**CREACION DE PROCESOS EN WINDOWS Y LINUX**

**Introducción**

Esta práctica de laboratorio ilustra la creación de procesos en Windows y Linux utilizando C. Es difícil escribir un documento técnico en español, pues me reuso 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 tiende a dificultarse. Es por ello que en este documento muchos de los términos importantes se dejaran en ingles, 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*

**Creación de procesos**

Un proceso puede levantar mas procesos utilizando el system call “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 identificador único o PID, Process Identifier, el cual generalmente es un número entero. En UNIX, podemos obtener una lista de todos los procesos utilizando el comando ps. Por ejemplo, ejecutando el comando ps –el, se mostrará en pantalla el listado de todos los procesos activos en el sistema. Con dicho listado, es fácil construir el “tree of processes”, recorriendo el listado recursivamente utilizando el identificador del parent process.

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, compartiéndolos entre todos los subprocesos. Como por ejemplo la memoria y los archivos. Restringir a un child process para que solo pueda accesar 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.

Muchos de los recursos lógicos y físicos que el proceso obtiene cuando es creado pueden ser transferidos desde el parent process hacia el child process (inicialización de datos, input, etc.). 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.

Cuando se crea a 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.

Deberá compilar y ejecutar dos programas que ejemplifican la creación de procesos en Linux y en Windows. Adicional debe responder a las interrogantes que se presentan en las actividades de cada sección. Al entregar su solución no olvide renombrar este archivo con su nombre y número de carnet. Utilice color azul para el color del font de su respuesta.

***Creación de procesos 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 addres space de su parent, CreateProcess() requiere especificar el nombre del programa a cargar en el address space del nuevo proceso. Ademas fork() no requiere parámetros, mientras que CreateProcess() espera no menos de 10 parámetros.

A continuación analizaremos un programa que ilustra la utilización de la función 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 podrían crear un proyecto en DevC ☺, o en cualquier compilador de C para Windows. (Nota, este programa es del tipo Console Application.)

#include <cstdlib>

#include <iostream>

#include <windows.h>

#include <stdio.h>

using namespace std;

int main( VOID )

{

STARTUPINFO si;

PROCESS\_INFORMATION pi;

ZeroMemory( &si, sizeof(si) );

si.cb = sizeof(si);

ZeroMemory( &pi, sizeof(pi) );

// Start the child process.

if( !CreateProcess( NULL, // No module name (use command line).

"C:\\WINDOWS\\system32\\mspaint.exe", // Command line.

NULL, // Process handle not inheritable.

NULL, // Thread handle not inheritable.

FALSE, // Set handle inheritance to FALSE.

0, // No creation flags.

NULL, // Use parent's environment block.

NULL, // Use parent's starting directory.

&si, // Pointer to STARTUPINFO structure.

&pi ) // Pointer to PROCESS\_INFORMATION structure.

)

{

printf( "CreateProcess failed (%d).\n", GetLastError() );

return -1;

}

// Wait until child process exits.

WaitForSingleObject( pi.hProcess, INFINITE );

// Close process and thread handles.

CloseHandle( pi.hProcess );

CloseHandle( pi.hThread );

}

*.*

Dos de los parámetros enviados a CreateProcess() son instancias de las estructuras STARTUPINFO y PROCESS\_INFORMATION. STARTUPINFO especifica 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 esta 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().

