Durum Değişkenleri

Durum Değişkenleri senkronizasyonun ilkel bir ikinci türdür. (Muteksler ilk olanlardır).Kesin koşullarla gerçeklenmek için beklemeye ihtiyacı olan threadler için kullanışlıdır.Pthreads olarak, durum değişkenleri içeren üç ilgili prosedür vardır:

*** pthread_cond_init(pthread_cond_t *cv);**

*** pthread_cond_wait(pthread_cond_t *cv, pthread_mutex_t *lock);**

*** pthread_cond_signal(pthread_cond_t *cv);**

Bunlardan ilki, sadece bir koşul değişkeni başlatır. ikincisi ile ilişkilidir.Thread'ın koşul doğru olana kadar bloklamak ve koşulun doğru olmasını beklediği zaman **Pthread_cond_wait()** fonksiyonu çağrılır.Bu şekilde ikinci parametre ile belirtilen muteksin kilitli olduğu kabul edilir. Sözkonusu mutex serbest bırakılır ve başka bir iş parçacığını bir pthread_cond_signal () çağrı tarafından bloklar.Bu Mutex uyandığı zaman bu mutexi alma zamanı kadar bekler ve bir kez alınıp, o pthread_cond_wait () çağrısından döndürür.

Pthread_cond_signal () belirtilen koşul değişkeni bekleyen herhangi bir konu olup olmadığını denetler. Aksi takdirde, sadece döner. Eğer bekleyen konuları varsa, sonra uyandırılır.Bu belirtilenin olup olmadığını pthread_cond_wait () parçacığının çağrı tarafından belirtilen kilitli mutex kendisi için gerekli olup olmadığını belirten pthread_cond_signal () fonksiyonunu çağırır.Ben böyle olmasını öneririm.

Not : pthread_cond_signal çağrıları tarafından uyandırılan sıra hakkında hiçbir şey düşünülmemelidir. Onların beklenen sırayla uyanmış olacağını varsaymak doğaldır. Ama bu durumda olmayabilir.

Basit Bir Örnek

Değişkenler koşulu kullanılarak basit program: barrier.c .

Burada 5 konu var ve biz hepsinin belli bir noktada senkronize olduğundan emin olmak istiyoruz. Threadler işlenmeye geçmeden önce bu bariyerde durdurkları için genellikle `` bariyer" olarak adlandırılır.Bekleyen thread sayısı **ndone** değişkeni içinde tutulur, **ndone NTHREADS'**e eşit olmadan önce thread bariyeri aşarsa **ts->cv** durum değişkeninde bekler.Son thread'de bariyeri aştığında diğer bütün thread'leri **pthread_cond_signal** çağrısı ile uyandırır.**barrier.c** programının çıktı olarak son thread'ın bariyeri aşmasına kadar bloklanan threadleri ve ondan sonra bariyeri aşan threadleri gösterir.

UNIX> barrier

Thread 0 – bariyer bekliyor

Thread 1 -- bariyer bekliyor

Thread 2 -- bariyer bekliyor

Thread 3 -- bariyer bekliyor

Thread 4 -- bariyer bekliyor

Thread 4 -- bariyer sonra

Thread 0 -- bariyer sonra

Thread 1 -- bariyer sonra

Thread 2 -- bariyer sonra

Thread 3 -- bariyer sonra

done

UNIX>

Calling pthread_join

(Thread'ın fork işleminde sonra **pthread_detach**() çağrısını çağırmamız bütün bu işlemleri anlamsız kılar.Ancak bu olay durum değişkenlerini kullanarak denemenin hala en iyisidir.)

