Universidad del Valle de Guatemala

Sistemas Operativos

Douglas de León Molina

Carné 18037

Laboratorio 2

# Ejercicio 1

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

R// En ambos procesos se crean 16 procesos.

R// Hay la misma cantidad de procesos en uno que en otro debido a que se llama el *fork()* 4 veces en ambos programas a pesar de que uno esté dentro de un *foor loop*.

# Ejercicio 2

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, application

Description automatically generated

R// Toma más tiempos el primero que no realiza los loops de forma concurrente.

R// La diferencia de tiempo se da debido a que el primero no realiza las tareas de forma concurrente.

# Ejercicio 3

Un cambio de context volutario ocurre cuando un proceso requiere de recursos que no están disponibles. Un cambio de contexto involuntario ocurre cuando el tiempo de duración se corta o se identifica un proceso de mayor prioridad.

Graphical user interface, application

Description automatically generated

R// Al momento de presionar las teclas o mover la ventana cambian principalmente Xorg, gnome-terminal, acpid, metacity y los eventos.

Graphical user interface, application, table

Description automatically generated

Application, table

Description automatically generated

R// En el primero se da una mayor cantidad de cambios de contexto involuntarios. Sin embargo, en el segundo se podría decir que se dan más en total si se suman los de todos los procesos, pero en promedio son menores. En el segundo igual se da un cambio voluntario que no se da en el primero.

R// Considero que esto se le puede atribuir a las llamadas a wait(NULL).

R// Los involuntarios se le pueden atribuir a otros procesos del sistema operativo que quiza en dado momento tienen mayor prioridad que nuestro programa.

R// En mi caso sí hubo cambios de contexto (se incrementé la cantidad de loops al for), pero sí hay uno que tiene menos que todos. Este podría ser el padre, puesto que este no realizaba el loop como los demás.

Graphical user interface, table

Description automatically generated with medium confidence

R// Al momento de escribir mientras se ejecutaba el programa la cantidad de cambios involuntarios incremento considerablemente.

# Ejercicio 4

Graphical user interface, table

Description automatically generated

R// Significa que el proceso es “Zombie”.

Graphical user interface, table

Description automatically generated

Table

Description automatically generated

R// En la ventana donde se ejecutó el programa no ocurre nada.

R// Cuando el proceso se queda huérfano, se vuelve hijo del proceso con PID 1.

# Ejercicio 5

Graphical user interface, text, application

Description automatically generated

R// La principal diferencia pienso que es la velocidad de la comunicación, puesto que es mucho más rápido acceder a algo en memoria primaria que a memoria secundaria.

R// No se puede utilizar el file descriptor de la otra instancia puesto que al momento de llamar este retorna -1, un error.

R// Sí, es posible. Lo hace a través de un file descriptor y memoria compartida entre proceso padre e hijo.

Graphical user interface, text, application

Description automatically generated

R// Se puede verificar puesto que al momento de intentar abrirlo, retorna un error con número 17. Este se puede validar dentro del código para saber de que el archivo ya está abierto.

R// La memoria aún puede ser utilizada desde los procesos que aún estén mapeadas, pero no se pueden agregar nuevos procesos.

R// Los punteros apuntan a registros de memoria.

R// Se puede liberar llamando shm\_unlink desde un programa independiente.

R// Según lo que investigué utilizar *usleep(1000)* al momento de hacer la llamada al *fork()* soluciona el problema.