Práctica 2.1: Introducción a la programación de sistemas UNIX

Objetivos

En esta práctica estudiaremos el uso básico del API de un sistema UNIX y su entorno de desarrollo. En particular, se usarán funciones para gestionar errores y obtener información.

Contenidos

Preparación del entorno para la práctica

Esta práctica únicamente requiere el entorno de desarrollo (compilador, editores y depurador), que está disponible en las máquinas virtuales de la asignatura y en la máquina física del laboratorio.

Se puede usar cualquier editor gráfico o de terminal. Además, se puede usar tanto el lenguaje C (compilador `gcc`) como C++ (compilador `g++`). Si fuera necesario compilar varios archivos, se recomienda el uso de `make`. Finalmente, el depurador recomendado en las prácticas es `gdb`. **No está permitido** el uso de IDEs como Eclipse.

Gestión de errores

Usar las funciones disponibles en el API del sistema (`perror(3)` y `strerror(3)`) para gestionar los errores en los siguientes casos. En cada ejercicio, añadir las librerías necesarias (`#include`).

Ejercicio 1. Añadir el código necesario para gestionar correctamente los errores generados por la llamada a `setuid(2)`. Consultar en el manual el propósito de la llamada y su prototipo.

```
int main() {
    if (setuid(0) == -1) {
        perror("Error");
    }
    return 1;
}
```

Ejercicio 2. Imprimir el código de error generado por la llamada del código anterior, tanto en su versión numérica como la cadena asociada.

```
#include <stdio.h>
#include <errno.h>
#include <string.h>

int main() {
    if (setuid(0) == -1) {
```

```
        printf("Codigo: %d Cadena: %s\n", errno, strerror(errno));
    }
    return 1;
}
```

Ejercicio 3. Escribir un programa que imprima todos los mensajes de error disponibles en el sistema. Considerar inicialmente que el límite de errores posibles es 255.

```
#include <errno.h>
#include <stdio.h>
#include <string.h>

int main(){
    int i = 0;
    char* s = "";
    for (i = 0; i < 135; i++) {
        printf("Codigo %d: Cadena: %s\n", i, s);
        s = strerror(i);
    }
    return 1;
}
```

Información del sistema

Ejercicio 4. El comando del sistema `uname(1)` muestra información sobre diversos aspectos del sistema. Consultar la página de manual y obtener la información del sistema.

```
uname - print system information
-a, --all
              print  all  information,  in the following order, except omit -p
              and -i if unknown:

[cursoredes@localhost ~]$ uname -a
Linux localhost.localdomain 3.10.0-862.11.6.el7.x86_64 #1 SMP Tue Aug 14 21:49:04
UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
```

Ejercicio 5. Escribir un programa que muestre, con `uname(2)`, cada aspecto del sistema y su valor. Comprobar la correcta ejecución de la llamada.

```
uname - get name and information about current kernel
#include <sys/utsname.h>
int uname(struct utsname *buf);
struct utsname {
               char sysname[];    /* Operating system name (e.g., "Linux") */
               char nodename[];   /* Name within "some implementation-defined
                                     network" */
               char release[];    /*Operating system release (e.g., "2.6.28")*/
               char version[];    /* Operating system version */
               char machine[];    /* Hardware identifier */
           #ifdef _GNU_SOURCE
               char domainname[]; /* NIS or YP domain name */
           #endif
```

```
          };

#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <sys/utsname.h>

int main(){
    struct utsname buf;
    if (uname(&buf) == -1){
        printf(strerror(errno));
    }
    else {
        printf("Sysname: %s\nNodename: %s\nRelease: %s\nVersion: %s\n
Machine: %s\n", buf.sysname, buf.nodename, buf.release,
buf.version, buf.machine);
    }
    return 1;
}
```

Ejercicio 6. Escribir un programa que obtenga, con `sysconf(3)`, información de configuración del sistema e imprima, por ejemplo, la longitud máxima de los argumentos, el número máximo de hijos y el número máximo de ficheros.

```
sysconf - get configuration information at run time
#include <unistd.h>
long sysconf(int name);

#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>

int main(){
    long l = 0;
    if ((l = sysconf(_SC_ARG_MAX)) == -1){
        printf(strerror(errno));
    }
    else {
        printf("Max number of arguments: %li\n", l);
    }
    if ((l = sysconf(_SC_CHILD_MAX)) == -1){
        printf(strerror(errno));
    }
    else {
        printf("Max number of childs: %li\n", l);
    }
    if ((l = sysconf(_SC_OPEN_MAX)) == -1){
        printf(strerror(errno));
    }
    else {
```

```
        printf("Max number of open files: %li\n", l);
    }
    return 1;
}
```

