PRACTICAS DE SO

**General**

#include <stdio.h> //Funciones de entrada/salida, como printf o perror

#include <stdlib.h> //exit, malloc o atoi void exit(int status) //Salir

exit(0), exit(EXIT\_SUCCESS) y return(0) exit(-1), exit(1) y exit(EXIT\_FAILURE)

void \*malloc(size\_t size) //Reserva memoria (malloc(5 \* sizeof(int)) -> Memoria de un array de 5 enteros) int atoi(const char \*str) //Convierte str a int

#include <string.h> //strerror

#include <errno.h> //Errores en la variable errno

char \*strerror(int errno) //Imprime la descripción del error errno errno //Entero relacionado con un error

**Procesos**

#include <unistd.h> //fork, getpid, getppid, sleep, execl, execv... #include <sys/types.h> //Todas las funciones pid\_t

#include <sys/wait.h> //wait, waitpid y sus macros (WIFEXITED, WIFSIGNALED...) pid\_t fork(void); //Crear proceso hijo identico al padre

fork=-1 //Error al crear hijo (EXIT\_FAILURE) fork=0 //Proceso hijo

fork>0 //Proceso es el padre

pid\_t getppid(void) //Devuelve el id del proceso padre pid\_t getpid(void) //Devuelve el id del proceso hijo

pid\_t wait(int \*status) //Detiene el proceso padre hasta que acabe el hijo

pid\_t waitpid(pid\_t pid, int \*status, int options) //Se puede usar para grupos de procesos pid

pid en concreto //Espera a recoger a un hijo en concreto

-1 //Espera a todos los hijos status //Guarda el estado options

WNOHANG //Si a terminado algun hijo=0 "Llamada no bloqueante" WUNTRACED //Detectar procesos detenidos por una señal

WCONTINUED //Detectar procesos que han sido reanudados tras una detención

WUNTRACED | WCONTINUED | WNOHANG //Si se quiere poner mas de uno se ponen entre " | "

//wait solo puede hacer WIFEXITED, waitpid todos WIFEXITED(status) //El proceso a salido exitosamente?

WEXITSTATUS(status) //Señal que lo ha terminado WIFSIGNALED(status) //El proceso termino por error?

WTERMSIG(status) //Señal que lo ha terminado WIFSTOPPED(status) //El proceso esta parado?

WSTOPSIG(status) //Señal que lo ha detenido

WIFCONTINUED(status) //El proceso ha sido reanudado tras una detención?

//Vacían el proceso en ejecución y carga con el programa pasado como parámetro

int execl(const char \*path, const char \*arg0, ..., const char \*argn, char \* /\*NULL\*/) //Ejecuta un programa con su ruta

path //Ruta al ejecutable

arg0 //Nombre del programa (normalmente es igual que path)

argn //Lista de argumentos separados con comas (arg1, arg2,...,argN) NULL //Debe acabar con NULL

int execlp(const char \*file, const char \*arg0, ..., const char \*argn, char \* /\*NULL\*/) //Ejecuta un programa con su ejecutable

file //Nombre del ejecutable

arg0 //Nombre del programa (normalmente es igual que path)

argn //Lista de argumentos separados con comas (arg1, arg2,...,argN) NULL //Debe acabar con NULL

int execv(const char \*path, char \*const argv[]) //Ejecuta un programa con su ruta path //Ruta al ejecutable

argv[] //Puntero a array de cadenas con argumentos (Ultimo elemento NULL o 0)

int execvp(const char \*file, char \*const argv[]) //Ejecuta un programa con su ejecutable file //Nombre del ejecutable

argv[] //Puntero a array de cadenas con argumentos (Ultimo elemento NULL o 0)

//Si se ejecutan con éxito no se devuelve nada, sino devuelve -1 y actualiza errno con el error

unsigned int sleep(unsigned int seconds) //Pausa la ejecución seconds //Segundos antes de que se ejecute la señal

//Devuelve 0 si se cumple correctamente y los segundos restantes si ha sido interrumpida

unsigned int alarm(unsigned seconds) //Programa una alarma que enviara SIGALRM

//Cada peticion anula la anterior

seconds //Segundos antes de que se ejecute la señal

//Si habia una alarma pendiente devuelve el restante para que acabase, sino devuelve 0

#include <signal.h> //Señales

void \*signal(int sig, void (\*func)(int)) sig //Identificador de la señal

SIGINT //Ctrl+C y termina el proceso.

