UNIVERSIDAD SAN CARLOS DE GUATEMALA

CENTRO UNIVERSITARIO DE OCCIDENTE

DIVISIÓN CIENCIAS DE LA INGENIERÍA

CARRERA DE INGENIERÍA CIENCIAS Y SISTEMAS

LABORATORIO DE SISTEMAS OPERATIVOS I

"**SÉPTIMO SEMESTRE"**

ING.: FRANCISCO ROJAS

ESTUDIANTE: LUIS ESTUARDO BOLAÑOS GONZÁLEZ - 201731766

PROYECTO: GESTIÓN DE PROYECTOS

FECHA: 26 de marzo de 2,021

DESCRIPCIÓN

El proyecto se basa en la creación de una aplicación en la cual se hace manejo y creación de procesos mediante árboles representativos, cada árbol se desglosa de la siguiente forma:

- El programa principal puede tener un máximo de 10 árboles.
- 1 árbol puede contener un máximo de 5 ramas.
- 1 rama puede contener un máximo de 10 hojas.

Cada árbol debe ser mostrado en pantalla según las instrucciones que el usuario escriba dentro de la interfaz.

Colores:

Cada uno de los componentes que forman a los arboles deben ir cambiando de color mientras sean mostrados en pantalla, la organizacion de colores se divide de la siguiente forma:

Tallo [1 a 10] Cambia los colores:

- Gris.
- Negro.

Rama [1 a 5] Cambia los colores:

- Todos los colores exceptuando los colores de los tallos y las hojas.

Hoja [1 a 10] Cambia los colores:

- Verde.
- Cafe.

Comandos aceptados dentro de la interfaz:

Donde a = # de planta, b = # de rama, c = # de hoja.

- (P, a, b, c) Instrucción de crear o reestructurar ramas.
- (P, a, b) Instrucción de eliminar ramas y no las hojas.
- (P, a, 0) La planta se seca solo el tallo.
- (P, a) Instrucción de crear una rama.
- (M, a) Muestra 1 planta en pantalla, queda cargado en la memoria para tener visible el árbol.
- (I, a) Imprime planta en archivo de texto con número de procesos y el árbol de procesos actual (pstree).

OBJETIVOS

Objetivos Generales:

- Implementar los conocimientos adquiridos relacionados a la creación de procesos.
- Implementar interfaz gráfica para la representación de procesos.

Objetivos Específicos:

- Crear procesos de manera síncrona.
- Manejar una buena comunicación entre procesos.
- Representar el manejo de procesos mediante los árboles representativos.
- Construir árboles de procesos y mostrarlos desde el comando pstree.

REQUERIMIENTOS TECNICOS

- Creación de procesos de manera síncrona.
- Desarrollo de interfaz amigable para el usuario.
- Imprimir el árbol de procesos del programa mediante pstree.
- Llevar un buen manejo de comunicación entre procesos.
- Eliminar los procesos creados de manera adecuada cuando sea necesario.
- Llevar un buen orden en la relación de procesos padre - hijos.

HERRAMIENTAS

- Lenguaje C/C++.
- Qt Creator (Community).
- Forks.
- Comando pstree.
- Archivos de texto.
- Semáforos.
- Pipes.

MARCO TEÓRICO

Lenguaje C:

Lenguaje de programación C. También conocido como “Lenguaje de programación de sistemas” desarrollado en el año 1972 por Dennis Ritchie para UNIX un sistema operativo multiplataforma. El lenguaje C es del tipo lenguaje estructurado como son Pascal, Fortran, Basic. Sus instrucciones son muy parecidas a otros lenguajes incluyendo sentencias como if, else, for, do y while... . Aunque C es un lenguaje de alto nivel (puesto que es estructurado y posee sentencias y funciones que simplifican su funcionamiento) tenemos la posibilidad de programar a bajo nivel (como en el Assembler tocando los registros, memoria etc.). Para simplificar el funcionamiento de el lenguaje C tiene incluidas librerías de funciones que pueden ser incluidas haciendo referencia la librería que las incluye, es decir que si queremos usar una función para borrar la pantalla tendremos que incluir en nuestro programa la librería que tiene la función para borrar la pantalla.

Al igual que B, es un lenguaje orientado a la implementación de sistemas operativos, concretamente Unix. C es apreciado por la eficiencia del código que produce y es el lenguaje de programación más popular para crear softwares de sistemas y aplicaciones.