Ejercicio 7. Escribir un programa que obtenga, con `pathconf(3)`, información de configuración del sistema de ficheros e imprima, por ejemplo, el número máximo de enlaces, el tamaño máximo de una ruta y el de un nombre de fichero.

```
fpathconf, pathconf - get configuration values for files
#include <unistd.h>
long fpathconf(int fd, int name);
long pathconf(char *path, int name);

#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>

int main(){
    char *path = "/home/cursoredes/Desktop/SO/a.txt";
    long l = 0;
    if ((l = pathconf(path, _PC_LINK_MAX)) == -1){
        printf("%d", errno);
    }
    else {
        printf("Max number of links: %li\n", l);
    }
    if ((l = pathconf(path, _PC_PATH_MAX)) == -1){
        printf("%d", errno);
    }
    else {
        printf("Max length of path: %li\n", l);
    }
    if ((l = pathconf(path, _PC_NAME_MAX)) == -1){
        printf("%d", errno);
    }
    else {
        printf("Max length of file name: %li\n", l);
    }
    return 1;
}
```

Información del usuario

Ejercicio 8. El comando `id(1)` muestra la información de usuario real y efectiva. Consultar la página de manual y comprobar su funcionamiento.

```
id - print real and effective user and group IDs
id [OPTION]... [USER]
-a, -Z, -g, -G, -n, -r, -u, -z
```

```
[cursoredes@localhost ~]$ id
uid=1000(cursoredes)  gid=1000(cursoredes)  groups=1000(cursoredes),  10(wheel),
983(wireshark)

[cursoredes@localhost ~]$ sudo id
uid=0(root) gid=0(root) groups=0(root)

[cursoredes@localhost ~]$ id -G
1000 10 983

[cursoredes@localhost ~]$ id -n -G
cursoredes wheel wireshark
```

Ejercicio 9. Escribir un programa que muestre, igual que `id`, el UID real y efectivo del usuario. ¿Cuándo podríamos asegurar que el fichero del programa tiene activado el bit *setuid*?

```
getuid, geteuid - get user identity

#include <unistd.h>
#include <sys/types.h>

uid_t getuid(void);
uid_t geteuid(void);

#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main(){
    uid_t uid = getuid();
    uid_t euid = geteuid();
    printf("Real UID: %d\nEffective UID: %d\n", uid, euid);
    return 1;
}

Cuando el UID es distinto al EUID el bit de setuid debe estar activo (Preguntar)
```

Ejercicio 10. Modificar el programa anterior para que muestre además el nombre de usuario, el directorio *home* y la descripción del usuario.

```
getpwnam, getpwnam_r, getpwuid, getpwuid_r - get password file entry

#include <sys/types.h>
#include <pwd.h>

struct passwd *getpwnam(const char *name);
struct passwd *getpwuid(uid_t uid);
int getpwnam_r(const char *name, struct passwd *pwd,
                  char *buf, size_t buflen, struct passwd **result);

int getpwuid_r(uid_t uid, struct passwd *pwd,
```

```
                  char *buf, size_t buflen, struct passwd **result);

struct passwd {
char *pw_name;   /* Nombre de usuario */
char *pw_passwd; /* Contraseña */
uid_t pw_uid;    /* Identificador de usuario */
gid_t pw_gid;    /* Identificador de grupo */
char *pw_gecos;  /* Descripción del usuario */
char *pw_dir;    /* Directorio “home” */
char *pw_shell;  /* Shell */
}

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <pwd.h>
#include <sys/types.h>

int main(){
    uid_t uid = getuid();
    uid_t euid = geteuid();
    printf("Real UID: %d\nEffective UID: %d\n", uid, euid);
    struct passwd *password = getpwuid(uid);
    printf("Username: %s\nHome: %s\nUser description: %s\n",
        password->pw_name, password->pw_dir, password->pw_gecos);
    return 1;
}
```

