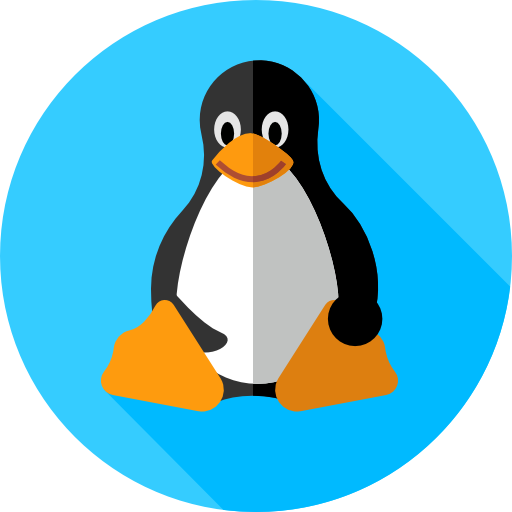
Logotipo

Descripción generada automáticamente

***Instituto Politecnico Nacional***

**Escuela Superior de Cómputo**

Práctica 3

**Práctica 3. Administrador de procesos en Linux y Windows (1)**

**Sistemas Operativos**

Grupo: 2CM12

Integrantes:

* Baldovinos Gutiérrez Kevin
* Bocanegra Heziquio Yestlanezi
* Castañares Torres Jorge David
* Hernández Hernández Rut Esther

Profesor

Jorge Cortes Galicia

Pantalla de computadora

Descripción generada automáticamente con confianza baja

# Introducción

## Fork

El proceso puede realizar más de una función al mismo tiempo, puede crear procesos totalmente independientes.

La mayoría de los sistemas operativos identifican los procesos según un identificador de proceso único (pid)

que suele ser un número entero.

En Unix la llamada al sistema fork se utiliza para crear un nuevo proceso, que se convierte en el proceso hijo

de la persona que llama. Ambos procesos se ejecutarán simultáneamente a partir de la siguiente instrucción

siguiente a la llamada al sistema fork(). Hay que distinguir el padre del hijo.

Cuando un proceso crea un nuevo proceso, existen dos posibilidades de ejecución:

* El padre continúa ejecutándose simultáneamente con sus hijos.
* El padre espera hasta que algunos o todos sus hijos hayan terminado.

También existen dos posibilidades en cuanto al espacio de direcciones del nuevo proceso:

* El proceso hijo es un duplicado del proceso padre (tiene el mismo programa
* y datos que el padre).
* El proceso hijo tiene un nuevo programa cargado en él.

## Execv

La llamada al sistema sustituye el programa que se está ejecutando por una imagen de programa recién cargada.

imagen del programa.

La sintaxis de execv sería

Execv (const char \*program, char \*\*args);

El primer argumento de execv() será una ruta al programa.

El segundo argumento de execv() será una matriz de punteros a cadenas terminadas en cero

que representan la lista de argumentos disponibles para el nuevo programa.

## Getpid

getpid() devuelve el ID del proceso (PID) del proceso que llama.

(Esto se utiliza a menudo por las rutinas que generan nombres de archivos temporales).

## Getppid

getppid() devuelve el ID del proceso padre del proceso que llama.

Este será el ID del proceso que creó de este proceso usando fork(), o, si ese proceso ya ha terminado, el ID del proceso al que este proceso ha sido repartido (ya sea init(1) o un proceso "subreaper" definido mediante

la operación PR\_SET\_CHILD\_SUBREAPER de prctl(2)).

## Wait

Wait comando que se utiliza para supervisar el proceso anterior, depende del proceso anterior estado de la devolución se le devolverá el estado de salida. Por ejemplo, si queremos esperar a completar un determinado proceso de IDENTIFICACIÓN de 13245, deberíamos usar «Wait 13245» cuando el proceso 13245 completa Wait comando de retorno de los valores de retorno de 13245 estado de salida.

