Josué Sagastume - 18173
Sistemas Operativos

LABORATORIO 2

Ejercicio 1

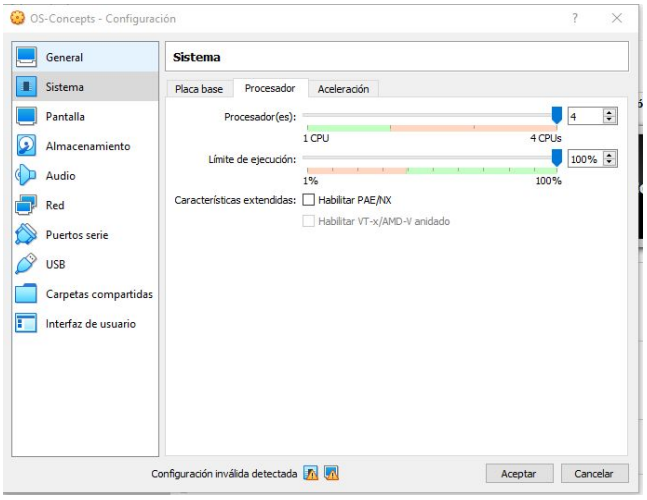

Imagen 1. Asignación de procesadores a la máquina virtual.

- ¿Cuántos procesos se crean en cada uno de los programas?

Para el primer programa se crearon 16 procesos y en el programa con fork dentro del ciclo for se crean 32 procesos.

- ¿Por qué hay tantos procesos en ambos programas cuando uno tiene cuatro llamadas fork() y el otro sólo tiene una?

En el primer programa es fácil explicar porqué, pues por cada fork se crea un proceso hijo, dándonos un total de 2^n procesos, que en este caso sería 2^4 = 16 procesos. Para el segundo programa, esto se complica un poco, pues ya que estamos agregando un ciclo for a la ecuación, este lo que provoca es que el proceso hijo y el proceso padre se ejecuten dos veces, duplicando así la cantidad de procesos, dándonos 32 procesos.

Ejercicio 2

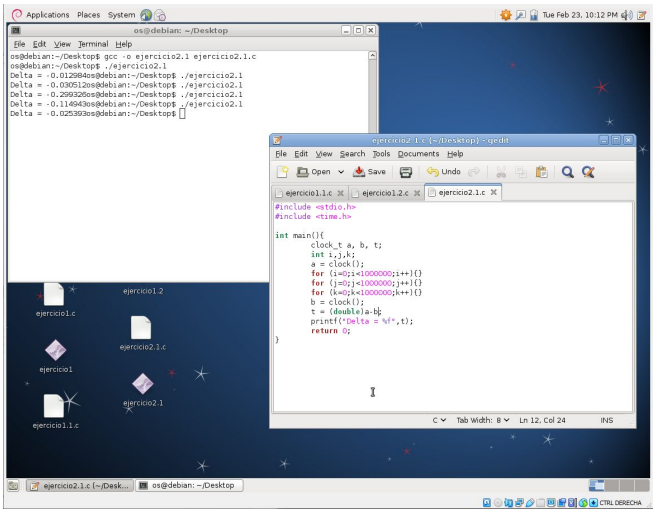

Imagen 2. Tiempo de ejecución del ejercicio2.1.c con un promedio de 0.0966316 segundos.

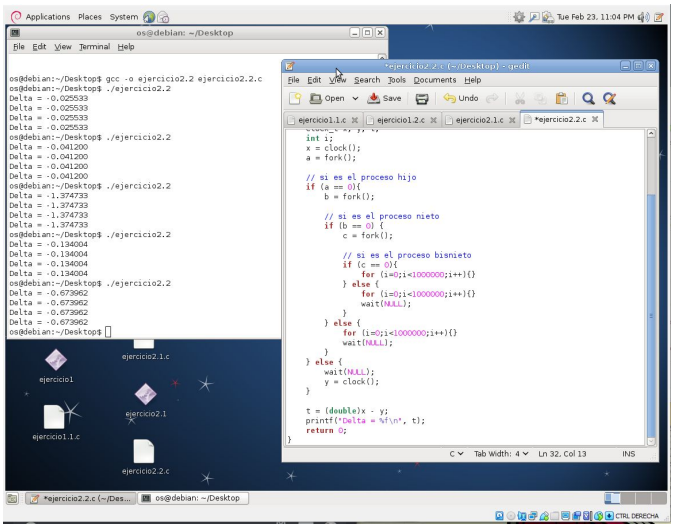

Imagen 3. Tiempo de ejecución del ejercicio 2.2.c con un promedio de 0.4498864 segundos.

