Ejercicios de Sistemas Operativos. Tema 2: Procesos
E.T.S. Ingeniería Informática. Universidad de Málaga
Profesor: Manuel Ujaldón
Dpto. de Arquitectura de Computadores

---

1. ¿Cuántos procesos se crean en el siguiente código (incluido el proceso inicial)? Dibuja el árbol de procesos etiquetando sus nodos como A, B y C según representen los correspondientes puntos del código.

```
int main() {
  fork(); // A
  fork(); // B
  fork(); // C
  return 0;
}
```

2. ¿Cuántos procesos se crean en el siguiente código (incluido el proceso inicial)? Dibuja el árbol de procesos.

```
int main() {
  pid_t pid;
  int i;
  for (i=0; i<n; i++) {
    pid = fork();
    if (pid > 0)
    {
      pid = wait(NULL);
      exit(0);
    }
    else
      printf("¡Hola papá!\n");
  }
}
```

3. Sabiendo que `sleep(X)` hace esperar a un proceso X segundos, indicar qué palabra escribe en pantalla el siguiente código C. Puedes ayudarte del árbol de procesos.

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

void imprime(char *C)
{
  printf("%s",C);
  fflush(stdout);
}

int main() {
  pid_t pid, pid2;
  pid = fork();
  if (pid > 0) {
    imprime("A");
    pid2 = fork();
    imprime("C");
    sleep(1);
    if (pid2 > 0) {
      imprime("I");
      wait(NULL);
      imprime("N");
      wait(NULL);
      exit(0);
    }
    sleep(1);
    imprime("O");
  }
  else
  {
    sleep(3);
    imprime("?");
  }
}
```

4. Sabiendo que `sleep(X)` hace esperar a un proceso X segundos, indicar qué palabra escribe en pantalla el siguiente código C. Puedes ayudarte del árbol de procesos.

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

void imprime(char *C)
{
  printf("%s",C);
  fflush(stdout);
}

int main() {
  pid_t pid, pid2;
  pid = fork();
  if (pid > 0) {
    sleep(1);
    imprime("A");
    wait(NULL);
    fork();
    imprime("R");
    sleep(1);
    pid2 = fork();
    imprime("I");
    if (pid2 > 0) {
      sleep(2);
      imprime("B");
      wait(NULL);
      imprime("!");
      exit(0);
    }
    sleep(4);
    imprime("A");
  }
  else
    imprime("P");
}
```

5. Determina la salida por pantalla del siguiente programa que lanza el proceso *init*, al que presuponemos PID 1. Considera además que el PID del proceso que está ejecutando el programa es 27, y el de su hijo es 30.

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

int main() {
  pid_t pidfork, mypid, myparent;
  pidfork = fork();
  mypid = getpid();
  myparent = getppid();
  if (pidfork > 0) {
    printf("Parent speaking: First PID is %d.\n", pidfork);
    printf("Parent speaking: Second PID is %d.\n", mypid);
    printf("Parent speaking: Third PID is %d.\n", myparent);
    wait(NULL);
  }
  else {
    sleep(1);  // Espera un segundo a que acabe el padre
    printf("Child speaking: First PID is %d.\n", pidfork);
    printf("Child speaking: Second PID is %d.\n", mypid);
    printf("Child speaking: Third PID is %d.\n", myparent);
  }
}
```

6. Determina la salida por pantalla del siguiente programa y razona el resultado:

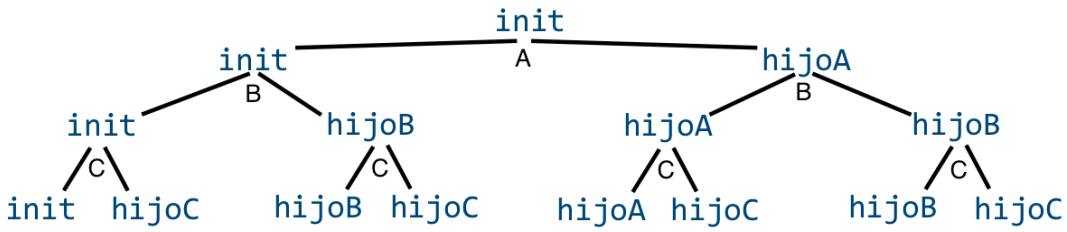
```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int value = 5;

int main() {
  pid_t pid;
  pid = fork();
  if (pid > 0) {
    wait(NULL);
    printf("Parent speaking: value is %d.\n", value);
  }
  else {
    value += 15;
    printf("Child speaking: value is %d.\n", value);
  }
}
```

7. Determina la salida por pantalla del siguiente programa y razona el resultado:

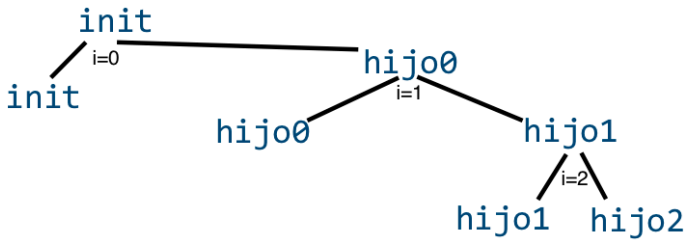
```
#include <sys/types.h>
#include <pthread.h>
#include <stdio.h>

int value = 5;

void *runner(void *param) {
  value += 15;
  pthread_exit(0);
}

int main() {
  pid_t pid;
  pthread_t tid;
  pthread_attr_t attr;