Se trata de un lenguaje de tipos de datos estáticos, débilmente tipado, de medio nivel, que dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a bajo nivel. Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan mezclar código en ensamblador con código C o acceder directamente a memoria o dispositivos periféricos.

Ventaja del lenguaje:

La programación en C tiene una gran facilidad para escribir código compacto y sencillo a su misma vez. En el lenguaje C no tenemos procedimientos como en otros lenguajes solamente tenemos funciones los procedimientos los simula y esta terminante mente prohibido escribir funciones , procedimientos y los comandos en mayúscula todo se escribe en minúsculas (a no ser las constantes J) Los archivos en la C se escriben en texto puro de ASCII del Dos si se escribe en WORD por ejemplo el mismo incluye muchos códigos no entendidos por el compilador y generara errores ;una vez escrito se debe pasar a compilar el archivo; los archivos tienen 2 Extensiones archivo.C que es el archivo a compilar el que contiene todas los procedimientos funciones y código de nuestro programa y archivo.h que es las librerías que contienen las funciones de nuestro programa. (NOTA : El compilador genera Archivos con extensión .EXE). Cada instrucción que pasemos a poner en C va seguida de un punto y coma para decirle al compilador que hasta ahí llega la instrucción simula un Enter del teclado. Ejemplo: clrscr(); /* borra la pantalla */

Lenguaje C++:

C + + es un lenguaje de programación que proviene de la extensión del lenguaje C para que pudiese manipular objetos. A pesar de ser un lenguaje con muchos años, su gran potencia lo convierte en uno de los lenguajes de programación más demandados en 2019.

Origen:

Fue diseñado a mediados de los años 80 por el danés Bjarne Stroustrup. Si intención fue la de extender el lenguaje de programación C (con mucho éxito en ese momento) para que tuviese los mecanismos necesarios para manipular objetos. Por lo tanto C++ contiene los paradigmas de la

programación estructurada y orientada a objetos, por lo que se le conoce como un lenguaje de programación multiparadigma.

Ventajas y Desventajas del lenguaje:

Las principales ventajas de programar en C++ son:

- Alto rendimiento: Es una de sus principales características, el alto rendimiento que ofrece. Esto es debido a que puede hacer llamadas directas al sistema operativo, es un lenguaje compilado para cada plataforma, posee gran variedad de parámetros de optimización y se integra de forma directa con el lenguaje ensamblador.

- Lenguaje actualizado: A pesar de que ya tiene muchos años, el lenguaje se ha ido actualizando, permitiendo crear, relacionar y operar con datos complejos y ha implementado múltiples patrones de diseño.

- Multiplataforma

- Extendido: C y C++ están muy extendidos. Casi cualquier programa o sistema están escritos o tienen alguna parte escrita en estos lenguajes (desde un navegador web hasta el propio sistema operativo).

Las principales desventajas de C++ es que se trata de un lenguaje muy amplio (con muchos años y muchas líneas de código), tiene que tener una compilación por plataforma y su depuración se complica debido a los errores que surgen. Además el manejo de librerías es más complicado que otros lenguajes como Java o .Net y su curva de aprendizaje muy alta. Puede consultarse más información en esta Página de referencia de C++.

Características:

Algunas de las características más importantes que posee el lenguaje C++ son:

Compatibilidad con bibliotecas: A través de bibliotecas hay muchas funciones que están disponible y que ayudan a escribir código rápidamente.

Orientado a Objetos: El foco de la programación está en los objetos y la manipulación y configuración de sus distintos parámetros o propiedades.

Rapidez: La compilación y ejecución de un programa en C++ es mucho más rápida que en la mayoría de lenguajes de programación.

Compilación: En C++ es necesario compilar el código de bajo nivel antes de ejecutarse, algo que no ocurre en otros lenguajes.

Punteros: Los punteros del lenguaje C, también están disponibles en C++.

Didáctico: Aprendiendo programación en C++ luego es mucho más fácil aprender lenguajes como Java, C#, PHP, Javascript, etc.

Fork:

Cuando se llama la función fork, esta genera un duplicado del proceso actual. El duplicado comparte los valores actuales de todas las variables, ficheros y otras estructuras de datos. La llamada a fork retorna al proceso padre el identificador del proceso hijo y retorna un cero al proceso hijo.

