Universidad Autónoma Metropolitana
Unidad Azcapotzalco

Sistemas Operativos

Práctica 02: Procesos con llamadas al sistema

1. Códigos y árboles de procesos

1.1. Un padre y un hijo y un subhijo

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main() {
        int idf;
        idf = fork();
        if ( idf == 0 ) {
                idf = fork();
        }
        printf("mi id es %d\n",(long) getpid());
        sleep(5);
        return 0;
}
```

Al ejecutar el código anterior e imprimir su árbol de procesos se obtiene lo siguiente:

```
[rebeca@rebeca fork]$ pstree -Ap 5136
fork(5136)---fork(5137)---fork(5138)
```

1.2. Un padre y dos hijos

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main() {
        int idf;
        idf = fork();
        if ( idf != 0 ) {
                fork();
        }
        printf("mi id es %d\n",(long) getpid());
        sleep(5);
        return 0;
}
```

Al ejecutar el código anterior e imprimir su árbol de procesos se obtiene lo siguiente:

```
[rebeca@rebeca fork]$ pstree -Ap 5492
fork2(5492)-+-fork2(5493)
            `-fork2(5494)
```

1.3. Un padre con una lista de N subhijos secuenciales

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main() {
        int n, idf;
        scanf("%d", &n);
        idf = fork();
        for (int i = 0; i < n-1; ++i) {
                if ( idf == 0 ) {
                        idf = fork();
                }
        }
        printf("mi id es %d\n",(long) getpid());
        sleep(5);
        return 0;
}
```

Al ejecutar el código anterior y darle un valor a N igual a 3, obtenemos el suguiente árbol de procesos:

```
[rebeca@rebeca fork]$ pstree -Ap 5590
fork3(5590)---fork3(5591)---fork3(5592)---fork3(5593)
```

1.4. Un padre y N hijos

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main() {
        int idf, n;
        scanf("%d", &n);
        idf = fork();
        for (int i = 0; i < n; ++i) {
                if ( idf != 0 ) {
                        idf = fork();
                }
        }
        printf("mi id es %d\n",(long) getpid());
        sleep(5);
        return 0;
}
```

Al ejecutar el código anterior y darle valor a N igual a 5, se obtiene el siguiente árbol de procesos:

```
fork4(6191)-+-fork4(6192)
            |-fork4(6193)
            |-fork4(6194)
            |-fork4(6195)
            |-fork4(6196)
            `-fork4(6197)
```

1.5. Un padre con dos hijos y un subhijo cada uno de ellos

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main() {
        int n, idf;
        idf = 1;
        for (int i = 0; i < 2; ++i) {
                if ( idf != 0 ) {
                        idf = fork();
                        if (idf == 0) {
                                fork();
                        }
                }
        }
        printf("mi id es %d\n",(long) getpid());
        sleep(5);
        return 0;
}
```

Al ejecutar el código anterior e imprimir su árbol de procesos se obtiene lo siguiente:

```
[rebeca@rebeca fork]$ pstree -Ap 6441
fork5(6441)-+-fork5(6442)---fork5(6444)
            `-fork5(6443)---fork5(6445)
```

2. Cuestionario

1. Mencione las dos maneras en las que se han creado procesos.

El primer proceso se genera al ejecutar un programa desde la terminal, cuando se ejecuta este programa y manda llamar la función $fork()$ se crea otro proceso mediante llamadas al sistema.

2. ¿Qué sucedió con la primera estructura que generó? ¿Implementó alguna manera para que el proceso padre esperara a los procesos hijo? En caso afirmativo, indique cuál.

Generó un hijo y un subhijo. Con la estructura de control if, sabiendo que la función $fork()$ regresa cero para el proceso que se acaba de crear se pudo condicionar la creación del nuevo procedimiento para que solamente el hijo del proceso inicial llamara de nuevo la función $fork()$ y se creará el subhijo del proceso inicial. No se solucionó que el padre esperara a los hijos, en el caso específico de estos programas, al ser pequeños los hijos siempre terminan antes que los padres.

3. Indique las llamadas al sistema utilizadas hasta ahora.

La función $fork()$ y en la práctica anterior se usó el comando *kill*

4. ¿Existe otra manera en la que un proceso padre espere a todos sus procesos hijos? Específique La función $waitpid(pid, \&statloc, opciones)$, espera a que un hijo termine, se le envía como primer parámetro el id del proceso hijo que se va a esperar.