UNIVERSIDAD SAN CARLOS DE GUATEMALA

CENTRO UNIVERSITARIO DE OCCIDENTE

DIVISIÓN CIENCIAS DE LA INGENIERÍA

CARRERA DE INGENIERÍA CIENCIAS Y SISTEMAS

SISTEMAS OPERATIVOS 1

"**SÉPTIMO SEMESTRE"**

ING.: FRANCISCO ROJAS

ESTUDIANTE: JOSÉ CARLOS SOBERANIS RAMÍREZ - 201730246

TRABAJO: DOCUMENTACIÓN PRIMER PROYECTO

FECHA: 25 de marzo de 2,021

ÍNDICE

INTRODUCCIÓN

El concepto principal del sistema operativo es el proceso, este es un programa en ejecución, incluyendo el valor del program counter, los registros y las variables, un recurso de procesador es alternado entre los diferentes procesos que existan en el sistema, dando la idea de que se ejecutan en paralelo.

Se realiza un proyecto capaz de administrar procesos, permitiendo la comunicación entre los mismos, sincronizando estos mediante el uso de semáforos, esto con la finalidad de lograr que cada uno acceda a una sección crítica simulada mediante un archivo de texto, para generar una interfaz gráfica equivalente.

Se busca aplicar conceptos sobre los procesos, la bifurcaciones de los mismos y también sobre la implementación de los semáforos en un proyecto real.

Dividimos la documentación del proyecto en la descripción del mismo, el marco teórico que lo sostiene, la metodología utilizada en conjunto con la codificación realizada.

OBJETIVOS

Objetivo General

- Desarrollar una aplicación para la administración de diferentes procesos creados a partir de un proceso padre los cuales deberán sincronizarse para acceder a una sección crítica representada por un archivo de texto.

Objetivos específicos

- Identificar el procedimiento(s) necesarios para lograr el multiprocesamiento y la sincronización entre estos.
- Diseñar un sistema de semáforos tal que no se bloqueen procesos y garantizar que no haya inanición entre los mismos, logrando una reacción en cadena.
- Sincronizar los accesos a la sección crítica, logrando modificar información sobre esta sin crear inconsistencias en otros procesos o con la sección misma.

DESCRIPCIÓN DEL PROYECTO

Descripción

Se desea realizar un proyecto para crear, modificar y eliminar distintos procesos, siendo estos capaces cada uno de acceder a una sección crítica o bien, sincronizar dichos procesos para poder representar un ‘árbol’ de procesos, en el que cada proceso cambiará el color de su homónimo en la representación gráfica.

Habrá un proceso padre, el cual podrá ser capaz de generar distintos procesos, denominados tallos, cada tallo podrá generar más procesos denominados ramas y cada rama podrá generar más procesos denominados hojas.

Cada uno de estos procesos deberá cambiar el color de su representación gráfica, y deben de poder ser observados desde el árbol de procesos generado por la instrucción pstree.

Requisitos/Funcionalidades

Instrucciones

El programa será capaz de reconocer distintas instrucciones, entre las que podemos mencionar:

- (P,n1,n2,n3): Que representa crear un tallo n1 (si no existe), con n2 ramas y n3 hojas para cada rama, deberán realizarse los cambios necesarios, es decir, si hay más ramas de las especificadas en n2, deberán eliminarse, si hay más hojas de las especificadas en n3, deberán eliminarse, si hacen falta deberán crearse.
- (P,n1): Crea un tallo n1 (si no existe)
- (P,n1,n2): Crea un tallo n1 (si no existe) con n2 ramas, sin modificar la cantidad de hojas que tenga cada rama.

- (M,n1): Muestra el tallo n1 en pantalla en la interfaz gráfica.
- (I, n1): Imprime en un archivo de texto la representación con formato pstree.

Cabe resaltar que cada tallo, rama u hoja es un proceso, por lo que deberá ser creado con la función fork(), y se debe realizar una administración de estos para evitar conflictos o bloqueos en el programa.