- ¿Cuál, en general, toma tiempos más largos?

  El segundo programa tiene un promedio de ejecución más largo.

- ¿Qué causa la diferencia de tiempo, o por qué se tarda más el que se tarda más?

Esto se debe a que los procesos padres tienen que esperar a que los procesos hijos terminen de ejecutarse para poder empezar ellos.

Ejercicio 3

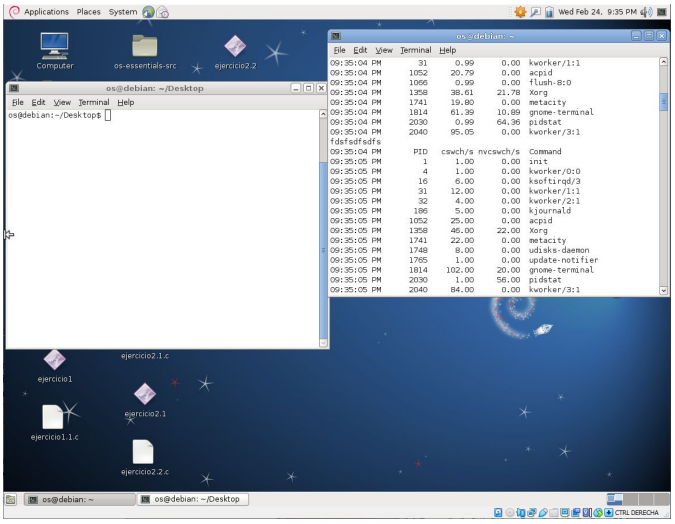

Imagen 4. Ejecución del comando pidstat -w 1.

- ¿Qué tipo de cambios de contexto incrementa notablemente en cada caso, y por qué?

Las cosas que cambian son el xorg, kworker y gnome terminal. Esto se debe a que gnome-terminal y xorg son cambios de contexto voluntarios, pues estos trabajan con el entorno gráfico. Para el caso de los cambios de contexto involuntarios, estos se dan debido al cambio en la interfaz gráfica, pues el sistema los reconoce de esta manera.

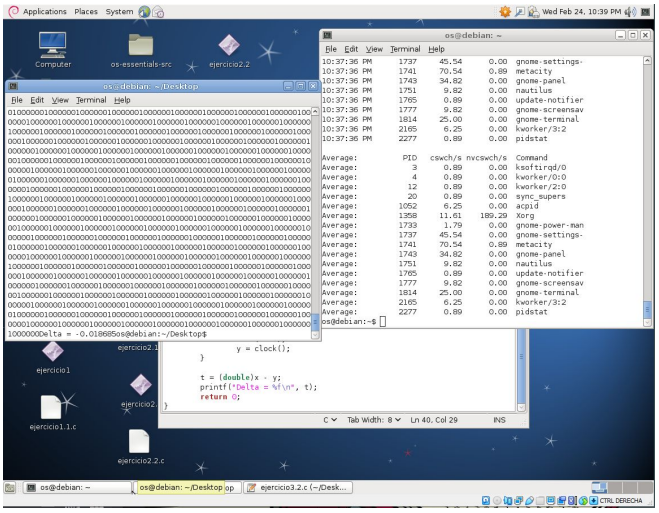

Imagen 5. Ejecución del ejercicio3.1.c junto con el comando pidstat -w X 1.