– Wait PID (PID – Proceso de IDENTIFICACIÓN de un comando, por lo que la utilidad es esperar a la terminación).  
– Wait JID (JID – ID de Trabajo que identifica un proceso en segundo plano que esperar, aplicable sólo para las invocaciones de Wait en el shell actual entorno de ejecución).

# Competencia.

El alumno aprende a familiarizarse con el administrador de procesos del sistema operativo Linux y Windows a través de la creación de nuevos procesos por copia exacta de código y/o por sustitución de código para el desarrollo de aplicaciones concurrentes sencillas.

# Desarrollo.

Sección Linux:

1. Introduzca los siguientes comandos a través de la consola del sistema operativo Linux:

# Ps

# Texto Descripción generada automáticamente

Imagen 1 Comando ps desde Virtual Box Ubuntu 20.0

**ps –fea**

**Texto

Descripción generada automáticamente**

Imagen 2 Comando ps -fea desde Virtual Box Ubuntu 20.0

Este comando nos proporciona poder visualizar el estado de los procesos que se están realizando en Linux.

1. A través de la ayuda en línea que proporciona Linux, investigue para que se utiliza el comando **ps** y mencione las opciones que se pueden utilizar con dicho comando. Además investigue el uso de las llamadas al sistema **fork(), execv(), getpid(), getppid()** y **wait()** en la ayuda en línea, mencione que otras funciones similares a execv() existen, reporte sus observaciones.

Véase las llamadas a funciones en la [introducción](#_Introducción) del mismo documento.

1. Capture, compile y ejecute los dos programas de creación de un nuevo proceso por copia exacta de código que a continuación se muestra. Observe su funcionamiento y experimente con el código.

#include <stdio.h> #include <unistd.h> #include <stdlib.h> int main(void)

{

int id\_proc; id\_proc=fork(); if (id\_proc == 0)

{

printf(“Soy el proceso hijo\n”);

}

else

{

printf(“Soy el proceso padre\n”);

}

printf(“Mensaje en ambos\n”); exit(0);

}

Código 2

#include <stdio.h> #include <unistd.h> #include <stdlib.h> int main(void)

{

int id\_proc; id\_proc=fork(); if (id\_proc == 0)

{

printf(“Soy el proceso hijo\n”); exit(0);

}

else

{

printf(“Soy el proceso padre\n”); exit(0);

}

}

Código 1

Texto

Descripción generada automáticamente

Imagen 3 VirtualBox Ubuntu 20.0 compilación código 1

Texto

Descripción generada automáticamente

Imagen 4 VirtualBox Ubuntu 20.0 compilación código 2

1. Programe una aplicación que cree el [árbol de procesos](#_Árbol_de_procesos) mostrado al final de este documento.
2. Programe una aplicación que cree cinco procesos (por copia exacta de código). El primer proceso se encargará de realizar la suma de dos matrices de 7x7 elementos tipo entero, el segundo proceso realizará la resta sobre esas mismas matrices, el tercer proceso realizará la multiplicación de las matrices, el cuarto proceso obtendrá las transpuestas de cada matriz. Cada uno de estos procesos escribirá un archivo con los resultados de la operación que realizó. El quinto proceso leerá los archivos de resultados y los mostrará en pantalla cada uno de ellos.

Programe la misma aplicación sin la creación de procesos, es decir de forma secuencial. Obtenga los tiempos de ejecución de las aplicaciones, compare estos tiempos y dé sus observaciones.

1. Capture, compile y ejecute el siguiente programa de creación de un nuevo proceso con sustitución de un nuevo código, así como el programa que será el nuevo código a ejecutar. Observe su funcionamiento y experimente con el código.