Interfaz gráfica

Debe mostrar la imagen de un árbol, esta se realizará con Labels, en el que cada label representara un "proceso" como lo pueden ser tallo, rama u hoja, los colores a intercambiar para cada uno de estos pueden ser:

- Tallos
    - Negro y café
- Ramas
    - Café y verde
- Hojas
    - Todos los colores

Entrada de texto

La entrada de texto cumple la misma función que la interfaz gráfica salvo por el hecho de que se realizan cambios de línea a línea en el archivo de texto y generalmente tendrá el objetivo de reflejarse en un archivo de texto de salida.

MARCO TEÓRICO

Algoritmo de Dekker

El algoritmo de Dekker es un algoritmo de programación concurrente para exclusión mutua, que permite a dos procesos o hilos de ejecución compartir un recurso sin conflictos. Fue uno de los primeros algoritmos de exclusión mutua inventados, implementado por Edsger Dijkstra.

Si ambos procesos intentan acceder a la sección crítica simultáneamente, el algoritmo elige un proceso según una variable de turno. Si el otro proceso está ejecutando en su sección crítica, deberá esperar su finalización.

Existen cinco versiones del algoritmo Dekker, teniendo ciertos fallos los primeros cuatro. La versión 5 es la que trabaja más eficientemente, siendo una combinación de la 1 y la 4.

- Versión 1: Alternancia estricta. Garantiza la exclusión mutua, pero su desventaja es que acopla los procesos fuertemente, esto significa que los procesos lentos atrasan a los procesos rápidos.
- Versión 2: Problema interbloqueo. No existe la alternancia, aunque ambos procesos caen a un mismo estado y nunca salen de ahí.
- Versión 3: Colisión región crítica no garantiza la exclusión mutua. Este algoritmo no evita que dos procesos puedan acceder al mismo tiempo a la región crítica.
- Versión 4: Postergación indefinida. Aunque los procesos no están en interbloqueo, un proceso o varios se quedan esperando a que suceda un evento que tal vez nunca suceda.

Multiprocesamiento

En el idioma español hacen sinónimo este término con el de multitareas (del inglés multitasking) el cual consiste en la ejecución de uno o más procesos concurrentes en un sistema. Así como la multitarea permite a múltiples procesos compartir una única CPU, múltiples CPU pueden ser utilizados para ejecutar múltiples procesos o múltiples hilos (threads) dentro de un único proceso.

Sección Crítica

Se denomina región crítica,(sección crítica y región crítica son denominaciones equivalentes) en programación concurrente de ciencias de la computación, a la porción de código de un programa de ordenador en la que se accede a un recurso compartido (estructura de datos o dispositivo) que no debe ser accedido por más de un proceso o hilo en ejecución. La sección crítica por lo general termina en un tiempo determinado y el hilo, proceso o tarea sólo tendrá que esperar un período determinado de tiempo para entrar. Se necesita un mecanismo de sincronización en la entrada y salida de la sección crítica para asegurar la utilización en exclusiva del recurso, por ejemplo un semáforo, monitores, el algoritmo de Dekker y Peterson, los candados.

Productor/Consumidor

En computación, el problema del productor-consumidor es un ejemplo clásico de problema de sincronización de multiprocesos. El programa describe dos procesos, productor y consumidor, ambos comparten un buffer de tamaño finito. La tarea del productor es generar un producto, almacenarlo y comenzar nuevamente; mientras que el consumidor toma (simultáneamente) productos uno a uno. El problema consiste en que el productor no añada más

productos que la capacidad del buffer y que el consumidor no intente tomar un producto si el buffer está vacío.

Exclusión mutua

Los algoritmos de exclusión mutua (comúnmente abreviada como mutex por mutual exclusion) se usan en programación concurrente para evitar que entre más de un proceso a la vez en la sección crítica. La sección crítica es el fragmento de código donde puede modificarse un recurso compartido.

La mayor parte de estos recursos son las señales, contadores, colas y otros datos que se emplean en la comunicación entre el código que se ejecuta cuando se da servicio a una interrupción y el código que se ejecuta el resto del tiempo. Se trata de un problema de vital importancia porque, si no se toman las precauciones debidas, una interrupción puede ocurrir entre dos instrucciones cualesquiera del código normal y esto puede provocar graves fallos.

Hilos

En sistemas operativos, un hilo o hebra (del inglés thread), proceso ligero o subproceso es una secuencia de tareas encadenadas muy pequeña que puede ser ejecutada por un sistema operativo.

