EBook Gratis

APRENDIZAJE pthreads

Free unaffiliated eBook created from
Stack Overflow contributors.

#pthreads

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: pthreads

It is an unofficial and free pthreads ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official pthreads.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con pthreads

Observaciones

Esta sección proporciona una descripción general de qué es pthreads y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema grande dentro de pthreads, y vincular a los temas relacionados. Dado que la Documentación para pthreads es nueva, es posible que deba crear versiones iniciales de esos temas relacionados.

Examples

Instalación o configuración

Instrucciones detalladas sobre cómo configurar o instalar pthreads.

Minimal "Hello World" con pthreads

```
#include <pthread.h>
#include <stdio.h>
#include <string.h>

/* function to be run as a thread always must have the same signature:
   it has one void* parameter and returns void */
void *threadfunction(void *arg)
{
  printf("Hello, World!\n"); /*printf() is specified as thread-safe as of C11*/
  return 0;
}

int main(void)
{
  pthread_t thread;
  int createerror = pthread_create(&thread, NULL, threadfunction, NULL);
  /*creates a new thread with default attributes and NULL passed as the argument to the start
routine*/
  if (!createerror) /*check whether the thread creation was successful*/
    {
      pthread_join(thread, NULL); /*wait until the created thread terminates*/
      return 0;
    }
  fprintf("%s\n", strerror(createerror), stderr);
  return 1;
}
```

Pasando argumentos a hilos

```
#include <stdio.h>
#include <pthread.h>
```

```
void *thread_func(void *arg)
{
    printf("I am thread #%d\n", *(int *)arg);
    return NULL;
}

int main(int argc, char *argv[])
{
    pthread_t t1, t2;
    int i = 1;
    int j = 2;

    /* Create 2 threads t1 and t2 with default attributes which will execute
    function "thread_func()" in their own contexts with specified arguments. */
    pthread_create(&t1, NULL, &thread_func, &i);
    pthread_create(&t2, NULL, &thread_func, &j);

    /* This makes the main thread wait on the death of t1 and t2. */
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    printf("In main thread\n");
    return 0;
}
```

Cómo compilar:

```
$ gcc -pthread -o hello hello.c
```

Esto imprime:

```
I am thread #1
I am thread #2
In main thread
```

Devolviendo el resultado del hilo

Se puede utilizar un puntero a un tipo de datos concreto, convertido a `void *` , para pasar valores y devolver los resultados de la función de subproceso.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

struct thread_args
{
    int a;
    double b;
};


struct thread_result
{
    long x;
    double y;
```

```
};

void *thread_func(void *args_void)
{
    struct thread_args *args = args_void;
    /* The thread cannot return a pointer to a local variable */
    struct thread_result *res = malloc(sizeof *res);

    res->x  = 10 + args->a;
    res->y = args->a * args->b;
    return res;
}

int main()
{
    pthread_t threadL;
    struct thread_args in = { .a = 10, .b = 3.141592653 };
    void *out_void;
    struct thread_result *out;

    pthread_create(&threadL, NULL, thread_func, &in);
    pthread_join(threadL, &out_void);
    out = out_void;
    printf("out -> x = %ld\tout -> b = %f\n", out->x, out->y);
    free(out);

    return 0;
}
```

En muchos casos, no es necesario pasar un valor de retorno de esta manera; por ejemplo, el espacio en la estructura de argumento también se puede usar para devolver resultados, o se puede pasar un puntero a una estructura de datos compartida al hilo y los resultados almacenados allí. .

Lea Empezando con pthreads en línea: https://riptutorial.com/es/pthreads/topic/5669/empezando-con-pthreads

Capítulo 2: Condición de carrera en pthreads.

Introducción

Al escribir aplicaciones de subprocesos múltiples, uno de los problemas más comunes que se experimentan son las condiciones de carrera. Así que documentamos el ¿Cómo los detectas? y como los manejas?

Examples

Ejemplo: Considerar tendrá dos hilos T1 y T2.

> ¿Cómo los detectas?

Si varios subprocesos pueden acceder a la misma *ubicación de variable / recurso / memoria* y al menos uno de ellos está cambiando el valor de *ubicación* de *variable / recurso / memoria* , entonces puede ocurrir la **Condición de Carrera** . Porque si un hilo está cambiando el valor de *la ubicación* de *variable / recurso / memoria* y otro hilo intenta leer lo mismo, entonces no obtendrá el valor actualizado.

Nota : si todos los subprocesos solo están leyendo la *ubicación de* la *variable / recurso / memoria,* entonces no se producirá la **Condición de Carrera** .

Ejemplo: Programa sufre de Condición de Carrera

