# Pthreads

Son procesos livianos que tienen su propio contador de programa y su pila de ejecución, pero no controla el contexto pesado. Algunos SO y lenguajes proveen mecanismos para permitir la programación de aplicaciones multithreading.

Al principio estos mecanismos fueron heterogéneos y poco portables, a mediados de los 90 la organización POSIX auspició el desarrollo de una biblioteca en C para multithreading

Con esta biblioteca se pueden usar threads, asignarles atributos, darlos por terminados, identificarlos, etc.

Pthreads provee funciones básicas para especificar concurrencia:

Texto

Descripción generada automáticamente

El “main” debe esperar a que todos los threads terminen

Inicialmente solo hay un hilo main, el resto de hilos los debe crear explícitamente el programador. Y los hilos que creo pueden crear más hilo , no hay dependencia entre ellos.

Las secciones criticas se implementan en Pthreads usando mutex locks por medio de variables mutex. Una variable mutex tiene dos estados: locked (bloqueado) y unlocked (desbloqueado). En cualquier instante un thread puede bloquear un mutex. Lock es una operación atómica.

Para entrar en la SC un Thread debe lograr tener control del mutex (ósea bloquearlo). Cuando un Thread sale de la SC debe desbloquear el mutex.

Texto

Descripción generada automáticamente

* lock y unlock: bloqueo y desbloqueo
* init es para inicializar la variable mutex. El segundo parámetro es una serie de atributos con valores por default.
* Una vez inicializado, los procesos pueden usar la variable mutex

Problema de productores y consumidores usando exclusión mutua:

* Un thread productor no debe sobreescribir el buffer compartido cuando el elemento anterior no ha sido tomado por un thread consumidor.
* Un thread consumidor no puede tomar nada de la estructura compartida hasta no estar seguro de que se ha producido algo anteriormente
* Los consumidores deben excluirse entre sí
* Los productores deben excluirse entre sí
* En este ejemplo el buffer es de tamaño 1.

MAIN:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Código de productores

Texto

Descripción generada automáticamente

Código de consumidores

Texto

Descripción generada automáticamente

Tipos mutex

Cuando se inicializa mutex se le pasan unos parámetros, usualmente se le pasan los default , pero hay uno que indica el tipo de Mutexs. Los tipos son Normal, Recursive y Error Check.

* Un mutex con el atributo **Normal** NO permite que un thread que lo tiene bloqueado vuelva a hacer un lock sobre él (deadlock)
* Un Mutex con el atributo **recursive** SI permite que un thread que lo tiene bloqueado vuelva a hacer un lock sobre él, incrementando una cuenta de control. Para desbloquearlo hay que hacer desbloqueos igual al numero de la cuenta de control
* Un Mutex con el atributo **ErrorCheck** responde con un reporte de error al intento de un segundo bloqueo por el mismo thread. En lugar de bloquear el proceso le devuelve un código de error.

Overhead de bloqueos por exclusión mutua

Los locks representan puntos de serialización, si dentro de una SC ponemos segmentos largos de programa tendremos una degradación importante de performance. A menudo se puede reducir el overhead por espera ociosa utilizando la función **pthread\_mutex\_trylock**. Retorna el control informando si pudo hacer o no el lock. Esto evita tiempos ociosos y es menos costos por no tener que manejar colas de espera.

Trylock: el profe recomienda usarlo cuando el proceso es completamente capaz de hacer otras instrucciones que NO requieran variables compartidas. Osea, intenta bloquear -> si puede, entra a SC y labura, sino hace el otro laburo sin SC. Si el proceso si o si requiere SC, entonces tiene más sentido usar el lock común.

Variables condición

Podemos utilizar variables condición para que un thread se autobloquee hasta que se alcance un estado determinado del programa. Cada variable esta asociada con un predicado. Cuando el predicado se convierte en verdadero la variable de condición da una señal para los threads que están esperando por el cambio de estado de condición.

Una única variable de condición puede asociarse a varios predicados, pero puede ser difícil el debug. Una variable condición siempre tiene un mutex asociada a ella. Cada thread bloquea este mutex y testea el predicado definido sobre la variable compartida.