La destrucción de los hilos antiguos por los nuevos es una característica que no permite a una aplicación realizar varias tareas a la vez (concurrentemente). Los distintos hilos de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, la situación de autenticación, etc. Esta técnica permite simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente.

Semáforos

Un semáforo es una variable especial (o tipo abstracto de datos) que constituye el método clásico para restringir o permitir el acceso a recursos compartidos (por ejemplo, un recurso de almacenamiento del sistema o variables del código fuente) en un entorno de multiprocesamiento (en el que se ejecutarán varios procesos concurrentemente). Fueron inventados por Edsger Dijkstra en 1965 y se usaron por primera vez en el sistema operativo THEOS.

Binarios

Un tipo simple de semáforo es el binario, que puede tomar solamente los valores 0 y 1. Se inicializan en 1 y son usados cuando sólo un proceso puede acceder a un recurso a la vez. Cuando el recurso está disponible, un proceso accede y decrementa el valor del semáforo con la operación P. El valor queda entonces en 0, lo que hace que si otro proceso intenta decrementarlo tenga que esperar. Cuando el proceso que decremento el semáforo realiza una operación V, algún proceso que estaba esperando comienza a utilizar el recurso.

Compuestos

Son semáforos generales a lo contrario del semáforo binario este tipo de semáforo puede tomar cualquier valor no negativo para poder hacer su proceso de sincronisacion siendo cero la condición de ejecución de un proceso y la espera de otro. En conclusión este tipo de semáforo puede tomar distintos valores tal ves con la finalidad de realizar conteos.

Signals

Una señal (del inglés signal) es una forma limitada de comunicación entre procesos empleada en Unix y otros sistemas operativos compatibles con POSIX. En esencia es una notificación asíncrona enviada a un proceso para informarle de un evento. Cuando se le manda

una señal a un proceso, el sistema operativo modifica su ejecución normal. Si se había establecido anteriormente un procedimiento (handler) para tratar esa señal se ejecuta este, si no se estableció nada previamente se ejecuta la acción por defecto para esa señal.

Pipe

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 consumidor recibe un dato, este se elimina de la tubería.

Fork

Una bifurcación o fork, cuando se aplica en el contexto de un lenguaje de programación o un sistema operativo, hace referencia a la creación de una copia de sí mismo por parte de un programa, que entonces actúa como un "proceso hijo" del proceso originario, ahora llamado "padre". Los procesos resultantes son idénticos, salvo que tienen distinto número de proceso (PID).

Más generalmente, una bifurcación en un entorno multihilo significa que un hilo de ejecución se bifurca.

HERRAMIENTAS Y CONCEPTOS

Hardware o características del sistema

- Sistema operativo: Manjaro 5.9/Debian 10.8
- Memoria Ram: 8 Gb
- Procesador intel core i7
- gcc, version (10.2.0)

Software

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.

Qt

Qt es un framework multiplataforma orientado a objetos ampliamente usado para desarrollar programas (software) que utilicen interfaz gráfica de usuario, así como también diferentes tipos de herramientas para la línea de comandos y consolas para servidores que no necesitan una interfaz gráfica de usuario.

Qt es desarrollada como un software libre y de código abierto a través de Qt Project, donde participa tanto la comunidad, como desarrolladores de Nokia, Digia y otras empresas.

Visual Studio Code

Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye soporte para la depuración, control integrado de Git,

resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias. Es gratuito y de código abierto, aunque la descarga oficial está bajo software privativo e incluye características personalizadas por Microsoft.

METODOLOGÍA

Análisis

Análisis inicial

En primera instancia y teniendo una idea básica sobre lo que se debe hacer se comenzó a realizar una primera solución que constaba del uso de pipes para crear comunicación entre procesos y así administrar la creación, modificación y eliminación de los mismos.

Este primer planteamiento constaba de crear una comunicación bidireccional entre cada padre y cada uno de sus hijos, dejando a cada hijo en ‘modo escucha’, esperando recibir un mensaje mediante el pipe conectado a su padre, esto provocaría una reacción en cadena, tal que.

El programa principal detecta cambios, enviamos señal a cada uno de los tallos para realizar los cambios, cada tallo envía una señal a cada rama para realizar los cambios y cada hoja recibe el mensaje para determinar si debe eliminarse o no.