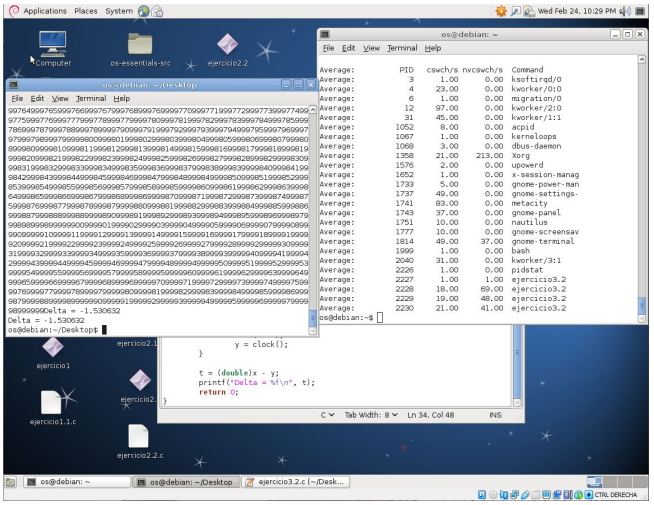

Imagen 6. Ejecución del ejercicio3.2.c junto con el comando pidstat -w X 1.

- ¿Qué diferencia hay en el número y tipo de cambios de contexto de entre programas?

El segundo programa que cuenta con los forks, este se es ejecutado más veces, pues por cada fork, este proceso es duplicado, generando más cambios de contexto.

- ¿A qué puede atribuirse los cambios de contexto voluntarios realizados por sus programas?

Al procesador, ya que para poder ser eficiente y estar trabajando y ocupado siempre, este provocó o hizo estos cambios de contexto voluntarios.

- ¿A qué puede atribuirse los cambios de contexto involuntarios realizados por sus programas?

Estos cambios se generaron al querer seguir ejecutándose los procesos, pero el sistema operativo los detiene para empezar a ejecutar otros procesos.

- ¿Por qué el reporte de cambios de contexto para su programa con forks muestra cuatro procesos, uno de los cuales reporta cero cambios de contexto?

Esto se debe a que durante la ejecución del programa se crearon cuatro forks, lo que hizo que se crearán cuatro procesos diferentes, y el que no produjo cambios de contexto es el padre de todos los procesos.

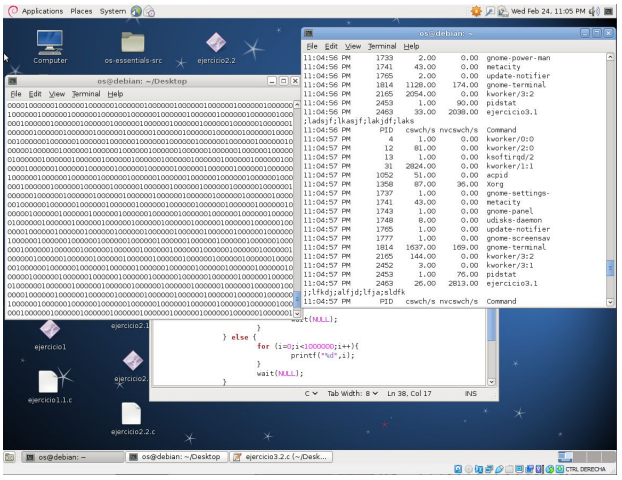

Imagen 7. Ejecución del ejercicio3.2.c junto con el comando pidstat -w 1.

- ¿Qué efecto percibe sobre el número de cambios de contexto de cada tipo?

Los cambios de contexto del programa en ejecución aumentaron bastante, pues normalmente eran de 200 aproximadamente, y cuando empecé a escribir en la terminal, estos aumentaron hasta más de 2500. Esto es debido a que los eventos de entrada y salida disminuyeron los cambios de contexto involuntarios, pero aumentaron los voluntarios.

