# Comunicación entre procesos(IPC)

## *Stat:*

int ***stat***(const char *nomb_arch, struct stat *buf) -- obtiene información acerca del archivo apuntado por nomb_arch. No se requieren permisos de lectura, escritura o ejecución, pero todos los directorios listados en nomb_arch deberán estar disponibles.

```
1.  struct stat
2.  {
3.      dev_t        st_dev;      /* dispositivo */
4.      ino_t        st_ino;      /* inodo */
5.      mode_t       st_mode;     /* proteccion */
6.      nlink_t      st_nlink;    /* numero de enlaces fisicos */
7.      uid_t        st_uid;      /* ID del usuario propietario */
8.      gid_t        st_gid;      /* ID del grupo propietario */
9.      dev_t        st_rdev;     /* tipo dispositivo (si es
10.                                 dispositivo inodo) */
11.     off_t        st_size;     /* tamaño total, en bytes */
12.     unsigned long st_blksize;  /* tamaño de bloque para el
13.                                   sistema de ficheros de E/S */
14.     unsigned long st_blocks;   /* numero de bloques asignados */
15.     time_t       st_atime;    /* hora ultimo acceso */
16.     time_t       st_mtime;    /* hora ultima modificacion */
17.     time_t       st_ctime;    /* hora ultimo cambio */
18. };
```

## Getrusage:

La función estándar de C **getrusage**, en la librería "sys/resource.h", tiene por objetivo indicar los recursos (tiempo, memoria y otros) consumidos por un proceso. Sin embargo, la implementación en Linux de la librería no rellena la información del uso de memoria.

La estructura rusage es gigante, nosotros usamos los siguientes atributos de la estructura:

```
printf("Tiempo reloj: %ld microsegundos\n", *t_total);

printf("Tiempo CPU sistema total: %ld microsegundos\n", ru->ru_stime.tv_usec);

printf("Tiempo CPU usuario total: %ld microsegundos\n", ru->ru_utime.tv_usec);

printf("Cantidad de Soft Page Faults: %ld \n", ru->ru_minflt);

printf("Cantidad de Hard Page Faults: %ld \n", ru->ru_majflt);

printf("Operaciones de entrada (en bloques): %ld \n", ru->ru_inblock);
```

```
printf("Operaciones de salida (en bloques): %ld \n", ru->ru_oublock);
printf("Mensajes IPC enviados: %ld \n", ru->ru_msgsnd);
printf("Mensajes IPC recibidos: %ld \n", ru->ru_msgrcv);
```

# Fifo:

## Primitiva:

int mkfifo(const char *pathname, mode_t mode);

Esta función recibe dos parámetros: “pathname” indica la ruta en la que queremos crear el FIFO, y “mode” indica el modo de acceso a dicho FIFO. Cualquier proceso es capaz de utilizar un FIFO siempre y cuando tengan los privilegios necesarios para ello.

### Para crearlo:

```
char *fifo1 = "/tmp/fifo1";
mkfifo(fifo1, 0666);
```

### Para eliminarlo:

```
Unlink(nomre_fifo);
```

# Shared Memory

## *Semáforos POSIX:*

**sem_wait**: operación P sobre un semáforo
Decrementa el contador del semáforo. Si el valor previo del semáforo es 0, sem_wait efectúa un bloqueo del proceso llamante hasta que el semáforo sea Incrementado.
Devuelve 0 si el semáforo pudo ser decrementado y -1 en caso de error
**sem_post**: incrementa el valor del contador del semáforo (operación V) .Devuelve 0 si el semáforo pudo ser incrementado y -1 en caso de error

**Se crea asi:**

```
sem_t *sem_x = sem_open("/sem_x", O_CREAT, 0666, 0),
```

**Se cierra asi:**

```
sem_close(sem_x);
```

**se elimina asi:**

```
sem_unlink("/sem_x");
```

## shmget() :

Permite acceder a una zona de memoria compartida y, opcionalmente, crearl a en caso de no existir. A esta llamada se le pasan tres argumentos: una clave, el tamaño del segmento a crear y el flag inicial de operación, y devuelve un entero, denominado shmid, que se utiliza para hacer referencia a dicho segmento.

## shmat():

Es la llamada que debe invocar un proceso para adjuntar una zona de memoria compartida dentro de su espacio de direcciones. Recibe tres parámetros: el identificador del segmento (shmid), una dirección de memoria (shmaddr) y las banderas descritas más adelante(que no usamos en el ejercicio).

## shmctl:

Proporciona una variedad de operaciones para el control de la memoria compartida.
En nuestro caso usamos ipc_rmid

- **IPC_RMID**: Marca una región de memoria compartida como destruida, aunque el borrado sólo se hará efectivo cuando el último proceso que la adjuntaba deje de hacerlo.

# Sockets

Hay varias formas de trabajar con sockets en C:

La primera de ellas es struct sockaddr, la cual contiene información del socket.

```
struct sockaddr
{
  unsigned short sa_family;  /* familia de la dirección */
  char sa_data[14];          /* 14 bytes de la dirección del protocolo */
};
```

Pero, existe otra estructura, struct sockaddr_in, la cual nos ayuda a hacer referencia a los elementos del socket (Esta es la que usamos nosotros ).

```
struct sockaddr_in
{
  short int sin_family;    /* Familia de la Dirección */
  unsigned short int sin_port; /* Puerto                */
  struct in_addr sin_addr;     /* Dirección de Internet      */
  unsigned char sin_zero[8];/* Del mismo tamaño que struct sockaddr*/
};
```

**Nota:** Dice Familia de la Dirección pero se refiere a la familia de sockets o conectores que se estén usando. Recuerden que en clase vimos que hay varios tipos a la hora de implementarlos.