```
#include <stdio.h>
#include <pthread.h>

int x= 0;

void* fun(void* in)
{
    int i;
    for ( i = 0; i < 10000000; i++ )
    {
        x++;
    }
}

int main()
{
    pthread_t t1, t2;
    printf("Point 1 >> X is: %d\n", x);

    pthread_create(&t1, NULL, fun, NULL);
    pthread_create(&t2, NULL, fun, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    printf("Point 2 >> X is: %d\n", x);
    return 0;
```

```
}
```

La salida en mi pantalla es:

```
Point 1 >> X is: 0
Point 2 >> X is: 9925047
```

Su salida puede variar. Pero seguro que no serán 20.000.000. Dado que ambos subprocesos ejecutan el mismo bucle y tienen una variable global `int x;`

```
for ( i = 0; i < 10000000; i++ )
{
   x++;
}
```

Entonces, el valor final de `x` en la línea `Point 2 >> X is: 9925047` debe ser 20,000,000. Pero no es así.

El estado de x se puede cambiar por otro hilo durante el tiempo entre x se está leyendo y cuando se vuelve a escribir.

Digamos que un hilo recupera el valor de x, pero aún no lo ha almacenado. ¡Otro hilo también puede recuperar el mismo valor de x (porque ningún hilo lo ha cambiado todavía) y luego ambos estarían almacenando el mismo valor (x + 1) de nuevo en x!

Ejemplo:

Hilo 1: lee x, el valor es 7

Hilo 1: agregue 1 a x, el valor ahora es 8

Hilo 2: lee x, el valor es 7

Hilo 1: almacena 8 en x

Hilo 2: agrega 1 a x, el valor ahora es 8

Hilo 2: almacena 8 en x

> ¿Cómo los manejas?

Las condiciones de carrera pueden evitarse empleando algún tipo de mecanismo de bloqueo antes del código que accede al recurso compartido o la exclusión mutua.

A continuación se modifica el programa:

Ejemplo: problema de condición de carrera resuelto

```
#include <stdio.h>
#include <pthread.h>
```

```
int x= 0;
//Create mutex
pthread_mutex_t test_mutex;

void* fun(void* in)
{
    int i;
    for ( i = 0; i < 10000000; i++ )
    {
        //Lock mutex before going to change variable
        pthread_mutex_lock(&test_mutex);
        x++;
        //Unlock mutex after changing the variable
        pthread_mutex_unlock(&test_mutex);
    }
}

int main()
{
    pthread_t t1, t2;
    printf("Point 1 >> X is: %d\n", x);

    //Initlize mutex
    pthread_mutex_init(&test_mutex, NULL);

    pthread_create(&t1, NULL, fun, NULL);
    pthread_create(&t2, NULL, fun, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    //Destroy mutex after use
    pthread_mutex_destroy(&test_mutex);
    printf("Point 2 >> X is: %d\n", x);
    return 0;
}
```

A continuación se muestra la salida:

```
Point 1 >> X is: 0
Point 2 >> X is: 20000000
```

Aquí, la respuesta sale como 20,000,000 cada vez.

Nota : el programa modificado, que está libre de error de condición de carrera, tardará mucho en ejecutarse. Porque hay sobrecarga en el bloqueo y desbloqueo `mutex` .

Lea Condición de carrera en pthreads. en línea:
https://riptutorial.com/es/pthreads/topic/8243/condicion-de-carrera-en-pthreads-

Capítulo 3: Variables condicionales

Introducción

Las variables condicionales son útiles en los casos en que desea que un subproceso espere algo que sucede en otro subproceso. Por ejemplo, en un escenario productor / consumidor con uno o más subprocesos productores y un subproceso consumidor, se pueden usar variables condicionales para indicar al subproceso consumidor que hay nuevos datos disponibles.

Observaciones

Proceso general

Una espera en una variable condicional (queueCond en el ejemplo productor / consumidor) siempre está acoplada a un mutex (el queueMutex en el ejemplo productor / consumidor), y siempre debería estar acoplada a una variable de estado "normal" también (queue.empty ( ) en el ejemplo productor / consumidor). Cuando se usa correctamente, esto asegura que no se pierdan datos nuevos en el consumidor.

En general, el proceso debe ser:

- Para el hilo de señalización:
    1. Bloquear el mutex
    2. Actualizar cualquier datos y variables de estado
    3. Señala la variable de condición
    4. Desbloquear el mutex
- Para el hilo de espera:
    1. Bloquear el mutex
    2. Hacer un `while` bucle en la variable de estado, de enlace, siempre y cuando los datos no está listo
    3. En el `while` de bucle, hacer una espera en la variable de condición con `pthread_cond_wait()`
    4. Cuando los `while` las salidas de bucle, ahora estamos seguros de que nuevos datos está listo, y que el mutex está bloqueado
    5. Hacer algo con los datos
    6. Desbloquea el mutex y repite.

Con este esquema, sin importar cuándo se programen los subprocesos de señalización y espera, el subproceso en espera nunca perderá datos (como en, nunca se quedará atascado esperando para siempre con los datos válidos listos). Esto se puede realizar al intentar ejecutar manualmente los pasos para el subproceso de señalización, registrando los estados del mutex, la condición y las variables de estado, para cada uno de los pasos en el subproceso en espera.

`pthread_cond_wait` y el mutex

Para facilitar el proceso anterior, es necesario llamar a `pthread_cond_wait()` con el mutex

bloqueado. Cuando se le llama, `pthread_cond_wait()` luego desbloqueará la exclusión mutua antes de poner el hilo en suspensión y, justo antes de regresar por cualquier motivo, la exclusión mutua se volverá a bloquear. Esto también significa que si algún otro subproceso tiene el mutex actualmente bloqueado, `pthread_cond_wait()` esperará a que el mutex se desbloquee, y hasta que el subproceso en espera adquiera el mutex, contenderá con otros hilos que intenten bloquear El mutex al mismo tiempo.

Despertares espurios

También, puede parecer como si el `while` bucle de espera en la variable de estado podría ser sustituido por un simple `if` declaración. Sin embargo, el `while` que se necesita bucle, como el estándar POSIX permite `pthread_cond_wait()` para hacer los llamados "falsos" activaciones durante la espera, sin llegar a ser señalado. Por lo tanto, el código debe volver a verificar la variable de estado para ver si `pthread_cond_wait()` devolvió debido a que realmente fue señalado, o debido a una de estas activaciones falsas.

Examples

Ejemplo productor / consumidor

```
pthread_mutex_t queueMutex;
pthread_cond_t queueCond;
Queue queue;

void Initialize() {
    //Initialize the mutex and the condition variable
    pthread_mutex_init(&queueMutex, NULL);
    pthread_cond_init(&queueCond, NULL);
}

void Producer() {
    //First we get some new data
    Data *newData = MakeNewData();

    //Lock the queue mutex to make sure that adding data to the queue happens correctly
    pthread_mutex_lock(&queueMutex);

    //Push new data to the queue
    queue.push(newData);

    //Signal the condition variable that new data is available in the queue
    pthread_cond_signal(&queueCond);

    //Done, unlock the mutex
    pthread_mutex_unlock(&queueMutex);
}


void Consumer() {

    //Run the consumer loop
    while(1) {

        //Start by locking the queue mutex
        pthread_mutex_lock(&queueMutex);
```

```
        //As long as the queue is empty,
        while(queue.empty()) {
            // - wait for the condition variable to be signalled
            //Note: This call unlocks the mutex when called and
            //relocks it before returning!
            pthread_cond_wait(&queueCond, &queueMutex);
        }

        //As we returned from the call, there must be new data in the queue - get it,
        Data *newData = queue.front();
        // - and remove it from the queue
        queue.pop();

        //Now unlock the mutex
        pthread_mutex_unlock(&queueMutex);

        // - and process the new data
        ProcessData(newData);
    }
}
```

Lea Variables condicionales en línea: https://riptutorial.com/es/pthreads/topic/8614/variables-condicionales

Creditos

S. No	Capítulos	Contributors
1	Empezando con pthreads	Armali, ArturFH, caf, Community, cse, EOF, nachiketkulk
2	Condición de carrera en pthreads.	cse, Mohan
3	Variables condicionales	sonicwave