UNIVERSIDAD NEBRIJA

Grado en Ingeniería Informática

Sistemas Operativos

Práctica 1

Índice

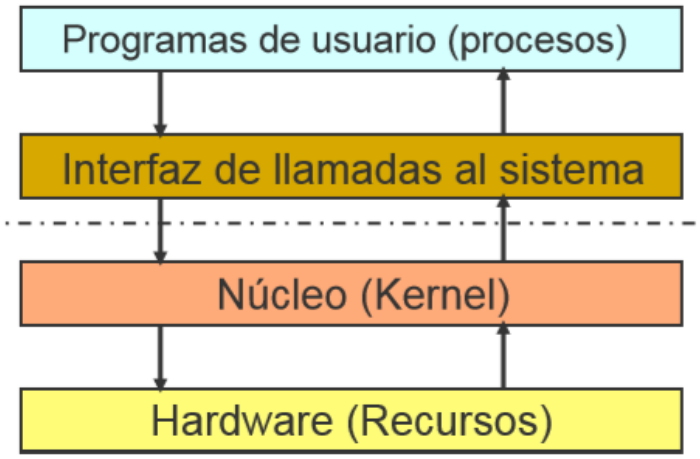
Práctica 1: Fundamentos Sistemas Operativos: POSIX

Introducción

La interfaz del sistema operativo con el programador es la que recupera los servicios y llamadas al sistema que los usuarios pueden utilizar directamente desde sus programas.
POSIX es el estándar de interfaz de sistemas operativos potables de IEEE basado en el sistema operativo UNIX. Aunque UNIX era prácticamente un estándar industrial, había bastantes, diferencias entre las distintas implementaciones de UNIX, lo que provocaba que las aplicaciones no se pudieran transportar fácilmente entre distintas plataformas UNIX. Este problema motivó a los implementadores y usuarios a desarrollar un estándar internacional con el propósito de conseguir la portabilidad de las aplicaciones en cuanto a código fuente.

POSIX es una interfaz basada en lenguaje C:

1. La concurrencia la proporciona el SO.
2. Los procesos solicitan servicios al SO (p.e. de E/S) mediante llamadas al sistema.



Ejemplo de utilización de llamadas al sistema: Crear un fichero “ejemplo” (si no existe) y escribir la cadena “Esto es un ejemplo”

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
int main (void)
{
  int fd, bytes_escritos;                    /*descriptor de fichero*/
  char buffer[100];
  mode_t modo = S_IRWXU; /* modo de r, w y x para el propietario*/

  strcpy(buffer, "Esto es un ejemplo\n");
  if ((fd = open ("ejemplo", O_RDWR | O_CREAT, modo))== -1)
        /*abre el fichero ejemplo en modo lectura/escritura o lo
        crea si no existe */
        perror ("Error al abrir o crear el fichero");
        /*muestra un mensaje de error si no puede abrir/crear el fichero*/
  else
        bytes_escritos = write(fd, buffer, strlen(buffer));
        /* escribe buffer de tamaño sizeof(buffer) en fd */
  exit(0);
}
```

Objetivos

- Gestión de procesos en un entorno UNIX.
- Estudio de las funciones aportadas por el estándar POSIX.

Entorno:

Previo a la implementación de los ejercicios propuestos, es necesario tener instalado un compilador como GCC en un entorno Unix como la distribución Ubuntu de Linux instalada en los laboratorios de la Escuela.

```
sudo apt update
sudo apt install gcc
gcc -version //compueba que la instalación ha resultado correcta
man gcc //manual de gcc
```

Compilación de programas

Compilación de un programa C:

```
gcc -o ejecutable fuente.c
```

Ejecución de programas

Como cualquier otro comando se puede ejecutar el archivo generado tras la compilación:

```
./ejecutable
```

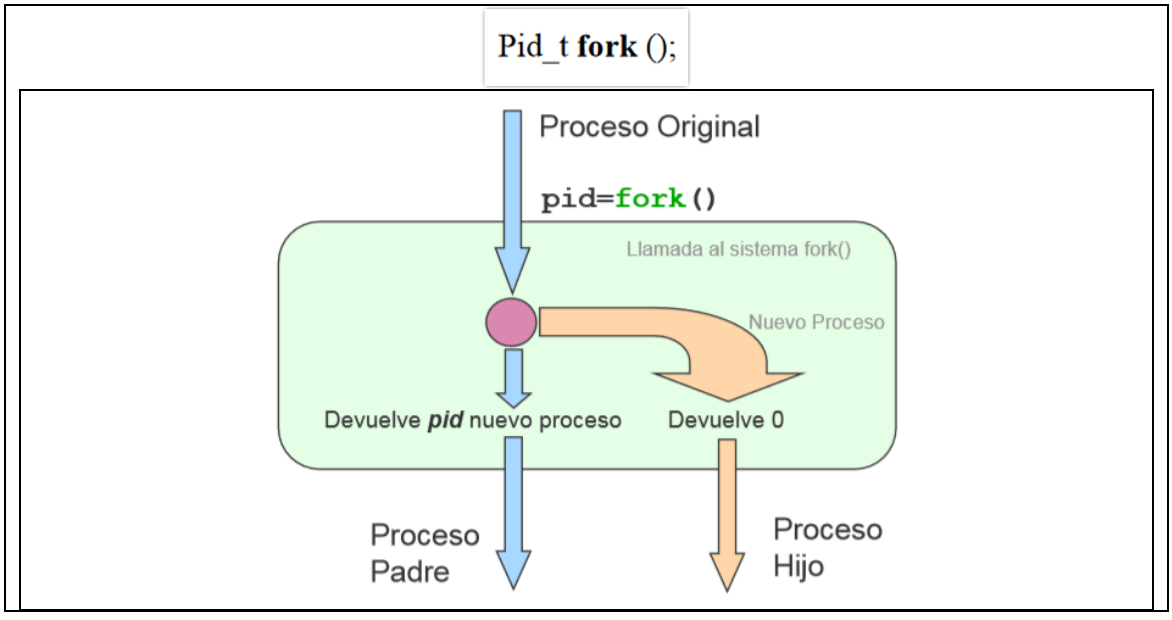
Funciones básicas

Funciones: los principales servicios que ofrece POSIX para la gestión de procesos se pueden resumir en la siguiente tabla:

Llamada	Función
fork	Crea un proceso
getpid	Obtiene el identificador del proceso
getppid	Obtiene el identificador del proceso padre
wait	Detiene un proceso hasta que alguno de sus hijos termina
waitpid	Detiene un proceso hasta que un determinado hijo termina
exec	(Conjunto de llamadas) Cambia la imagen de memoria de un proceso
exit	Fuerza la terminación del proceso que la invoca
kill	Solicita la terminación de otro proceso

Ejemplos gestión de procesos

La forma de crear un proceso en un sistema operativo que ofrezca la interfaz POSIX es invocando el servicio fork.



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

int main (void)
{
  pid_t pid;
  pid = fork();

  switch (pid)
  {
        case -1: perror ("No se ha podido crear el hijo");
                 break;
        case 0:  printf("Soy el hijo, mi PID es %d y mi PPID es %d\n",
                                 getpid(), getppid());
                 break;

        default: printf ("Soy el padre, mi PID es %d y el PID de mi
                          hijo es %d\n", getpid(), pid);
  }

  exit(0);
}
```

```
[/u0/sitr/sitr001/procesos]p1
Soy el hijo, mi PID es 15673 y mi PPID es 15672
Soy el padre, mi PID es 15672 y el PID de mi    hijo es 15673
[/u0/sitr/sitr001/procesos]
```

Bifurcación

Ejemplo Espera entre Procesos

```
#include <sys/types.h>
#include <stdlib.h>
#include <sys/wait.h>
int main(void)
{
   pid_t childpid, childdead;
   int i;
   childpid = fork();

   if (childpid == -1)
   {
          perror("fork no pudo crear el hijo");
          exit(1);
   }
   else  if (childpid == 0)
   {
        printf ("Soy el hijo (PID %d) y voy a contar hasta 100000 \n", getpid());
        for (i=0;i<100000; i++) {}
   }
   else
   {    printf("Soy el padre (PID %d) y voy a esperar a mi hijo (PID %d)\n",
                                                 getpid(),childpid);
        if ((childdead = wait(0))==-1)
           perror("No he podido esperar al hijo");
        else
           printf("Mi hijo con pid %d, ha muerto\n",childdead);
   }

   exit(0);
}
```

```
[/u0/sitr/sitr001/procesos]p2
Soy el hijo (PID 18296) y mi padre es (PID 18295), voy a contar hasta 100000
Soy el padre (PID 18295) y voy a esperar a mi hijo (PID 18296)
Mi hijo con pid 18296, ha muerto
[/u0/sitr/sitr001/procesos]
```

Bifurcación

Entregables

Ejercicio_1:

Llamada al sistema para la apertura y lectura del fichero “ejemplo” creado en el primer programa mostrado.

Ejercicio_2:

Implementar una aplicación concurrente que calcule el cuadrado de los 20 primeros números naturales y almacene el resultado, repartiendo la tarea entre dos procesos: Para ello se deben crear dos procesos Hijos:

- Hijo1 realiza la operación sobre los números pares.
- Hijo2 realiza la operación sobre los números impares.

El proceso padre espera la terminación de los hijos, ‘obtiene’ el resultado de cada hijo y muestra los valores ordenados en pantalla.

Ejercicio_3:

Observa el siguiente código y escribe la jerarquía de procesos resultante. Ahora compila y ejecuta el código para comprobarlo. Presta atención al orden de terminación de los procesos, ¿qué observas? ¿por qué?

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

int main (int argc, char *argv[]) {

  int num;
  pid_t pid;

  srandom(getpid());
  for (num= 0; num< 3; num++) {
    pid= fork();
    printf ("Soy el proceso de PID %d y mi padre tiene %d de PID.\n",
              getpid(), getppid());
    if (pid== 0)
      break;
  }
  if (pid== 0)
    sleep(random()%5);
  else
    for (num= 0; num< 3; num++)
      printf ("Fin del proceso de PID %d.\n", wait (NULL));

  return 0;
}
```

Ejercicio_4:

Implementar un código que genere la estructura de procesos de la siguiente figura: