UNIVERSIDAD SAN CARLOS DE GUATEMALA

CENTRO UNIVERSITARIO DE OCCIDENTE

DIVISIÓN CIENCIAS DE LA INGENIERÍA

CARRERA DE INGENIERÍA CIENCIAS Y SISTEMAS

LABORATORIO SISTEMAS OPERATIVOS 1

ING: FRANCISCO ROJAS

ESTUDIANTE:

201730919 - Bryan René Gómez Gómez

TEMA: “Manual Técnico - Primer Proyecto”

FECHA: 28 de marzo de 2,021

OBJETIVOS

General

- Realizar una aplicación, que pueda crear procesos que trabajen en paralelo.

Específicos

1. Aplicar conceptos recibidos en clase magistral y laboratorio acerca de multiprocesos.
2. Utilizar el concepto de semáforos para poder sincronizar, los procesos en paralelo.
3. Generar procesos en paralelo, para poder solucionar el problema propuesto.

ÍNDICE

DESCRIPCIÓN DEL PROBLEMA

Se necesita de una aplicación que pueda crear y modificar árboles, mediante instrucciones, donde las hojas, ramas y los tallos puedan trabajar como un proceso independiente donde cambiaran de color constantemente.

El tiempo de espera para realizar un cambio de color es de 1s, cada hoja, rama y tallo seran procesos en paralelo, donde se pueden crear de uno a diez tallos, cada uno de estos permitira la creacion de cinco ramas como maximo y un minimo de una, y cada rama debera de tener como minimo una hoja y maximo diez, ya que estos son procesos trabajando paralelamente deberan de funcionar hasta que sea eliminado, y para observar el funcionamiento deberan de ir cambiando de color, a continuacion se describen los colores:

Tallo:

- Gris
- Negro

Rama:

- Todos los colores

Hojas:

- Verde
- Cafe

Los comandos a utilizar son los siguientes:

Creación y modificación de una planta

- (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, únicamente quedará el tallo.
- (P, a) Intrusión de crear una planta.

Donde:

- a = Número de planta o tallo
- b = Número rama

- c = Número de hoja (Se le asigna a todas las ramas)

Comando para mostrar una planta:

(M, a) Muestra la planta o tallo “a” en pantalla, queda y las modificaciones realizadas en otras plantas se verán reflejadas hasta que se muestran.

Comando para mostrar la estructura de los procesos mediante un archivo de texto:

(I, a) Imprime la planta o tallo “a” en un archivo de texto con el número de procesos y el árbol de procesos actual (pstree).

REQUERIMIENTOS TÉCNICOS

Requerimientos Mínimos de Hardware

- **Procesador:** Core i5
- **Memoria RAM:** 4 Gigabytes (GB)
- **Almacenamiento:** 50 Gigabytes (GB)

Requerimientos Mínimos de Software

- **Sistema Operativo:** GNU / Linux (Distribución derivada de Debian)
    - Ubuntu 20.04
- **QT Creator**
- **Compilador de C/C++**
- **CMake**
- **QMake**

HERRAMIENTAS PARA EL DESARROLLO

Debian GNU/Linux

Es un sistema operativo libre, desarrollado por miles de voluntarios de todo el mundo, que colaboran a través de Internet.

La dedicación de Debian al software libre, su base de voluntarios, su naturaleza no comercial y su modelo de desarrollo abierto la distingue de otras distribuciones del sistema operativo GNU. Todos estos aspectos y más se recogen en el llamado Contrato Social de Debian.

Debian GNU/Linux puede utilizar distintos mecanismos de instalación, como son: DVD, CD, USB, e incluso directamente desde la red (este último depende de la velocidad de la red del usuario).

Ubuntu

Ubuntu es un sistema operativo de software libre y código abierto. Es una distribución de Linux basada en Debian. Puede correr en computadores de escritorio y servidores. Está orientado al usuario promedio, con un fuerte enfoque en la facilidad de uso y en mejorar la experiencia del usuario. Está compuesto de múltiples software normalmente distribuido bajo una licencia libre o de código abierto. Estadísticas web sugieren que la cuota de mercado de Ubuntu dentro de las distribuciones Linux es, aproximadamente, del 52 %, y con una tendencia a aumentar como servidor web.

Su patrocinador, Canonical, es una compañía británica propiedad del empresario sudafricano Mark Shuttleworth. Ofrece el sistema de manera gratuita, y se financia por medio de servicios vinculados al sistema operativo y vendiendo soporte técnico. Además, al mantenerlo libre y gratuito, la empresa es capaz de aprovechar los desarrolladores de la comunidad para mejorar los componentes de su sistema operativo. Extraoficialmente, la comunidad de desarrolladores proporciona soporte para otras derivaciones de Ubuntu, con otros entornos gráficos, como Kubuntu, Xubuntu, Ubuntu MATE, Edubuntu, Ubuntu Studio, Mythbuntu, Ubuntu GNOME y Lubuntu.

Canonical, además de mantener Ubuntu, provee una versión orientada a servidores, *Ubuntu Server*, una versión para empresas, *Ubuntu Business Desktop Remix*, una para

televisores, *Ubuntu TV*, otra versión para tabletas *Ubuntu Tablet*, también *Ubuntu Phone* y una para usar el escritorio desde teléfonos inteligentes, *Ubuntu for Android*.

Cada seis meses se publica una nueva versión de Ubuntu. Esta recibe soporte por parte de Canonical durante nueve meses por medio de actualizaciones de seguridad, parches para *bugs* críticos y actualizaciones menores de programas. Las versiones LTS (*Long Term Support*), que se liberan cada dos años, reciben soporte durante cinco años en los sistemas de escritorio y de servidor.

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.

C/C++

C++ es un lenguaje de programación diseñado en 1979 por Bjarne Stroustrup. La intención de su creación fue extender al lenguaje de programación C mecanismos que permiten la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, C++ es un lenguaje híbrido.

Posteriormente se añadieron facilidades de programación genérica, que se sumaron a los paradigmas de programación estructurada y programación orientada a objetos. Por esto se suele decir que el C++ es un lenguaje de programación multiparadigma.

Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos. Existen también algunos intérpretes, tales como ROOT.

El nombre "C++" fue propuesto por Rick Mascitti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Antes se había usado el nombre "C con clases". En C++, la expresión "C++" significa "incremento de C" y se refiere a que C++ es una extensión de C.

CMake

CMake es una herramienta multiplataforma de generación o automatización de código. El nombre es una abreviatura para "cross platform make" (make multiplataforma); más allá del uso de "make" en el nombre, CMake es una suite separada y de más alto nivel que el sistema make común de Unix, siendo similar a las autotools.

CMake es un software libre y de código abierto multiplataforma para gestionar la automatización de la construcción del software utilizando un método independiente del compilador. Soporta jerarquías de directorios y aplicaciones que dependen de múltiples bibliotecas. Se utiliza en conjunción con entornos de construcción nativos como Make, Qt Creator, Ninja, Xcode de Apple, y Microsoft Visual Studio. Tiene dependencias mínimas, requiriendo sólo un compilador C++ en su propio sistema de construcción.

qmake

qmake es un programa que automatiza la creación de ficheros Makefiles. Los ficheros Makefiles son usados por el programa Make para compilar el código de manera automática. Make hace mucho más cómoda la compilación de proyectos relativamente medianos, pero a medida que la complejidad del proyecto aumenta los ficheros Makefile pueden llegar a crecer y complicarse demasiado. Y aquí es dónde entra en juego qmake, automatizando el proceso a un nivel superior.

qmake crea Makefiles que se adaptan a la plataforma deseada por lo que soporta diferentes sistemas operativos, entre los que están: Linux, Apple Mac OS X, Symbian, Android y Microsoft Windows.

qmake fue creado por Trolltech (ahora propiedad de Digia), y forma parte del framework de Qt. Aunque posee características adicionales que facilitan el desarrollo con Qt, automatiza la creación de los códigos moc (meta object compiler) y rcc (resource compiler), puede usarse para desarrollar cualquier proyecto software.

INSTALACIONES Y CONFIGURACIONES DEL SOFTWARE DE DESARROLLO

Cómo instalar GCC el compilador de C en GNU/Linux Ubuntu

GNU Compiler Collection es un sistema de compilación desarrollado para soportar varios lenguajes de programación. Es un compilador estándar utilizado en la mayoría de los proyectos relacionados con GNU y Linux, por ejemplo, el kernel de Linux. El objetivo de este tutorial es instalar GCC el compilador de C en Ubuntu 20.04 LTS Focal Fossa Linux. Esto se logrará utilizando el apt install comando.

