

ResumFinal.pdf



abcd_1234



Sistemas Operativos



2º Grado en Ingeniería Informática



Facultad de Informática de Barcelona (Fib)
Universidad Politécnica de Catalunya



[Accede al documento original](#)



Escuela de
Organización
Industrial

Contigo que evolucionas.
Contigo que lideras. Contigo que transformas.

Esto es EOI.
Mismo propósito,
nueva energía.



Descubre más aquí



EOI Escuela de
Organización
Industrial

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio



Necesito concentración

ali ali oohh
esto con 1 coin me
lo quito yo...

WUOLAH

SO: Resumen Final

T3 - Memoria

Reservar / Liberar memoria dinámica

Llamada a sistema:

```
sbrk(tamaño_variación_heap);
```

- > 0 → aumenta el heap
- < 0 → reduce el hep
- $= 0$ → no se modifica el heap

//ejemplo:

```
int main(int argc, char *argv[]) {
    int num_procs = atoi(argv[1]);
    int *pids;
    pids = sbrk(num_procs * sizeof(int)); // reservamos
    for(int i=0; i<num_procs; i++) {
        pids[i] = fork();
        if(pids[i] == 0) {
            // codigo hijo
        }
    }
    sbrk(-1 * num_procs * sizeof(int)); // liberamos el
}
```

Librería en C:

```
malloc(tamaño_reservado);
```

//ejemplo:

```

int main(int argc, char *argv[]) {
    int num_procs = atoi(argv[1]);
    int *pids;
    pids = malloc(num_procs * sizeof(int)); // reservamos espacio
    for(int i=0; i<num_procs; i++) {
        pids[i] = fork();
        if(pids[i] == 0) {
            // codigo hijo
        }
    }
    free(pids); // liberamos el espacio reservado por pids
}

```

Asignación de Memoria

Conceptos:

- **Fragmentación Interna:** Ocurre cuando un proceso reserva un espacio de memoria que no acaba utilizando por completo y quedan bytes libres sin uso. Otro proceso puede quedarse sin memoria libre porque hay memoria reservada sin usarse.
- **Fragmentación externa:** Ocurre cuando un proceso dinámicamente va reservando espacios de memoria repartidos en varias direcciones. Otro proceso puede quedarse sin espacio continuo suficiente.
- **Paginación:**
 - Espacio dividido en bloques fijos (páginas)
 - Memoria física dividida en particiones del mismo tamaño (marcos)
 - MMU, Tabla de Páginas, TLB

Puede haber fragmentación interna si hace falta reservar una página para un espacio menor que esta.

- **Segmentación:** Espacio dividido en segmentos de tamaño variable ajustado a lo que se necesite.

- Puede haber fragmentación externa.

- **Esquema mixto:**
 - Combina paginación con segmentación
 - Espacio lógico del proceso dividido en segmentos
 - Segmentos divididos en páginas

Optimizaciones:

- **COW (Copy on Write):** cuando se hace un fork, en vez de copiar todo el espacio de memoria de variables y código, se aprovecha el que ya se ha reservado para el padre para leer variables, el mismo código, etc. y sólo cuando se va a hacer una modificación de alguna variable o se va a reservar más espacio se hace una copia de el entorno para ese hijo que lo está haciendo.
- **Memoria Virtual:**
 - "Sacar de memoria los espacios que no están bajo demanda
 - Permite mejor concurrencia de procesos en ejecución
 - **Swapping:** Mover bloques de memoria de procesos que no se están ejecutando a disco cuando el proceso en ejecución necesite más espacio (swap area). Para seleccionar un proceso "víctima se usan técnicas como el LRU.

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio



Necesito concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH

- **Thrashing:** Ocurre cuando un proceso consume la mayoría o gran parte de su tiempo de ejecución en resolver fallos de página cuando las intenta recuperar del swap area. Solución: Memoria Prefetch → Anticipar las páginas que va a necesitar el siguiente proceso.

T4 - Entrada / Salida

Dispositivos

Todos los dispositivos (físicos, virtuales, y lógicos) se tratan de la misma forma. Esto permite el uso de las mismas llamadas a sistema independientemente del tipo.