#include <stdio.h> #include <unistd.h> #include <sys/types.h>

/\* Programa creador de un nuevo proceso \*/

int main()

{

pid\_t pid; char \*argv[3];

argv[0]="/usuario/hola"; /\*cambiar por ruta propia\*/ argv[1]="Desde el Hijo";

argv[2]=NULL; if((pid=fork())==-1)

printf("Error al crear el proceso hijo\n"); if(pid==0)

{

printf("Soy el hijo ejecutando: %s\n", argv[0]); execv(argv[0],argv);

}

else

{

wait(0);

printf("Soy el Padre\n"); exit(0);

}

}

/\* hola.c Programa que será invocado \*/

#include<stdio.h>

int main(int argc, char \*argv[])

{

char mensaje[100]; strcpy(mensaje,"Hola Mundo "); strcat(mensaje,argv[1]); printf("%s\n",mensaje);

exit(0);

}

Obtenga el ejecutable hola del programa hola.c antes de ejecutar el programa creador de un nuevo proceso.

1. Programe una aplicación que cree un proceso hijo a partir de un proceso padre, el hijo creado a su vez creará tres procesos hijos más. Cada uno de los tres procesos

generados ejecutará tres programas diferentes mediante sustitución de código, el primer programa resolverá una ecuación algebraica de segundo grado mediante la fórmula general, el segundo programa mostrará la serie de Fibonacci para un número N, y el tercer programa obtendrá la multiplicación de dos matrices de 7x7 elementos de tipo entero. Observe el funcionamiento de su programa detalladamente y responda la siguiente pregunta ¿es posible un funcionamiento 100% concurrente de su aplicación? Explique porque sí o no es 100% concurrente su aplicación.

1. Programe la aplicación desarrollada en el punto 5 de la sección de Linux utilizando esta vez la creación de procesos por sustitución de código.

Sección Windows:

1. Inicie sesión en Windows.
2. Para esta práctica se utilizará el ambiente de programación Dev C/C++.
3. Capture y compile el programa de creación de un nuevo proceso que a continuación se muestra.

#include <windows.h> #include <stdio.h>

int main(int argc, char \*argv[])

{

STARTUPINFO si; PROCESS\_INFORMATION pi;

int i;

ZeroMemory(&si, sizeof(si)); si.cb = sizeof(si); ZeroMemory(&pi, sizeof(pi)); if(argc!=2)

{

/\* Estructura de información inicial para Windows \*/

/\* Estructura de información del adm. de procesos \*/

printf("Usar: %s Nombre\_programa\_hijo\n", argv[0]); return;

}

// Creación proceso hijo

if(!CreateProcess(NULL, argv[1], NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))

{

printf( "Fallo al invocar CreateProcess (%d)\n", GetLastError() ); return;

}

// Proceso padre printf("Soy el padre\n");

WaitForSingleObject(pi.hProcess, INFINITE);

// Terminación controlada del proceso e hilo asociado de ejecución CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

}

1. Capture y compile el programa que contendrá al proceso hijo que a continuación se muestra.

#include <windows.h> #include <stdio.h>

int main(void)

{

printf("Soy el hijo\n"); exit(0);

}

1. Ejecute el primer código pasando como argumento el nombre del archivo ejecutable del segundo código capturado. Observe el funcionamiento del programa, reporte sus observaciones y experimente con el código.
2. Compare y reporte tanto las diferencias como similitudes que encuentra con respecto a la creación de procesos por sustitución de código en Linux.
3. Programe una aplicación que cree un proceso hijo a partir de un proceso padre, el hijo creado a su vez creará 5 procesos hijos más. A su vez cada uno de los cinco procesos creará 3 procesos más. Cada uno de los procesos creados imprimirá en pantalla su identificador. **CONSEJO: INVESTIGUE LA FUNCIÓN GetCurrentProcessId() DEL API WIN32.**
4. Programe las aplicaciones desarrolladas en el punto 5 de la sección de Linux (tanto la de procesos como la secuencial) utilizando esta vez la creación de procesos en Windows.

# Árbol de procesos del punto 4

main()