Requisitos de software y convenciones utilizados

Categoría	Requisitos, convenciones o versión de software utilizada
Sistema	Instalado o actualizado Ubuntu 20.04 Focal Fossa
Software	CCG
Otro	Acceso privilegiado a su sistema Linux como root o mediante el sudo comando.
Convenciones	# : requiere que los comandos de Linux se ejecuten con privilegios de root, ya sea directamente como usuario root o mediante el uso del sudo comando **$** : requiere que los comandos de Linux se ejecuten como un usuario normal sin privilegios

Aunque puede instalar el compilador de C por separado mediante la instalación del gcc paquete, la forma recomendada de instalar el compilador de C en Ubuntu es mediante la instalación de todo el paquete de desarrollo build-essential.

1. Instale el compilador de C instalando el paquete de desarrollo build-essential:

   ```
   $ sudo apt install build-essential
   ```

2. Verifique la versión del compilador C:

```
$ gcc --version
```

3. Crea una fuente de código C básica. Por ejemplo, creamos el programa hello world C. Guarde el siguiente código como hello.c archivo de texto:

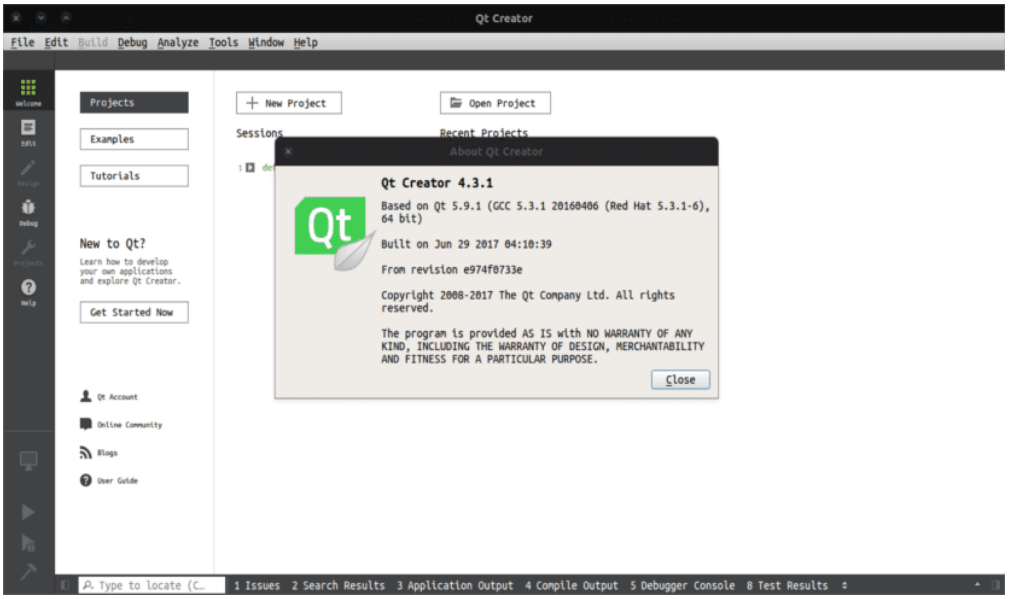
```
#include
int main()
{
  printf("Hello, World!");
  return 0;
}
```

4. Compila y ejecuta el hello.c código C:

```
$ gcc -o hello hello.c

$ ./hello
Hello, World!
```

Instalar Qt Creator



Qt 5.9.1, (con Qt Creator 4.3.1 incluido) instalación en Ubuntu

Para empezar vamos a **instalar Build Essential**, si es que no lo tienes ya instalado. Este es un paquete que va a permitirnos a los usuarios instalar y utilizar herramientas de c ++ en Ubuntu. Para proceder a la instalación, abrimos una terminal (Ctrl+Alt+T) y primero actualizaremos el software disponible para después instalar el paquete escribiendo:

```
1  sudo apt update; sudo apt install build-essential
```