Ejercicio 4

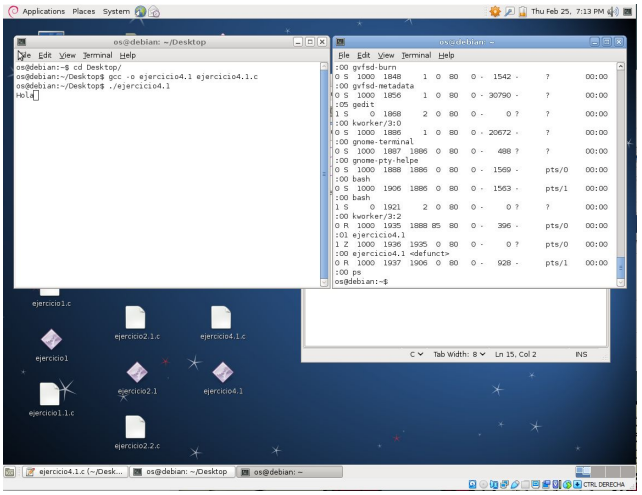

Imagen 8. Ejecución del ejercicio4.1.c junto con el comando ps-ael.

- ¿Qué significa la Z y a qué se debe?

La Z es de “zombie”, este es un proceso que ha completado su ejecución pero aún tiene una entrada en la tabla de procesos, permitiendo al proceso que lo ha creado leer el estado de su salida.

- ¿Qué sucede en la ventana donde ejecutó su programa?

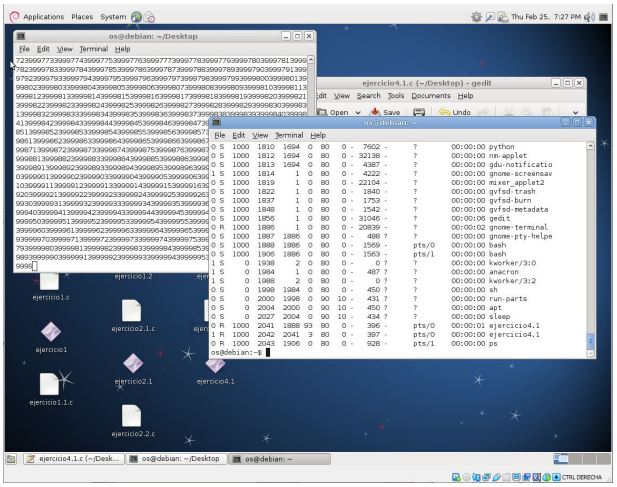

Imagen 9. Ejecución del ejercicio4.1.c con conteo junto al comando ps-ael.

- ¿Qué sucede en la ventana donde ejecutó su programa?

El programa sigue ejecutándose ya que el hijo sigue funcionando igual a pesar de que el proceso padre fue asesinado.

- ¿Quién es el padre del proceso que quedó huérfano?

Se puede decir que al quedar estos procesos huérfanos, son adoptados por el init, por lo que ahora su ppid es 1.

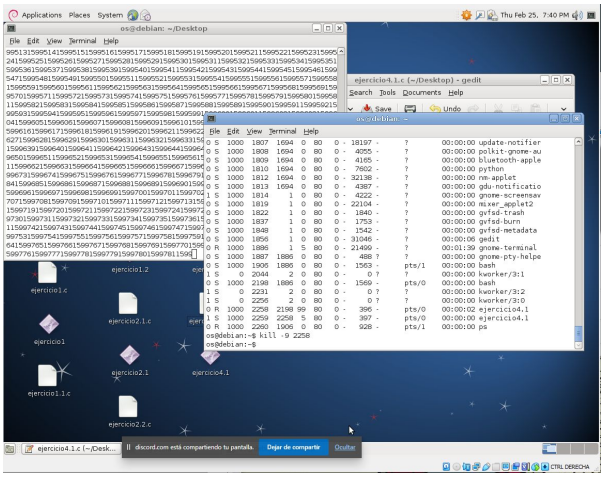

Imagen 10. Ejecución del comando kill -9 al proceso padre.

