Boletín de ejercicios 2.1
Ejercicios de llamadas para gestión de procesos

July 13, 2016

1. Sea el siguiente programa escrito en C:

```
#include<stdio.h>
#include <stdlib.h>
main()
{  if (fork()!=0){
      if (fork()!=0){
         printf("A\n");
      } else {
      printf("B\n");
      }
   } else
      printf("C\n");
   exit(0);
}
```

   a) Dibuja un gráfico de la jerarquía de procesos que genera la ejecución de este código.

b) Muestra la salida que genera la ejecución de este código. ¿Puede producirse otra salida? Justifica la respuesta.

c) Modifca el programa para que la salida sea:

```
C
B
A
```

2. Sea el siguiente programa escrito en C:

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

main()
{  int i;

   for (i=0; i<3; i++)
      if (fork()!=0){
            printf("i=%d, soy padre\n",i);
            exit(0);
      } else {
            printf("i=%d, soy hijo\n",i);
            exit(0);
      }
}
```

a) Muestra un gráfico que ilustre la jerarquía de procesos que se crea durante su ejecución. Asigna un código identificativo a cada proceso.

b) Muestra la salida que genera la ejecución de este código, utilizando los códigos asignados en el gráfico anterior. ¿Es única? Justifica la respuesta.

c) Añade el código necesario para que el orden de ejecución sea tal que los respectivos procesos padre sean los últimos que se ejecuten.

3. Sea el siguiente programa escrito en C:

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

main()
{  int i;
   for (i=0; i<3; i++)
      if (fork()!=0) break;

   printf("PID=%d PPID=%d\n",getpid(),getppid());
   exit(0);
}
```

a) Muestra un gráfico que ilustre la jerarquía de procesos que se crea durante su ejecución. Asigna un código identificativo a cada proceso.

b) Muestra la salida que genera la ejecución de este código, utilizando los códigos asignados en el gráfico anterior. ¿Es única? Justifica la respuesta.

c) Añade el código necesario para que el orden de ejecución sea tal que los respectivos procesos padre sean los últimos que se ejecuten.

4. Sea el siguiente programa escrito en C:

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

main()
{  int i;

   for (i=0; i<3; i++)
      if (fork() == 0) break;

   printf("Soy el proceso PID %d y mi padre %d\n",
           getpid(),getppid());
   exit(0);
}
```

a) Muestra un gráfico que ilustre la jerarquía de procesos que se crea durante su ejecución. Asigna un código identificativo a cada proceso.

b) Muestra la salida que genera la ejecución de este código, utilizando los códigos asignados en el gráfico anterior. ¿Es única? Justifica la respuesta.

c) Añade el código necesario para que el orden de ejecución sea tal que los respectivos procesos padre sean los últimos que se ejecuten y para que la salida de la ejecución sea única.

5. Sea el siguiente programa escrito en C:

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

main()
{  int i;

   for (i=0; i<3; i++)
      if (fork() == 0) break;
   fork();
   printf("Soy el proceso PID %d y mi padre %d\n", getpid(),getppid());
   exit(0);
}
```

a) Muestra un gráfico que ilustre la jerarquía de procesos que se crea durante su ejecución. Asigna un código identificativo a cada proceso.

b) Muestra la salida que genera la ejecución de este código, utilizando los códigos asignados en el gráfico anterior. ¿Es única? Justifica la respuesta.

c) Añade el código necesario para que el orden de ejecución sea tal que los respectivos procesos padre sean los últimos que se ejecuten.

6. Sea el siguiente programa escrito en C:

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

main()
{  int nuevo,i;
   for (i=0; i<2; i++){
      nuevo = fork();
      if (nuevo == 0) break;
   }
   for (i=0; i<2; i++){
      nuevo = fork();
      if (nuevo != 0) break;
   }
   printf("Soy el proceso PID %d y mi padre %d\n",
           getpid(),getppid());
   exit(0);
}
```

a) Muestra un gráfico que ilustre la jerarquía de procesos que se crea durante su ejecución. Asigna un código identificativo a cada proceso.

b) Muestra la salida que genera la ejecución de este código, utilizando los códigos asignados en el gráfico anterior. ¿Es única? Justifica la respuesta.

c) Añade el código necesario para que el orden de ejecución sea tal que los respectivos procesos padre sean los últimos que se ejecuten.

7. Responde a las siguientes preguntas a partir del código que aparece a continuación:

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

int main(){
  int i, v=2;

  for (i=1;i<4;i++){
     if (fork() != 0){
         printf("Proceso=%d, iteracion=%d, v=%d\n",getpid(),i,v);
```

```
            break;
        } else {
            v = v * i;
            printf("Proceso=%d, iteracion=%d, v=%d\n",getpid(),i,v);

        }
    }
    exit(0);
}
```

a) Muestra en un gráfico la jerarquía de procesos que se genera durante su ejecución. Asigna un identificador a cada proceso.

b) Los procesos creados con este programa muestran por pantalla su salida en orden indefinido. Modifícalo para que, utilizando la función `wait`, impriman por pantalla en el mismo orden siempre.

c) Una vez realizados los cambios del apartado b), ¿cuál sería la salida mostrada por pantalla? Justifica la respuesta utilizando para cada proceso los mismos códigos que has utilizado en el apartado a).