PID:

Las siglas PID, del inglés Process IDentifier, se usan para referirse al identificador de proceso.

La función getppid:

El hijo puede obtener el identificador del proceso padre llamando a la función getppid .

La variable $$:

La variable especial $$ contiene el PID del proceso actual. Es de sólo lectura.

Semáforos:

Los semáforos son un mecanismo de sincronización de procesos inventados por Edsger Dijkstra en 1965. Los semáforos permiten al programador asistir al planificador del sistema operativo en su toma de decisiones de manera que permiten sincronizar la ejecución de dos o más procesos. A diferencia de los cerrojos, los semáforos nos ofrecen un mecanismo de espera no ocupada.

Los semáforos son un tipo de datos que están compuestos por dos atributos:

- Un contador, que siempre vale >= 0.
- Una cola de procesos inicialmente vacía.

Tipos de semáforos:

Dependiendo de la función que cumple el semáforo, vamos a diferenciar los siguientes tipos:

- Semáforo de exclusión mutua, inicialmente su contador vale 1 y permite que haya un único proceso simultáneamente dentro de la sección crítica.

- Semáforo contador, permiten llevar la cuenta del número de unidades de recurso compartido disponible, que va desde 0 hasta N.
- Semáforo de espera, generalmente se emplea para forzar que un proceso pase a estado bloqueado hasta que se cumpla la condición que le permite ejecutarse. Por lo general, el contador vale 0 inicialmente, no obstante, podría tener un valor distinto de cero.

Ventajas e inconvenientes:

La principal ventaja de los semáforos frente a los cerrojos es que permiten sincronizar dos o más procesos de manera que no se desperdician recursos de CPU realizando comprobaciones continuadas de la condición que permite progresar al proceso.

Los inconvenientes asociados al uso de semáforos son los siguientes:

- Los programadores tienden a usarlos incorrectamente, de manera que no resuelven de manera adecuada el problema de concurrencia o dan lugar a interbloqueos.
- No hay nada que obligue a los programadores a usarlos.
- Los compiladores no ofrecen ningún mecanismo de comprobación sobre el correcto uso de los semáforos.
- Son independientes del recurso compartido al que se asocian.

Debido a estos inconvenientes, se desarrollaron los monitores.

PIPES:

En informática, una tubería (pipeline o cauce) consiste en una cadena de procesos conectados de forma tal que la salida de cada elemento de la cadena es la entrada del próximo. Permiten la comunicación y sincronización entre procesos. Es común el uso de búfer de datos entre elementos consecutivos.

La comunicación por medio de tuberías se basa en la interacción productor/consumidor, los procesos productores (aquellos que envían datos) se comunican con los procesos consumidores (que reciben datos) siguiendo un orden FIFO. Una vez que el proceso consumidor recibe un dato, este se elimina de la tubería.

Las tuberías están implementadas en forma muy eficiente en los sistemas operativos multitarea, iniciando todos los procesos al mismo tiempo, y atendiendo automáticamente los requerimientos de lectura de datos para cada proceso cuando los datos son escritos por el proceso anterior. De esta manera el planificador de corto plazo va a dar el uso de la CPU a cada proceso a medida que pueda ejecutarse minimizando los tiempos muertos.

Para mejorar el rendimiento, la mayoría de los sistemas operativos implementan las tuberías usando búferes, lo que permite al proceso proveedor generar más datos que lo que el proceso consumidor puede atender inmediatamente.En informática, una tubería (pipeline o cauce) consiste en una cadena de procesos conectados de forma tal que la salida de cada elemento de la cadena es la entrada del próximo. Permiten la comunicación y sincronización entre procesos. Es común el uso de búfer de datos entre elementos consecutivos.

La comunicación por medio de tuberías se basa en la interacción productor/consumidor, los procesos productores (aquellos que envían datos) se comunican con los procesos consumidores (que reciben datos) siguiendo un orden FIFO. Una vez que el proceso consumidor recibe un dato, este se elimina de la tubería.

Las tuberías están implementadas en forma muy eficiente en los sistemas operativos multitarea, iniciando todos los procesos al mismo tiempo, y atendiendo automáticamente los requerimientos de lectura de datos para cada proceso cuando los datos son escritos por el proceso anterior. De esta manera el planificador de corto plazo va a dar el uso de la CPU a cada proceso a medida que pueda ejecutarse minimizando los tiempos muertos.

