BLGM 343 – DENEY 4 *
İŞ PARÇACIKLARI

Amaçlar

1. POSIX tabanlı iş parçacıkları hakkında bilgi edinilmesi
2. İş parçacıklarıyla paralellik

Ön Bilgi

Bu deneyin amacı iş parçacıkları hakkında bilgi edinmektir. İş parçacıkları sayesinde sayesinde, bir program yapacağı işleri aynı anda çalıştırabilir. Çoklu işlemcili bilgisayarlarda iş parçacıkları kullanıldığında programın çalışma süresini de kısaltacaktır. Benzer bir durum alt işlemler kullanılarak da yapılabilir, ancak, bu işlemlerin kendi aralarında veri paylaşması, iş parçacıklarına göre daha zordur. Ayrıca yeni açılan işlemlerin hafızada kendilerine ait bir alanı bulunduğu için, her açılan işlem hafızada ek bir yer kaplamaktadır. İş parçacıkları ise aynı hafıza alanında çalışırlar.

Linux ve benzeri bir çok işletim sistemi POSIX tabanlı iş parçacıklarını desteklemektedir. Ancak POSIX standardını desteklemese de, neredeyse tüm işletim sistemleri iş parçacıklarını desteklemektedir.

Bu deneyde aşağıdaki aynı işi yapan ancak farklı paralelizme sahip programların incelemesi beklenmektedir.

1. Bir ana ve iki ek fonksiyonu olan, tek işlem ve iş parçasıyla çalışan bir program
2. Yukarıdaki programın üç iş parçasıyla çalıştırılması (bir ana iki de alt iş parçacığı)
3. İlk programın üç ayrı işlem ile çalıştırılması

Tüm bu çalışma modları tam olarak aynı işlemi yapmaktadır. Bu modlardan ilk ikisinin programı ekte verilmiştir.

* BLGM 343 dersi için Bahar 2011/2012 döneminde verilen CMPE 343 deneylerinden Gürcü Öz ve Cem Kalyoncu tarafından çevirilmiştir

Deneyler

1. Linux hesabınıza giriş yaparak gerekli dosyaları bölüm sitesinden indiriniz. İndirdiğiniz arşivdeki dosyaları, daha sonra kolayca bulabileceğiniz şekilde bir dizine yerleştiriniz. Eğer kendi bilgisayarlarınızı kullanacaksanız, pthread geliştirme kütüphanelerinin kurulu olduğunda emin olunuz.

2. Tek iş parçasıyla çalışmakta olan programı derleyiniz. Birden fazla kod dosyası içerdiği için bu dosyaları da derlerken belirtmeniz gerekmektedir (not: verilen makefile'ı da kullanabilirsiniz)

```
gcc -o singlethread singlethread.c func1.c func2.c
```

3. Derlediğiniz programı çalıştırarak çıktıları defterinize yazınız. Kaynak kodunu inceleyerek geçen toplam geçen süreyi analiz ediniz.

4. Çoklu iş parçacıklı programı derleyeniz. Bunun için pthread geliştirme kütüphanesi gerekmektedir:

```
gcc -o multithread multithread.c func1.c func2.c -lpthread
```

5. Derlediğiniz programı çalıştırarak sonuçları defterinize yazınız. Kaynak kodu inceleyerek geçen toplam süreyi analiz ediniz.

6. Biri tek iş parcacıklı olan program, diğeri çoklu iş parcacıklı çalışan program için defterinize iki zaman çizelgesi çiziniz. Bu iki programın çalışma sistemleri arasındaki farkı anlatınız.

7. multithread.c dosyasındaki pthread_join komutlarını silerek programı tekrar derleyip çalıştırınız. Sonuçları not aldıktan sonra 5. adımda çalışan programa göre, niye daha kısa sürdüğünü tespit ediniz.

8. multithread.c dosyasındaki pthread_join fonksiyonlarını tekrar yerlerine koyun. for(j=1;...) döngüsünde, sleep sistem çağırısından önce, r1 genel değişkenine 100+j değerini atayın. Daha sonra, func1.c ve func2.c dosyalarındaki printf komutlarına r1 değişkenin değerini yazması için gerekli eklentiyi yapınız. Ardından programı derleyerek çalıştırınız. Bu deneyden anladığınız sonucu belirtiniz.

9. Bir önceki adımda değiştirdiğiniz multithread.c dosyasını örnek olarak kullanarak aşağıdaki işlemleri yapan yeni bir program yazınız

a. iki alt işlemin, hata kontrolleri yapılarak yaratılması

b. birinci alt işlemin fun1, ikinci alt işlemin ise fun2 fonksiyonlarını çalıştırmasının sağlanması

c. Ana işlemin gerekli işlemleri tamamladıktan sonra, alt işlemleri bekleyerek çıkmasının sağlanması.

10. Bir önceki adımda yazdığını programı derleyiniz.

11. Programı çalıştırarak tüm bilgileri not alınız. Sonuçlarınızı tek ve çoklu iş parcacıklı programların sonuçlarıyla karşılaştırınız. Ayrıca, daha önce yaptığınız gibi bir zaman çizelgesi çıkartınız.

