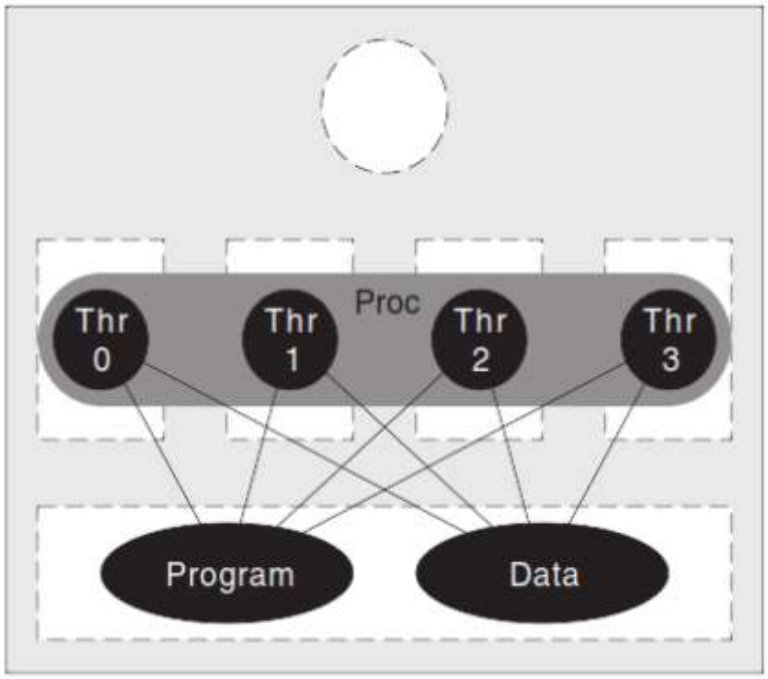
Thread

Bir işletim sisteminde ya da yazılan bir programda birden fazla işin aynı anda yapılabilmesi için kullanılır. Threadlerin varlıkları bir işleme bağlıdır ve lightweighted process olarak anılabilirler.



ŞEKIL 1. SMP PARALLEL PROGRAM

Pthread

C'nin thread kullanımına imkan tanıyan POSIX kütüphanesidir

Bir thread başladıktan sonra;

- Hazır durumuna geçer,
- Scheduling algoritmasına bağlı olarak çalışır duruma geçebilir,
- Çalışma işlemi sırasında thread işini bitirirse veya iptal olursa/edilirse thread sonlanma durumuna geçer,
- Ya da çalışma durumundayken, bir sebeple bekleme durumuna geçebilir: devam etmek için bir başka thread'i beklemesi gerekebilir ya da bir süre uyutulmuş olabilir ya da bir kaynağa erişmek istiyordur ama kaynak başka bir thread tarafından kullanımdadır onu bekleyebilir.
- Bekleme durumuna sebep olan durumlar ortadan kalktığında hazır durumuna tekrar geçer ve hazır kuyruğunda beklemeye başlar.

Pthread Fonksiyonları

Fonksiyon ismi	Fonksiyonun Görevi
int pthread_create(pthread_t *tid, cons pthread_attr_t *attr, void* funk, void * arg)	Thread oluşturmak için kullanılan fonksiyondur. 4 parametre alır; thread id, thread attribute, thread'in çalışacağı fonksiyon, fonksiyona verilecek parametreler. 0 dönerse başarılı, dönmezse hata var demektir.
int pthread_exit(pthread_t *tid)	Parametre olarak aldığı threadi sonlandırır.
int pthread_equal(pthread_t *tid1, pthread *tid2)	Parametre olarak verilen 2 thread'i karşılaştırır ve eğer birbirlerine eşitlerse 0 döndürür; değillerse duruma göre başka değerler döndürür.

Fonksiyon ismi	Fonksiyonun Görevi
int pthread_join(pthread_t tid, void** ptr)	Verilen thread için işlemi bitene kadar bekler ve döndürülen değeri 2. Parametresine alır.
int pthread_detach(pthread_t tid)	Thread'in memory'deki kopyasını siler.
int pthread_cance(pthread_t tid)	Thread'i iptal eder.
pthread_t pthread_self()	O anda çalışan Thread'in bilgisini döndürür.
int sched_yield()	Çağırılan Thread'in çalışma durumundan alınarak bekleme durumuna geçirilmesini sağlar, bu sayede aynı kaynak için bekleyen başka bir Thread kaynağa erişebilecektir.

Örnek Pthread - 1

Yukarıda anlatılan threadlerin durum geçişleriyle ilgili örnek bir kod aşağıdaki şekilde yazılabilir.

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

void *thread_routine(void *arg){
        printf("Newly produced thread is working. \n");
}
int main(void) {
        pthread_t thread_id; void *thread_result;
        pthread_create(&thread_id, NULL, thread_routine, NULL);
        printf(" Main Thread is working. \n");
        pthread_join(tread_id, &thread_result);
        return 0;
}
```

```
lab4 ./pthread_1
Main thread is working.
Newly produced thread is working.
```

Yukarıda görüldüğü üzere önce pthread_t veri yapısından thread_id isimli bir değişken tanımlanmıştır. Daha sonra pthread_create fonksiyonu kullanılarak yeni threadimiz thead_id de parametre verilerek oluşturulmuştur ve çağırılana kadar hazır durumunda beklemektedir. Daha sonra printf fonksiyonu işletilmiştir. En son da pthread_join fonksiyonu ile thread_routine fonksiyonuna dallanılarak üretilen thread çalıştırılmıştır.

Örnek Pthread - 2

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

void *thread_routine(void *arg){
        printf("Newly produced thread is working. \n");
        return(void*); strdup("1 is over.");
}
```

```
İnt main(void) {
        pthread_t thread_id; void *thread_result;
        pthread_create(&thread_id, NULL, thread_routine, NULL);
        printf(“ Main Thread is working. \n");
        pthread_join(tread_id, &thread_result);
        if(thread_result != 0)
                printf(“Thread: %s came back to main thread. \n”, thread_result);
        return 0;
}
```

```
lab4 ./pthread_2
Main thread is working.
Newly produced thread is working.
Thread: 1 is over. Came back to main thread!
```

Örnek Pthread - 3

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

void *thread_routine(void *arg){
        printf(“Thread is created \n”);
        sleep(30);
        printf(“After sleep.”);
}
İnt main(void) {
        pthread_t thread_id; void *thread_result;
        pthread_create(&thread_id, NULL, thread_routine, NULL);
        sleep(3);
        printf(“ Main Thread is here \n");
        pthread_cancel(thread_id);
        printf(“The End”);
        return 0;
}
```

```
lab4 ./pthread_3
Thread is created
Main thread is here
The End
```

Yukarıda görüldüğü üzere main thread 3 saniyelik uykudan sonra “Main Thread” mesajını yayınlayacak ve subthread daha uykudan uyanıp “After sleep” mesajını basamadan onun görevini sonlandıracaktır.

Mutex

Pthread Mutex Fonksiyonları

Fonksiyon ismi	Fonksiyonun Görevi
pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);	Yeni bir mutex oluştururken kullanılır ve verilen değişken ile ifade edilen ve ikinci parametrede verilen özellikleri taşıyan yeni bir thread mutex oluşturur. Bu değişken daha sonradan lock ve unlock için kullanılır.
int pthread_mutex_destroy(pthread_mutex_t *mutex)	Mutex kilidini kaldırır ve yok eder.
int pthread_mutex_lock(pthread_mutex_t *mutex)	Lock işlemini gerçekleştirir ve lock'ı alan thread dışında tüm threadler unlock işlemi gerçeklenene kadar beklerler.
int pthread_mutex_unlock(pthread_mutex_t *mutex)	Kritik bölgede işi biten Thread unlock ile kritik bölgeye başka bekleyen bir Thread'in girmesine imkan tanır.
int pthread_mutex_trylock(pthread_mutex_t *mutex)	Kilidi elde etmeden önce kilit alınmak için uygun mu diye sorar. Eğer zaten kilitli ise busy mesajı döner.

Örnek Pthread - 4

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

pthread_t_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
int shared data = 1;

void *consumer(void *arg){
        for(int i = 0; i<30;i ++){
                pthread_mutex_lock(&mutex);
                shared_data —;
                pthread_mutex_unlock(&mutex);
        }
        printf("Returning from Consumer= %d\n", shared_data);
}
İnt main(void) {
        pthread_t thread_id;
        pthread_create(&thread_id, NULL, thread_routine, NULL);
        for(int i = 0; i<30; i++){
                pthread_mutex_lock(&mutex);
                shared_data ++;
                pthread_mutex_unlock(&mutex);
        }
        printf("End of Main = %d\n", shared_data);
        return 0;
}
```

```
lab4 ./pthread_4
End of Main=2
```

```
Returning from Consumer= 1
```

Yukarıdaki kodda producer ve consumer problemi için bilinen 2 problemden yalnızca birisi çözülmüştür. Progress probleminin çözülebilmesi için conditional variable kullanılmalıdır. Buna göre problemde şayet üretim için ayrılan alan kalmadıysa üreticilerin durup tüketicilerden en az birisinin bir tüketim yapmasını ve yeni alan açılmasını beklemesi gerekir. Benzer şekilde şayet hiç ürün yoksa, bu durumda da tüketicilerin durup en az bir üreticinin bir ürün üreterek hatta koymasını beklemesi gerekir. Burada ise yalnızca kritik bölgeye bir anda tek bir thread'in gireceği lock mekanizması ile garanti edilmiştir.

Referanslar

1. Mehmet Sıddık Aktaş "Parallel and Distributed Computing Lecture Notes"
2. Alan Kaminsky " Building Parallel Programs"