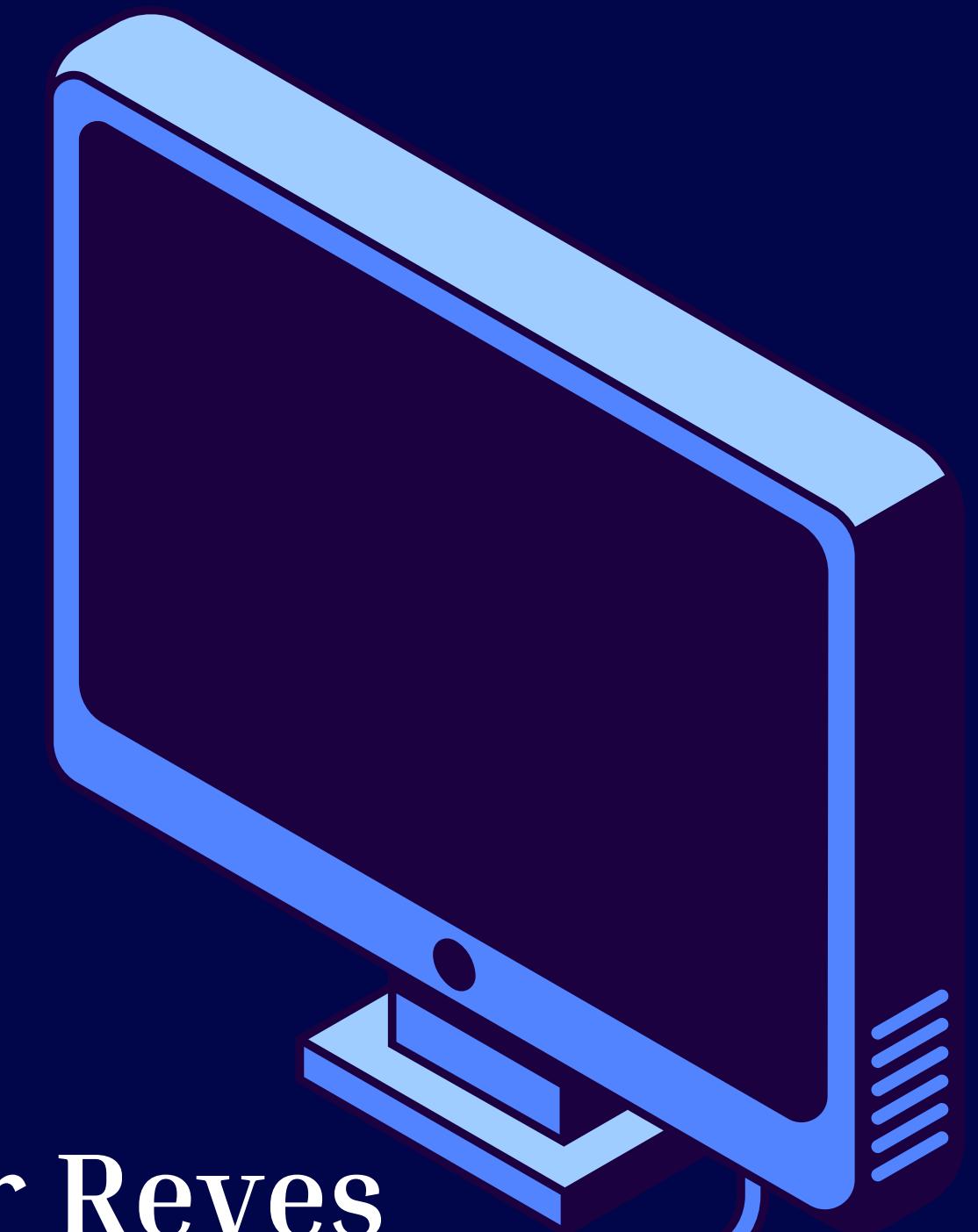


Sistemas Operativos

Ayudantía 3: Pipes, Scheduling y Tarea 1

Profesores: Martín Gutierrez y Víctor Reyes

Ayudantes: Diego Banda y Dante Hortuvia



Contacto

diego.banda@mail_udp.cl



darKlouds

github.com/DiegoBan/S02025-1



dante.hortuvia@mail_udp.cl

doshuertos



Variables entre procesos

```
int main(){
    int contador = 0;
    pid_t pid = fork();

    if(pid > 0){
        for(int i = 0;i < 10 ; i++){
            contador++;
            printf("Soy el proceso padre y llevo la cuenta de %d\n",contador);
        }
    }else if(pid == 0){
        for(int i = 0;i < 10 ; i++){
            contador++;
            printf("Soy el proceso Hijo y llevo la cuenta de %d\n", contador);