Problema

Al momento de tener este tipo de comunicación, no tenemos una forma concreta de que cada procesos pueda realizar cambios a la interfaz por defecto, dado que al hacer un fork obtenemos una copia del espacio en el que nos encontramos en conjunto con todas sus variables, esto es un problema, dado que no nos permite realizar lo que queremos y eso es que cada proceso

realice los cambios sobre sus propias características y que estos cambios se vean reflejados en la interfaz.

También se tenía un problema de sincronización, debido a que algunos pipes no se administraban correctamente, por lo que hacía falta un mejor manejo de estos.

Segundo análisis

Al tratar de resolver el problema de la mala administración de las comunicaciones entre procesos, se intentó realizar utilizando un objeto <Tallo> que se encargaría de cargar 56 pipes (aunque no sean todas utilizadas) esta cantidad se debe a que es la cantidad máxima de procesos que un proceso tallo podría tener, dado que existe la limitante de 10 tallos, 5 ramas por tallo y 10 hojas por rama.

Este objeto se encargaba de enviar el pipe de comunicación a un objeto <Proceso>, que se creaba al realizar el Fork() correspondiente, entrando en un bucle tal que esperaba una señal que se enviaría mediante el pipe del padre al hijo.

De esta forma lograremos una comunicación del programa principal a cada uno de los procesos creados, es decir, desde el programa principal podíamos enviar mensajes a los procesos tallos, a los procesos ramas y a los procesos hojas.

Problema

Similar al análisis anterior, no se está realizando un cambio constante en la interfaz, por lo que no se está cumpliendo con uno de los requerimientos del sistema.

Tercer análisis

Intentaremos implementar un nuevo sistema, tal que utilizaremos un archivo de texto generado por el comando actual que se ingrese al sistema, una vez recibido un cambio

procederemos a realizar los cambios en cada uno de los procesos correspondientes, esto se logrará mediante el uso de semáforos.

Garantizando la integridad de nuestro archivo de texto (denominado la sección crítica del sistema) procederemos a escribir al recibir un comando, y posteriormente enviar una señal en cadena entre procesos para que cada proceso se encargue de modificar el archivo, de esta forma al finalizar el recorrido de procesos obtendremos un archivo de texto modificado por cada proceso.

Ahora procedemos a leer la versión final del archivo de texto generado y a hacer una representación gráfica de estos cambios, de esta forma garantizamos que cada proceso realice sus propias modificaciones y que estas se vean reflejadas en la interfaz gráfica.

Problemas

Puede llegar a ser necesario tener un orden específico para poder acceder a los diferentes procesos, tal que se recorran las hojas, posteriormente las ramas y de último los tallos, haciendo que nuestro sistema se vuelva complejo, pues se es necesario controlar el flujo en el que se deberán recorrer los procesos.

Cuarto análisis

Se plantea crear una clase para almacenar la información de cada tallo, rama y hoja; también la estructura tallo tiene una lista de semáforos para enviar señales a cada uno de sus hijos y cada una de las ramas tiene una lista de semáforos para enviar señales a sus nodos hojas, de esta forma podemos solucionar el error en el flujo del análisis, garantizando que cada hoja verifique antes que la rama, y que cada rama verifique antes que el tallo.

Problemas

Existen conflictos con las librerías existentes de Qt, esto se debe a una diferente implementación de algunas funciones a utilizar, como lo pueden ser para el uso de hilos y el uso de variables y métodos estáticos.

Diseño

En primer lugar necesitamos que un semáforo nos indique cuándo dejar de actualizar para poder realizar cambios significativos en el archivo de texto, pero hace falta definir cómo se compondrá este archivo; este tendrá la siguiente estructura:

```
<idTallo>,<idRama>,<idHoja>,<color>
...
<idTallo>,<idRama>,<idHoja>,<color>
```

En este podremos observar que cada línea representará un proceso, el cual está compuesto por el identificador del tallo al que pertenece, a la rama y a la hoja, es importante resaltar que cuando idHoja = 0, se refiere a que la línea representa un proceso del tipo rama y cuando idHoja = 0 && idRama = 0, se refiere a un proceso tipo Tallo.

