return-statement-vs-exit-in-main>

<https://stackoverflow.com/questions/3463551/what-is-the-difference-between-exit-and-return/3463562>

- Dado que los procesos hijos heredan los descriptors del padre y en cada proceso hijo montamos y desmontamos el dispositivo virtual, tendremos que modificar las funciones `bmount()` y `bumount()` de `bloques.c`:

```
int bmount(const char *camino) {
    if (descriptor > 0) {
        close(descriptor);
    }
    ...
}

int bumount() {
    descriptor = close(descriptor);
    ...
}
```

- Se pueden provocar las **esperas** mediante la función `usleep()`, especificando el tiempo en **microsegundos**. Hay que tener en cuenta que esta función es cancelada por la llegada de alguna señal (por ejemplo cuando un hijo acaba y se envía la señal SIGCHLD), y por eso la ejecución de la simulación dura menos de 15 segundos ($100 \times 0,15$). Si se deseara forzar las esperas con la duración establecida (**que no es nuestro propósito**), se podría usar `nanosleep()`, especificando el tiempo en **nanosegundos**, de la siguiente manera:

```
void my_sleep(unsigned msec) { //recibe tiempo en milisegundos
    struct timespec req, rem;
    int err;
    req.tv_sec = msec / 1000; //conversión a segundos
    req.tv_nsec = (msec % 1000) * 1000000; //conversión a nanosegundos
    while ((req.tv_sec != 0) || (req.tv_nsec != 0)) {
        if (nanosleep(&req, &rem) == 0)
            // rem almacena el tiempo restante si una llamada al sistema
            // ha sido interrumpida por una señal
            break;
        err = errno;
        // Interrupted; continue
        if (err == EINTR) {
            req.tv_sec = rem.tv_sec;
            req.tv_nsec = rem.tv_nsec;
        }
    }
}
```

- Se aconseja **borrar el dispositivo virtual** antes de crear otro con el mismo nombre y empezar las pruebas con **hasta 3 procesos** que realizaran **de 3 a 10 escrituras** (mostrando la realización de cada una). Luego ir aumentando el nº de procesos y nº de escrituras, e ir comprobando los resultados poco a poco, antes de expandir la simulación a los 100 procesos y 50 escrituras.

Ejemplo de testing¹⁰:

```
*** SIMULACIÓN DE 3 PROCESOS REALIZANDO CADA UNO 10 ESCRITURAS ***
[simulación.c → Escritura 1 en /simul_20210524104218/proceso_9779/prueba.dat]
[simulación.c → Escritura 2 en /simul_20210524104218/proceso_9779/prueba.dat]
[simulación.c → Escritura 3 en /simul_20210524104218/proceso_9779/prueba.dat]
[simulación.c → Escritura 4 en /simul_20210524104218/proceso_9779/prueba.dat]
[simulación.c → Escritura 1 en /simul_20210524104218/proceso_9780/prueba.dat]
[simulación.c → Escritura 5 en /simul_20210524104218/proceso_9779/prueba.dat]
[simulación.c → Escritura 2 en /simul_20210524104218/proceso_9780/prueba.dat]
[simulación.c → Escritura 6 en /simul_20210524104218/proceso_9779/prueba.dat]
[simulación.c → Escritura 3 en /simul_20210524104218/proceso_9780/prueba.dat]
[simulación.c → Escritura 7 en /simul_20210524104218/proceso_9779/prueba.dat]
[simulación.c → Escritura 4 en /simul_20210524104218/proceso_9780/prueba.dat]
[simulación.c → Escritura 8 en /simul_20210524104218/proceso_9779/prueba.dat]
[simulación.c → Escritura 1 en /simul_20210524104218/proceso_9781/prueba.dat]
[simulación.c → Escritura 5 en /simul_20210524104218/proceso_9780/prueba.dat]
[simulación.c → Escritura 9 en /simul_20210524104218/proceso_9779/prueba.dat]
[simulación.c → Escritura 2 en /simul_20210524104218/proceso_9781/prueba.dat]
[simulación.c → Escritura 6 en /simul_20210524104218/proceso_9780/prueba.dat]
[simulación.c → Escritura 10 en /simul_20210524104218/proceso_9779/prueba.dat]
[simulación.c → Escritura 3 en /simul_20210524104218/proceso_9781/prueba.dat]
[simulación.c → Escritura 7 en /simul_20210524104218/proceso_9780/prueba.dat]
[Proceso 1: Completadas 10 escrituras en /simul_20210524104218/proceso_9779/prueba.dat]
[simulación.c → Escritura 4 en /simul_20210524104218/proceso_9781/prueba.dat]
[simulación.c → Escritura 8 en /simul_20210524104218/proceso_9780/prueba.dat]
[simulación.c → Escritura 5 en /simul_20210524104218/proceso_9781/prueba.dat]
[simulación.c → Escritura 9 en /simul_20210524104218/proceso_9780/prueba.dat]
[simulación.c → Escritura 6 en /simul_20210524104218/proceso_9781/prueba.dat]
[simulación.c → Escritura 10 en /simul_20210524104218/proceso_9780/prueba.dat]
[simulación.c → Escritura 7 en /simul_20210524104218/proceso_9781/prueba.dat]
[Proceso 2: Completadas 10 escrituras en /simul_20210524104218/proceso_9780/prueba.dat]
[simulación.c → Escritura 8 en /simul_20210524104218/proceso_9781/prueba.dat]
[simulación.c → Escritura 9 en /simul_20210524104218/proceso_9781/prueba.dat]
[simulación.c → Escritura 10 en /simul_20210524104218/proceso_9781/prueba.dat]
[Proceso 3: Completadas 10 escrituras en /simul_20210524104218/proceso_9781/prueba.dat]
```

