APÉNDICE A LA BIBLIA DE LOS SISTEMAS OPERATIVOS

Por Álvaro Ramos Fustero.

C

MAIN(), ARGC, ARGV...

La firma típica de la función main es int main(int argc, char *argv[]).

- argc (argument count) es un entero que representa el número de argumentos pasados al programa desde la línea de comandos, incluyendo el nombre del propio programa.
- argv (argument vector) es un array de punteros a cadenas de caracteres (strings) que contienen los argumentos individuales pasados al programa.

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    // Imprimir el número total de argumentos
    printf("Número total de argumentos: %d\n", argc);

    // Imprimir el nombre del programa (primer argumento)
    printf("Argumento 0: %s\n", argv[0]);

    // Imprimir los argumentos adicionales, si los hay
    if (argc > 1) {
        printf("Argumentos adicionales:\n");
        for (int i = 1; i < argc; i++) {
            printf("Argumento %d: %s\n", i, argv[i]);
        }
    } else {
        printf("No se han proporcionado argumentos adicionales.\n");
    }
    return 0;
}
```

Partir un comando en los argumentos que lo componen

```c
char *vector[100];
vector[0] = strtok(comando, " ");
int i = 0;
while(vector[i] != NULL){
    i++;
    vector[i]=strtok(NULL, " ");
    }
execvp(vector[0], &vector[0]);
```

Leer de un sitio y escribir en otro

```
const int BUFSIZE = ...;
char buffer[BUFSIZE];

//Función para leer de una tubería y escribir en un fichero
while((bytes = read(fd_pipe[0], &buffer, BUFSIZE)) > 0){
    write(fd_salida, &buffer, bytes);
}
```

Leer línea

```
int leerLinea(int fd, char* linea){
    char c;
    int i=0;
    int bytesLeidos = read(fd, &c, 1);
    while((bytesLeidos != 0) && (c!='\n')){
        linea[i]=c;
        i++;
        bytesLeidos = read(fd, &c, 1);
    }
    linea[i] = '\0';
    return i;
}
```

LLAMADAS AL SISTEMA

FORK()

Programa que crea un hijo

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main(){
    // Crear un nuevo proceso hijo
    pid_t child_pid = fork();

    // Verificar si la creación del hijo fue exitosa
    if (child_pid < 0) {
        fprintf(stderr, "Error al crear el proceso hijo.\n");
        exit(EXIT_FAILURE);
    } else if (child_pid == 0) {
        // Código ejecutado por el proceso hijo
        printf("Proceso hijo creado con PID %d\n", getpid());
        exit(EXIT_SUCCESS);
    } else {
        // Código ejecutado por el proceso padre
        // Esperar a que el hijo termine
```

```
        wait(NULL);
        printf("Proceso padre con PID %d esperó a que el hijo terminara.\n", get
    }

    return 0;
}
```

Programa que crea n hijos

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    // Iterador para el bucle
    int i;

    // Bucle para crear 10 hijos
    for(i=0; i<10; ++i){
        // Crear un nuevo proceso hijo
        int pid=fork();

        // Verificar si la creación del hijo fue exitosa
        if(pid < 0){
            fprintf(stderr, "Error al crear el proceso hijo.\n");
            exit(EXIT_FAILURE);
        }else if(pid == 0){
            // Código ejecutado por el proceso hijo
            printf("Hijo %d con PID %d\n", i + 1, getpid());
            exit(EXIT_SUCCESS); // El hijo termina aquí
        }
    }

    // Código ejecutado por el proceso padre
    // Esperar a que todos los hijos terminen
    for(i=0; i<10; ++i){
        wait(NULL);
    }

    printf("Proceso padre con PID %d termina.\n", getpid());

    return 0;
}
```

SEÑALES

Hacer una pausa con SIGALRM

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

void handler_alarma(int signum) {
```

```
    printf("Recibida SIGALRM.\n");
    signal(SIGALRM, manejador_alarma);/*Volvemos a capturar SIGALRM, ya que
comportamiento por defecto es matar el proceso. En principio aquí esto no se
usa ya que solo llamamos a la señal una vez*/
}

int main() {
    // Configurar el manejador de señales para SIGALRM
    signal(SIGALRM, handler_alarma);
    // Establecer una alarma para que genere SIGALRM después de 5 segundos
    alarm(5);

    printf("Esperando la señal SIGALRM...\n");
    pause();

    return 0;
}
```