Una vez definido esto, estableceremos el funcionamiento del programa, tal que al iniciar pondremos a correr dos hilos, uno esperará el ingreso de información y cuando se ingrese solicitará poner en espera el hilo que se encargará de actualizar la información de los procesos en el archivo de texto, esto se logrará utilizando semáforos.

Inicializamos un semáforo tal que una de las partes estará en constante actualización, esto funcionará de la siguiente manera

```
proceso entrada(){
      scanf(); //esperamos que el usuario ingrese el comando a realiar
```

```
	semaforo.wait(); //solicitamos el acceso a la sección crítica una vez que haya comando a ejecutar
	//escribimos en la nueva estructura en el archivo de texto
	semaforo.signal(); //enviamos la señal de que terminamos de acceder a la región crítica
}

proceso actualizacion(){
	semaforo.wait();
	//comenzamos la reacción en cadena para actualizar el color en el archivo de texto de proceso en proceso
	semaforo.signal()//enviamos la señal para repetir el proceso o dar paso a la modificación por un nuevo comando
}
```

Codificación

Clase Main

Se encarga de administrar los procesos principales del sistema que son entrada() y actualización(), estos métodos funcionan de forma ‘intercalada’, aunque la actualización será más recurrente, dado que entrada esperara una entrada por consola para solicitar el acceso a la sección crítica.

En este archivo se encuentran las funciones básicas y también las variables iniciales para administrar los procesos hijos, que serían tallos, en conjunto con su comunicación.

Clase Tallo

Almacena un identificador compuesto de tres números, junto con un listado de 5 semáforos para la comunicación con cada una de sus ramas, así como un semáforo para comunicarse con su padre.

Esta clase verificará el listado de semáforos y al finalizar analizará el estado de ese proceso en el archivo de texto.

Clase Rama

Almacena un identificador compuesto de tres números, junto con un listado de 10 semáforos para la comunicación con cada una de sus hojas, así como un semáforo para comunicarse con su padre.

Esta clase verificará el listado de semáforos y al finalizar analizará el estado de ese proceso en el archivo de texto.