¹⁰ Se ha limitado a 3 procesos y 10 escrituras/proceso para ver el intercalado. Se ha utilizado un color para cada proceso para facilitar el seguimiento visual de la ejecución.

- Una vez testado el funcionamiento para unos pocos procesos y unas pocas escrituras podéis **eliminar las impresiones entre corchetes de cada escritura** y lanzarlo, para los 100 procesos con 50 escrituras cada una, con el comando **time**¹¹ delante para contrastar vuestro tiempo de ejecución. **Es imprescindible la impresión del nombre del directorio de simulación para la posterior verificación de las escrituras ya que lo necesitaremos como parámetro si introducimos los comandos a mano en la consola.**

```
uib:~$ ./mi_mkfs disco 100000
```

```
uib:~$ time ./simulacion disco
```

```
*** SIMULACIÓN DE 100 PROCESOS REALIZANDO CADA UNO 50 ESCRITURAS ***
```

```
[Proceso 1: Completadas 50 escrituras en /simul_20220525155523/proceso_30788/prueba.dat]
[Proceso 2: Completadas 50 escrituras en /simul_20220525155523/proceso_30797/prueba.dat]
[Proceso 3: Completadas 50 escrituras en /simul_20220525155523/proceso_30798/prueba.dat]
[Proceso 4: Completadas 50 escrituras en /simul_20220525155523/proceso_30799/prueba.dat]
[Proceso 5: Completadas 50 escrituras en /simul_20220525155523/proceso_30800/prueba.dat]
[Proceso 6: Completadas 50 escrituras en /simul_20220525155523/proceso_30801/prueba.dat]
[Proceso 7: Completadas 50 escrituras en /simul_20220525155523/proceso_30802/prueba.dat]
[Proceso 8: Completadas 50 escrituras en /simul_20220525155523/proceso_30803/prueba.dat]
[Proceso 9: Completadas 50 escrituras en /simul_20220525155523/proceso_30804/prueba.dat]
[Proceso 10: Completadas 50 escrituras en /simul_20220525155523/proceso_30805/prueba.dat]
[Proceso 11: Completadas 50 escrituras en /simul_20220525155523/proceso_30806/prueba.dat]
[Proceso 12: Completadas 50 escrituras en /simul_20220525155523/proceso_30807/prueba.dat]
[Proceso 13: Completadas 50 escrituras en /simul_20220525155523/proceso_30808/prueba.dat]
[Proceso 14: Completadas 50 escrituras en /simul_20220525155523/proceso_30809/prueba.dat]
[Proceso 15: Completadas 50 escrituras en /simul_20220525155523/proceso_30810/prueba.dat]
[Proceso 16: Completadas 50 escrituras en /simul_20220525155523/proceso_30811/prueba.dat]
[Proceso 17: Completadas 50 escrituras en /simul_20220525155523/proceso_30812/prueba.dat]
[Proceso 18: Completadas 50 escrituras en /simul_20220525155523/proceso_30813/prueba.dat]
[Proceso 19: Completadas 50 escrituras en /simul_20220525155523/proceso_30816/prueba.dat]
[Proceso 20: Completadas 50 escrituras en /simul_20220525155523/proceso_30817/prueba.dat]
[Proceso 21: Completadas 50 escrituras en /simul_20220525155523/proceso_30818/prueba.dat]
[Proceso 22: Completadas 50 escrituras en /simul_20220525155523/proceso_30819/prueba.dat]
[Proceso 23: Completadas 50 escrituras en /simul_20220525155523/proceso_30820/prueba.dat]
[Proceso 24: Completadas 50 escrituras en /simul_20220525155523/proceso_30821/prueba.dat]
[Proceso 25: Completadas 50 escrituras en /simul_20220525155523/proceso_30822/prueba.dat]
[Proceso 26: Completadas 50 escrituras en /simul_20220525155523/proceso_30823/prueba.dat]
[Proceso 27: Completadas 50 escrituras en /simul_20220525155523/proceso_30824/prueba.dat]
[Proceso 28: Completadas 50 escrituras en /simul_20220525155523/proceso_30825/prueba.dat]
[Proceso 29: Completadas 50 escrituras en /simul_20220525155523/proceso_30828/prueba.dat]
```