Los procesos tienen 3 salidas estándar:

- stdin → 0
- stdout → 1
- stderr → 2

Dispositivo Virtual: Aísla al usuario de la complejidad de gestión de dispositivos físicos. Representación en el sistema operativo: `/dev/dispX`

Dispositivo Lógico: Establece la relación entre un dispositivo físico y virtual. Establece los premisos del dispositivo. Se asocia a un archivo.

Dispositivo Físico: Traduce las comunicaciones adaptadas al dispositivo en específico.

Drivers: Permiten comunicarse y controlar un dispositivo de hardware específico.

Creación de Dispositivos

- Compilar el Driver de Dispositivo DD. →
- Instalar el driver: `insmod archivo_driver`
- Crear dispositivo lógico: `mknod /dev/dispX c menor mayor`
- Crear dispositivo virtual: `open("/dev/dispX", ...);`

```
mknod nombre [c | b] mayor menor
```

- nombre → nombre del dispositivo.
- [c | b]
 - c → dispositivo de tipo carácter.
 - b → dispositivo de tipo bloque.
- p → Pipe (no hace falta mayor o menor).
- mayor → identificador de la clase de dispositivo.
- menor → identificador de la instancia del dispositivo de esa clase.

Inodos

Estructura de datos que almacena metadatos de archivos y apunta a su contenido.

Contienen: Tamaño, Tipo, Permisos de acceso, Propietario y Grupo, Tiempos de acceso al archivo, Numero de links y Punteros a su contenido.




























Llamadas a Sistema

```
fd = open(char* pathname, int flags, mode_t mode);
```

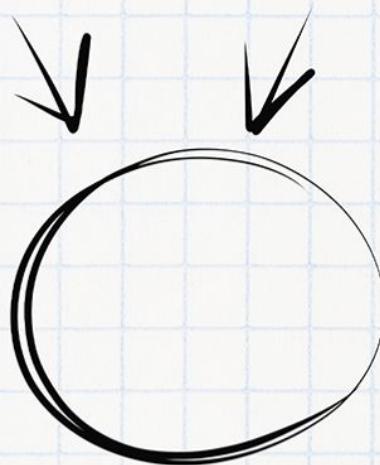
- Abre o crea un archivo.
- Devuelve un File Descriptor del archivo que abrimos.
- Usage: (**abrir archivo**)
 - pathname: "archivo.txt"
 - flags → si hay más de uno se ponen con una OR |
 - O_RDONLY → Permisos de lectura.
 - O_WRONLY → Permisos de escritura.
 - O_RDWR → Permisos de lectura y escritura.
 - O_TRUNC → truncar contenido del archivo.
 - O_APPEND → añadir al contenido del archivo.
- Usage: (**crear archivo**)
 - pathname: "archivo.txt"

Imagínate aprobando el examen

Necesitas tiempo y concentración

Planes	 PLAN TURBO	 PLAN PRO	 PLAN PRO+
 Descargas sin publi al mes	10 	40 	80 
 Elimina el video entre descargas			
 Descarga carpetas			
 Descarga archivos grandes			
 Visualiza apuntes online sin publi			
 Elimina toda la publi web			
 Precios Anual <input type="checkbox"/>	0,99 € / mes	3,99 € / mes	7,99 € / mes

Ahora que puedes conseguirlo,
¿Qué nota vas a sacar?



WUOLAH

- flags
O_RDONLY , O_WRONLY , O_RDWR , O_TRUNC , O_APPEND
O_CREAT → crear archivo si no existe.
- mode → si hay más de uno se ponen con una OR |
S_IRWXU → Permisos de lectura, escritura y ejecución para el usuario.
S_IRUSR → Permisos de lectura para el usuario.
S_IWUSR → Permisos de escritura para el usuario.
0600 → COMÚN: Permisos de lectura y escritura para el usuario.

```
bytes_read = read(int fd , type buf , size_t count );
```