8. A partir del siguiente programa escrito en C, muestra y justifica el valor que imprimirá cada proceso para la variable `contador`.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>

int main()
{
  int i,pid;
  int contador=1;

  contador = contador + contador;

  for (i=0;i<3;i++){
    contador = contador * 2;
    if (fork()==0) {
      contador = contador + (i+1);
      break;
    } else
      wait(NULL);
  }
  printf ("Proceso %d, contador = %d\n",getpid(),contador);
  exit(0);
}
```

9. A partir de los siguientes programas, muestra y justifica el valor que imprimirá cada proceso para la variable `contador` cuando se ejecuta el código del fichero `programa.c`.

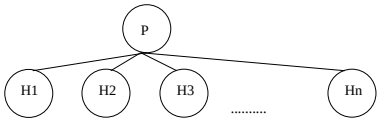
```
$ cat programa.c
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>


int main()
{
  int i,pid;
  int contador=1;

  contador = contador + contador;

  if (fork()!=0) {
    wait(NULL);
    for (i=0;i<3;i++){
      contador = contador * 2;
    }
    printf ("Proceso %d, contador = %d\n",getpid(),contador);
    exit(0);
  } else {
    execlp("prueba","prueba",NULL);
    perror("exec prueba");
    exit(-1);
  }
}
$ cat prueba.c
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>

int main()
{
   int contador;

   contador = contador * 4;
   printf("Proceso %d, contador = %d\n",getpid(),contador);
   exit(0);
}
```

10. A partir del siguiente programa escrito en C, muestra la jerarquía de procesos que se genera y lo que imprime cada proceso. Asocia a cada proceso un identificador. Justifica la respuesta.

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

#define PROCESOS 5

int contador=0;

int main()
   { int i;

   for (i=0;i<PROCESOS;i++)
   {  if (fork() != 0)
         wait(NULL);
      else
      {  contador = contador + i;
         printf("Proceso: %d contador = %d\n",getpid(), contador);
         exit(0);
      }
   }
   printf("Proceso: %d contador = %d\n",getpid(), contador);
   exit (0);
}
```

11. Implementa un programa en C que cree tres procesos hijos. El proceso padre debe esperar a que acaben los procesos hijos e imprimr un mensaje indicándolo. No es necesario que utilices un bucle `for`.

12. Diseña un programa en C que cree la jerarquía de procesos que se muestra en la siguiente figura, donde el número de hijos que se crean (H1, H2,...) es un valor que se introduce como argumento al programa.



Cada uno de los procesos hijos ejecutará un programa distinto cuyo nombre se introduce al programa como argumento y que se supone que está almacenado en el directorio de trabajo o en un directorio incluído en la variable `$PATH`.

Como ejemplo, si la ejecución del programa se realiza de la siguiente forma:
`$ programa 4 prog1 prog2 prog3 prog4`
el primero en ejecutarse será el programa `prog1`, después `prog2`, `prog3` y `prog4`. Las salidas de los programas no deben mezclarse. Y el proceso padre

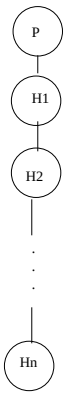
será el último que finaliza la ejecución. El código de cada uno de estos programas podría ser, por ejemplo:

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

#define N 15
main()
{
   int i;
   for (i=0;i<N;i++)
       printf("prog1 i=%d\n",i);
   exit(0);
}
```

El código de los restantes programas podría ser similar a este variando el valor de `N` y la palabra `prog1`.

Recuerda que las llamadas al sistema del tipo `exec` permiten que un proceso ejecute un código diferente al que estaba ejecutando hasta entonces.

13. Diseña un programa en C que cree la jerarquía de procesos que se muestra en la siguiente figura, donde el número de hijos que se crean (H1, H2,...) vendrá dado por el número de argumentos que se introducen al programa.



Cada uno de los procesos hijos ejecutará un programa distinto cuyo nombre se introduce al programa como argumento y que se supone que está almacenado en el directorio de trabajo o en un directorio incluído en la variable `$PATH`. El orden de ejecución de los comandos será el que se establece en la lista de argumentos del programa. Así, si la ejecución del programa se realiza de la siguiente forma:
`$ programa ls pwd date`
primero se ejecutará el comando `ls` y mostrará su salida en pantalla. Después el comando `pwd` y, finalmente, `date`. El proceso padre será el que termina en último lugar.

Recuerda que las llamadas al sistema del tipo `exec` permiten que un proceso ejecute un código diferente al que estaba ejecutando hasta entonces.

14. Diseña un programa en C que cree un proceso hijo que ejecute el comando que se le pasa como argumento al programa. El número de argumentos de dicho comando puede variar. Así, por ejemplo, si se invoca el programa de la siguiente forma:
`$ prog grep "hola" fichero.c`
el proceso hijo ejecutará el comando `grep` con todos los argumentos que se le pasan a continuación en la invocación del programa.

    El programa finalizará cuando el proceso hijo finalice su ejecución.

    Recuerda que las llamadas al sistema del tipo `exec` permiten que un proceso ejecute un código diferente al que estaba ejecutando hasta entonces.

15. Sea el código que aparece a continuación:

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

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

   if (argc < 2) {printf("Número de parámetos incorrecto\n"); exit(-1);}

   printf("\nSoy el padre %d y espero a que acabe mi hijo\n",getpid());
   if (fork()!=0)
         wait(NULL);
   else
   {
      printf("\nSoy %d y ejecuto %s:\n",getpid(),argv[1]);
      execvp(argv[1],&argv[1]);
      perror("El error es");
      exit (-1);
   }
   exit(0);
}
```

    a) Ejecútalo introduciendo comandos inválidos (por ejemplo, `lls`). ¿Qué salida genera el programa en ese caso y qué función o comando la genera?

    b) Ejecútalo pasándole como argumento `ls -l fich`, siendo `fich` el nombre de un fichero que no existe. ¿Qué salida genera? ¿Qué función o comando genera dicha salida?

16. Indica las transiciones de estado que generan las siguientes llamadas al sistema (si las hay):

(a) `wait()`

(b) `getpid()`

(c) `fork()`

(d) `exit()`

(e) `execlp()`

Explica en qué casos la rutina que atiende el servicio solicitado por estas llamadas llama al planificador del SO.

17. Responde a las siguientes preguntas a partir del código que aparece a continuación:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(){
  int i, v=1;

  for (i=0;i<3;i++){
     if (fork() == 0)
     {   v = v + 1;
         printf("proceso=%d, iteracion=%d, v=%d\n", getpid(), i, v);
         fork();
         v = v + 1;
         break;
     } else v = 3 * v;
  }
  printf("proceso=%d, iteracion=%d, v=%d\n", getpid(), i, v);
  exit(0);
}
```

(a) Muestra en un gráfico la jerarquía de procesos que se genera durante su ejecución. Asigna un identificador a cada proceso.

(b) Los procesos creados con este programa muestran por pantalla su salida en orden indefinido. Modifícalo para que, utilizando la función `wait`, impriman por pantalla en el mismo orden siempre.

(c) Una vez realizados los cambios del apartado b), ¿cuál sería la salida mostrada por pantalla? Justifica la respuesta utilizando para cada proceso los mismos códigos que has utilizado en el apartado a).

18. Responde a las siguientes preguntas a partir del código que aparece a continuación:

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

```
int main(){
  int i, v=1;

  for (i=0;i<2;i++){
     if (fork() != 0)
          if (fork() != 0) break;
     v = v + i;
  }
  printf("Proceso=%d, padre=%d, iteracion=%d, v=%d\n",getpid(),getppid(),i,v);
  exit(0);
}
```

(a) Muestra en un gráfico la jerarquía de procesos que se genera durante su ejecución. Asigna un identificador a cada proceso.

(b) Los procesos creados con este programa muestran por pantalla su salida en orden indefinido. Modifícalo para que, utilizando la función `wait`, impriman por pantalla en el mismo orden siempre.

(c) Una vez realizados los cambios del apartado b), ¿cuál sería la salida mostrada por pantalla?

19. Responde a las siguientes preguntas a partir del código que aparece a continuación:

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

int main(){
  int i, v=1;

  for (i=0;i<2;i++){
     fork();
     if (fork() != 0)  break;
     v = v + i;
  }
  printf("Proceso=%d, padre=%d, iteracion=%d, v=%d\n",getpid(),getppid(),i,v);
  exit(0);
}
```

(a) Muestra en un gráfico la jerarquía de procesos que se genera durante su ejecución. Asigna un identificador a cada proceso.

