Kristen Brandt 171482

Lab 2

Graphical user interface, text, application, email

Description automatically generated

**Ejercicio 1:**

Text

Description automatically generated

Text

Description automatically generated

Se crean 2^n procesos para los forks() donde n es el numero de forks que se hacen. En el primer caso se crean 16 procesos. En el primer programa hay tantos forks porque cada fork tiene dos child processes.

Text

Description automatically generated

Text

Description automatically generated

En el segundo solo se crea una llamada al sistema, un proceso. Por lo que solo se imprime 1 la palabra fork una sola vez.

**Ejercicio 2:**

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text, chat or text message

Description automatically generated

Compare los resultados de tiempos de cada uno de sus programas, y responda:

¿Cuál, en general, toma tiempos más largos?

En general el primer programa toma tiempos mas largos.

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

La diferencia de tiempo es causada por la concurrencia. En el primer programa se hace un proceso a la vez, y cuando este se termine ya se sigue con el siguiente proceso. En el segundo programa no pasa esto, haciéndolo mas rápido.

**Ejercicio 3:**

Text, letter

Description automatically generated

Contextos voluntarios e involuntario:

Un cambio voluntario es cuando se realiza una llamada a el sistema y hay que esperar por cada evento. Estos procesos voluntarios pasan de estar en ejecución a estar bloqueados. El motivo es la eficiencia en el uso de procesador.

Un cambio involuntario es cuando el sistema operativo le quieta UCP al proceso. Estos procesos pasan de en ejecución a listo. El motivo es el repartimiento del procesador.

Table

Description automatically generated with medium confidence

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

El tipo de cambios de contexto que incrementa es el pidstat y Xorg al solo teclear y abrir otras pantallas. Xorg incrementa porque este esta encargado de la interfaz gráfica.

Utilizando los prints:

 primer programa

 segundo programa

Pidstat, Xorg, gnome-terminal son los procesos utilizados al correr los programas. Al correr el primer programa los procesos involuntarios aumentan y en el segundo programa los procesos involuntarios también aumentan.

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

La diferencia que hay en el numero y tipo de cambios de contexto entre programas es que el segundo programa por los forks concurrentes sube el numero de cambios voluntarios. El numero de cambios involuntarios es menor en el segundo programa que en el primero.

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

Se le puede atribuir a los cambios de contexto voluntarios a los fork. Esto es porque uno no empieza hasta que el otro termine.

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

Se le puede atribuir a los cambios de contexto involuntarios a los procesos que tiene el sistema operativo.

¿Por qué el reporte de cambios de contexto para su programa con fork()s muestra cuatro procesos, uno de los cuales reporta cero cambios de contexto?

El reporte de cambios de contexto para el programa con forks muestra cuatro forks. Uno reporta cero cambios de contexto porque son los hijos los que se están ejecutando.

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

El efecto que observe del número de cambios de contexto es que aumentan los cambios involuntarios.

**Ejercicio 4:**

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

¿Qué significa la Z y a qué se debe?

La Z significa que es un proceso que ya terminado, pero falta hacer algo con el, esta en un estado zombie.

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

¿Qué sucede en la ventana donde ejecutó su programa?

En la ventada donde ejecute mi programa sale la palabra Killed y termina de correr el proceso.

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

¿Quién es el padre del proceso que quedó huérfano?

El padre del proceso que quedo huérfano es el primero de cada 2 procesos cuando se trata de un fork.

**Ejercicio 5:**

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

La diferencia entre realizar comunicación usando memoria compartida en lugar de usando un archivo de texto es que en el archivo de texto se haría información duplicada en la memoria. La memoria compartida se va a utilizar por los programas ejecutados y esta no va a tener información duplicada.

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

File descriptor no se debe usar de la memoria compartida para realizar el mmap porque son instancias diferentes. Como son instancias diferentes usan distintas partes/espacio de la memoria.

¿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 enviar el output de un programa ejecutado con exec a otro proceso por medio de un pipe. La ejecucion de ls | less. El comando ls imprime todo lo que esta en el directorio y less se utiliza para mostrar el contenido de un archivo o una salida de comando una pagina a la vez. Esto nos deja poder navegar en el archivo.

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

Uno se puede asegurar que ya se ha abierto un espacio de memoria compartida con cierto nombre porque se puede utilizar este espacio para hacer operaciones. Si este espacio no esta abierto sale un mensaje de error. Errno es corto para "error number", este es una variable que guarda el numero de error que se produjo. Dependiendo del numero que errno guarda el error.

¿Qué pasa si se ejecuta shm\_unlink cuando hay procesos que todavía están usando la memoria compartida?

El comando shm\_unlink está encargado de eliminar el nombre del objeto de la memora compartida. Si hay referencias al objeto de memoria compartida cuando se utiliza el comando shm\_unlink el nombre se elimina antes de que regrese la función. No se hace una eliminación de la memoria hasta que el mapa al objeto de memoria compartida sea eliminado.

¿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 de un espacio en memoria al que apunta un puntero ya que este se tiene que inicializar y luego se le asigna un valor correspondiente del espacio de memoria a donde apunta.

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”?

Se puede liberar el espacio de memoria compartida "manualmente" utilizando la función munmap. Esta función esta encargada de eliminar cualquier mapeo que contiene parte del espacio de direcciones del proceso. Si no hay memoria compartida asignada en el rango de direcciones que fue asignada la función munmap no hay efectos.

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.

La realización de un fork() es extremadamente rápido, aproximadamente un milisegundo. Si se quisiera averiguar el tiempo exacto se puede utilizar clock\_gettime (fork()).

La función usleep se utiliza para retrasar la ejecución del programa por un numero determinado de microsegundos. Al usar usleep en el programa nos aseguramos de que las dos instancias no terminen mucho antes que la otra.