Si no tienes instalado el paquete Qt Creator que contiene la IU y las herramientas de línea de comandos para la creación y ejecución del proyecto Qt, escribe en la misma terminal:


```
1  sudo apt install qtcreator
```

Si quieres que Qt5 se use como la versión predeterminada de Qt Creator, ejecuta el siguiente comando:


```
sudo apt install qt5-default
```

Para implementar proyectos más complejos, habrá que **instalar la documentación y los ejemplos de Qt**. Esto se puede hacer escribiendo en la terminal:

```
sudo apt-get install qt5-doc qtbase5-examples qtbase5-doc-html
```

El paquete *qt5-doc* contiene la documentación API de Qt 5. *Qtbase5-examples* contiene los ejemplos de Qt Base 5 y *qtbase5-doc-html* contiene la documentación HTML para las bibliotecas base de Qt 5.

Cómo instalar qt5-qmake en Ubuntu

Instalar qt5-qmake

Para instalar qt5-qmake en Ubuntu ejecutar los siguientes comandos:

```
sudo apt-get update
sudo apt-get install qt5-qmake
```

Información del paquete qt5-qmake

Nombre	qt5-qmake
Sección	devel
Descripción	Qt 5 qmake Makefile generator tool
Sitio	qt-project.org/
Responsable	ubuntu-devel-discuss@lists.ubuntu.com

Cómo instalar cmake en Ubuntu

Instalar cmake

Para instalar cmake en Ubuntu ejecutar los siguientes comandos:

```
sudo apt-get update
sudo apt-get install cmake
```

Información del paquete cmake

Nombre	cmake
Sección	devel
Descripción	cross-platform, open-source make system
Sitio	cmake.org/
Responsable	ubuntu-devel-discuss@lists.ubuntu.com

Código de los Semáforos

Antes de nada, queda claro que únicamente pretendo dar una idea sencilla de cómo funcionan los semáforos. No detallo aquí todas las opciones de las funciones a utilizar ni explico todas las posibilidades. Tampoco puedo garantizar que la sintaxis de las funciones y de los parámetros sea correcta al 100%, aunque sí suministró dos fuentes de ejemplo que compilan y funcionan. Hay, por tanto, que leer este texto con intención de hacer únicamente un uso sencillo de semáforos.

Para utilizar los semáforos en un programa, deben seguirse los siguientes pasos:

- Obtener una clave de semáforo. En general, una clave de recurso compartido, ya que la función que nos sirve para obtener dicha clave también vale para memoria compartida y colas de mensajes. Para ello se utiliza la función **key_t ftok(char *, int)** a la que se suministra como primer parámetro el nombre y path de un fichero cualquiera que exista y como segundo un entero cualquiera. Todos los procesos que quieran compartir el semáforo, deben suministrar el mismo fichero y el mismo entero. En los programas de ejemplo sem1.c y sem2.c se utilizan "/bin/ls" y el entero 33.

- Obtener un array de semáforos. La función **int semget (key_t, int, int)** nos permite obtener un array de semáforos. Se le pasa como primer parámetro la clave obtenida en el paso anterior, el segundo parámetro es el número de semáforos que queremos y el tercer parámetro son flags.

  Estos flags permiten poner los permisos de acceso a los semáforos, similares a los ficheros, de lectura y escritura para el usuario, grupo y otros. También lleva unos modificadores para la obtención del semáforo.

  En nuestro ejemplo pondremos **0600 | IPC_CREATE** que indica permiso de lectura y escritura para el propietario y que los semáforos se creen si no lo están al llamar a **semget()**. Es importante el 0 delante del 600, así el compilador de C interpretará el número en octal y pondrá correctamente los permisos.

  La función **semget()** nos devuelve un identificador del array de semáforos.
- Uno de los procesos debe inicializar el semáforo. La función a utilizar es **int semctl (int, int, int, int)**. El primer parámetro es el identificador del array de semáforos obtenido anteriormente, el segundo parámetro es el índice del semáforo que queremos inicializar dentro del array de semáforos obtenido. Si sólo hemos pedido uno, el segundo parámetro será 0.

  El tercer parámetro indica qué queremos hacer con el semáforo. En función de su valor, los siguientes parámetros serán una cosa u otra. En nuestro ejemplo, que queremos inicializar el semáforo, el valor del tercer parámetro es **SETVAL**.

