SO sesión 3: procesos

lunes, 9 de noviembre de 2020 10:05

Para leer en el man	Descripción básica	Parámetros/argumentos principales que practicaremos
getpid	Retorna el PID del proceso que la ejecuta	
fork	Crea un proceso nuevo, hijo del que la ejecuta	
exit	Termina el proceso que ejecuta la llamada	
waitpid	Espera la finalización de un proceso hijo	
execlp	Ejecuta un programa en el contexto del mismo proceso	
perror	Escribe un mensaje del último error producido	
ps	Devuelve información de los procesos	-a, -u, -o
proc	Pseudo-file system que ofrece información de datos del kernel	cmdline, cwd, environ exe, status

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


int main(int argc,char *argv[]) {
    int pid;
    char buffer[80];
    pid = fork();
    /* A partir de esta línea de código tenemos 2 procesos */
    sprintf(buffer,"Soy el proceso %d\n",getpid());
    write(1, buffer, strlen(buffer));
    return 0;
}
```

Se crean dos procesos, el padre y el hijo (se ejecutan a la vez, concurrentemente

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

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

```c
int main(int argc,char *argv[]) {
    int pid;
    char buffer[80];
    sprintf(buffer,"Antes del fork: Soy el proceso %d\n",getpid());
    write(1, buffer, strlen(buffer));
    pid = fork();
    switch (pid) { /* Esta linea la ejecutan tanto el padre como el hijo */

        case 0: /* Escribe aqui el codigo del proceso hijo */
            sprintf(buffer,"HIJO: Soy el proceso %d\n",getpid());
            write(1, buffer, strlen(buffer));
            break;

        case -1:/* Se ha producido un error */
            strcpy(buffer,"Se ha prodcido une error\n");
            write(1, buffer, strlen(buffer));
            break;

        default: /* (pid != 0) && (pid != -1) */
            /* Escribe aqui el codigo del padre */
            sprintf(buffer,"PADRE: Soy el proceso %d\n",getpid());
            write(1, buffer, strlen(buffer));
    }
    sprintf(buffer,"Los DOS: Soy el proceso %d\n",getpid());
    write(1,buffer, strlen(buffer));
    return 0;
}
```

También se crean dos procesos (padre e hijo), pero hay trozos que sólo lo ejecuta el hijo, tr
lo ejecuta el padre y trozos que lo ejecutan los dos. Se ejecutan concurrentemente

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
int main(int argc,char *argv[]) {
    int pid;
    char buffer[80];
    sprintf(buffer,"Antes del fork: Soy el proceso %d\n",getpid());
    write (1, buffer, strlen(buffer));
    pid = fork();
    switch (pid) { /* Esta linea la ejecutan tanto el padre como el hijo */
        case 0: /* Escribe aqui el codigo del proceso hijo */
            sprintf(buffer,"HIJO: Soy el proceso %d\n",getpid());
            write(1, buffer, strlen(buffer));
            /* Termina su ejecución */
            exit(0);
        case -1: /* Se ha producido un error */
            sprintf(buffer,"Se ha prodcido une error\n");
            write(1, buffer, strlen(buffer));
            break;
        default: /* (pid !=0) && (pid !=-1) */
            /* Escribe aqui el codigo del padre */
            sprintf(buffer,"PADRE: Soy el proceso %d\n",getpid());
            write(1, buffer, strlen(buffer));
    }
    sprintf(buffer,"Solo lo ejecuta el padre: Soy el proceso %d\n",getpid());
    write(1, buffer, strlen(buffer));
```

ozos que solo

```
    write(1, buffer, strlen(buffer));
    return 0;
}
```

Se crean dos procesos (hijo y padre), pero cuando el hijo finaliza su trozo de código, hace u
decir, acaba. Por lo que la última parte del código sólo lo ejecuta el proceso padre

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
int main(int argc,char *argv[]) {
    int pid,i;
    char buffer[80];
    sprintf(buffer,"Antes del fork: Soy el proceso %d\n",getpid());
    write (1, buffer, strlen(buffer));
    pid = fork();
    switch (pid) { /* Esta linea la ejecutan tanto el padre como el hijo */
        case 0: /* Escribe aqui el codigo del proceso hijo */
            sprintf(buffer,"HIJO: Soy el proceso %d\n",getpid());
            write(1, buffer, strlen(buffer));
            for (i = 0; i < 10000; i++) ; /* bucle vacío sólo para que el hijo pierda tiempo */
            sprintf(buffer,"HIJO acaba: Soy el proceso %d\n",getpid());
            write(1, buffer, strlen(buffer));
            /* Termina su ejecución */
            exit(0);
        case -1: /* Se ha producido un error */
            sprintf(buffer,"Se ha prodcido une error\n");
            write(1, buffer, strlen(buffer));
            break;
        default: /* (pid !=0) && (pid !=-1) */
            /* Escribe aqui el codigo del padre */
            sprintf(buffer,"PADRE: Soy el proceso %d\n",getpid());
            write(1, buffer, strlen(buffer));
    }
    sprintf(buffer,"Solo lo ejecuta el padre: Soy el proceso %d\n",getpid());
    write(1, buffer, strlen(buffer));
    return 0;
}
```

