SİSTEM PROGRAMLAMA

Deney#4 : İş-parçacıklarını (Thread) Anlama

GİRİŞ

Bu deneyde amaç thread ve thread-programlamayi öğrenmektir. Thread' lere yüklediğimiz işler **func1.c** and **func2.c** olarak harici olarak verilmiştir. Programların başlangıç halleri zip içeriğinde (ayrıca appendiks te) verilmiştir. Verilen Kodlar da hatalar çıkarsa düzeltiniz(bazı yazım hataları çıkabilir). Geliştirdiğiniz kodları, sorulara olan cevapları ve ekran görüntülerini **bir pdf raporu olarak hazırlayıp, sisteme yükleyiniz.**

Deneyler

1. Linux home dizininizde **Lab3a** isminde bir dizin açın ve zip dosyasinda verilen (appendikste türkçe versionları var) **singlethread.c**, **multithread.c , func1.c**, **func2.c** dosyalarını oraya kopyalayın.
2. Tek-threadli programı (a) **singlethread.c**, **func1.c** ve **func2.c derleyin ve birleştirin**. Bu amaçla aşağıdaki satırı kullanabilirsiniz (derleyicinize göre farklı komutlar vermek zorunda kalabilirsiniz).

   cc -o singlethread singlethread.c func1.c func2.c

   Hatırlatma : singlethread programı 3 tane dosyaya bağımlı olduğu için bağımlı listesini (singlethread.c , func1.c ve func2.c) derleme satırına ekledik.
3. Bu programı (**./singlethread**) çalıştırıp inceleyin. Kaynak kodları analiz edip **singlethread** programı ne yapıyor birkaç cümle ile açıklayın.

4. Benzer şekilde çok-threadli programı aşağıdaki gibi çalıştırıp benzer şekilde inceleyin

   cc -o multithread multithread.c func1.c func2.c /usr/lib/libpthread.a

   multithread programı ne yapıyor birkaç cümle ile açıklayın. İki programın çalışma süreleri hakkında yorum yapınız ve sebebini anlatınız.
   Hatırlatma: Birleştirme esnasında thread kütüphanesine bağımlı kod için en sonda bağımlı olduğu kütüphaneyi yazdık. Ama sizde belki başka bir kütüphane isteyebilir bu durumda **libpthread.so** veya **libpthread.so.1** deneyin.

5. **multithread.c**, programında iki tane **pthread_join ifadesini kaldırın**, derleyip çalıştırın. Sonucu anlamaya çalışın. Neden programın toplam çalışma zamanı orjinaline gore daha az oldu.

6. **multithread.c** kodunu değiştirerek yeni bir program yazın. Programın adı **multiprocess.c olsun**. Bu programda func1 ve func2 işini parent ve child prosese yükleyin ve iki iş için geçen toplam çalışma süresini bulun. Parent process, işini bitirdikten sonra child process için beklesin ve sonlansın. Programı çalıştırarak çok-processli bir program ile aynı işi yapan çok-threadli bir program arasındaki farkı benzer şekilde yorumlayın.

7. **multithread.c** kodunu değiştirerek yeni bir program yazınız. Programın adı **multithreadparam.c olsun.** Bu programda **r1 ve r2** global değişkenlerin adreslerini parametre olarak **func1** ve **func2** fonksiyonlarına geçiriniz. Tüm threadlerin (ana thread de dahil) sleepten önce **r1** ve sleepten sonra **r2** değerini bir arttırmasını ve son olarak ana threadin **r1** ve **r2** değerlerini ekrana yazmasını sağlayınız. Sonuç değerlerde bir yanlışlık var mı? Yorum yapınız.

APPENDIX
(Source texts of functions)

```
/*******************************************************
 *
 *
 * Tek-process (sadece ana threadli)
 *
 *  Dosya: singlethread.c
 *
 *
 *******************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>
#include <pthread.h>

/* Fonk. Tanımları */
 void func1(void);
 void func2(void);

int r1 = 0, r2 = 0; /* Global variables, gerekli olduğunda fonksiyonlar
kullanacak*/

int
main(void)
{
  pthread_t td1, td2; /* 2 child thread için tanımlayıcılar*/
  int p; /* burada kullanılmıyor */
  int j;
  float delta;
  struct timeval time1, time2;

  printf("Tek-Threadli process başlıyor...\n");
   gettimeofday(&time1, 0); /* Başlama zamanı */

/* 2 child thread yaratılması
  p = pthread_create(&td1,
             NULL,
                 (void *)func1,
             NULL);

  if (p != 0) {perror("Thread 1 creation problem"); exit(1);}

  p = pthread_create(&td2,
                 NULL,
                 (void *)func2,
                 NULL);
  if (p != 0) {perror("Thread 2 creation problem"); exit(1);}
*/

/* şimdi tek process, tek ana threadi ile bir şeyler yapıyor */
  for (j=1; j<=4; ++j)
    {
    printf("Processin ana fonksiyonu: %d\n", j);
    sleep(3); /* 3 saniye uyu */
```

```
    }

/* Tüm processler sıra ile iki fonksiyonu çağırıyorlar */
  func1();
  func2();

/* Processin toplam çalışma zamanlarını hesaplama */
   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("Ana thread sonlandı...\n");

   return 0;
}

/*********************************************************
 *
 *
 * Threadlerin kullanımı : basit bir örnek
 *
 * Dosya: multithread.c
 *
 *
 *********************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>
#include <pthread.h>

/* Fonk. Tanımları */
 void func1(void);
 void func2(void);

int r1 = 0, r2 = 0; /* Global variables, gerekli olduğunda fonksiyonlar
kullanacak*/

int
main(void)
{
  pthread_t       td1, td2; /* 2 child thread için tanımlayıcılar*/
  int p;
  int j;
  float delta;
  struct timeval time1, time2;

  printf("Çok-threadli process ana thread olarak başlıyor...\n");
   gettimeofday(&time1, 0); /* Başlama zamanı */

/* 2 child thread yaratılması */
  p = pthread_create(&td1,
             NULL,
                 (void *)func1,
             NULL);
  if (p != 0) {perror("Thread 1 creation problem"); exit(1);}
```

```
  p = pthread_create(&td2,
                NULL,
                (void *)func2,
                NULL);
  if (p != 0) {perror("Thread 2 creation problem"); exit(1);}

/* Şimdi ana thread bir şeyler yapıyor */
  for (j=1; j<=4; ++j)
    {
    printf("Ana thread çalışıyor: %d\n", j);
    sleep(3); /* Sleep 3 seconds */
    }

   pthread_join(td1, NULL); /* child thread1 in sonlanmasını bekle */
   pthread_join(td2, NULL); /* child thread2 in sonlanmasını bekle */

/* Çok-threadli processin toplam çalışma zamanlarını hesaplama */
   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("Ana thread sonlandı...\n");

   return 0;
}


/*********************************************************
 *
 * Dosya: func1.c
 *  Periyodik olarak uyuyan basit bir fonksiyon
 *
 *
 *
 *********************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

void func1(void)
{
  int i;

  for (i = 1; i <10; ++i)
    {
    printf("Fonksyion func1 yazar ve 4 saniye uyur: %d \n", i);
    sleep(4);
    }
  return;

}


/*********************************************************
```

```
 *
 * Dosya: func2.c
 * Periyodik olarak uyuyan basit bir fonksiyon
 *
 *
 *
 ********************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

void func2(void)
{
  int i;

  for (i = 1; i < 8; ++i)
    {
    printf("Fonksyion func2 yazar ve 3 saniye uyur: %d \n", i);
    sleep(3);
    }
  return;

}
```