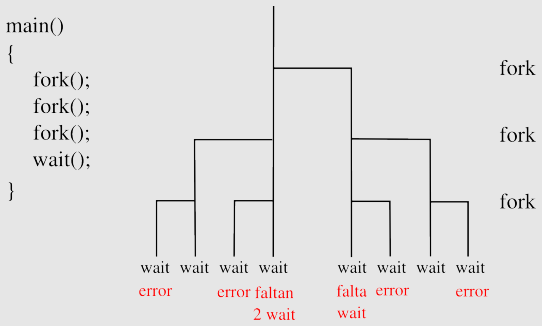
1. Se realiza el código adjunto. Realiza un diagrama con los procesos creados. Explica a través del diagrama cómo finalizan los procesos, si quedan procesos zombies o huérfanos y cuáles son si los hubiese.

```
main(){
    fork();
    fork();
    fork();
    wait();
}
```

Solución:



A modo de ilustración, este primer ejemplo se explicará con todo detalle.

- Con el primer fork() se lanza un proceso hijo (**H1**).
- En el segundo fork(), tanto el proceso padre como el proceso hijo lanza cada uno un proceso y por tanto hay un proceso padre (**P**), dos procesos hijos (**H1** y **H2**) y un proceso nieto (**N1**).
- En el tercer fork() el proceso padre, los dos procesos hijos y el proceso nieto lanzan un nuevo proceso cada uno de ellos. Así quedan un proceso padre (**P**), tres procesos hijos (**H1**, **H2** y **H3**), tres procesos nietos (**N1**, **N2** y **N3**, dos descendientes de **H1** y uno descendiente de **H2**) y un proceso biznieto (**B1**, descendiente del abuelo **H1**).
- Cada uno de los ocho procesos hace una llamada a la función wait(). Como se puede ver en la figura:
  - El proceso biznieto (**B1**), dos nietos (**N2** y **N3**) y el proceso hijo (**H3**) no tienen descendientes. De esta forma, la llamada a la función wait() devuelve un error al no detectar procesos hijos y los cuatro procesos pueden finalizar quedando en estado zombie hasta que su proceso padre pueda recogerlo y se libere definitivamente. *Metafóricamente, imagina un niño esperando en la puerta del colegio, una vez finalizada la jornada escolar (liberación de los recursos y memoria), hasta que su padre pasa a recogerlo (finalización del proceso).*
  - De igual forma el proceso hijo **H2** y el proceso nieto **N1** hacen una llamada a la función wait() y como sólo tienen un descendiente y es uno de los anteriores (**N2** o **N3**, según el orden de ejecución, para el proceso **H2** y **B1** para el proceso **N1**), recogerán a sus procesos hijos (*estos son los padres que recogen a los niños que están esperando en el colegio*). A su vez los procesos **H2** y **N1** enviarán la notificación a su padre de terminación. **H2** y **N1** pueden quedar en estado zombie o huérfano dependiendo de la dinámica de ejecución de sus procesos padre.

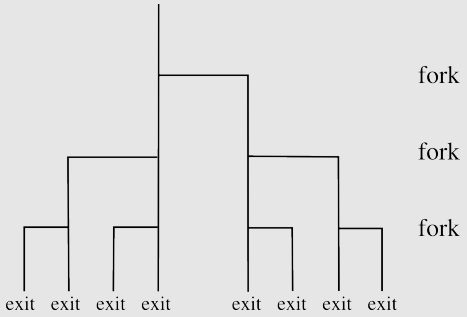
- El proceso hijo **(H1)** tiene dos descendientes (**N1** y (**N2** o **N3**, según el orden de ejecución)) y una única llamada a la función wait(), solo podrá finalizar completamente uno de los procesos. El otro proceso quedará huérfano, siendo el proceso init el que finalmente termine la ejecución del proceso huérfano.
- El proceso padre tiene tres descendientes y un sólo wait(). Como sus descendientes terminan correctamente eso implica que al menos dejará dos de sus descendientes huérfanos que serán adoptados por el proceso init.