```c
int ma
    int
    cha
    spr
    wri
    pid
    swi
    }
    wai
    spr
    wri
    ret
}
```

Se crean dos procesos (padre e hijo) y ejecutan sus respectivos códigos, no se sabe cuál se termina antes, ya que se ejecutan concurrentemente.

Pero s
termin
perfec

```c
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
char variable_global='A';
int main(int argc,char *argv[]) {
    int pid;
    char variable_local='a';
    char buffer[80];
    sprintf(buffer,"Antes del fork: Soy el proceso %d\n",getpid());
    write(1, buffer, strlen(buffer));
    pid = fork();
    switch (pid) {/* Esta linea la ejecutan tanto el padre como el hijo */
        case 0: /* Escribe aqui el codigo del proceso hijo */
            sprintf(buffer,"HIJO: Soy el proceso %d\n",getpid());
            write(1,buffer,strlen(buffer));
            /* Comprobamos que tenemos acceso a las variables */
            variable_global = 'B';
            variable_local = 'b';
            sprintf(buffer,"HIJO:La variable_global vale %c y la local %c\n", variable_global,variable_local);
```

un exit es

```
in(int argc,char *argv[]) {
t pid,i;
ar buffer[80];
rintf(buffer,"Antes del fork: Soy el proceso %d\n",getpid());
ite (1, buffer, strlen(buffer));
d = fork();
itch (pid) { /* Esta linea la ejecutan tanto el padre como el hijo */
  case 0: /* Escribe aqui el codigo del proceso hijo */
      sprintf(buffer,"HIJO: Soy el proceso %d\n",getpid());
      write(1, buffer, strlen(buffer));
      for (i = 0; i < 10000; i++) ; /* bucle vacío sólo para que el hijo pierda tiempo */
      sprintf(buffer,"HIJO acaba: Soy el proceso %d\n",getpid());
      write(1, buffer, strlen(buffer));
      /* Termina su ejecución */
      exit(0);
  case -1: /* Se ha producido un error */
      sprintf(buffer,"Se ha prodcido une error\n");
      write(1, buffer, strlen(buffer));
      break;
  default: /* (pid !=0) && (pid !=-1) */
      /* Escribe aqui el codigo del padre */
      sprintf(buffer,"PADRE: Soy el proceso %d\n",getpid());
      write(1, buffer, strlen(buffer));

itpid(-1, NULL, 0);
rintf(buffer,"Solo lo ejecuta el padre: Soy el proceso %d\n",getpid());
ite(1, buffer, strlen(buffer));
turn 0;
```

i le insertamos un waitpid (-1, NULL, 0) el proceso padre esperará a que
ne el proceso hijo para ejecutar su código, en este caso, sabemos
ctamente que el proceso hijo termina antes

```
            write (1, buffer, strlen(buffer));
            /* Termina su ejecución */
            exit(0);
        case -1: /* Se ha producido un error */
            sprintf(buffer,"Se ha prodcido une error\n");
            write (1, buffer, strlen(buffer));
            break;
        default: /* Escribe aqui el codigo del padre */
            sprintf(buffer,"PADRE: Soy el proceso %d\n",getpid());
            write (1, buffer, strlen(buffer));
            waitpid(-1, NULL, 0);
            /* Comprobamos que tenemos acceso a las variables */
            sprintf(buffer,"PADRE:La variable_global vale %c y la local %c\n", variable_global,variable_local);
            write (1, buffer, strlen(buffer));
    }
    sprintf(buffer,"Solo lo ejecuta el padre: Soy el proceso %d\n",getpid());
    write(1, buffer, strlen(buffer));
    return 0;
}
```

No podemos asegurar que dos ejecuciones tiene el mismo contenido de salida porque no
velocidad a la que van los procesos, por lo tanto puede que uno se adelante más que otr
los recursos que el ordenador tenga disponiblese

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

char variable_global='A';

void Escribe_variables(char *variable_local) {
    char buffer[80];
    sprintf(buffer,"Funcion:La variable_global vale %c y la local %c\n",
    variable_global,*variable_local);
    write(1, buffer, strlen(buffer));
}
int main(int argc,char *argv[]) {
    int pid;
    char buffer[80];
    char variable_local='a';
    char *variable_localc = &variable_local;
    sprintf(buffer,"Antes del fork: Soy el proceso %d\n",getpid());
    write(1, buffer, strlen(buffer));
    pid = fork();
    switch (pid){ /* Esta linea la ejecutan tanto el padre como el hijo */
        case 0: /* Escribe aqui el codigo del proceso hijo */
            sprintf(buffer,"HIJO: Soy el proceso %d\n",getpid());
            write(1,buffer,strlen(buffer));
            /* Probamos si tenemos acceso desde una función */
            Escribe_variables(variable_localc);
            /* Termina su ejecución */
            exit(0);
        case -1: /* Se ha producido un error */
            sprintf(buffer,"Se ha prodcido une error\n");
            write(1,buffer,strlen(buffer));
            break;
        default: /* Escribe aqui el codigo del padre */
            sprintf(buffer,"PADRE: Soy el proceso %d\n",getpid());
            write(1, buffer, strlen(buffer));
            /* Probamos si tenemos acceso desde una función */
            Escribe_variables(variable_localc);
```