Información horaria del sistema

Ejercicio 11. El comando `date(1)` muestra la hora del sistema. Consultar la página de manual y familiarizarse con los distintos formatos disponibles para mostrar la hora.

```
date - print or set the system date and time

date [OPTION]... [+FORMAT]
date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]
Display the current time in the given FORMAT, or set the system date.

-d=STRING, -f=DATAFILE, -I=TIMESPEC, -r=FILE, -R, --rfc-3339=TIMESPEC, -s=STRING,
-u
```

Ejercicio 12. Escribir un programa que muestre la hora, en segundos desde el Epoch, usando la función `time(2)`.

```
time - Obtener el tiempo en segundos desde el Epoch

#include <time.h>
time_t time(time_t *t);

#include <stdio.h>
```

```
#include <string.h>
#include <errno.h>
#include <time.h>

int main(){
    time_t t = time(NULL);
    if (t == -1){
        printf(strerror(errno));
    }
    else{
        printf("Seconds passed from epoch: %ld\n", t);
    }
    return 1;
}
```

Ejercicio 13. Escribir un programa que mida, en microsegundos usando la función `gettimeofday(2)`, lo que tarda un bucle que incrementa una variable un millón de veces.

```
gettimeofday, settimeofday - get / set time

#include <sys/time.h>

int gettimeofday(struct timeval *tv, struct timezone *tz);
int settimeofday(const struct timeval *tv, const struct timezone *tz);

struct timeval {
time_t      tv_sec;     /* seconds */
suseconds_t tv_usec;    /* microseconds */
};

and  gives  the number of seconds and microseconds since the Epoch (see
time(2)).  The tz argument is a struct timezone:

struct timezone {
int tz_minuteswest;     /* minutes west of Greenwich */
int tz_dsttime;         /* type of DST correction */
};

#include <stdio.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>

int main(){
    int i;
    struct timeval tv1, tv2;
    gettimeofday(&tv1, NULL);
    for (i = 0; i < 1000000; i++);
    gettimeofday(&tv2, NULL);
    printf("Elapsed time: %d\n", tv2.tv_usec - tv1.tv_usec);
    return 1;
```

```
}
```

Ejercicio 14. Escribir un programa que muestre el año usando la función `localtime(3)`.

```
#include <time.h>

struct tm *localtime(const time_t *timep);
struct tm *localtime_r(const time_t *timep, struct tm *result);

struct tm {
int tm_sec;         /* seconds */
int tm_min;         /* minutes */
int tm_hour;        /* hours */
int tm_mday;        /* day of the month */
int tm_mon;         /* month */
int tm_year;        /* year */
int tm_wday;        /* day of the week */
int tm_yday;        /* day in the year */
int tm_isdst;       /* daylight saving time */
};

#include <stdio.h>
#include <string.h>
#include <time.h>

int main(){
    time_t time_now = time(NULL);
    struct tm *time_tm = localtime(&time_now);
    printf("Year: %d\n", time_tm->tm_year + 1900);
    return 1;
}
```

Ejercicio 15. Modificar el programa anterior para que imprima la hora de forma legible, como `"lunes, 29 de octubre de 2018, 10:34"`, usando la función `strftime(3)`.

```
strftime - format date and time

#include <time.h>
size_t strftime(char *s, size_t max, const char *format, const struct tm *tm);


setlocale - set the current locale

#include <locale.h>
char *setlocale(int category, const char *locale);

#include <stdio.h>
#include <string.h>
#include <time.h>
#include <locale.h>

int main(){
```

```
    setlocale(LC_ALL, "es_ES");
    time_t time_now = time(NULL);
    struct tm *time_tm = localtime(&time_now);
    printf("Year: %d\n", time_tm->tm_year + 1900);

    char *date[100];
    strftime(date, 100, "%A, %d de %B de %Y, %H:%M", time_tm);
    printf("Date: %s", date);
    return 1;
}
```

Nota: Para establecer la configuración regional (*locale*, como idioma o formato de hora) en el programa según la configuración actual, usar la función `setlocale(3)`, por ejemplo, `setlocale(LC_ALL, "")`. Para cambiar la configuración regional, ejecutar, por ejemplo, `export LC_ALL="es_ES"`, o bien, `export LC_TIME="es_ES"`.