  El cuarto parámetro, aunque está definido como un entero, en realidad admite una unión bastante liada. Si no queremos complicarnos la vida, bastará pasar como cuarto parámetro un 1 si queremos el semáforo en "verde" o un 0 si lo queremos en "rojo". En el código de ejemplo sem1.c y sem2.c se utiliza la unión, pero el resultado es el mismo.

  Debo advertir además que cuando utilizé semáforos con Solaris (versión de UNIX de Sun Microsystems) la unión estaba definida en los #**include** adecuados, pero en linux (Mandrake), no lo está y hay que definirla en el código (según indica el man).
- Ya está todo preparado, ahora sólo queda usar los semáforos. El proceso que quiera acceder a un recurso común debe primero decrementar el semáforo. Para ello utilizará la función **int semop (int, strcut sembuf *, size_t)**. El

primer parámetro es el identificador del array de semáforos obtenido con **semget()**. El segundo parámetro es un array de operaciones sobre el semáforo. Para incrementarlo, bastará con un array de una única posición. El tercer parámetro es el número de elementos en el array, es decir, 1.

La estructura del segundo parámetro contiene tres campos:

- **short sem_num** que es el índice del array del semáforo sobre el que queremos actuar. En nuestro caso, con un solo semáforo, el índice será 0.
- **short sem_op** que es el valor en el que queremos decrementar el semáforo. En nuestro caso, -1.
- **short sem_flg** son flags que afectan a la operación. En nuestro caso, para no complicarnos la vida, pondremos 0.

- Al realizar esta operación, si el semáforo se vuelve negativo, nuestro proceso se quedará "bloqueado" hasta que alguien incremente el semáforo y lo haga, como mínimo, 0.
- Cuando el proceso termine de usar el recurso común, debe incrementar el semáforo. La función a utilizar es la misma, pero poniendo 1 en el campo **sem_op** de la estructura **strcut sembuf**.

Como ejemplo de todo esto, están los programas sem1.c y sem2.c que ya se han mencionado antes. Se compilan com **make sem1** y **make sem2**. Para ejecutarlos, debe ejecutarse primero en una ventana de shell y luego en otra ventana de shell distinta.

sem1 tiene un bucle infinito para entrar en el semáforo. Escribe en pantalla cuando entra y cuando sale. Como el semáforo está en "rojo" por defecto, **sem1** entra en él y no sale hasta que **sem2** lo indica.

sem2 tiene un bucle de 1 a 10 en el que pone en verde el semáforo y espera un segundo. El resultado es que **sem1** entra en el semáforo, queda "bloqueado" un segundo y sale del semáforo para volver a entrar en él y repetir el proceso 10 veces.

Puesto que las llamadas a la función **semop()** son bastante "engorrosas" por aquello de rellenar la estructura, suele ser útil hacer un para de funciones **espera_semaforo()** y **eleva_semaforo()** que realicen este código. Se les podría pasar el identificador del semáforo y el índice dentro del array de semáforos.

MARCO TEÓRICO

Multiproceso, Multitarea y Multiusuario

Multiproceso, multitarea y multiusuario son características de los sistemas operativos. A continuación se pasará a estudiar estos conceptos en profundidad:

Multiproceso. Un sistema operativo multiproceso es aquel que puede ejecutar varios procesos de forma concurrente (a la vez). Para que se puedan ejecutar varios procesos a la vez es necesario tener un sistema multiprocesador (con varios procesadores).

El objetivo de utilizar un sistema con varios procesadores no es ni más ni menos que aumentar la potencia de cálculo y por lo tanto rendimiento del mismo. El sistema va repartiendo la carga de trabajo entre los procesadores existentes y también tendrá que gestionar la memoria para poder repartirla entre dichos procesadores. Generalmente

Los sistemas multiprocesador se utilizan en workstations, servidores, etc.

