YZM 3102
İşletim Sistemleri

Yrd. Doç. Dr. Deniz KILINÇ
Celal Bayar Üniversitesi
Hasan Ferdi Turgutlu Teknoloji Fakültesi
Yazılım Mühendisliği

BÖLÜM – 6.2

Bu bölümde,

- Mutex ve Semafor Farkları
- Klasik Senkronizasyon Problemleri ve Çözümleri
  - Producer/Consumer
  - Reader/Writer
  - Dining Philosopher
- Monitörler

konularına değinilecektir.

Mutex ve Semafor Farkları

- Semaphore mutex'in aksine senkronize erişimi tamamen bekletmek değil (***busy waiting***), sınırlandırmak için kullanılır.
- Örneğin, sınırlı bir kaynağımız var, aynı anda 5 thread'in erişimi sistem için sorun olmuyor ama 5'ten fazla thread tarafından o kaynak kullanılmak istendiğinde performans sorunu yaşanıyor.
- Dolayısıyla kodun o bölümünü bütün thread'lerin erişimine açmak istemiyoruz. Bu durumda **sayaçlı bir lock mekanizmasına** ihtiyacımız var. İşte o mekanizmanın adı Semaphore.

Mutex ve Semafor Farkları (devam…)

- Mutex'i sadece kilitleyen thread/proses kilidi kaldırabiliyorken semaphore üzerindeki kilidi herhangi bir thread/proses kaldırabilir.
- Diğer bir deyişle **semaphore'un sahibi yoktur**.
- Semaphore *lock etmiş* **herhangi bir thread** tarafından da *unlock edilebilir*.
- Mutex ise *sadece lock etmiş thread tarafından unlock edilebilir*.

.NET Semafor Örneği

```
static void Main(string[] args)
{
    var threadApple = new Thread(() => WriteMessageMutex("Apple"));
    threadApple.Start();

    var threadBanana = new Thread(() => WriteMessageMutex("Banana"));
    threadBanana.Start();
}

private static Mutex m = new Mutex();
2 references
static void WriteMessageMutex(string message)
{
    try
    {
        m.WaitOne();
        for (int i = 0; i < 100; i++)
        {
            Thread.Sleep(100);
            Console.Write(message);
        }
    }
    finally
    {
        m.ReleaseMutex();
    }
}
```

Busy Waiting

Kaynağımıza sadece iki thread/prosesin erişimine izin verecek kilit mekanizması oluşturalım.

.NET Semafor Örneği (devam…)

```
static void Main(string[] args)
{
    var threadApple = new Thread(() => WriteMessageSem("Apple"));
    threadApple.Start();

    var threadBanana = new Thread(() => WriteMessageSem("Banana"));
    threadBanana.Start();
}
private static Semaphore sem = new Semaphore(2, 2, "SemTest");
2 references
static void WriteMessageSem(string message)
{
    try
    {
        sem.WaitOne();
        for (int i = 0; i < 100; i++)
        {
            Thread.Sleep(100);
            Console.Write(message);
        }
    }
    finally
    {
        sem.Release(1);
    }
}
```

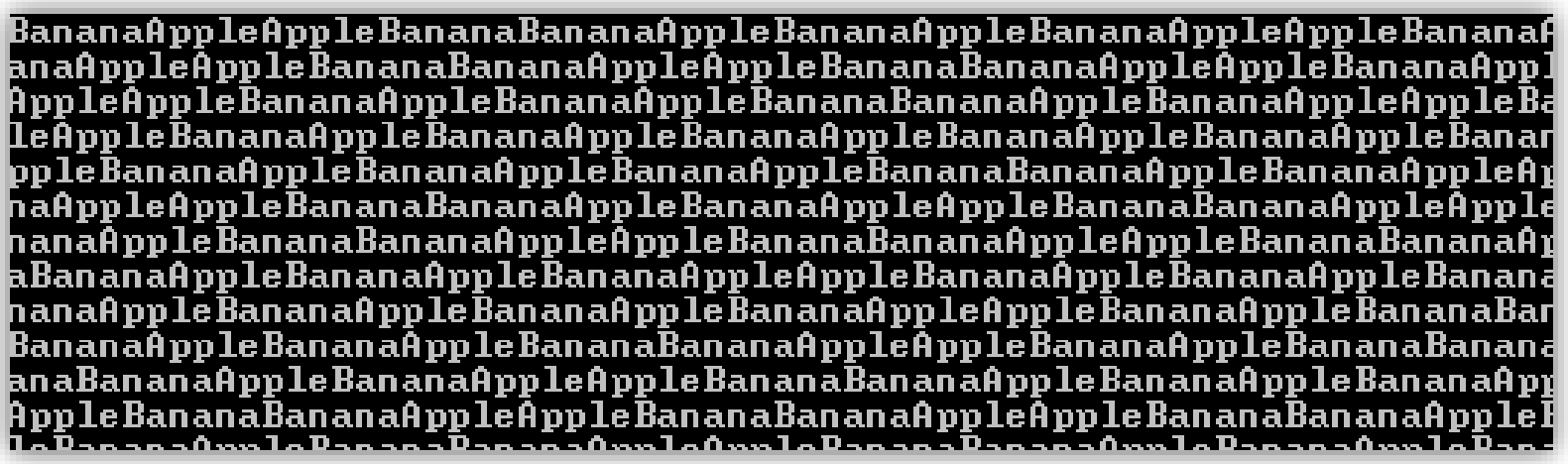
- **Başlangıç değeri: 2**
- **Maks. thread erişim değeri: 2**
- **Semafor adı: «Semtest»**