¹¹ Time muestra 3 tiempos:

- el **real** de ejecución: es el tiempo total transcurrido desde que ha invocado el comando (incluyendo intervalos de tiempo utilizados por otros procesos y tiempos de espera para E/S). Se le denomina a veces como **tiempo de reloj**, porque es tiempo que ha transcurrido en nuestro reloj.
- el de **usuario**: es sólo el tiempo de CPU utilizado en la ejecución del proceso.
- el de **sistema**: es la cantidad de tiempo consumido en el kernel, que es el tiempo empleado en contestar peticiones del sistema.

[illegible]

```
[Proceso 84: Completadas 50 escrituras en /simul_20220525155523/proceso_30896/prueba.dat]
[Proceso 86: Completadas 50 escrituras en /simul_20220525155523/proceso_30898/prueba.dat]
[Proceso 87: Completadas 50 escrituras en /simul_20220525155523/proceso_30899/prueba.dat]
[Proceso 85: Completadas 50 escrituras en /simul_20220525155523/proceso_30897/prueba.dat]
[Proceso 88: Completadas 50 escrituras en /simul_20220525155523/proceso_30900/prueba.dat]
[Proceso 91: Completadas 50 escrituras en /simul_20220525155523/proceso_30903/prueba.dat]
[Proceso 90: Completadas 50 escrituras en /simul_20220525155523/proceso_30902/prueba.dat]
[Proceso 92: Completadas 50 escrituras en /simul_20220525155523/proceso_30904/prueba.dat]
[Proceso 89: Completadas 50 escrituras en /simul_20220525155523/proceso_30901/prueba.dat]
[Proceso 93: Completadas 50 escrituras en /simul_20220525155523/proceso_30905/prueba.dat]
[Proceso 94: Completadas 50 escrituras en /simul_20220525155523/proceso_30908/prueba.dat]
[Proceso 95: Completadas 50 escrituras en /simul_20220525155523/proceso_30909/prueba.dat]
[Proceso 96: Completadas 50 escrituras en /simul_20220525155523/proceso_30910/prueba.dat]
[Proceso 98: Completadas 50 escrituras en /simul_20220525155523/proceso_30912/prueba.dat]
[Proceso 97: Completadas 50 escrituras en /simul_20220525155523/proceso_30911/prueba.dat]
[Proceso 99: Completadas 50 escrituras en /simul_20220525155523/proceso_30913/prueba.dat]
[Proceso 100: Completadas 50 escrituras en /simul_20220525155523/proceso_30914/prueba.dat]
```

```
real    0m11,855s12
user    0m0,376s
sys     0m0,391s
```

- **Observación:** Un proceso puede acabar antes que otro iniciado antes que él.

¹² Con llamadas a `usleep()`. Si se utilizara la función propia `my_sleep()` podría tardar unos 18s y no observaríamos el intercalado de la finalización de procesos.