Contar segundos con SIGALRM

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

void manejador_alarma(int signum) {
    printf("Han pasado 2 segundos\n");
    alarm(2);
    signal(SIGALRM, manejador_alarma);/*Volvemos a capturar SIGALRM, ya que
comportamiento por defecto es matar el proceso.*/
}

int main() {
    // Configurar el manejador de señales para SIGALRM
    signal(SIGALRM, manejador_alarma);

    // Establecer una alarma para que genere SIGALRM después de 5 segundos
    alarm(2);

    printf("Esperando la señal SIGALRM...\n");

    // Entrar en un bucle de espera (pause) hasta que se reciba una señal
    while (1) {
        pause();
    }

    return 0;
}
```

Hacer que el programa finalice al cabo de x segundos

```
//Lo mismo pero sin capturar SIGALRM
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

```c
#include <signal.h>


int main() {
    // Configurar el manejador de señales para SIGALRM
    //signal(SIGALRM, SIG_DFL); Esto no haría falta ya que al empezar el program
    // tiene su valor por defecto.

    // Establecer una alarma para que genere SIGALRM después de 5 segundos
    alarm(2);

    printf("Esperando la señal SIGALRM para morirme...\n");

    // Entrar en un bucle de espera (pause) hasta que se reciba una señal
    while (1) {
        pause();
    }

    return 0;
}
```

Recibir una señal SIGUSR1 creada por el usuario.

```c
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>

// Variable global para indicar el modo actual
int modo = 0;

// Función de manejo de señal
void manejar_SIGUSR1(int signo) {
    if (signo == SIGUSR1) {
        // Cambiar de modo
        modo = (modo + 1) % 2;

        // Imprimir el nuevo modo
        printf("Cambiando a modo %d\n", modo);
    }
}

int main() {
    // Configurar el manejador de señales para SIGUSR1
    if (signal(SIGUSR1, manejar_SIGUSR1) == SIG_ERR) {
        perror("Error al configurar el manejador de señales");
        exit(EXIT_FAILURE);
    }

    // Imprimir el PID del proceso
    printf("PID del proceso: %d\n", getpid());

    // Ciclo infinito simulando la ejecución del programa
    while (1) {
        // Realizar alguna tarea según el modo actual
        if (modo == 0) {
            printf("Modo 0: Realizando tarea A\n");
        } else {
            printf("Modo 1: Realizando tarea B\n");
```

```
        }

        // Hacer pausa para simular la ejecución del programa
        sleep(2);
    }

    return 0;
}
```

Para enviarle la señal desde la terminal hacemos

`kill -SIGUSR1 12345`

TUBERÍAS

Redirigir salida estándar de un programa a una tubería, y a fichero

```
//Programa redirecciona la salida estándar y de error del programa especificado
// en su segundo (y sucesivos) argumentos en el fichero especificado en el
// primero de ellos
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <string.h>
#include <fcntl.h>


int main(int argc, char *argv[]){
    int pid, fd_pipe[2], fd_salida, bytes;
    const int BUFSIZE = 512;
    pipe(fd_pipe);
    pid = fork();
    if(pid==0){//Hijo (ejecuta)
        close(fd_pipe[0]);//Cerramos el extremo de lectura de la tubería
        close(1);//Cerramos la entrada estándar
        dup(fd_pipe[1]);//Redirigimos la salida estandar a la tubería
        close(2);//Cerramos el error estándar
        dup(fd_pipe[1]);//Redirigimos el error estándar a la tubería
        execvp(argv[2], &argv[2]);//Ejecutamos el programa con los argumentos da
        exit(0);
    }else{//Padre (redirecciona tubería a archivo)
        close(fd_pipe[1]);//Cerramos el extremo de escritura de la tubería
        fd_salida = open(argv[1], O_CREAT|O_WRONLY, 0777);
        char buffer[BUFSIZE];
        //Función para leer de una tubería y escribir en un fichero
        while((bytes = read(fd_pipe[0], &buffer, BUFSIZE)) > 0){
            write(fd_salida, &buffer, bytes);
        }
        wait(NULL);//Esperamos a que el hijo termine
        exit(0);
    }
}
```