Practicas Segundo Parcial Programación y Administración de Sistemas

Práctica 3. Programación en POSIX.

Procesado de línea de comandos

```
-o      #Llamada básica
-o <parámetro>      -o<parámetro>      #Llamada con parámetro
-abc = -a -b -c      #Si varias opciones no dependen de parámetros pueden agruparse
getopt(argc, argv, "abc:")      #abc: Son los argumentos
        #c espera argumentos, por eso va acompañado de :, si quisiésemos argumento en b seria ab:c
optind          #Indice del siguiente elemento a procesar de argv[]
optarg          #Valor del argumento obligatorio de una opción
optopt          #Valor de la opción cuando es desconocida
opterr          #Mensaje de error

getopt_long(argc, argv, "abd:c:f:", long_options, NULL)      #Llamar a variables con nombres largos
static struct option long_options[] = {
        {"add", no_argument, NULL, 'a'},      #Solo puedes asignar una palabra a 1 valor
                #{<nombre largo>, <recibe/no recibe argumento>, NULL, <nombre corto>}
        {0, 0, 0, 0}      #Final de las opciones
};
```

Variables de entorno

```
env     #Muestra todas las variables de entorno del sistema operativo
getenv()        #Obtener una variable de entorno concreta
        #getenv("LANG") devuelve lenguaje o getenv("HOME") la carpeta home del usuario
```

Obtención de información de un usuario

```
/etc/passwd             #Mantiene información sobre los usuarios
/etc/group              #Mantiene información sobre los grupos
/usr/include/pwd.h              #Funciones y estructuras de acceso a la información de usuarios
/usr/include/grp.h              #Funciones y estructuras de acceso a la información de grupos
getpwuid()      #Devuelve una estructura con información del UID especificado
getpwnam()      #Devuelve una estructura con información del nombre especificado
getgrgid()      #Devuelve una estructura con información de un grupo pasado su GID
getgrnam()      #Devuelve una estructura con información de un grupo pasado su nombre

pw = getpwnam(lgn)
        pw->pw_gecos)       #Nombre de usuario asociado al login
        pw->pw_name         #Login
        pw->pw_passwd       #Password
        pw->pw_uid          #UID
        pw->pw_dir          #Home
        pw->pw_shell        #Shell
        pw->pw_gid      #Número de grupo principal
gr = getgrgid(pw->pw_gid)
        gr->gr_name         #Nombre del grupo principal
```

Creación de procesos (fork y exec)

```
pid_t fork(void);        #Crear proceso hijo idéntico al padre
   fork=-1       #Error al crear hijo (EXIT_FAILURE)
   fork=0        #Proceso hijo
   fork>0        #Proceso es el padre

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 está 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
```

Señales entre procesos

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

SIGHUP #Se envía automáticamente cuando la terminal asociada al proceso se cierra

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.

Comunicación entre procesos POSIX

Tipos de tuberías:

- Tuberías anónimas: Se crean desde bash de forma temporal intercomunicando dos procesos.
- Tuberías con nombre: Crear tuberías dentro del sistema de archivos para ser accedida por procesos.

int pipe(int fildes[2]); #Crea una tubería anónima y devuelve dos descriptores de fichero uno para leer (fildes[0]) y otro para escribir (fildes[1]). Usando read y write. Se cierra con close

Colas de mensajes

mqd_t mq_open(const char *name, int oflag, mode_t mode, struct mq_attr *attr); #Crear o abrir una cola

name #Siempre tendra una barra al inicio, "/nombrecola"

oflag #Flags binarios que se pueden especificar como un OR a nivel de bits de distintas macros.

O_CREAT | O_WRONLY #Debe crearse si no existe y solo escritura

O_RDONLY #Solo lectura

O_RDWR #Lectura y escritura

O_ NONBLOCK #No espera, da error si no hay espacio o no hay mensajes a leer

mode #Permisos de la cola

attr #Puntero a una estructura mq_attr (Solo en O_CREAT)

mq_maxmsg #Número máximo de mensajes acumulados en la cola

mq_msgsize #Tamano máximo de dichos mensajes

#Devuelve un mqdes (id). Si falla devueve -1 y errno

ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len, unsigned *msg_prio); #Recibir mensajes

mqdes #Id de la cola

msg_ptr #Almacen del mensaje

msg_len #Longitud del mensaje

msg_prio #Prioridad del mensaje

int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned msg_prio); #Enviar mensajes
    msg_prio #Prioridad mayor a 0 adelanta la cola al recibir
#0 si se envia y -1 si no (errno)

int mq_close(mqd_t mqdes); #Cierra la cola pero sigue disponible para otros procesos (mantiene los mensaje si los tiene). Devuelvere 0 o -1 si hay error (errno)

int mq_unlink(const char *name); #Eliminar una cola permanentemente. Devuelvere 0 o -1 si hay error.