La velocidad relativa de los procesos es la que va a determinar lo que realmente ocurra. A priori, no podemos decir con exactitud qué proceso va a terminar primero y cuál el último. Sí que sabemos con certeza que el proceso padre no será el último.

2. Realiza un diagrama con los procesos creados a partir de la ejecución del siguiente código. Identifica cada proceso con una letra e indica lo que imprimirán por la terminal cada uno de los procesos. Indica también si quedan procesos zombies o huérfanos y si se puede determinar cuáles.

```
main(){
    static int a[2]={0,0};
    static int c = 0;

    if(c == 3) exit(0);
    ++c;

    if (fork()) ++a[0];
    else ++a[1];
    main();

    printf ("(%d,%d)",a[0],a[1]);

    if(c != 3) wait();
}
```

Solución:



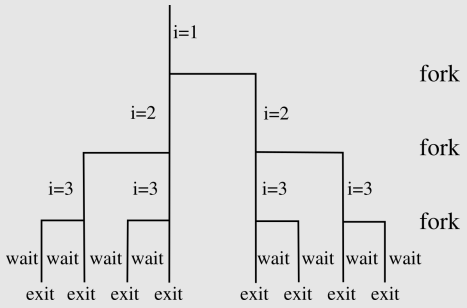
Todos los procesos generados, incluido el padre ejecutan la sentencia if(i == 3) exit(0) y por tanto ninguno ejecuta ningún wait. Esto hará que todos todos los procesos salvo el padre puedan convertirse en zombies o huérfanos dependiendo de la dinámica de los procesos.

No se imprimirá nada porque nunca llegará a la sentencia printf. Las dos últimas sentencias nunca se ejecutan.

3. Dibuja el árbol de procesos y determina qué procesos quedan zombies o huérfanos para el siguiente código:

```
main(){
    static int i=0;

    if(i != 3){
        ++i;
        fork();
        main();
    } else wait();
    exit(0);
}
```

Solución:



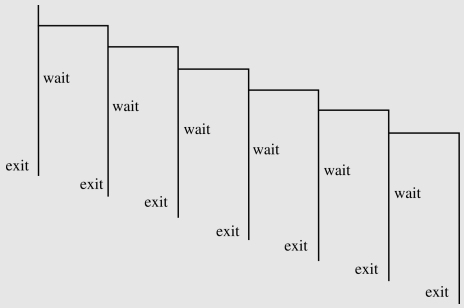
Todos los procesos generados realizan un wait. Los biznietos que no tienen hijos el sistema operativo detecta que no tienen hijos y por tanto el wait ni se bloquea ni produce error y el hijo termina correctamente. Todos los nietos hacen un wait y tienen un hijo con lo que todos los biznietos terminarán correctamente quedando en el estado de zombies sólo si terminan antes de que su padre haga el wait y dejarán de estarlo cuando el padre haga wait.

El problema principal proviene del padre y sus hijos. Todos ellos tienen más de un hijo y sólo hacen un wait. Eso implica que para cada uno de ellos, el hijo que termine primero finalizará correctamente como en el caso anterior pero el resto de los hijos podrán quedar huérfanos temporalmente y cuando realicen el exit todos quedarán zombies.

4. Dibuja un esquema en forma de árbol de los procesos que se crean según el código que se indica más abajo. Explica también si los procesos terminan correctamente o no lo hacen y por qué motivos.

```
main(){
    for (i=0; i< 6; ++i)
        if(fork()) break;
    if(i<5) wait();
    exit();
}
```

Solución:



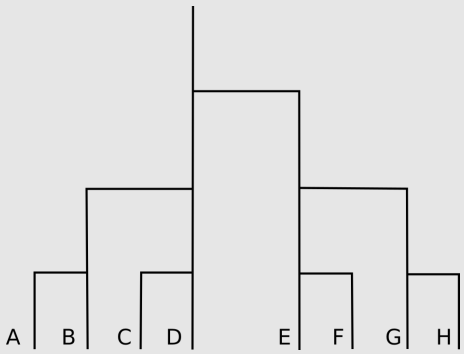
Todos los procesos menos el último realizan un wait antes de terminar. Todos los procesos terminarán correctamente y sólo temporalmente los hijos estarán en estado zombie si es que terminan antes de que el padre realice el wait. Además, terminan de forma ordenada desde el último hijo hasta el padre.