El realizar un sistema operativa que pueda trabajar con varios procesadores de forma concurrente es una tarea complicada y generalmente se diseña para que trabajen de varias formas:

- **Forma asimétrica.** Se designa un procesador como el procesador master (maestro) y los demás serán slave (esclavos) de este. En el procesador maestra se ejecuta el sistema operativo y se encargará de repartir el trabajo entre los demás procesadores esclavos. Este sistema tiene la ventaja de que es más simple de implementar ya
  que se centraliza la gestión en un procesador. Por el contrario, el que existan procesadores que realicen distinto trabajo hace que el sistema no sea tan eficiente como un sistema operativo que trabaje de forma simétrica.
- **Forma simétrica.** En este tipo de sistema operativo todos los procesadores realizan las mismas funciones. Es más difícil de implementar pero son más eficientes que los sistemas operativos multiprocesos asimétricas. El poder balancear la carga de trabajo entre todos los procesadores existentes hace que el tiempo de inactividad de los mismos sea mucho menor y por lo tanto la

productividad mucho más alta. Otra ventaja es que si un procesador falla. gracias al balanceo de carga, los demás procesadores se hacen cargo de las tareas del procesador que ha fallado.

- **Multitarea** En un sistema operativo multitarea, varios procesos se pueden estar ejecutando aparentemente al mismo tiempo sin que el usuario la perciba. La gran mayoría de sistemas operativos actuales son multitarea. Un sistema multitarea permite a la vez estar escuchando música, navegando por internet y realizando una videoconferencia. El sistema operativo fracciona el tiempo de CPU y lo va repartiendo entre los procesos que lo necesitan de la mejor forma posible. Además de la multitarea aparece el concepto de multihilo.
- **Multihilo.** En ocasiones, un proceso puede tener varios hilos de ejecución de tal manera que un proceso pueda ejecutar varias tareas a la vez El multihilo se utiliza a veces por eficiencia, debido a que el creer muchos procesos implica la asignación de muchos recursos mientras que muchos hilos pueden compartir los recursos y memoria de un proceso. El multihilo evita además la pérdida de tiempo en cambios de contexto al ejecutarse todos los hilos en el mismo contexto.
- **Multiusuario**. Un sistema operativo multiusuario es un sistema que puede dar servicio a varios usuarios de forma simultánea. Hay que tener en cuenta que un sistema con varias cuentas de usuario no es necesariamente un sistema multiusuario. Por ejemplo, las versiones domésticas de los sistemas Windows permiten tener varias cuentas de usuario en el mismo sistema operativo pero no permiten que haya varios usuarios trabajando en el sistema al mismo tiempo Las versiones Server de Windows si permiten esta característica a través de terminal server. Por el contrario, las versiones Linux si son multiusuarios desde hace mucho tiempo.

Semáforos en C para Linux

A veces es necesario que dos o más procesos o hilos (threads) accedan a un recurso común (escribir en un mismo fichero, leer la misma zona de memoria, escribir en la misma pantalla, etc). El problema es que si lo hacen simultáneamente y de forma incontrolada, pueden "machacar" el uno la operación del otro (y dejar el fichero o la memoria con un contenido inservible o la pantalla ilegible).

Para evitar este problema, están los semáforos. Un semáforo da acceso al recurso a uno de los procesos y se lo niega a los demás mientras el primero no termina. Los semáforos, junto con la memoria compartida y las colas de mensajes, son los recursos compartidos que suministra UNIX para comunicación entre procesos.

El funcionamiento del semáforo es como el de una variable contador. Imaginemos que el semáforo controla un fichero y que inicialmente tiene el valor 1 (está "verde"). Cuando un proceso quiere acceder al fichero, primero debe decrementar el semáforo. El contador queda a 0 y como no es negativo, deja que el proceso siga su ejecución y, por tanto, acceda al fichero.

Ahora un segundo proceso lo intenta y para ello también decrementa el contador. Esta vez el contador se pone a -1 y como es negativo, el semáforo se encarga de que el proceso quede "bloqueado" y "dormido" en una cola de espera. Este segundo proceso no continuará por tanto su ejecución y no accede al fichero.

Supongamos ahora que el primer proceso termina de escribir el fichero. Al acabar con el fichero debe incrementar el contador del semáforo. Al hacerlo, este contador se pone a 0. Como no es negativo, el semáforo se encarga de mirar la cola de procesos pendientes y "desbloquear" al primer proceso de dicha cola. Con ello, el segundo proceso que quería acceder al fichero continua su ejecución y accede al fichero.

Cuando este proceso también termine con el fichero, incrementa el contador y el semáforo vuelve a ponerse a 1, a estar "verde".

