Laboratorio

Tree of Processes

Ciencias de la Computación VII

El objetivo de este laboratorio es experimentar cómo se comporta la creación de un Proceso en Windows y Linux utilizando C.

Es difícil escribir un documento técnico en español, pues me rehuso a traducir algunos términos y conceptos importantes relacionados a los tópicos de computación, pues en el proceso de traducción generalmente se pierde algo del concepto original, y la lectura se tiende a dificultar. Es por ello que en este documento muchos de los términos importantes se dejarán en inglés, para facilitar la comprensión y la búsqueda del contenido, y evitar malas traducciones. A pesar que por momentos la combinación de los dos idiomas puede resultar hasta cómica, prefiero eso a escribir cosas como: El ordenador, el cortafuegos, el proceso padre, el fichero, etc.

– Etson Guerrero

Introducción

Creación de procesos

Para levantar más procesos utilizamos **syscall o System Call** estos es un mecanismo utilizado para pedir un Servicio al Sistema Operativo, en este caso el **Create Process**. El proceso que crea a los otros procesos se le conoce como el **Parent process** y los nuevos procesos que se levantan generalmente son llamados **Children processes**. Cada uno de los nuevos procesos también pueden crear otros procesos, formando lo que se conoce como el **Tree of Processes** la mayoría de los sistemas operativos identifican los procesos con un **Process Identifier**, el cual generalmente es un número entero positivo.

En general, un proceso necesitará de ciertos recursos para completar una tarea. Cuando un proceso crea a un subproceso, este último puede obtener sus recursos directamente desde el OS, o puede ser limitado a un subconjunto de los recursos del **Parent process**, compartiendolos entre todos los subprocesos. Como por ejemplo la memoria y los archivos. Restringir a un **Child process** para que solo pueda acceder a un subconjunto de los recursos del parent process es una forma que se emplea para evitar que la creación de subprocesos sobrecargue al sistema.

Por ejemplo, consideremos un proceso cuya función es la de desplegar en pantalla el contenido de la imagen img.jpg y dicho proceso es creado desde la línea de comando enviándole como parámetro el nombre del archivo, por lo que este proceso tomará como input desde su parent process el nombre del archivo img.jpg, y entonces abrirá el archivo para mostrarlo en pantalla.

Al crear un nuevo proceso, podemos tener dos posibilidades:

1. El parent continúa ejecutándose concurrentemente con su subproceso.
2. El parent espera hasta que alguno o todos sus subprocesos hayan terminado.

Y tenemos dos posibilidades en términos del **Address Space**:

1. El nuevo proceso es un duplicado del parent process. Es decir, tiene el mismo programa y data que su parent.
2. El nuevo proceso es un nuevo programa cargado a memoria.

Instrucciones:

Deberá compilar y ejecutar los dos programas que ejemplifican la creación de procesos en Linux y en Windows que están adjuntos a este Laboratorio. Adicional debe responder a las interrogantes que se presentan en las actividades de cada sección.

Descripción de Creación de Procesos:

En Linux

Los procesos creados con la función fork(), después de crear un nuevo **Child process**, ambos procesos (Child and Parent) ejecutarán la siguiente instrucción después del Syscall. Un proceso secundario usa el mismo PC (Program Counter), los mismos registros de CPU, los mismos archivos abiertos que se usan en el proceso primario.

En el archivo **linux.c** se utiliza fork(), el cual al ser llamado crea un subproceso que ejecuta **ls** (**ls**t files and documents).

En Windows

Ahora analicemos la creación de procesos en Windows. Los procesos son creados utilizando el API Win32, con la función **CreateProcess()**, y se diferencia del **fork()** en el hecho que en vez de heredar el **address space** de su parent, **CreateProcess()** requiere especificar el nombre del programa a cargar en el **address space** del nuevo proceso. Además **fork()** no requiere parámetros, mientras que **CreateProcess()** espera no menos de 10 parámetros.

En el archivo **windows.c** se utiliza **CreateProcess()**, el cual crea un nuevo proceso que carga en memoria **mspaint.exe**. De momento utilizaremos los valores default para la mayoría de parámetros del CreateProcess(). Para probar este código podría crear un proyecto en **DevC** , o en cualquier **compilador de C** para Windows. (Nota, este programa es del tipo Console Application.)

Dos de los parámetros enviados a CreateProcess() son instancias de las estructuras **STARTUPINFO** y **PROCESS_INFORMATION**. **STARTUPINFO** específica muchas de las propiedades para el nuevo proceso, como por ejemplo el tamaño y apariencia de la ventana, así como los handlers para el input y output. La estructura **PROCESS_INFORMATION** contiene el handler e identificadores asociados al proceso que se está creando y a su hilo de ejecución. La función **ZeroMemory()** sirve para reservar la memoria para cada una de estas estructuras antes de invocar al método **CreateProcess()**.

Los dos primeros parámetros que se envían al **CreateProcess()** son el nombre de la aplicación y los parámetros de línea de comando. Si el nombre de la aplicación se envía como **NULL**, el segundo parámetro especifica la aplicación a cargar. En este caso se está levantando una instancia del **mspaint.exe**.

Actividades:

Coloque sus respuestas en un archivo .txt (**respuestas.txt**)

Procesos en Windows:

1. ¿Cuál es el resultado de este programa?
2. Ejecute el programa **procexp.exe** que se encuentra en la carpeta Soporte a los Laboratorios del GES. Tome su tiempo para revisar y explorar las funcionalidades de la aplicación.
3. Con el **procexp.exe** corriendo, vuelva a ejecutar el código anterior. Analice y revise la estructura del árbol de procesos. ¿Qué relación tiene su programa con el **mspaint.exe**?
4. ¿Qué sucede en el árbol de procesos si cierra la ventana de consola asociada a su programa? ¿**mspaint.exe** continua activo?
5. Para analizar el funcionamiento de la instrucción **WaitForSingleObject(),** pruebe comentar la llamada a dicha función y observe el comportamiento de su programa en el árbol de procesos.
6. ¿Existe alguna similitud entre las instrucciones **wait()** y **WaitForSingleObject()** de los programas que ejecuta en Linux y Windows?

Procesos en Linux:

1. ¿Cuál es el resultado de este programa?
2. Describa paso a paso lo que sucede en este programa.
3. ¿Qué sucede si comenta la instrucción **wait(NULL)**?, ¿Cambian los resultados? ¿Por qué?
4. ¿Cuál es la función de los **if** que validan la variable **pid** (process id)?
5. ¿Qué sección de código o programa se ejecuta cuando se crea el nuevo proceso? (Indique lo que sucede al momento de ejecutar las instrucciones **fork()** y **execlp()**)

Entrega:

- El laboratorio debe ser entregado por medio del GES, con todos los archivos en un **ZIP**. No se calificarán laboratorios entregados tarde o por medio de URL externo.
- El laboratorio puede tener una calificación de -100 si se detecta.

Para este Laboratorio se adjunta un archivo ZIP con el código a compilar en ambos sistemas, además de procexp.exe para windows, puede hacer uso de ello para realizar su laboratorio.