5. Se realiza el código adjunto. Realiza un diagrama con los procesos creados. Identifica cada proceso con una letra e indica lo que imprimirán por la terminal cada uno de los procesos.

```
main(){
    int a[2];

    a[0] = a [1] = 0;
    for ( i = 0; i < 3; ++i){
        if (fork()) ++a[0];
        else ++a[1];
    }
    printf ("(%d,%d)",a[0],a[1]);
}
```

Indica qué problema de diseño tiene el código presentado. Procesos huérfanos, procesos zombies, etc.

Solución:



Todos los procesos se quedan zombies menos el padre. Pueden haber estado incluso en el estado de huérfano si el padre de cada hijo termina antes que el hijo.

Lo que imprime cada proceso es lo siguiente:
A: 1,2; **B**: 2,1; **C**: 2,1; **D**: 3,0; **E**: 2,1; **F**: 1,2; **G**: 1,2; **H**: 0,3.

6. Se realiza el código adjunto. Realiza un diagrama con los procesos creados.

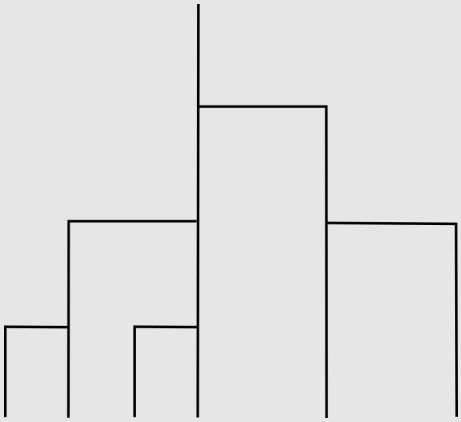
```
int main(){
    if( !fork()){
        fork();
        wait();
    } else {
        fork();
        fork();
        wait();
    }
    exit(EXIT_SUCCESS);
}
```

Se pide:

A. Dibujar la jerarquía de procesos que resulta de la ejecución de dicho código. Identifica cada proceso con una letra o numeración distinta. ¿Cuántos procesos, contando el proceso padre, se habrán generado en la ejecución del código?

B. Indica qué problema de diseño tiene el código presentado. Procesos huérfanos, procesos zombies, etc ...

Solución:



Todos los procesos realizan un wait. Los procesos que no tienen hijos su wait no produce error ni se bloquea y finalizan. Los procesos que sólo tienen un hijo realizan el wait de su hijo que, si acaba antes de que el padre llegue a ejecutar el wait estarán temporalmente zombies pero finalizarán correctamente. El padre es el único proceso que tiene más de un hijo y como sólo realiza un wait recogerá correctamente al hijo que finalice primero pero los otros dos procesos pueden quedar huérfanos si el padre y el primer hijo al que espera finalizan antes. Estos procesos cuando finalicen quedarán definitivamente zombies.

7. Se realiza el código adjunto. Realiza un diagrama con los procesos creados. Identifica cada proceso con una letra e indica lo que imprimirán por la terminal cada uno de los procesos.

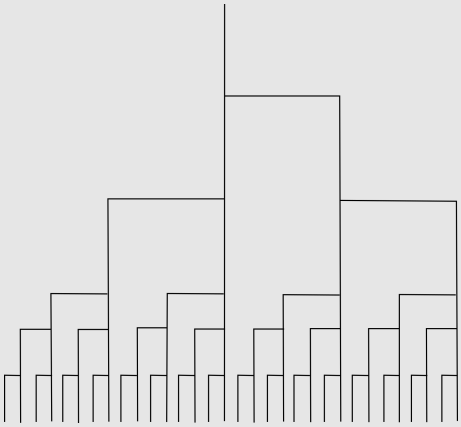
```
int main(){
      int i, pid;

      for ( i = 0; i < 3; i++){
          pid=fork();
          if(i%2) fork();
      }

      for( i = 0; i < 5 ; ++i) wait();
      exit(EXIT_SUCCESS);
  }
```

Se pide:

A. Dibujar la jerarquía de procesos que resulta de la ejecución de dicho código. Identifica cada proceso con una letra o numeración distinta. ¿Cuántos procesos, contando el proceso padre, se habrán generado en la ejecución del código?