- **Semafor değ. 1 azalt**

- **Semafor değ. 1 arttır**

.NET Semafor Örneği (devam…)

Sonuç



Klasik Senkronizasyon Problemleri

- Klasikleşmiş üç tane senkronizasyon problemi üzerinde çalışılacaktır.
  1. The Producer-Consumer Problem
  2. The Readers–Writers Problem
  3. The Dining-Philosophers Problem

Producer-Consumer Problemi

- **Semafor** kullanarak, problemi sınırlı buffer durumunda çözmeye çalışalım
- Üretici ve tüketici birer proses olarak yaratılmaktadır.
- Burada 3 önemli kısıtlama vardır:
  - Buffer'a aynı anda sadece bir proses erişebilir (mutual exclusion). Bu amaçla **mutex** isimli **ikili semafor** kullanılacaktır.
  - Eğer Buffer boş ise tüketici, üreticinin Buffer'a veri girmesini bekler. Üretici ve tüketici **empty** isimli semaforu **senkronizasyon amacıyla** kullanacaklardır.
  - Eğer Buffer dolu ise üretici, tüketicin Buffer'dan veri almasını bekler. Üretici ve tüketici **full** isimli **semaforu senkronizasyon amacıyla** kullanacaklardır.

Producer-Consumer Problemi (devam…)

```
# define N 100                          / * Buffer'daki eleman sayısı */
# define TRUE 1
typedef int semaphore;                  / * Semaforlar int bir tür olarak tanımlanıyor */
semaphore mutex = 1;                    / * kritik kısmı karşılıklı dışlamak */
semaphore empty = N;                    / * Buffer'daki boş yer sayısı */
semaphore full = 0;                     / * Buffer'daki dolu yer sayısı */
producer()
{
     int item;                          / * yerel değişken */
     while (TRUE) {
          produce_item(&item);          / * Buffer'a konulacak veri oluşturuluyor */
          wait(empty);                  / * Buffer dolu ise bekle yoksa bos yer sayısını 1 azalt */
          wait(mutex);                  / * Kritik kısma girebilmek için vize al */
          enter_item(item);             / * Veriyi Buffer'a gir (Kritik Kısım) */
          signal(mutex);                / * Kritik Kısımdan çıktığını belirt */
          signal(full);                 / * Bekleyen tüketici varsa uyandır yoksa Buffer'daki */
                                        / * dolu yer sayısını 1 arttır */
     }

}
consumer(){
     int item;                          / * yerel değişken */
     while (TRUE) {
          wait(full);                   / * Buffer boş ise bekle yoksa dolu yer sayısını 1 azalt */
          wait(mutex);                  / * Kritik kısma girebilmek için vize al */
          remove_item(&item);           / * Veriyi Buffer'dan al (Kritik Kısım) */
          signal(mutex);                / * Kritik Kısımdan çıktığını belirt */
          signal(empty);                / * Bekleyen üretici varsa uyandır yoksa Buffer'daki */
                                        / * dolu boş sayısını 1 arttır */
          consume_item(item);           / * Buffer'dan alınan veriyi kullan */
     }
}
```

Producer-Consumer Problemi (devam…)

```
Paylaşılan: semaforlar: mutex, empty, full;
mutex = 1;  /* for mutual exclusion*/
empty = N; /* number empty buf entries */
full = 0;      /* number full buf entries */
```

Producer

```
do {
    . . .
    // produce an item in nextp
    . . .
    wait(empty);
    wait(mutex);
    . . .
    // add nextp to buffer
    . . .
    signal(mutex);
    signal(full);
} while (true);
```