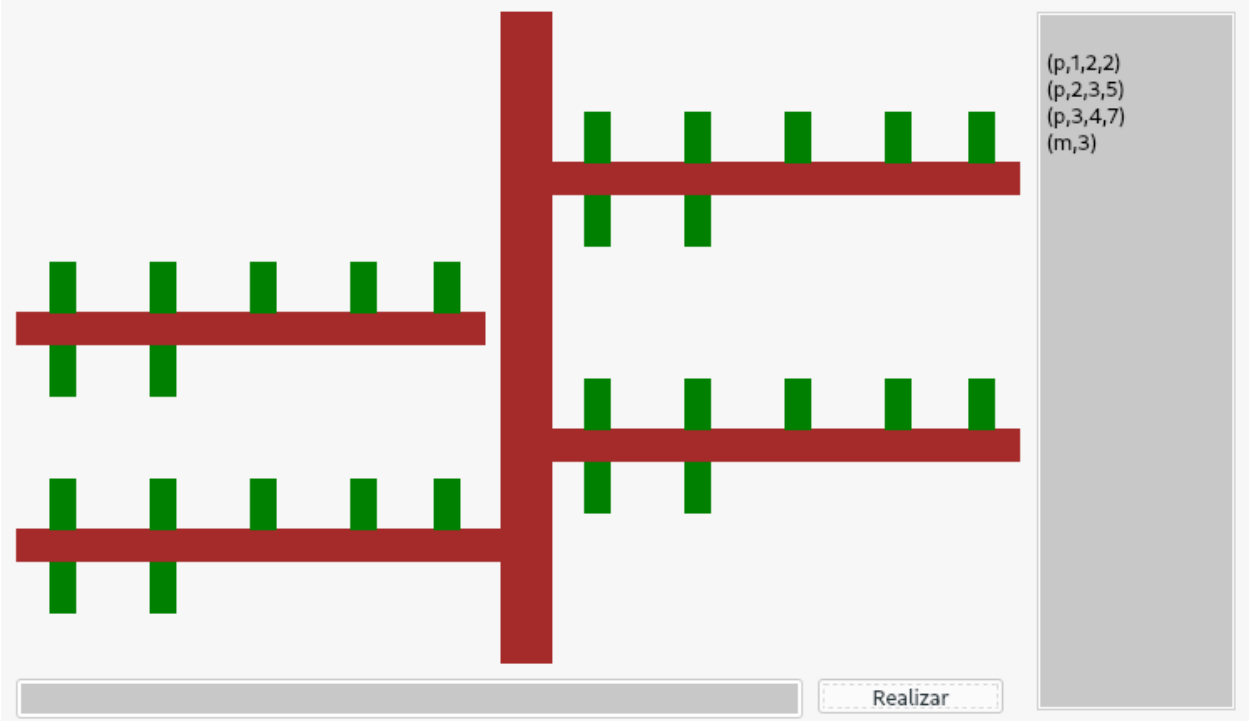
Clase Hoja

Almacena un identificador compuesto por 3 números, también un semáforo para comunicarse con su proceso padre, esta se encargará de ubicar el proceso equivalente en el archivo de texto y modificar el color que se desea pintar en la interfaz.

RESULTADOS

Código 1

El código de este entregable se basa en la creación de procesos mediante las instrucciones establecidas no se permite modificación de los mismos, se acepta el comando mostrar y el comando imprimir, siendo lo que se muestra en pantalla una representación de los procesos creados, aunque estos no afecten directamente la interfaz misma.



Este se encuentra en la carpeta “7mo_SO_Proyecto” y se compila usando el comando make.

Código 2

Hace uso de una ‘sección crítica’ haciendo que únicamente un proceso haga uso de esta, esto se logró mediante la implementación de semáforos, aunque sin éxito absoluto al llevar un orden concreto en el recorrido de los procesos.

Este se encuentra en la carpeta de “semáforo” y se compila usando el comando

```
gcc -o salida main.cpp -pthread -fno-exceptions
```

REFERENCIAS

- MULTIPROCESO, MULTITAREA Y MULTIUSUARIO. (2021). Retrieved 27 March 2021, from https://sisoperativoutp2587.wordpress.com/2016/08/12/multiproceso-multitarea-y-multiusuario/
- pipe() System call - GeeksforGeeks. (2021). Retrieved 27 March 2021, from https://www.geeksforgeeks.org/pipe-system-call/
- Procesos e hilos en C de Unix/Linux. (2021). Retrieved 27 March 2021, from http://www.chuidiang.org/clinux/procesos/procesoshilos.php
- qmake - Wikipedia, la enciclopedia libre. (2021). Retrieved 27 March 2021, from https://es.wikipedia.org/wiki/Qmake
- Qt Creator - Wikipedia, la enciclopedia libre. (2021). Retrieved 27 March 2021, from https://es.wikipedia.org/wiki/Qt_Creator
- Ubuntu - Wikipedia, la enciclopedia libre. (2021). Retrieved 27 March 2021, from https://es.wikipedia.org/wiki/Ubuntu
- (2021). Retrieved 27 March 2021, from https://blog.desdelinux.net/como-matar-procesos-facilmente/
- (2021). Retrieved 27 March 2021, from https://www.youtube.com/watch?v=vvI2-CZ1Bik
- Comunicación entre Procesos UNIX. (2021). Retrieved 27 March 2021, from http://www.lsi.us.es/cursos/seminario-1.html

- 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
- Semáforos en C para Unix/Linux. (2021). Retrieved 27 March 2021, from http://www.chuidiang.org/clinux/ipcs/semaforo.php
- (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
- C++ - Wikipedia, la enciclopedia libre. (2021). Retrieved 27 March 2021, from https://es.wikipedia.org/wiki/C%2B%2B
- CMake - Wikipedia, la enciclopedia libre. (2021). Retrieved 27 March 2021, from https://es.wikipedia.org/wiki/CMake
- Cómo instalar. (2021). Retrieved 27 March 2021, from https://howtoinstall.co/es/qt5-qmake
- Cómo instalar. (2021). Retrieved 27 March 2021, from https://howtoinstall.co/es/cmake
- 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/

- 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/
- fork() in C - GeeksforGeeks. (2021). Retrieved 27 March 2021, from https://www.geeksforgeeks.org/fork-system-call/
- linux c pthread multiproceso - programador clic. (2021). Retrieved 27 March 2021, from https://programmerclick.com/article/35581399098/