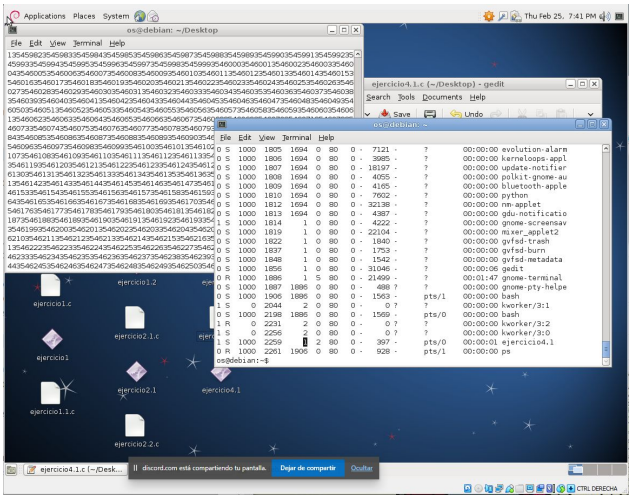

Imagen 11. El ppid del proceso huérfano ahora es 1.

Ejercicio 5

- ¿Qué diferencia hay entre realizar comunicación usando memoria compartida en lugar de usando un archivo de texto común y corriente?

El cambio más visible es la velocidad de comunicación, pues ahora los procesos comparten memoria. Pero al tener que manejar la memoria por código, este ahora es más susceptible y vulnerable a errores.

- ¿Por qué no se debe usar el file descriptor de la memoria compartida producido porotra instancia para realizar el mmap?

Al contar con diferentes file descriptors, estos tienen la capacidad de poder detectar el mismo archivo. Esto puede ocasionar que ambos puedan modificar y sobreescribir el mismo archivo, generando así problemas.

- ¿Es posible enviar el output de un programa ejecutado con exec a otro proceso por medio de un pipe? Investigue y explique cómo funciona este mecanismo en la terminal (e.g.,la ejecución de ls | less).

Si es posible, pues la terminal puede manipular los file descriptors de un hijo al redirigir la entrada. Dado a que un proceso recién creado, este puede acceder a los mismos archivos que el proceso padre. Por lo que si ejecutamos un fork luego de crear un pipe, tanto el hijo como el padre pueden comunicarse entre sí por medio del pipe.

- ¿Cómo puede asegurarse de que ya se ha abierto un espacio de memoria compartida con un nombre determinado? Investigue y explique el error.

Para esto se utiliza la librería errno, pues el valor contenido en esta variable sólo es significativo cuando el valor de retorno de la llamada indicó un error. Este error solo podría darse cuando se crea una memoria compartida ya existente.

- ¿Qué pasa si se ejecuta shm_unlink cuando hay procesos que todavía están usando la memoria compartida?

Solo se dejaría de manejar la memoria compartida, pues se desvinculan, aunque algún proceso aún esté utilizando el objeto.

- ¿Cómo puede referirse al contenido de un espacio en memoria al que apunta un puntero? Observe que su programa deberá tener alguna forma de saber hasta dónde ha escrito su otra instancia en la memoria compartida para no escribir sobre ello.

Se puede referir al contenido utilizando los comandos "&" y "*", con estos podemos acceder tanto al valor como a la dirección de una variable a través de un puntero.

- Imagine que una ejecución de su programa sufre un error que termina la ejecución prematuramente, dejando el espacio de memoria compartida abierto y provocando que nuevas ejecuciones se queden esperando el file descriptor del espacio de memoria compartida. ¿Cómo puede liberar el espacio de memoria compartida "manualmente"?

Para esto se puede crear otro programa que se encargue de desconectar la memoria compartida. Se puede utilizar shmctl(shm_id, IPC_RMID, NULL);.

- Observe que el programa que ejecute dos instancias de ipc.c debe cuidar que una instancia no termine mucho antes que la otra para evitar que ambas instancias abran y cierren su propio espacio de memoria compartida. ¿Aproximadamente cuánto tiempo toma la realización de un fork()? Investigue y aplique usleep.

Esto depende mucho de la velocidad de los procesadores utilizados por el sistema. Usualmente suele ser un tiempo de orden x10^-6, es decir, de microsegundos. Al utilizar usleep se puede solucionar este problema de tiempos entre instancias.