Se crean dos procesos (padre e hijo) hay dos variables (local y global) y observamos que si cambiamos el valor de estas variables en el proceso hijo, no cambia de valor en el proceso padre

o controlamos la
o en función de

```
    }
    sprintf(buffer,"Solo lo ejecuta el padre: Soy el proceso %d\n",getpid());
    write(1, buffer, strlen(buffer));
    return 0;
}
```

Para utilizar una variable local en otro función, se ha de inicializar un puntero con la dire
variable y pasarle a la función como parámetro

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

void main(){
    char buffer[80];
    sprintf(buffer, "Soy el proceso: %d\n", getpid());
    write(1,buffer,strlen(buffer));
    execlp("ls", "ls", "-l", (char *) 0);
    sprintf(buffer, "Soy el proceso: %d\n", getpid());
    write(1,buffer,strlen(buffer));
}
```

Ejemplo de execlp: Ejecuta las 3 primeras líneas y se muta, ejecutando ls -l. Imprime p
primer mensaje de "soy el proceso…" ya que el segundo no lo ejecuta, porque ya se h
proceso, a no ser que se haya producido un error de mutación se ejecutaría el segund
el proceso…"

Se consultan a la sección 2 del man(fork, exit, waitpid, execlp,….) ya que son llamadas

Ps: ps -e muestra procesos
Ps -u muestra procesos de usuarios
Ps -u nombre_usuario muestra los procesos del nombre_usuario

Directorios: color azul marino

En el fichero environ y el comando env coinciden tanto el PATH como el PWD

En el caso del proceso hijo:
Cwd -> /home/so1/Desktop/so3 señala a la carpeta donde se encuentra el ejecutab
Exe -> /home/so1/Dektop/so3/myPS_v0 señala la localización exacta del ejecutabl
No podemos ver el contenido de environ y el de cmdline está vacío ya que no hemo
comando en el terminal para ejecutar el proceso Porque el proceso hijo está en esta
en running se podría ver

ección de la

por pantalla sólo el
a mutado a otro
lo mensaje de "soy

s a sistema

ble
e
s escrito ningún
ado zombie, si está

```
//Ejemplo esquema secuencial
for (i=0;i<num_hijos;i++){
      pid=fork();
      if (pid==0) {
            // código hijo
            exit(0);// Solo si el hijo no muta y queremos que termine
      }
      // Esperamos a que termine antes de crear el siguiente
      waitpid(…); // los parámetros depende de lo que queramos
}
```

```
//Ejemplo esquema concurrente
for (i=0;i<num_hijos;i++){
      pid=fork();
      if (pid==0) {
            // código hijo
            exit(0);// Solo si el hijo no muta y queremos que termine
      }
}
// Esperamos a todos los procesos
while (waitpid(…)>0); // los parámetros depende de lo que queramos
```

```
void usage() {
    char buff[256];
    sprintf(buff, "Usage: actualizar_fecha fichero_1
    write(1, buff, strlen(buff));
    exit(1);
}
```

```
void error_y_exit (char *msg,int exit_status)
{
      perror(msg);
      exit(exit_status);
}

void usage()
{
    char buff[256];

    sprintf(buff, "Usage: actualizar_fecha fichero_1 .. fichero_n\n");
    write(1, buff, strlen(buff));
    exit(1);
```

```
.. fichero_n\n");
```

```
}


    all: actualizar_fecha signals

    actualizar_fecha: actualizar_fecha.c
            gcc -o actualizar_fecha actualizar_fecha.c

    signals: signals.c
            gcc -o signals signals.c

    clean:
            rm actualizar_fecha signals
```