Es posible hacer que el valor inicial del semáforo sea, por ejemplo, 3, con lo que pasarán los tres primeros procesos que lo intenten. Pueden a su vez quedar muchos procesos encolados simultáneamente, con lo que el contador quedará con un valor negativo grande. Cada vez que un proceso incrementa el contador (libere el recurso común), el primer proceso encolado despertará. Los demás seguirán dormidos.

Como vemos, el proceso de los semáforos requiere colaboración de los procesos. Un proceso debe decrementar el contador antes de acceder al fichero e incrementarlo cuando termine. Si los procesos no siguen este "protocolo" (y pueden no hacerlo), el semáforo no sirve de nada.

Procesos / fork() en C

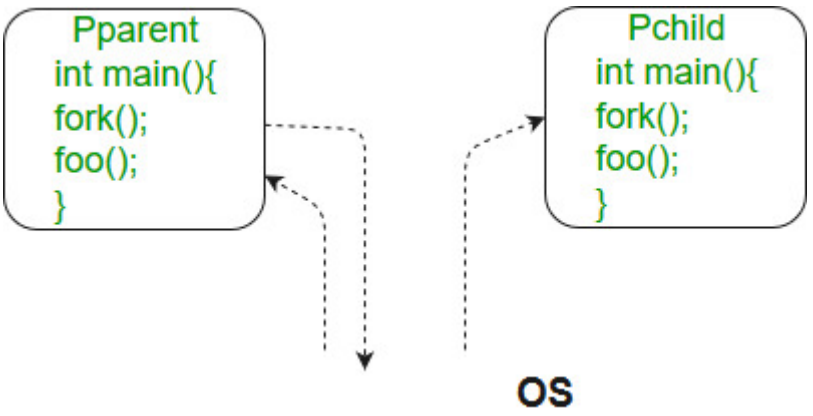
La llamada al sistema fork se utiliza para crear un nuevo proceso, que se denomina proceso hijo, que se ejecuta al mismo tiempo que el proceso que realiza la llamada fork () (proceso padre). Después de que se crea un nuevo proceso hijo, ambos procesos ejecutarán la siguiente instrucción después de la llamada al sistema fork (). Un proceso hijo usa la misma PC (contador de programa), los mismos registros de CPU, los mismos archivos abiertos que usa en el proceso padre.

No toma parámetros y devuelve un valor entero. A continuación se muestran diferentes valores devueltos por fork ().

Valor negativo: la creación de un proceso hijo no tuvo éxito.

Cero: devuelto al proceso hijo recién creado.

Valor positivo: devuelto al padre o la persona que llama. El valor contiene el ID de proceso del proceso hijo recién creado.



pipe () Llamada al sistema

Requisito previo: Llamadas al sistema de E / S

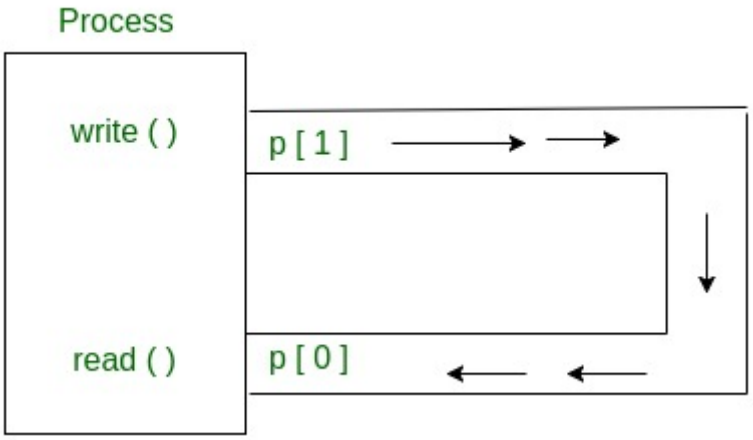
Conceptualmente, una tubería es una conexión entre dos procesos, de modo que la salida estándar de un proceso se convierte en la entrada estándar del otro proceso. En el sistema operativo UNIX, las tuberías son útiles para la comunicación entre procesos relacionados (comunicación entre procesos).