Consumer

```
do {
    wait(full);
    wait(mutex);
    . . .
    // remove item to nextc
    . . .
    signal(mutex);
    signal(empty);
    . . .
    // consume item in nextc
    . . .
} while (true);
```

Prod.-Cons. Problemi C Çözümü

Global Tanımlar

```
#include<stdio.h>
#include<semaphore.h>
#include<pthread.h>
#include<stdlib.h>
#define buffersize 10

pthread_mutex_t mutex;
pthread_t tidP[20],tidC[20];
sem_t full,empty;
int counter;
int buffer[buffersize];
```

Prod.-Cons. Problemi C Çözümü (devam…)

main()

```
void initialize()
{
   pthread_mutex_init(&mutex,NULL);
   sem_init(&full,1,0);
   sem_init(&empty,1,buffersize);
   counter=0;
}

int main()
{
   int n1,n2,i;
   initialize();
   printf("\nProducer sayisi gir: ");
   scanf("%d",&n1);
   printf("\nConsumer sayisi gir: ");
   scanf("%d",&n2);
   for(i=0;i<n1;i++)
        pthread_create(&tidP[i],NULL,producer,NULL);
   for(i=0;i<n2;i++)
        pthread_create(&tidC[i],NULL,consumer,NULL);
   for(i=0;i<n1;i++)
        pthread_join(tidP[i],NULL);
   for(i=0;i<n2;i++)
        pthread_join(tidC[i],NULL);
   sleep(5);
   exit(0);
}
```

Prod.-Cons. Problemi C Çözümü (devam…)

Producer

```
void write(int item)
{
   buffer[counter++]=item;
}

void * producer (void * param)
{
   int item;
   item=rand()%5;
   // Buffer dolu ise bekle yoksa bos yer sayısını 1 azalt
   sem_wait(&empty);
   // Kritik bölge kilitle
   pthread_mutex_lock(&mutex);
   printf("\nProducer üretti, item: %d\n",item);
   // Buffer'a item ekle
   write(item);
   // Kritik bölge kilidi aç
   pthread_mutex_unlock(&mutex);
   //Bekleyen tüketici varsa uyandır, yoksa Buffer'daki dolu yer sayısını 1 arttır
   sem_post(&full);
}
```

Prod.-Cons. Problemi C Çözümü (devam…)

Consumer

```
int read()
{
   return(buffer[--counter]);
}

void * consumer (void * param)
{
   int item;
   // Buffer boş ise bekle yoksa dolu yer sayısını 1 azalt
   sem_wait(&full);
   // Kritik bölge kilitle
   pthread_mutex_lock(&mutex);
   // Buffer'dan item oku
   item=read();
   printf("\nConsumer tüketti, item: %d\n",item);
   // Kritik bölge kilidi aç
   pthread_mutex_unlock(&mutex);
   // Bekleyen üretici varsa uyandır yoksa Buffer'daki boş yer sayısını 1 arttır
   sem_post(&empty);
}
```

Readers-Writers Problemi

- Bir veri tabanının **birden fazla proses** tarafından ***paylaşıldığını*** varsayalım.
- Bazı prosesler, verileri sadece **okumak isterken**,
- Diğerleri **veriler üzerinde değişiklik** yapmak isteyeceklerdir. (***Readers ve Writes***)
- İki işlem, **aynı anda okuma yapmak isterse problem oluşmaz** fakat, bir writer, reader ya da writer ile *aynı anda veriye erişir* ise **problem oluşabilir**.
- Bu problemin önüne geçmek için **writer prosesler** erişimde ***ayrıcalıklı olmalıdır***.
- Bu senkronizasyon problemi **Readers/Writers** olarak adlandırılır.

Readers-Writers Problemi (devam…)

- Readers/Writers Problemi (**2 problem**):
  - **Problem 1:** reader proses beklemez iken, writer proses, veri üzerinde işlem yapmak için izin istemek zorundadır.
  - **Problem 2:** Eğer bir writer veriye erişmek için bekliyorsa, hiçbir yeni reader okuma işlemine başlayamaz.
- İki şekilde de reader ve writer işlemler beklemek (starvation) zorunda kalacaklardır.
- **İlk problemde** writer prosesler gecikebilir.
- **İkinci problemde** reader prosesler gecikebilir.

Readers-Writers Problemi (devam…)

Writer Proses

```
semaphore rw_mutex = 1;
semaphore mutex = 1;
int read_count = 0;
```

```
do {
   wait(rw_mutex);
      . . .
   /* writing is performed */
      . . .
   signal(rw_mutex);
} while (true);
```

Figure 5.11 The structure of a writer process.