        }
    }
    return 0;
}
```

Cada proceso tiene un contador propio, que no afecta al otro.

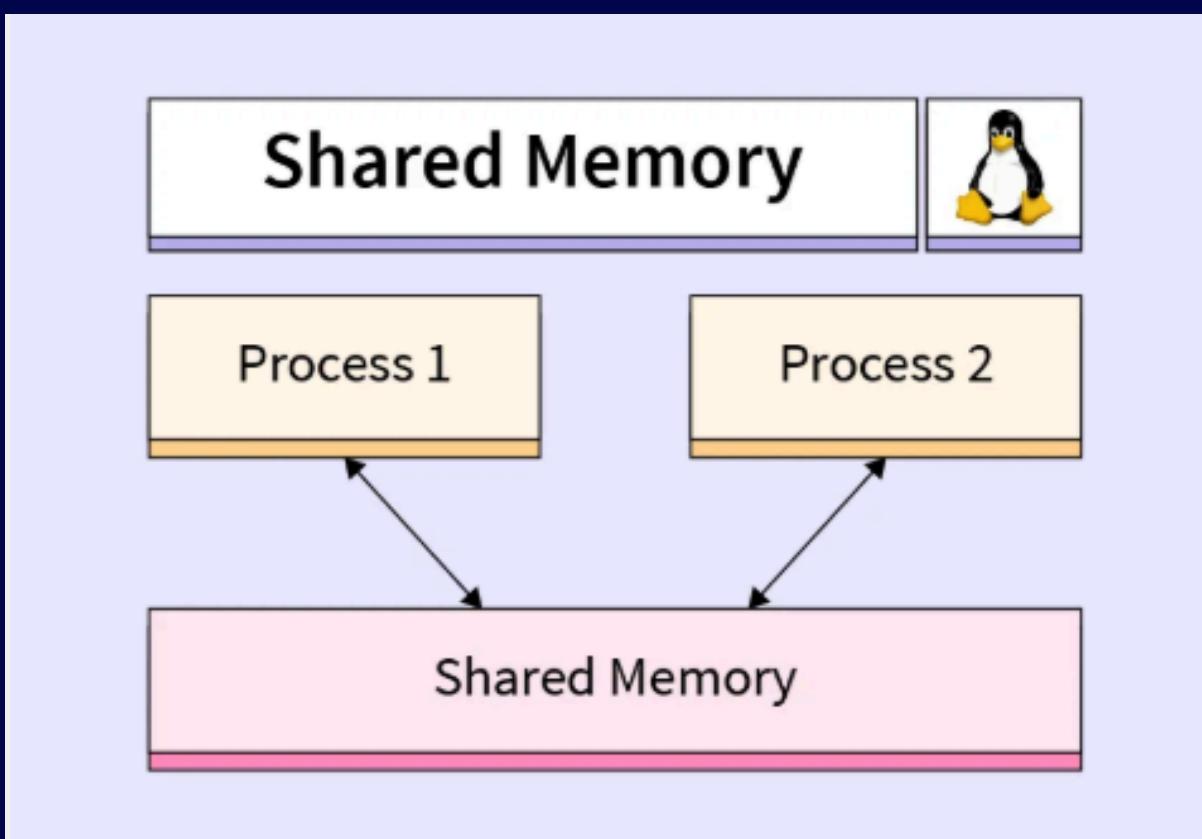
¿Los procesos padre e hijo comparten el contador? ¿O cada uno tiene uno propio?

```
Soy el proceso padre y llevo la cuenta de 1
Soy el proceso Hijo y llevo la cuenta de 1
Soy el proceso padre y llevo la cuenta de 2
Soy el proceso Hijo y llevo la cuenta de 2
Soy el proceso Hijo y llevo la cuenta de 3
Soy el proceso padre y llevo la cuenta de 3
Soy el proceso Hijo y llevo la cuenta de 4
Soy el proceso padre y llevo la cuenta de 4
Soy el proceso Hijo y llevo la cuenta de 5
Soy el proceso padre y llevo la cuenta de 5
```