B. Indica qué problema de diseño tiene el código presentado. Procesos huérfanos, procesos zombies, etc ...

Solución:



Todos los procesos realizan cinco waits. Como ningún proceso tiene más de 5 hijos pueden sobrarles waits pero como el sistema operativo sabe que no tienen hijos los wait no se bloquean y los porcesos finalizan correctamente. Si cualquier hijo termina antes de que el padre realice el wait estará temporalmente zombie hasta que su padre realice un wait.

8. Se realiza el código adjunto. Realiza un diagrama con los procesos creados. Identifica cada proceso con una letra e indica lo que imprimirán por la terminal cada uno de los procesos.

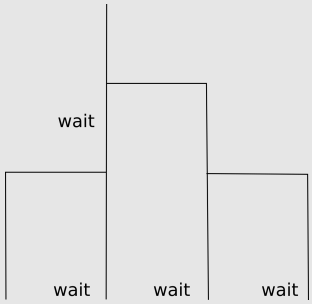
```
main(){
      if(fork()){
          wait();
          if(fork()) wait();
      } else {
          fork();
          wait();
      }
      exit(0);
  }
```

Se pide:

A. Dibujar la jerarquía de procesos que resulta de la ejecución de dicho código. Identifica cada proceso con una letra o numeración distinta. ¿Cuántos procesos, contando el proceso padre, se habrán generado en la ejecución del código?

B. Indica qué problema de diseño tiene el código presentado. Procesos huérfanos, procesos zombies, etc ...

Solución:



El hijo nieto realiza un wait de más pero como el sistema operativo sabe que no tiene hijos ni se bloquea ni devuelve error. Este proceso queda zombie temporalmente si termina antes de que su padre realice el wait. Cuando este proceso recoge a su hijo puede terminar correctamente y por tanto terminar (en las mismas condiciones de zombie temporal). En el momento en el que el padre de todos realiza el wait y su hijo ha terminado entonces puede continuar y lanzar un nuevo hijo para el que hace un nuevo wait. Cuando el proceso hijo termine el padre lo recogerá correctamente (en el mismo posible estado de zombie temporal).

9. Se realiza el código adjunto.

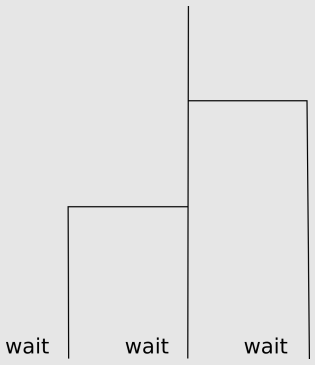
```
main(){
    int i=0;
    pid_t pid;

    while((pid = fork()) != 0 && i < 1){
        if(pid) fork();
        else wait();
        ++i;
    }
    wait();
}
```

Se pide:

A. Dibuja la jerarquía de procesos que resulta de la ejecución de dicho código. Identifica cada proceso con una letra o numeración distinta. ¿Cuántos procesos, contando el proceso padre, se habrán generado en la ejecución del código?

B. Indica qué problema de diseño tiene el código presentado. Procesos huérfanos, procesos zombies, etc ...

Solución:



Los dos hijos hacen un wait que no se bloquea ni devuelve error porque no tienen hijos. Sin embargo el padre sólo hace un wait para dos hijos. El primero que termine podrá estar temporalmente zombie hasta que el padre haga el wait y el segundo podrá estar huérfano temporalmente hasta que muera en cuyo caso estará zombie porque el padre no hace más waits.