Para mejorar el rendimiento, la mayoría de los sistemas operativos implementan las tuberías usando búferes, lo que permite al proceso proveedor generar más datos que lo que el proceso consumidor puede atender inmediatamente.

Signal:

Una señal es un "aviso" que puede enviar un proceso a otro proceso. El sistema operativo unix se encarga de que el proceso que recibe la señal la trate inmediatamente. De hecho, termina la línea de código que esté ejecutando y salta a la función de tratamiento de señales adecuada. Cuando termina de ejecutar esa función de tratamiento de señales, continua con la ejecución en la línea de código donde lo había dibujado.

El sistema operativo envía señales a los procesos en determinadas circunstancias. Por ejemplo, si en el programa que se está ejecutando en una shell nosotros apretamos Ctrl-C, se está enviando una señal de terminación al proceso. Este la trata inmediatamente y sale. Si nuestro programa intenta acceder a una memoria no válida (por ejemplo, accediendo al contenido de un puntero a NULL), el sistema operativo detecta esta circunstancia y le envía una señal de terminación inmediata, con lo que el programa "se cae".

Las señales van identificadas por un número entero.

Memoria compartida:

La memoria compartida, junto con los semáforos y las colas de mensajes, son los recursos compartidos que pone unix a disposición de los programas para que puedan intercambiarse información.

En C para unix es posible hacer que dos procesos (dos programas) distintos sean capaces de compartir una zona de memoria común y, de esta manera, compartir o comunicarse datos.

La forma de conseguirlo en un programa es la siguiente:

- En primer lugar necesitamos conseguir una clave, de tipo key_t, que sea común para todos los programas que quieran compartir la memoria. Para ello existe la función key_t ftok (char *, int). A dicha función se le pasa un fichero que exista y sea accesible y un entero. Con ellos construye una clave que nos devuelve. Si todos los programas utilizan el mismo fichero y el mismo entero, obtendrán la misma clave.

  Es habitual como primer parámetro pasar algún fichero del sistema que sepamos seguro

de su existencia, como por ejemplo "/bin/ls", que es el "ls" del unix.

Para el entero, bastaría con poner un #define en algún fichero.h de forma que todos los programas que vayan a utilizar la memoria compartida incluyan dicho fichero y utilicen como entero el del #define

- Una vez obtenida la clave, se crea la zona de memoria. Para ello está la función int shmget (key_t, int, int). Con dicha función creamos la memoria y nos devuelve un identificador para dicha zona.

  Si la zona de memoria correspondiente a la Clave key_t ya estuviera creada, simplemente nos daría el identificador de la memoria (siempre y cuando los parámetros no indiquen lo contrario).

  - El primer parámetro es la clave key_t obtenida anteriormente y que debería ser la misma para todos los programas.
  - El segundo parámetro es el tamaño en bytes que deseamos para la memoria.
  - El tercer parámetro son unos flags. Aunque hay más posibilidades, lo imprescindible es:

    - 9 bits menos significativos, son permisos de lectura/escritura/ejecución para propietario/grupo/otros, al igual que los ficheros. Para obtener una memoria con todos los permisos para todo el mundo, debemos poner como parte de los flags el número 0777. Es importante el cero delante, para que

el número se interprete en octal y queden los bits en su sitio (En C, cualquier número que empiece por cero, se considera octal). El de ejecución no tiene sentido y se ignora.

- IPC_CREAT. Junto con los bits anteriores, este bit indica si se debe crear la memoria en caso de que no exista.

  Si está puesto, la memoria se creará si no lo está ya y se devolverá el identificador.

  Si no está puesto, se intentará obtener el identificador y se obtendrá un error si no está ya creada.

- En resumen, los flags deberían ser algo así como 0777 | IPC_CREAT

- El último paso poder usar la memoria consiste en obtener un puntero que apunte la zona de memoria, para poder escribir o leer en ella. Declaramos en nuestro código un puntero al tipo que sepamos que va a haber en la zona de memoria (una estructura, un array, tipos simples, etc) y utilizamos la función char * shmat (int, char *, int).

