Tabla de contenido

1. Objetivos

- Avanzar en la programación en c.
- Utilizar de forma correcta las llamadas al sistema de C para la gestión de procesos en Linux.
- Aprender a gestionar procesos: crear, asignar tareas a realizar, consultar su información.
- Hacer uso de los mecanismos básicos de comunicación entre procesos (IPC).
- Aplicar la gestión de señales entre procesos.

2. Estructura básica de un programa en C

C es el lenguaje más utilizado en la programación de sistemas Linux. Una gran parte del sistema está programado en c. Este es un manual muy básico que intenta ser de ayuda para el objetivo que aquí se plantea que es la gestión de procesos.

```
#include <stdio.h>   //Incluir de los ficheros de cabecera que dan acceso a llamadas al sistema.
#include <stdlib.h>

int multiplica(int a);    //Declaración de prototipo de funciones

int main(int argc, char *argv[])  //Función principal de c
{
 int  a,b;
 printf("Prototipo de programa C. \n"); // Visualización mensaje por pantalla (stdout->salida estándar)
 printf("Introduce un valor entero: ");
 scanf("%d",&b);    // Petición de dato por teclado al usuario (stdin->entrada estándar)
 a=multiplica(b);   // Llamada a función, parámetro pasado por valor y recogida resultado
 printf("Resultado multiplicación: %d \n",a); // Visualización de contenido de una variable
 exit(0);
}

int multiplica(int a)   // Declaración de una función
{
 a=a*2;
 return a;     //Devolución de resultado
}
```

Finalizado el programa podemos compilarlo y ejecutarlo desde el terminal:

Compilación: gcc –o programa_ejecutable programa_fuente.c
Ejecución: ./programa_ejecutable

3. Trabajo con el proceso generado por el programa ejecutado

Fichero de cabecera: **unistd.h**

Llamadas al sistema: **getpid(), getppid(), getuid(), getgid(), geteuid(), getegid()**

- **getpid**: devuelve el indentificador del proces o que hace la llamada.
  Idproc=getpid();
- **getppid()**: devuelve el identificador del proceso padre del proceso que hace la llamada. Si el proceso padre ha finalizado, devuelve el identificador del proceso que le ha sido asignado como padre. Se le puede haber asignado el proceso Init (1) u otro proceso ancestor.
  Idprocpadre=getppid();

- **getuid():** devuelve el identificador del usuario real del proceso actual, el que ejecuta el programa.
  Idguid=getuid();
- **getgid():** devuelve el identificador del grupo real del proceso actual.
  Idgid=getgid();
- **geteuid():** devuelve el identificador del usuario propietario del proceso actual.
  Ideuid=geteuid();
- **getegid();** devuelve el identificador del grupo propietario del proceso actual.
  Idegid=getegid();

Ejemplo de uso de fork():

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


int main(int argc, char *argv[])
{
  printf("Programa C que al ejecutarse es un proceso más en el sistema.\n");
  printf("Identificador de proceso: %d y mi padre es: %d \n",getpid(), getppid());
  printf("Termino mi ejecución.\n");
  exit(0);
}
```

4. Ejercicio propuesto 1

Tenemos el siguiente programa:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(int argc, char *argv[])
{
  printf(“Trabajo con procesos\n”);
// fork();
  printf("Mensaje después del fork 1\n");
// fork();
printf("Mensaje después del fork 2 \n");
//fork()
printf(“Mensaje después del fork 3 \n”);

 exit(0);
}
```

- Ejecute el programa tal cual está, ¿qué resultado obtiene?
- Quite ahora el comentario del primer fork() y analice el resultado
- Quite el comentario del segundo fork() y vuelva a analizar el resultado
- Ahora ejecute sin ningún fork() comentado ¿qué resultado se obtiene?

Tomad notas de las dudas que os surgen para resolverlas en clase.

5. Ejercicio propuesto 2

Ahora analice el siguiente programa:

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

int main(int arg, char *argv[]) {
 int i,pid;
 printf("Antes del bucle:\n");
 for (i = 0; i < 5 ; i++ )
  {
   pid=fork();
   printf("%d ", i)
  }
 printf("Proceso %d \n",getpid());
 exit(0);
}
```

Sin ejecutar el programa intente dar respuesta a las siguientes preguntas:

- ¿Cuántas veces aparecerá el mensaje que aparece dentro del bucle?
- ¿Y el mensaje antes de la sentencia exit(0)?
- Justifique las respuestas que ha dado

Una vez contestadas las preguntas, implemente el programa y responda de nuevo a las mismas preguntas justificando los resultados.