Técnicas Digitales III

Trabajo práctico: Procesos

Creación de procesos

1. Compile y ejecute prc01.c

Compile el programa	`$ gcc -o prc01 prc01.c`
Ejecute	`$ ./prc01`

   ¿Qué es el PID ? ¿Cómo es el PID del padre respecto del PID del hijo?, ¿por qué?

2. Compile y ejecute prc02.c. Luego, desde otra consola, ejecute el comando "`pstree -p`" e identifique los procesos en ejecución.

Compile el programa	`$ gcc -o prc02 prc02.c`
Ejecute	`$ ./prc02`
Visualice el árbol de procesos	`$ pstree -p`

3. El programa prc03.c ejecuta 2 veces la función `fork()` y hace una espera activa de 30 segundos con la función sleep(). Compile y ejecute prc03.c. Luego, desde otra consola ejecute el comando `pstree -p` e identifique los procesos en ejecución.

4. La función `fork()` devuelve el pid del proceso hijo cuando lo ejecuta el padre; y devuelve 0 (cero) cuando lo ejecuta el proceso hijo. ¿Qué estructura de bifurcación de C le parece más conveniente para implementar que padre e hijo ejecuten diferente código (`if, while, for, case`)? Modifique prc02.c con la estructura de bifurcación seleccionada. Compile y ejecute el programa.

5. Compile y ejecute prc05.c. La variable *x* es inicializada en 100. Luego es decrementada por el proceso hijo e incrementada por el proceso padre. ¿Por qué *x* nunca retorna a su valor original?

6. Modifique el programa prc05.c de tal manera de que además de imprimir el valor de la variable *x*, imprima la dirección en memoria de esta variable (*&x*). Ejecute el programa y explique la salida del mismo.

7. Tome proc02.c y ponga las funciones de las líneas 14 y 15 dentro de un bucle que se repita 3 veces. Imprima también el valor de la variable de control del bucle (variable i). Analice y deduzca cuántos hijos son creados. Justifique su respuesta. ¿Qué sucede con el valor de i?.

Finalización de procesos

8. Tome el programa del Ej. 4 y fuerce a que el proceso hijo haga una espera activa de 30 segundos con la función `sleep()`. El proceso padre debe terminar antes que el

proceso hijo. ¿Qué sucede con el proceso hijo?. Ejecute "`pstree -p`" e identifique si persisten los procesos en cuestión. Observe los números de pid.

9. Tome el programa del Ej. 8 y agregue al final del código del proceso padre la función `wait(NULL)`. Ejecute "`pstree -p`" e identifique si persisten los procesos en cuestión. Observe los números de pid.

10. Tome el programa del Ej. 5 y fuerce a que el proceso hijo entre en un bucle infinito con la función `while(1)`. ¿Qué sucede con el proceso hijo? ¿Con qué combinación de teclas puede terminar el proceso hijo?

11. Similar a lo hecho en el Ej. 10, ahora fuerce a que el proceso padre entre en un bucle infinito con la función `while(1).` Visualice luego los procesos en ejecución con "`pstree -p`" e identifique el proceso "zombie".

Manejo de flujos (streams)

12. Escriba y compile un programa que imprima un texto en el flujo *stdout* y otro texto en el flujo *stderr* mediante la función `fprintf()`.

```
fprintf(stdout, "Texto stdout\n");
fprintf(stderr, "Texto stderr\n");
```

Ejecute el programa desde la consola. ¿A dónde está direccionado cada flujo?.

13. Desde la consola o a través de un script ejecute los siguientes comandos:

```
Ejecute                    ls -al  > ./stdout
Ejecute                    cat stdout
```

¿Qué operaciones se han realizado?. Luego de ejecutar el primer comando, ¿observa algo por consola?, ¿por qué?.

14. Ejecute el programa creado en el Ej. 12 de la siguiente manera:

```
Ejecute                    ./prc12 2> err.txt
```

Compare el resultado con la salida del Ej. 12. ¿Qué observa por consola?. ¿Cuál es el contenido del archivo err.txt? ¿Cuál es la función del operador "`2>`" ?

Función execl()

15. Compile y ejecute prc15.c. ¿Qué sucede al ejecutar la función execl()?

16. En el archivo prc15.c, comente la línea 13 y descomente la línea 14. Compile y ejecute. ¿Qué observa por consola? ¿Por qué ha cambiado la salida respecto al ej. 15?.