  - El primer parámetro es el identificador de la memoria obtenido en el paso anterior.
  - Los otros dos bastará rellenarlos con ceros.
  - El puntero devuelto es de tipo char *. Debemos hacerle un "cast" al tipo que queramos, por ejemplo, (mi_estructura *)shmat (...);

    Esta función lo que en realidad hace, además de darnos el puntero, es asociar la memoria compartida a la zona de datos de nuestro programa, por lo que es necesario llamarla sólo una vez en cada proceso. Si queremos más punteros a la

zona de memoria, bastará con igualarlos al que ya tenemos.

- Ya estamos en condiciones de utilizar la memoria. Cualquier cosa que escribamos en el contenido de nuestro puntero, se escribirá en la zona de memoria compartida y será accesible para los demás procesos.
- Una vez terminada de usar la memoria, debemos liberarla. Para ello utlizamos las funciones int shmdt (char *) e int shmctl (int, int, struct shmid_ds *)

  La primera función desasocia la memoria compartida de la zona de datos de nuestro programa. Basta pasarle el puntero que tenemos a la zona de memoria compartida y llamarla una vez por proceso.

  La segunda función destruye realmente la zona de memoria compartida. Hay que pasarle el identificador de memoria obtenido con shmget(), un flag que indique que queremos destruirla IPC_RMID, y un tercer parámetro al que bastará con pasarle un NULL A esta función sólo debe llamarla uno de los procesos.

Hasta ahora se ha indicado lo mínimo y más básico para poder utilizar memoria compartida. Hay más funciones que permiten hacer más cosas con la memoria compartida. Además, para un uso más "serio" y "correcto", sería necesario añadir algo como semáforos.

Por ejemplo, si los datos que deseamos compartir son muy grandes, es bastante probable que mientras un proceso intenta escribir en la memoria, otro esté leyendo. Si lo intentan de forma totalmente descontrolada y a la vez, los datos que leen pueden ser incoherentes (hay otro que está escribiéndolos en ese momento y todavía no ha terminado).

Para evitar esto existen los semáforos. Se pondría un semáforo para acceder a la memoria, de forma que cuando un proceso lo está haciendo, el semáforo se pone "rojo" y ningún otro proceso puede acceder a ella. Cuando el proceso termina, el semáforo se pone "verde" y ya podría acceder a otro proceso.

QT Creator:

Qt Creator es un IDE multiplataforma programado en C++, JavaScript y QML creado por Trolltech el cual es parte de SDK para el desarrollo de aplicaciones con Interfaces Gráficas de Usuario (GUI por sus siglas en inglés) con las bibliotecas Qt, Los sistemas operativos que soporta en forma oficial son:

- GNU/Linux 2.6.x, para versiones de 32 y 64 bits con Qt 4.x instalado. Además hay una versión para Linux con gcc 3.3.
- Mac OS X 10.4 o superior, requiriendo Qt 4.x
- Windows XP y superiores, requiriendo el compilador MinGW y Qt 4.4.3 para MinGW.

Editor avanzado de código:

Qt Creator se centra en proporcionar características que ayudan a los nuevos usuarios de Qt a aprender, también aumentar la productividad.

- Editor de código con soporte para C++, QML y ECMAscript
- Herramientas para la rápida navegación del código
- Resaltado de sintaxis y auto-completado de código

- Control estático de código y estilo a medida que se escribe
- Soporte para refactoring de código
- Ayuda sensitiva al contexto
- Plegado de código (code folding)
- Paréntesis coincidentes y modos de selección

Depurador Visual:

El depurador visual (visual debugger) para C + +. Qt Creator muestra la información en bruto procedente de GDB de una manera clara y concisa.[3]

- Interrupción de la ejecución del programa.
- Ejecución línea por línea o instrucción a instrucción.
- Puntos de interrupción (breakpoints).
- Examinar el contenido de llamadas a la pila (stack), los observadores y de la *variables locales y globales.

BIBLIOGRAFÍA

https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)

https://openwebinars.net/blog/que-es-cpp/

https://campusvirtual.ull.es/ocw/pluginfile.php/2173/mod_resource/content/0/perlexamples/node66.html

https://1984.lsi.us.es/wiki-ssoo/index.php/Sem%C3%A1foros#:~:text=Los%20sem%C3%A1foros%20son%20un%20mecanismo,de%20dos%20o%20m%C3%A1s%20procesos.

https://es.wikipedia.org/wiki/Tuber%C3%ADa_(inform%C3%A1tica)

http://www.chuidiang.org/clinux/senhales/senhales.php

http://www.chuidiang.org/clinux/ipcs/mem_comp.php