¿Cómo Podemos compartir variables?

Mediante Memoria Compartida:

Una forma de comunicación entre los procesos , donde se permite compartir memoria entre ellos.



- `shmget`: se utiliza para crear o acceder a un espacio de memoria compartida, necesita una llave, un tamaño y opciones.
- `shmid` es el id de la memoria.
- `shmaddr`: se utiliza para especificar la dirección de la memoria, donde se conecta el proceso.
- `shmctl` se utiliza para realizar operaciones de control en la memoria.
- `shmdt` desconecta la memoria de un proceso.

Ejemplo Memoria compartida

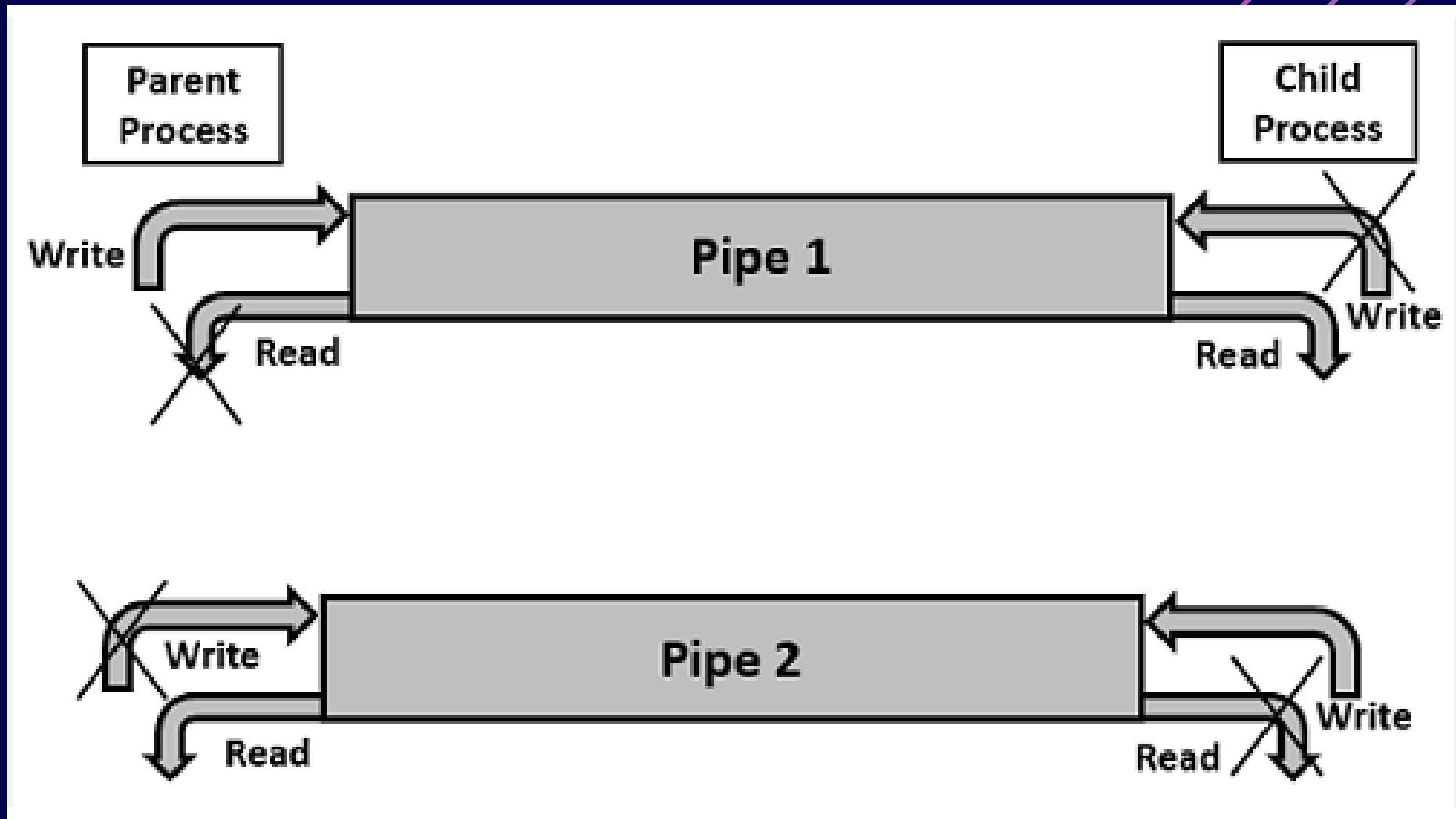
```
key_t key = 1234; //Se establece llave del espacio de memoria  
int shmid; //espacio del memoria  
  
shmid = shmget(key, sizeof(int), IPC_CREAT | 0666);  
if(shmid == -1) { //Manejo de errores  
    perror("shmget");  
    exit(EXIT_FAILURE);  
}  
int *contador = (int*) shmat(shmid, NULL, 0); //Creacion de la variable  
if(contador == (int*) -1) { // manejo de errores  
    perror("shmat");  
    exit(1);  
}  
*contador = 0; //Inicio en 0  
  
for(int i = 0; i < 10 ; i++){  
    (*contador)++;  
    printf("Soy el proceso padre y llevo la cuenta de %d\n", *contador);  
}  
else if(pid == 0){  
    for(int i = 0; i < 10 ; i++){  
        (*contador)++;  
        printf("Soy el proceso Hijo y llevo la cuenta de %d\n", *contador);  
    }  
    if(shmdt(contador) == -1){  
        perror("shmdt");  
        exit(EXIT_FAILURE);  
    }  
    if(shmctl(shmid, IPC_RMID, NULL) == -1){  
        perror("shmctl");  
        exit(EXIT_FAILURE);  
    }  
}
```