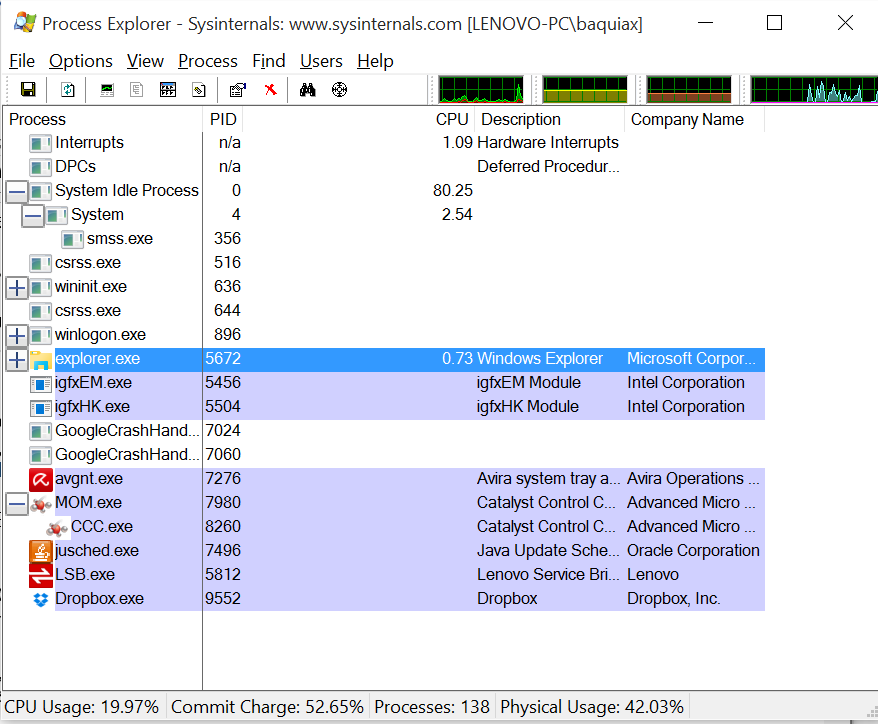
Los dos primeros parámetros que se envían al CreateProcess() son el nombre de la aplicación y los parámetros de línea de comando. Si el nombre de la aplicación se envía como NULL, el segundo parámetro especifica la aplicación a cargar. En este caso se esta levantando una instancia del mspaint.exe.

**Procesos en Windows: Actividades**

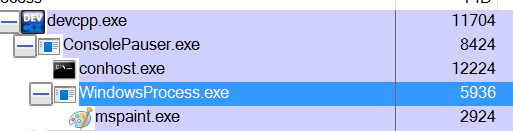
1) ¿Cuál es el resultado de este programa?

**Se ha abierto MS Paint.**

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.

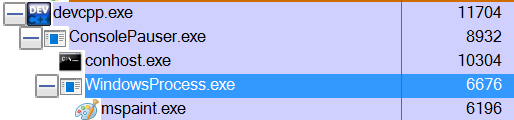


3) Con el procexp.exe corriendo, vuelva a ejecutar el código anterior. Analice y revise la estructura del árbol de procesos. ¿Qué relación tiene su programa con el mspaint.exe? ,



**En este caso nuestro programa es el “parent process” de mspaint.exe**

4) ¿Qué sucede en el árbol de procesos si cierra la ventana de consola asociada a su programa? ¿Mspaint.exe continua activo?



**Se cierra la ventana de consola y…**

**im2.PNG**

**MSPaint sigue abierto.**

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.

**En esta ocasión WindowsProcess(mi binario) sólo abrió mspain y se cerro, sin esperar si mspaint se cerró o no.**





6) ¿Existe alguna similitud entre las instrucciones wait() y WaitForSingleObject() de los programas que ejecutó en Linux y Windows?

**La similitud que veo es que ambos nos sirven para identificar cuando un proceso hijo ha terminado de ejecutarse.**

***Creación de procesos en UNIX y Linux.***

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

int main()

{

pid\_t pid;

/\* fork a child process \*/

pid = fork();

if (pid < 0) { /\* error occurred \*/

fprintf(stderr, "Fork Failed\n");

exit(-1);

}

else if (pid == 0) { /\* child process \*/

printf("I am the child %d\n",pid);

execlp("/bin/ls","ls",NULL);

}

else { /\* parent process \*/

/\* parent will wait for the child to

complete\*/

printf("I am your father %d\n",pid);

wait(NULL);

printf("Child Complete: I’ll never join you!! \n");