La tubería es solo comunicación unidireccional, es decir, podemos usar una tubería de modo que un proceso escriba en la tubería y el otro proceso lea desde la tubería. Abre una tubería, que es un área de la memoria principal que se trata como un "archivo virtual".

El proceso de creación puede utilizar la tubería, así como todos sus procesos secundarios, para leer y escribir. Un proceso puede escribirse en este "archivo virtual" o canalización y otro proceso relacionado puede leer desde él.

Si un proceso intenta leer antes de que se escriba algo en la tubería, el proceso se suspende hasta que se escriba algo.

La llamada al sistema de tuberías encuentra las dos primeras posiciones disponibles en la tabla de archivos abiertos del proceso y las asigna para los extremos de lectura y escritura de la tubería.



BIBLIOGRAFÍA

1. (2021). Retrieved 27 March 2021, from https://ubunlog.com/instala-qt-creator-compila-programa/?utm_source=feedburner&utm_medium=%24%7Bfeed%2C+email%7D&utm_campaign=Feed%3A+%24%7BUbunlog%7D+%28%24%7BUbunlog%7D%29
2. C++ - Wikipedia, la enciclopedia libre. (2021). Retrieved 27 March 2021, from https://es.wikipedia.org/wiki/C%2B%2B
3. CMake - Wikipedia, la enciclopedia libre. (2021). Retrieved 27 March 2021, from https://es.wikipedia.org/wiki/CMake
4. Cómo instalar. (2021). Retrieved 27 March 2021, from https://howtoinstall.co/es/qt5-qmake
5. Cómo instalar. (2021). Retrieved 27 March 2021, from https://howtoinstall.co/es/cmake
6. Cómo instalar GCC el compilador de C en Ubuntu 20.04 LTS Focal Fossa Linux. (2021). Retrieved 27 March 2021, from https://goto-linux.com/es/2020/6/15/como-instalar-gcc-el-compilador-de-c-en-ubuntu-20.04-lts-focal-fossa-linux/
7. Como instalar Ubuntu 20.04 (u otra distro Linux) en tu equipo | Oficina de Software Libre. (2021). Retrieved 27 March 2021, from https://osl.ugr.es/2020/05/20/como-instalar-ubuntu-1/
8. fork() in C - GeeksforGeeks. (2021). Retrieved 27 March 2021, from https://www.geeksforgeeks.org/fork-system-call/
9. linux c pthread multiproceso - programador clic. (2021). Retrieved 27 March 2021, from https://programmerclick.com/article/35581399098/

10. MULTIPROCESO, MULTITAREA Y MULTIUSUARIO. (2021). Retrieved 27 March 2021, from https://sisoperativoutp2587.wordpress.com/2016/08/12/multiproceso-multitarea-y-multiusuario/
11. pipe() System call - GeeksforGeeks. (2021). Retrieved 27 March 2021, from https://www.geeksforgeeks.org/pipe-system-call/
12. Procesos e hilos en C de Unix/Linux. (2021). Retrieved 27 March 2021, from http://www.chuidiang.org/clinux/procesos/procesoshilos.php
13. qmake - Wikipedia, la enciclopedia libre. (2021). Retrieved 27 March 2021, from https://es.wikipedia.org/wiki/Qmake
14. Qt Creator - Wikipedia, la enciclopedia libre. (2021). Retrieved 27 March 2021, from https://es.wikipedia.org/wiki/Qt_Creator
15. Ubuntu - Wikipedia, la enciclopedia libre. (2021). Retrieved 27 March 2021, from https://es.wikipedia.org/wiki/Ubuntu
16. (2021). Retrieved 27 March 2021, from https://blog.desdelinux.net/como-matar-procesos-facilmente/
17. (2021). Retrieved 27 March 2021, from https://www.youtube.com/watch?v=vvI2-CZ1Bik
18. Comunicación entre Procesos UNIX. (2021). Retrieved 27 March 2021, from http://www.lsi.us.es/cursos/seminario-1.html
19. fork(), C., & Ortiz, S. (2021). Cómo funciona la función fork(). Retrieved 27 March 2021,https://es.stackoverflow.com/questions/179414/como-funciona-la-funci%C3%B3n-fork
20. Semáforos en C para Unix/Linux. (2021). Retrieved 27 March 2021, from http://www.chuidiang.org/clinux/ipcs/semaforo.php