Resultado

Soy el proceso padre y llevo la cuenta de 1
Soy el proceso Hijo y llevo la cuenta de 2
Soy el proceso padre y llevo la cuenta de 3
Soy el proceso Hijo y llevo la cuenta de 4
Soy el proceso padre y llevo la cuenta de 5
Soy el proceso Hijo y llevo la cuenta de 6
Soy el proceso padre y llevo la cuenta de 7
Soy el proceso Hijo y llevo la cuenta de 8
Soy el proceso padre y llevo la cuenta de 9
Soy el proceso Hijo y llevo la cuenta de 10
Soy el proceso padre y llevo la cuenta de 11
Soy el proceso Hijo y llevo la cuenta de 12

Pipes

Pipes anónimos: Tienen una comunicación unidireccional entre dos procesos, se utiliza entre los procesos padre e hijo.



- Para crear pipes se hace atreves de la syscall pipe(), se pasa un array de dos elementos tipo int.
- se utiliza 0 para leer (READ) y 1 para escribir (WRITE).
- Las dos syscall necesitan : Archivo que represente al pipe. Espacio de memoria. Tamaño.

Ejemplo Pipes sin nombre

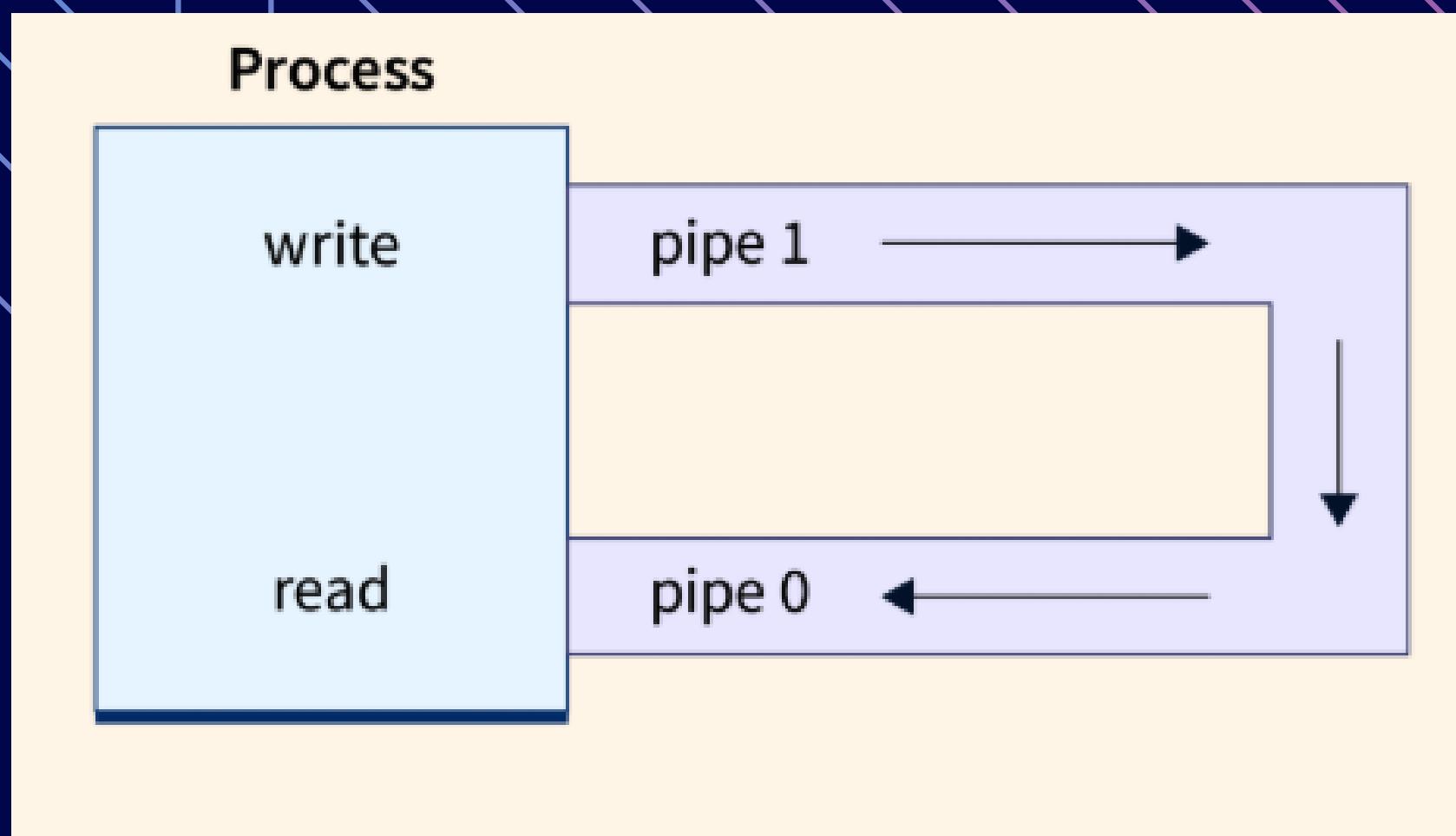
```
int main(){
    int fd1[2];
    pid_t pid;
    pipe(fd1); // Creacion de pipes
    pid = fork();

    if(pid == 0){
        close(fd1[1]);
        char buffer[100];
        int Buffer_read = read(fd1[0], buffer, sizeof(buffer));
        if(Buffer_read == -1){ // Manejo de errores
            perror("READ PIPE");
            exit(-1);
        }
        buffer[Buffer_read] = '\0'; // señal de termino
        printf("Llegaron las %s\n",buffer);
    } else {
        close(fd1[0]);
        char *mensaje = "Pipshass";
        ssize_t bytes_escrito = write(fd1[1],mensaje ,strlen(mensaje));
        if(bytes_escrito == -1){
            perror("WRITE PIPE");
            exit(-1);
        }
        printf("Enviado\n");
        close(fd1[1]);
    }
}
```