SIGFPE //Cuando se produce un error en coma flotante

SIGTERM //El proceso termina, puede ejecutar un manejador programado por el usuario SIGKILL //El proceso termina

SIGSTOP //Para su ejecución SIGCONT //Reanuda su ejecución

SIGTSTP //CTRL+Z y para su ejecución

SIGALRM //Reciba la señal una vez transcurrido un tiempo prefijado, puede establecer un manejador para esta señal

SIGUSR1 y SIGUSR2 //El programador el que decide lo que debe hacer el proceso cuando recibe estas

señales

func //Cómo se maneja la señal

SIG\_DFL //Manejo por defecto para esa señal SIG\_IGN //Señal ignorada

funcion\_manejador //Función personalizada creada por el programador (void funcion\_manejador(int))

//Si se cumple la solicitud devuelve el nombre de func para la señal sig especificada, sino devolverá SIG\_ERR y se almacenará un valor

//positivo en errno.

void kill(pid\_t pid, int sig)

pid //Identificador del proceso al que enviamos la seña

pid>0 //Lo enviamos al proceso cuyo id coincide con el numero sig //Señal a enviar

**Hilos**

#include <pthread.h> //Hilos y mutex

int pthread\_create(pthread\_t \*thread, pthread\_attr\_t \*attr, void \* (\*start\_routine) (void \*), void \*arg)

//Crear hilos

thread //Puntero a hilo

attr //Puntero con atributos del hilo o NULL (atributos por defecto) start\_routine //Función que ejecuta el hilo (void \*func(void \*)) arg //Pasa por argumentos start\_routine o NULL (no pasa nada)

//Si ejecuta con exito devuelve 0, si hay un error sera distinto

int pthread\_join (pthread\_t thread, void \*\*thread\_return) //Espera a que el hilo termine y lo recoge thread //Hilo al que se espera

thread\_return //Retorno del hilo o NULL (si no necesitas el retorno)

//Devuelve 0 si todo está correcto o un valor diferente si hubo algún error

void pthread\_exit (void \*retval) //Termina el hilo de forma controlada retval //Devuelve el resultado, debe estar guardado con malloc

pthread\_t pthread\_self(void) //Devuelve el id del hilo que lo invoca

int pthread\_kill(pthread\_t thread, int sig) //Enviar una señal a un hilo thread //Id del hilo

sig //Identificador de la señal (#include <signal.h>)

//Si ejecuta con exito devuelve 0, si hay un error sera distinto

**Mutex**

#include <pthread.h> //Hilos y mutex

int pthread\_mutex\_init(pthread\_mutex\_t \*mutex, const pthread\_mutexattr\_t \*attr) //Inicializar un mutex mutex //Puntero al mutex

attr //Tipo de mutex PTHREAD\_MUTEX\_NORMAL //Por defecto PTHREAD\_MUTEX\_RECURSIVE //Recursivo

PTHREAD\_MUTEX\_ERRORCHECK //Detecta errores NULL //Igual a PTHREAD\_MUTEX\_NORMAL

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

//Si ejecuta con exito devuelve 0, si hay un error sera distinto EAGAIN //Se han agotado los recursos del sistema

ENOMEM //No hat memoria disponible EPERM //No hay permisos para los attr EINVAL //Parametros invalidos

int pthread\_mutex\_lock(pthread\_mutex\_t \*mutex) //Bloquea un mutex mutex //Puntero al mutex

//Si ejecuta con exito devuelve 0, si hay un error sera distinto

int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex) //Libera el bloquea de un mutex mutex //Puntero al mutex

//Si ejecuta con exito devuelve 0, si hay un error sera distinto

int pthread\_mutex\_destroy(pthread\_mutex\_t \*mutex) //Libera la memoria del mutex mutex //Mutex a destruir

//Si ejecuta con exito devuelve 0, si hay un error sera distinto

struct semaforo{ //Semaforo general int cuenta=1;

tipoCola cola;

}

cuenta>=0 //Abierto (Si es 1 puede entrar 1 proceso, si es 2 pueden 2...) cuenta<0 //Cerrado

semWait(sem) //Decrementa el valor del semaforo sem //Semaforo

semSignal(sem) //Incrementa el valor del semaforo sem //Semaforo

**Semáforos generales**

#include <semaphore.h> //Semaforos generales

int sem\_init(sem\_t \*sem, int pshared, int value) //Iniciar un semaforo sem //Puntero al semaforo

pshared //El semáforo es compartido entre procesos? pshared==0 //No

pshared!=0 //Si

value //Valor inicial del semaforo

//Si ejecuta con exito devuelve 0 o -1 si hay un error

int sem\_wait(sem\_t \*sem) //Decrementa el semaforo sem //Puntero al semaforo

//Si ejecuta con exito devuelve 0 o -1 si hay un error

int sem\_post(sem\_t \*sem) //Incrementa el semaforo sem //Puntero al semaforo

//Si ejecuta con exito devuelve 0 o -1 si hay un error

int sem\_destroy(sem\_t \*sem) //Destruye el semaforo sem //Puntero al semaforo

//Si ejecuta con exito devuelve 0 o -1 si hay un error