12. En son yazmış olduğunuz programda ana işlemin, alt işlemleri beklemesi için yazmış olduğunuz kodu kaldırınız. Programı tekrar çalıştırarak daha önce elde etmiş olduğunuz sonuçla karşılaştırınız.

Kaynakça

1. Nichols, B. et al., *Pthreads Programming*, O'Reiily & Assoc., 1996, pp. 1 - 27.

Sorular

1. İş parçacığı nedir?

2. Programda iş parçacığının kullanım amacı nedir?

3. İşlemlerle iş parçacıkları arasında fark var mıdır? Açıklayınız.

4. Linux çekirdeğinde mevcut olan iş parcacığı standardı nedir?

5. İş parçacıkları Windows sınıfı işletim sistemlerinde mevcut mudur?

6. Bir program başladığında kaç tane iş parçacığına sahip olur?

7. Bir programda nasıl yeni bir iş parçacığı oluşturulur?

8. Bir programdaki ana iş parçacığının iki alt iş parçacığı oluşturduğunu varsayın. Eğer ana iş parçacığının işine, alt iş parçacıkları işlerini tamamladıktan sonra dönmesini istiyorsak ne yapmalıyız?

9. Birden fazla iş parcacığının aynı fonksiyonu çalıştırması mümkün müdür?

10. Eğer ana iş parçacığı, bir veya daha fazla iş parçacığı oluşturup sonlanırsa, bu yeni oluşturulan iş parçacıklarına ne olur? Çalışmaya devam ederler mi? Açıklayınız.

11. Eğer birden fazla iş parçacığı aynı genel değişkeni kullanmaya kalkarsa ne gibi problemler oluşur? Bu durumda ne gibi önlemler alınmalıdır.

Ek: 2. Programların kodları

func1.c

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



void func1(void)
{
    int i;

    for (i = 1; i <10; ++i)
    {
        printf("func1 fonksiyonu bu yazıyı yazıp 4san uyuyor: %d\n", i);
        sleep(4);
    }
    return;

}
```

func2.c

```

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


void func2(void)
{
    int i;

    for (i = 1; i < 8; ++i)
    {
        printf("func2 bu yazıyı yazıp 3san uyuyor: %d\n", i);
        sleep(3);
    }
    return;

}
```

singlethread.c

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>
#include <pthread.h>


/* Diğer dosyalarda bulunan fonksiyonlar */
```

```
void func1(void);
void func2(void);

int r1 = 0, r2 = 0; /* Genel değişkenler, diğer fonksiyonlar
                       tarafından erişilebilir */

int main(void)
{
    int j;
    float delta;
    struct timeval time1, time2;


    printf("Singlethreaded process starts  here...\n");
    gettimeofday(&time1, 0); /* Starting time */


    /* Önce ana işlem çalışıyor */
    for (j=1; j<=4; ++j)
    {
        printf("Ana işlem çalışıyor: %d\n", j);
        sleep(3); /* 3 saniye bekle */
    }

    /* Daha sonra iki fonksiyon da sırayla çağırılıyor */
    func1();
    func2();



    /* Toplam geçen sürenin hesaplanması */
    gettimeofday (&time2, 0);
    delta = (float)((1000000*time2.tv_sec + time2.tv_usec) -
                    (1000000*time1.tv_sec + time1.tv_usec))/1000000;
    printf("Toplam geçen zaman = %f saniye\n", delta);

    printf("İşlem tamamlandı...\n");

    return 0;
}
```

multithread.c

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>
#include <pthread.h>

/* Diğer dosyalarda bulunan fonksiyonlar */
void func1(void);
void func2(void);

int r1 = 0, r2 = 0; /* Genel değişkenler, diğer fonksiyonlar
                       tarafından erişilebilir */

int main(void)
{
    pthread_t       td1, td2; /* iş parçacık tanımlayıcıları */
    int p;
    int j;
    float delta;
    struct timeval time1, time2;
```

```
    printf("Ana işlem ana ,ş parcacığı olarak başlıyor...\n");
    gettimeofday(&time1, 0); /* Başlama zamanı */

    /* İki alt izgenin oluşturulması */
    p = pthread_create(&td1,
                       NULL,
                       (void *)func1,
                       NULL);

    if (p != 0) {
        perror("İş parcacığı 1 oluşurulurken hata");
        exit(1);
    }

    p = pthread_create(&td2,
                       NULL,
                       (void *)func2,
                       NULL);
    if (p != 0) {
        perror("İş parcacığı 2 oluşurulurken hata");
        exit(1);
    }

    /* Ana işlem birşeyler yapıyor */
    for (j=1; j<=4; ++j)
    {
        printf("Ana işlem çalışıyor: %d\n", j);
        sleep(3); /* 3 saniye uyu */
    }


    pthread_join(td1, NULL); /* 1. izgeyi bekle */
    pthread_join(td2, NULL); /* 2. izgeyi bekle */

    /* Toplam geçen sürenin hesaplanması */
    gettimeofday (&time2, 0);
    delta = (float)((1000000*time2.tv_sec + time2.tv_usec) -
                    (1000000*time1.tv_sec + time1.tv_usec))/1000000;
    printf("Toplam geçen zaman = %f saniye\n", delta);

    printf("İşlem tamamlandı...\n");

    return 0;
}
```