Pipes con nombre

Pipes FIFO (Con nombre) : Tienen una comunicación bidireccional, no requieren una ser padre e hijo.



- Se crean con la syscall mkfifo(), se le tiene que dar un nombre y respectivos permisos en octal.
- Tiene read() y write()
- Eliminar al final de la ejecucion el pipe, ya que este permanece luego de haber finalizado la ejecución.



**Deben utilizarlo
en la tarea**

Ejemplo Pipes con Nombre

```
int main(){
    const char *fifo_path = "./my_fifo2"; // Nombre de la ruta del pipe

    int fd;
    int num;

    mkfifo(fifo_path, 0666); // Creacion del pipe
    fd = open(fifo_path, O_WRONLY); // Abertura del pipe

    printf("Ingrese un numero para multiplicar por 100\n");
    scanf("%d", &num);

    write(fd, &num, sizeof(num)); // Envio de mensaje
    close(fd);
    unlink(fifo_path);

    return 0 ;
}
```

Código 1

```
int main(){
    const char *fifo_path = "./my_fifo2"; // Nombre de la ruta del pipe

    int fd;
    int num;

    mkfifo(fifo_path, 0666); // Creacion del pipe
    fd = open(fifo_path, O_RDONLY); // Abertura del pipe

    read(fd, &num, sizeof(num)); // recibo del mensaje

    printf("su numero multiplicado por 100 es: %d\n", num);

    close(fd);
    unlink(fifo_path);

    return 0 ;
}
```