Eğer beni sevdiyseniz, thread'ı temizlemek için **pthread_join()** fonksiyonunu kullanmak sizin için sürekli bir eziyet olacaktır.Aslında thread'ın işini bitirip bitirmediğini önemsemiyorsanız bu fonksiyonu çağırmanıza gerek yoktur.Fakat birçok thread yaratıp pthread_join() fonksiyonunu çağırmazsanız bu büyük bir kaynak tahsisi problemine yol açacaktır.örnek için **bigfork.c'** ye bakıyoruz:

```
void *thread(void *arg)
{
  return NULL;
}

main()
{
  pthread_t tid;
  int i;
  int j;

  j = 0;
  while(1) {
    printf("j = %d\n", j);
    j++;
    for (i = 0; i < 1000; i++) {
      if (pthread_create(&tid, NULL, thread, NULL) != 0) {
        perror("pthread_create");
        exit(1);
      }
    }
    sleep(2);
  }
}
```

Bu program 1000 adet threadı yineleyerek yaratır ve iki saniye bekler.Thread'ler keni kendilerine basitçe geri dönerler. 1000 adet thread iki saniye içinde tamamlanmış olmalı ki main thread bekleyebilsin, böylece her bir yinelemenin sonunda sadece main thread olacaktır.(nasıl test edeceğinizi düşünün)

```
UNIX> bigfork
j = 0
j = 1
j = 2
_alloc_chunk(): _mmap failed: Not enough space
pthread_create: Not enough space
UNIX>
```

2000 ve 3000 adet thread arasında dallanırken bellek dışına çıkarız.bu yüzdendir ki her thread kendi stack yapısı kadar alan tahsis eder. Bu 2000 adetten fazla thread **pthread_join()** çağrısını çağırmadan kendi stack kapasitelerini serbest bırakmazlar.Örneğin bigfork2.c uyku durumundan sonra her 1000 thread grubu için gruplara katılır.Görebileceğiniz gibi bellek alanını aşmadan sürekli çalışabilir.

```
UNIX> bigfork2
j = 0
j = 1
j = 2
j = 3
...
UNIX>
```

şimdi jthread.c ye bakalım

Jthread.c'de 3 adet procedure tanımlıdır.

jthread_system_init(): bu prosedürü diğer prosedürleri kullanmak için mutlaka kullanmalıyız.

jthread_create(void (*func)(void *), arg):Bu çağrı thread'ın fonksiyon ve parametreleriyle fork işlemini gerçekler.'0' döndürürse başarılı işlem '1' döndürürse hata demektir.size "tid" vermediğini not edin.Jthread_create() çağrısı ile yaratılan threadler üstünde join çağrısı çalışmaz.Kendi kendilerini otomatik olarak temizlerler.Bu sebepten dolayı fonkisyon geri değer döndürmez

jthread_exit(): Bu sizin ana thread'den veya dallanmış thread'den çıkmak için çağrı yaptığınızı gösterir.thread bulunduğu zamanlar otomatik olarak temizler.

Bu prosedürlerin tanımları jthread.c içindedir.Bu konuyu henüz açıklamayacağız.Bunun yerine hello_world.c ve bigfork3.c örneklerine bakınız.katılımsız dallanma için jthread.h ve jthread.c nin her ikiside kullanılır.Her ikisininde iyi çalıştığını göreceksiniz.Kullanışlı değil mi?

Şimdi , jthread.c'nin kısa bir açıklaması jthread_system_init global mutex'in, durum değişkenlerinin, dlist'in ve varolan threadlerin sayılarını ilklendirir.Ardından bir garbage-collecting thread yaratır. Dlist'teki bütün threadlar için thread'ın **pthreads_join** çağrısı çağrılır, dlist boş ise bekler.jthread_create sayaçı arttırır, sonra dallanmış prosedür

jthread_starter() çağrısını çağırır. Bu thread istenilen fonksiyonu ve argumanı çağırır ve eğer geri dönerse jthread_exit() prosedürünü çağırır.böylece , eğer thread thread_exit() prosedürünü direkt çağırırsa yada dönerse jthread_exit() çağıracak.
şimdi **jthread_exit() pthread_self()'**i onun tid'ini almak için çağıracak ve onu dlistin sonuna ekleyerek garbage-collecting threadının **pthread_cond_signal()**'i ile çağrılmasını bekleyecek.Bu şekilde bütün threadler kaynaklarını serbest bırakacaklar.