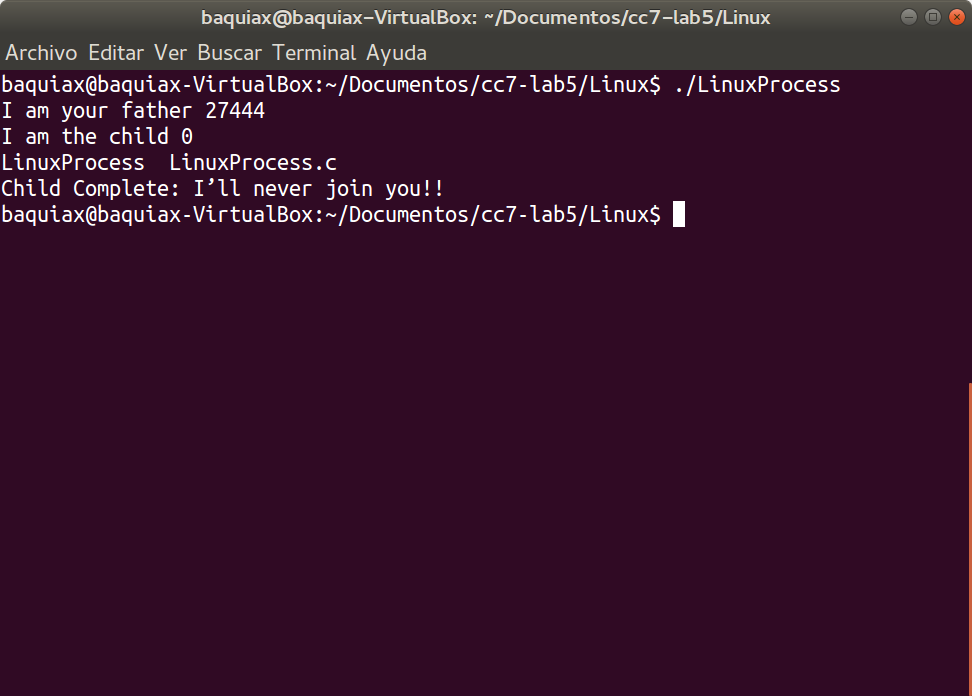
exit(0);

}

}

**Procesos en Linux: Actividades**

1) ¿Cuál es el resultado de este programa?



2) Describa paso a paso lo que sucede en este programa.

1. **Primero se define una variable de tipo (struct) pid\_t que definirá el identificador del proceso a crear.**
2. **Se hace luego una llamada al fork() que crea un nuevo proceso retornando un elemento de tipo pid\_t**

**El valor de retorno de fork() puede tomar tres valores**

* 1. **ID negativo: La creación del proceso no se realizó con éxito.**
  2. **ID cero: devuelve un cero al proceso recién creado.**
  3. **ID positivo: devuelve el PID al padre del proceso recién creado.**

1. **Si el resultado del fork() fue negativo, entonces se imprime en pantalla un mensaje descriptivo que denota ese error. Al final se envía un exit(-1) que reporta al sistema una salida con estado -1.**
2. **Si la respuesta del fork() es 0, entonces se imprime un mensaje que describe que quien recibió ese mensaje fue el “child process” recién creado. Adicional a esto se hace un “ls”.**
3. **Por último si la respueta del fork() es un número pisitivo, entonces se imprime una frase alegórica y se h6ace una llamada a wait() que tiene como función detener la ejecución hasta que al menos un “child process” termine. Si esto es así se imprime un último mensaje.**

3) ¿Qué sucede si comenta la instrucción wait(NULL)?, ¿Cambian los resultados? ¿Por qué?

**El programa nunca termina, a diferencia de tenerlo descomentado. Y esto es porque al no ejecutar wait() nunca se notifica que el “child process” ha terminado de ejecutarse.**

4) ¿Cuál es la función de los if que validan la variable pid (process id)?

**Están ahí porque al ejecutar fork(), tanto como el “parent process” como el “child proces” ejecutarán las instrucciones inmediatamente después de fork(), por ello es necesario distinguirlos con las sentencias if colocadas.**

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() )

**Se ejecuta todo el código inmediatamente después de fork(). Como mencioné anteriormente, fork() crear un nuevo proceso y permite que ambos ejecuten las instrucciones que están después de esta llamada. Cuando quien esté ejecutando el bloque de código sea el child process, entonces se hace un ls ejecutado a través de execlp().**

**Actividades Generales**

1. Discuta sobre las similitudes y diferencias entre la creación de procesos en Linux y en Windows.

**Puede notarse que la creación de un proceso en Linux es más sencilla. La diferencia más grande la tiene el createProcess de Windows y el fork() de Linux. fork() define de alguna manera, qué puede ejecutar el child process y también permite compartir espacios de memoria entre parent y child.**