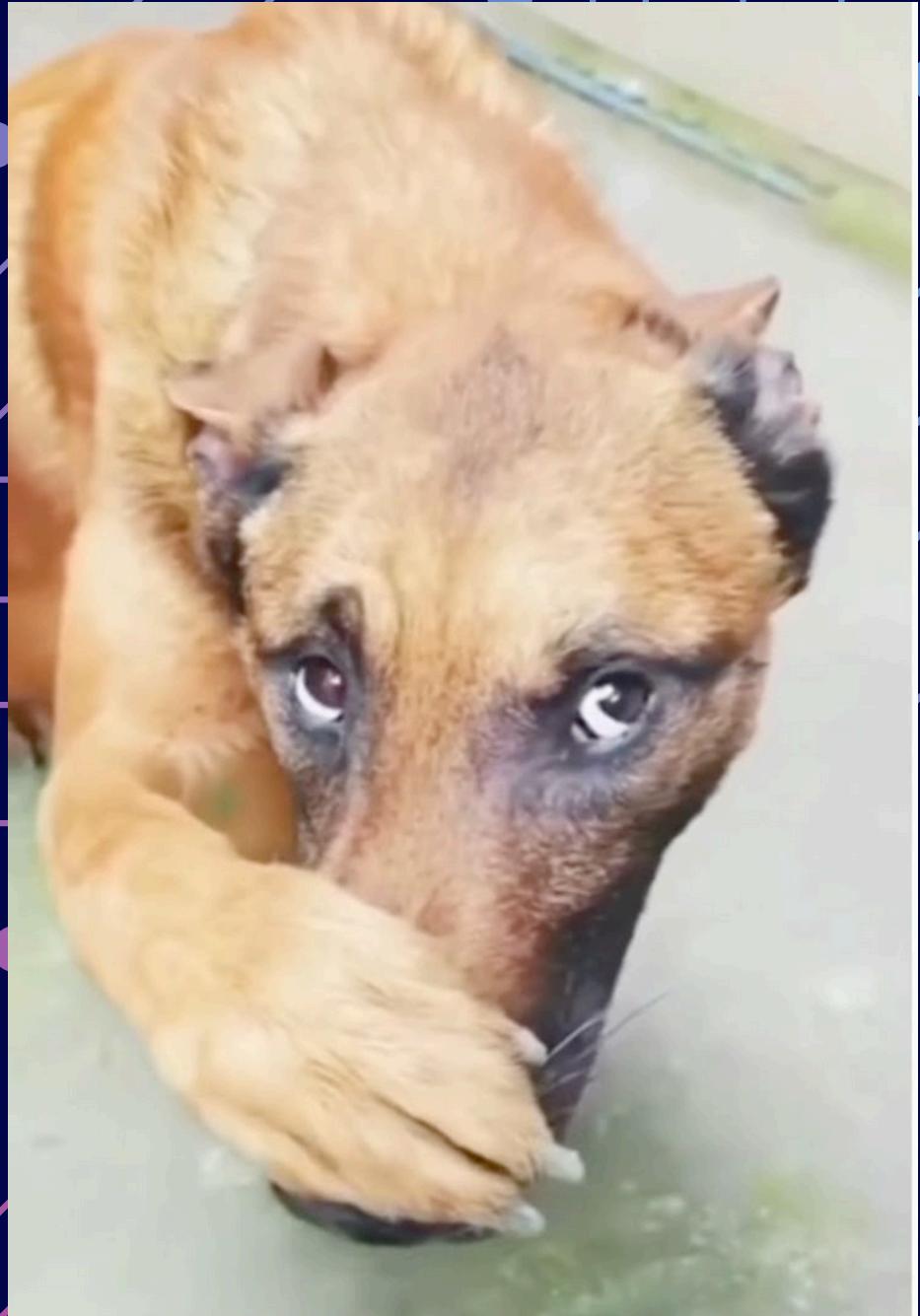
Código 2

Scheduling

Imaginando que se tiene un procesador de un solo núcleo...

Se necesita que cada proceso actual tenga tiempo de CPU.

Si tu PC tiene más programas que núcleos el procesador, ¿cómo parece ejecutar todo simultáneamente?



Scheduling: Recordando...



Scheduling

Long term scheduler

Procesos que quedan en la ready queue, esto define la cantidad también.

Medium term scheduler

Modifica temporalmente el grado de multiprogramación y hace swapping (RAM a HDD y viceversa).

Short term scheduler

Envía procesos de la ready queue a CPU, realiza cambio de contexto.



Tipos de scheduling

- **Scheduling expropiativo:** Puede quitar un proceso que está con tiempo de CPU para darle tiempo de CPU a otro.
- **Scheduling no-expropiativo:** Una vez asignó tiempo de CPU a un proceso, este debe terminar para seguir con otro.
- **Batch Scheduling:** Se ejecutan bajo cierta secuencia.
- **Interactive Scheduling:** Principalmente para dar respuesta rápida entre máquina/usuario y procesos paralelos.
- **Real-time Scheduling:** Responder ante tiempos de término es crítico.

Algoritmos de scheduling

No-Expropiativos

Batch

First come
first served
Shortes job
first

Expropiativos

Real time
Earliest
deadline
first

Interactive

Round
Robin

Interactive

Priority

Ejercicios Scheduling

Considere la siguiente tabla de proceso:

Proceso	CPU Burst	Arrival Time
P_0	3	7
P_1	10	1
P_2	3	0
P_3	6	5

Determinar el Turnaround y end time, considerando para ello el algoritmo Short Job First no expropiativo y Round-Robin con un quantum de 2.

Ejercicios Scheduling.

Considere la siguiente tabla de proceso::

Proceso	Arrival Time	CPU Burst	Deadline
P_0	1	2	28
P_1	6	4	27
P_2	2	7	35
P_3	0	6	32
P_4	7	5	23

Para cada uno de los procesos, determinar el Turnaround y end time, considerando para ello el algoritmo Round-Robin con un quantum de 3.

**¡Gracias
por su
atención!**