Si el predicado es falso el thread espera en la variable condición utilizando la función **pthread\_cond\_wait**

Texto

Descripción generada automáticamente

* Init -> inicializa (se llama antes de cualquier uso)
* destroy -> destruye la variable.
* wait -> para mimir el proceso en la variable condición (primer parámetro), el segundo parámetro es el mutex que se tuvo que usar. Primero se bloquea el mutex para modificar las variables compartidas q forman parte de la condición, una vez desbloqueado el mutex, se duerme el proceso en la cola de condición.
  + Si se despierta, se tiene que volver a bloquear la variable mutex del segundo parámetro, ya que se durmió en SC, al despertarse sigue en SC. Al bloquear el mutex y usar la SC, termina de usar la SC y ahí sigue la cosa.
* timedwait -> funciona parecido, pero la diferencia es el 3er parámetro que es una cantidad X de tiempo que se espera para ser despertado, si se excede dicho tiempo el proceso se despierta automáticamente.
* signal -> despierta al primer proceso mimido en la variable condición pasada por parámetro.
* broadcast -> despierta TODOS los procesos mimidos en la variable condición pasada por parámetro.

Problema de productores consumidores pero usando variables condición:

MAIN

Texto, Carta

Descripción generada automáticamente

Codigo de productores:

Texto, Carta

Descripción generada automáticamente

Hay un while (hayElemento == 1) porque puede ocurrir que si despiertan a un productor dormido otro productor que no estaba dormido tome el control de la SC dejando cosas en el buffer. De esta forma apenas se despierta el productor dormido se rechequea que haya o no elementos en el buffer, así sabe si dormir de nuevo no.

Codigo de consumidores:Texto, Carta

Descripción generada automáticamente

El while es el mismo funcionamiento pero para consumidores.

Atributos y sincronización en pthreads

La API Pthreads permite que se puede cambiar los atributos por defecto de las entidades usando attributes objects.

Un attribute object es una estructura de datos que describe las propiedades de la entidad en cuestión. Una vez que estas propiedades están establecidas el attribute object es pasado al método que inicializa la entidad.

Tiene como ventaja que mejora la modularidad y da facilidad de modificación del código.

Texto

Descripción generada automáticamente

Texto, Carta

Descripción generada automáticamente

# Semáforos en pthreads

Los threads se pueden sincronizar por semáforos usando la librería semaphore.h

He aquí su declaración y operaciones:

Texto

Descripción generada automáticamente

Ejemplo de productores y consumidores con semáforos.

Las funciones de productor y consumidor serán ejecutadas por threads independientes, que acceden a un buffer compartido. El productor deposita una secuencia de enteros en el buffer y el consumidor busca estos valores y los suma.

Los semáforos vacio y lleno garantizan el acceso alternativo de productor y consumidor sobre el buffer.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente con confianza media

Hacen lo mismo que Pthreads en cuanto a exclusión mutua, en cuanto a sincronización por condición las variables cond tienden a ser más eficientes por tener menos overhead.

# Monitores en Pthreads

Pthreads nos permite manejar la exclusión mutua por medio de las variables mutex y nos permite manejar la sincronización por condición utilizando variables condición para que un thread se autobloquee hasta que se alcance un estado determinado del programa. Una variable de condición siempre tiene un mutex asociada a ella.

Pthreads no posee “Monitores” pero con las dos herramientas anteriores se puede simular: con mutex se hace la exclusión mutua que nos brindaba el monitor de forma implícita y con las variables condición la sincronización.

El acceso exclusivo al monitor se simula usando una variable mutex la cual se bloquea antes de la llamada al procedure y se desbloquea al terminar el mismo. Cada llamado de un proceso a un procedure de un monitor debe ser reemplazado por el código de ese procedure.

Ejemplo de lectores y escritores usando Pthreads.

Texto, Carta

Descripción generada automáticamente

Como solo hay un monitor se pone solo una variable mutex que se inicializa en el main del programa

Texto, Carta

Descripción generada automáticamente

Se agrega en los procesos el bloqueo y desbloqueo de mutex en los llamados a los procedure del monitor

Diagrama

Descripción generada automáticamente con confianza media

Ahora toca reemplazar los llamados al monitor por el código de los procedure

Imagen que contiene Escala de tiempo

Descripción generada automáticamente