  pid = fork();
  if (pid > 0) {
    wait(NULL);
    printf("Parent speaking: value is %d.\n", value);
  }
  else {
    pthread_attr_init(&attr);
    pthread_create(&tid, &attr, runner, NULL);
    pthread_join(tid, NULL);
    printf("Child speaking: value is %d.\n", value);
  }
}
```

SOLUCIONES A LOS EJERCICIOS:

1. Se crean un total de 8 procesos, tal y como muestra el siguiente diagrama:



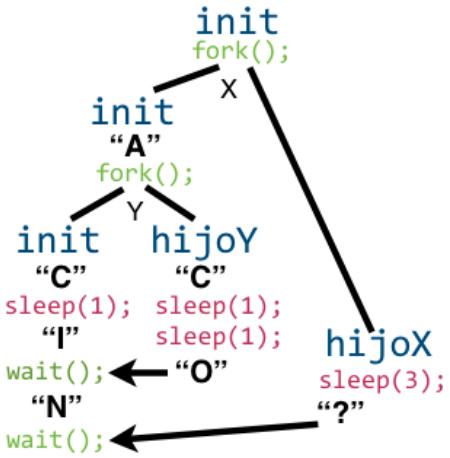
2. En general, se crea un proceso en cada una de las n iteraciones. Como se indica que se considere también el proceso inicial, la respuesta es $n+1$ procesos, tal y como muestra el siguiente diagrama:



3. Se escribe en la pantalla la palabra ACCION?.

```
pid = fork();           // Creación de un hijo X
if (pid > 0) {
  imprime("A");         // En pantalla: "A"
  pid2 = fork();        // Creación de un hijo Y
  imprime("C");         // En pantalla: "ACC"
  sleep(1);
  if (pid2 > 0) {
    imprime("I");       // En pantalla: "ACCI"
    wait(NULL);         // El padre recoge al hijo Y
    imprime("N");       // En pantalla: "ACCION"
    wait(NULL);         // El padre recoge al hijo X
    exit(0);            // El padre acaba
  }
  sleep(1);             // Por aquí continúa el hijo Y
  imprime("O");         // En pantalla: "ACCIO"
}
else
{
  sleep(3);             // Arranca el hijo X
  imprime("?");         // En pantalla: "ACCION?"
}
```

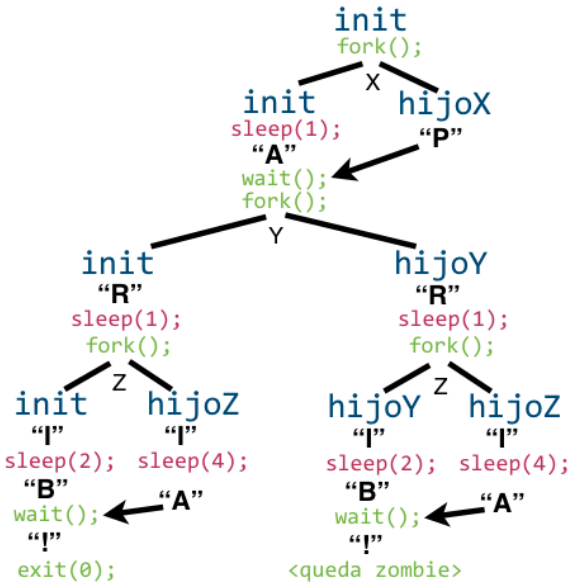
El árbol de procesos es el siguiente:



4. Se escribe en la pantalla la palabra PARRIIIIBBAA!!.

```
pid = fork();           // Creación de un hijo X
if (pid > 0) {
  sleep(1);
  imprime("A");         // En pantalla: "PA"
  wait(NULL);           // Recogida del hijo X
  fork();               // Creación de un hijo Y
  imprime("R");         // En pantalla: "PARR"
  sleep(1);
  pid2 = fork();        // Creación de dos hijos Z
  imprime("I");         // En pantalla: "PARRIIII"
  if (pid2 > 0) {
    sleep(2);
    imprime("B");       // En pantalla: "PARRIIIIBB"
    wait(NULL);         // Recogida de los dos hijos Z
    imprime("!");       // En pantalla: "PARRIIIIBBAA!!"
    exit(0);            // El padre acaba. El hijo Y queda zombie
  }
  sleep(4);             // Por aquí continúan los dos hijos Z
  imprime("A");         // En pantalla: "PARRIIIIBBAA"
}
else                    // Arranca el hijo X
  imprime("P");         // En pantalla: "P"
```

El árbol de procesos es el siguiente:



5. Se escribe en la pantalla la siguiente secuencia:

```
Parent speaking: First PID is 30.
Parent speaking: Second PID is 27.
Parent speaking: Third PID is 1.
Child speaking: First PID is 0.
Child speaking: Second PID is 30.
Child speaking: Third PID is 27.
```

Vemos que el padre barre la jerarquía completa, es decir, primero imprime el PID de su hijo, luego el suyo y finalmente el de su padre (que es el proceso `init`). El hijo hace lo mismo, salvo que el primer PID que imprime es 0 (no tiene hijos).

6. Veremos en pantalla los siguientes mensajes:

```
Child speaking: Value is 20.
Parent speaking: Value is 5.
```

Esto es así porque el padre no ve el espacio de direcciones de memoria del hijo. Por lo tanto, el incremento de la variable sólo tiene lugar en la copia local que alberga el hijo.

7. Veremos en pantalla los siguientes mensajes (aunque no necesariamente en ese orden):

```
Child speaking: Value is 20.
Parent speaking: Value is 5.
```

El proceso padre crea un proceso hijo que incrementa el valor de una variable que el padre no puede ver, ya que el hijo actúa en su espacio de direcciones privado (y clonado del padre).

El proceso hijo crea un hilo que incrementa el valor de una variable. Cuando el hilo acaba, el hijo ve ese incremento porque comparte la memoria con dicho hilo.