- Lee contenido de archivos.
- Devuelve el numero de bytes leídos.
- Usage:
 - fd → File Descriptor del archivo que queremos leer.
 - buf → Variable en la que guardamos el contenido leído.
 - count → Tamaño en bytes de lo que vamos a leer.

```
bytes_written = write(int fd, type buf, size_t count);
```

- Escribe en archivos.
- Devuelve el numero de bytes escritos.
- Usage:
 - fd → File Descriptor del archivo que queremos escribir.
 - buf → Variable con el contenido a escribir.
 - count → Tamaño en bytes de lo que vamos a escribir.

```
newFd = dup(fd);
```

- Crea una copia del file descriptor al primer espacio disponible en la tabla de fds.
- Devuelve el nuevo File Descriptor.

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins?

Plan Turbo: barato

Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH

- Usage:

- fd → File Descriptor que queremos duplicar.

```
newFd = dup2(fd, newFd);
```

- Cierra fd y reemplaza su posición en la tabla de fds por newFd.
- "Remapeamos el canal de newFd.
- Devuelve el nuevo File Descriptor.

- Usage:

- fd → File Descriptor que queremos cerrar y reemplazar.
- newFd → Nuevo File Descriptor que queremos asignar.

```
close(int fd);
```

- Cierra el fd.

- Usage:

- fd → File Descriptor que queremos cerrar.

```
pipe(int[2] vector_fd);
```

- Crea una pipe sin nombre y asigna sus File Descriptors a vector_fd.
- Pipe accesible por procesos hijo si se ha declarado antes del fork().
- Usage:

- vector_fd → Vector al que asignamos los File Descriptors de la pipe
 - vector_fd[0] → File Descriptor de lectura de la pipe
 - vector_fd[1] → File Descriptor de escritura de la pipe

```
mknod(char* pipe_name, mode_t mode, dev_t dev);
```

- Crea una pipe con nombre.
- Todos los procesos pueden acceder a la pipe.
- Los procesos tienen que usar `open()` para obtener un File Descriptor para la pipe y `read()` y `write()` para escribir.

- Usage:
 - pipe_name → String con el nombre de la pipe que queremos crear.
 - mode → Separamos estos dos parámetros con una OR |
 - S_IFIF O → Especificamos tipo named pipe.
 - 0600 → Permisos para el usuario.
 - dev → 0

```
n = lseek(int fd, off_t offset, int origin);
```

- Mueve el puntero fd tantos bytes como indica el offset.
- Devuelve la posición a la que se ha movido en el archivo.
- Usage:
 - fd → File Descriptor que quedemos mover.
 - offset → Bytes que queremos avanzar.
 - origin → Posición de referencia.
 - SEEK_SET → fd apunta al inicio del archivo + offset.
 - SEEK_CURR → Se hace el offset a partir de fd.
 - SEEK_END → fd apunta al final del archivo + offset.

Si el fd ya apunta a un archivo antes de hacer `fork()` y tanto el padre como el hijo

usan el fd, los movimientos con

lseek, read y write modifican el valor de fd en los dos procesos de forma cruzada.

Hacer `close(fd)` en uno de los dos ficheros no afecta al otro, moverlo sí.

Esto ocurre porque al hacer `fork()` se copia la **FDT** (File Descriptor Table) con los file descriptors apuntando a la misma posición en la tabla **FT** (File Table). Los cambios en el offset se comparten dado que se configura en la **FT**, mientras que los File Descriptors se cierran en la FDT asociada a cada proceso que inicialmente era un duplicado pero que no se comparte.

Pipes

Lectura

Mientras haya contenido escrito en la pipe se lee el contenido.

Si hay un escritor en la pipe, el proceso en lectura quedará bloqueado hasta que:

- El proceso escribiendo escriba más y se lea.
- El proceso escribiendo cierre su File Descriptor.

Es importante cerrar los File Descriptors de escritura cuando ya no se vayan a usar más.

Escritura

Si hay espacio para escribir datos se escribe en la pipe.

Si **NO** hay espacio el proceso se quedará en estado **SLEEP** hasta que se vacíe la pipe.

Si no hay lectores el proceso recibe un **SIGPIPE**.