(b) Los procesos creados con este programa muestran por pantalla su salida en orden indefinido. Modifícalo para que, utilizando la función `wait`, impriman por pantalla en el mismo orden siempre.

(c) Una vez realizados los cambios del apartado b), ¿cuál sería la salida mostrada por pantalla?

20. Indica de manera justificada qué se mostrará por pantalla al ejecutar el programa `add.c`. Asume que `add2.c` está almacenado en el directorio de trabajo o en un directorio incluído en la variable `$PATH`.

```
$ cat add.c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

main()
{  int suma=4;;

   printf("Proceso %d: suma vale %d\n", getpid(), suma);
   if (fork()== 0)
   {
     printf("Proceso %d: suma vale %d\n", getpid(), suma);
     execlp("add2","add2", "5","3",NULL);
   }
   else wait(NULL);
   suma = suma + 2;
   printf("Proceso %d: suma vale %d\n", getpid(), suma);
   exit(0);
}
$ cat add2.c
#include <stdio.h>

main(int argc, char *argv[])
{  int a, b, suma;

   a = atoi(argv[1]);
   b = atoi(argv[2]);
   suma = a + b;
   printf("Proceso %d: suma vale %d\n", getpid(), suma);
   return(suma);
}
```

21. Explica qué cambios de estado se pueden producir en un proceso cuando este ejecuta cada una de las siguientes funciones:

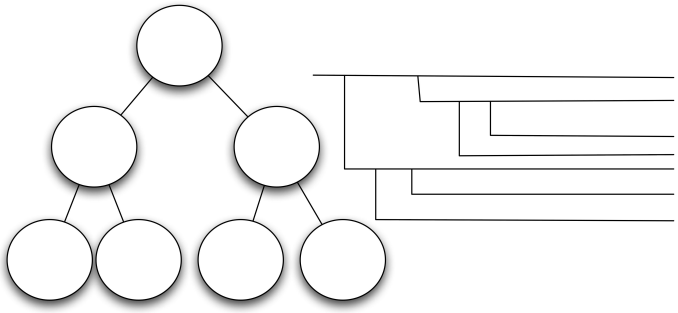
    (a) `fork()`

    (b) `execvp()`

    (c) `wait()`

    (d) `getpid()`.

22. Implementa un programa en lenguaje C que cree la siguiente jerarquía de procesos:



23. Responde a las siguientes preguntas a partir del código que aparece a continuación:

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

int main(){
  int i, v=0;

  for (i=0;i<2;i++)
  {
     if (fork() != 0) break;
     v = v + 2;
  }
  for (i=0;i<2;i++)
  {
     if (fork() == 0) break;
     v = v + 1;
  }
  printf("Proceso=%d, padre=%d, i=%d, v=%d\n", getpid(), getppid(), i, v);
  exit(0);
}
```

   (a) Muestra en un gráfico la jerarquía de procesos que se genera durante su ejecución. Asigna un identificador a cada proceso.

   (b) Los procesos creados con este programa muestran por pantalla su salida en orden indefinido. Modifícalo para que, utilizando la función `wait`, impriman por pantalla en el mismo orden siempre.

   (c) Una vez realizados los cambios del apartado b), ¿cuál sería la salida mostrada por pantalla? Utiliza para cada proceso los mismos códigos que has utilizado en el apartado a).

24. Indica de manera justificada si la siguiente información se guarda en el descriptor de un proceso (BCP):

(a) La tabla de ficheros abiertos por el proceso.

(b) Los punteros de lectura/escritura de los ficheros abiertos por el proceso.

(c) El estado del procesador cuando el proceso está en ejecución.

(d) El estado del proceso cuando el proceso está bloqueado.

25. Responde a las siguientes preguntas a partir del código que aparece a continuación:

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

int main(){
  int i, n;

  for (i=3;i>0;i--)
  {
     n=fork();
     if ((i==1) && (n==0)) break;
  }
  printf("Proceso=%d, padre=%d, i=%d\n", getpid(), getppid(), i);
  exit(0);
}
```

(a) Muestra en un gráfico la jerarquía de procesos que se genera durante su ejecución. Asigna un identificador a cada proceso.

(b) Los procesos creados con este programa muestran por pantalla su salida en orden indefinido. Modifícalo para que, utilizando la función `wait()`, impriman por pantalla en el mismo orden siempre.

(c) Una vez realizados los cambios del apartado b), ¿cuál sería la salida mostrada por pantalla? Utiliza para cada proceso los mismos identificadores que has utilizado en el apartado a).