Readers-Writers Problemi (devam…)

Reader Proses

```
do
{
   wait(mutex)  : reader mutex üzerinde bekliyor
   read_count++: okuma yapmak isteyen proc sayısı
   if (read_count==1)           :ilk reader ise
         wait(rw_mutex);        :1 reader bekliyor
   signal(mutex)
   ......
   //okuma işlemini yap
   .......
   wait(mutex)  : n-1 reader mutex üzerinde bekliyor
   read_count--: 1 reader okuma işlemini bitirdi

   if (read_count==0 )          :son reader ise
         signal(rw_mutex)
   signal(mutex)
}while (true);
```

Readers-Writers Problemi C Çözümü

Global Tanımlar

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

sem_t readCountAccess;
sem_t databaseAccess;
int readCount=0;

void *Reader(void *arg);
void *Writer(void *arg);
```

Readers-Writers Problemi C Çözümü

main()

```
int main()
{
    int i=0;
    int NumberofReaderThread=0;
    int NumberofWriterThread;
    sem_init(&readCountAccess,0,1);
    sem_init(&databaseAccess,0,1);

    pthread_t Readers_thr[100],Writer_thr[100];
    printf("\nReader thread sayisi gir(MAX 10): ");
    scanf("%d",&NumberofReaderThread);
    printf("\nWriter thread sayisi gir(MAX 10): ");
    scanf("%d",&NumberofWriterThread);

    for(i=0;i<NumberofReaderThread;i++)
         pthread_create(&Readers_thr[i],NULL,Reader,(void *)i);
    for(i=0;i<NumberofWriterThread;i++)
         pthread_create(&Writer_thr[i],NULL,Writer,(void *)i);
    for(i=0;i<NumberofWriterThread;i++)
         pthread_join(Writer_thr[i],NULL);
    for(i=0;i<NumberofReaderThread;i++)
         pthread_join(Readers_thr[i],NULL);
    sem_destroy(&databaseAccess);
    sem_destroy(&readCountAccess);
    return 0;
}
```

Readers-Writers Problemi C Çözümü

Writer Proses

```
void * Writer(void *arg)
{
   sleep(1);
   int temp=(int)arg;
   printf("\nWriter %d degisiklik yapmak icin veritabanina erismeye calisiyor...",temp);
   sem_wait(&databaseAccess);
   printf("\nWriter %d veritabanina yaziyor...",temp);
   printf("\nWriter %d veritabanindan ayriliyor...");
   sem_post(&databaseAccess);
}
```

Readers-Writers Problemi C Çözümü

Reader Proses

```
void *Reader(void *arg)
{
    sleep(1);
    int temp=(int)arg;
    printf("\nReader %d veri okumak icin veritabanina erismeye calisiyor...",temp);
    sem_wait(&readCountAccess);
    //okuma yapmak isteyen proses sayısı
    readCount++;
    if(readCount==1) //ilk reader ise
    {
         sem_wait(&databaseAccess);
         printf("\nReader %d veriyi okuyor...",temp);
    }
    sem_post(&readCountAccess); //n-1 reader mutex üzerinde bekliyor
    sem_wait(&readCountAccess);
    readCount--; //1 reader okuma işlemini bitirdi
    if(readCount==0) //son reader ise
    {
         printf("\nReader %d veritabanindan ayriliyor...",temp);
         sem_post(&databaseAccess);
    }
    sem_post(&readCountAccess);
}
```

The Dining-Philosophers Problem

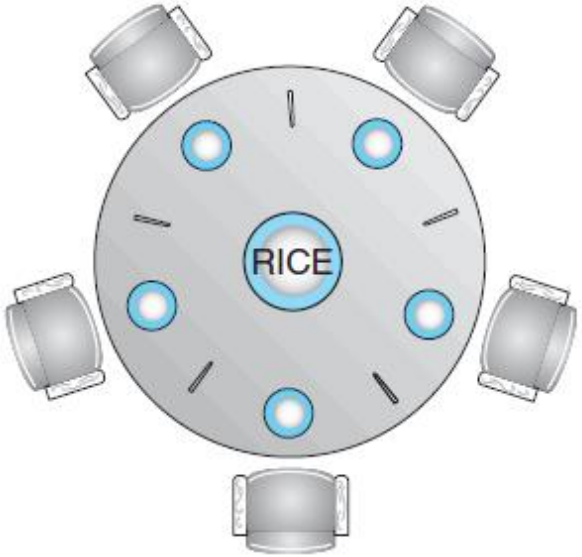


Figure 5.13 The situation of the dining philosophers.

- 5 filozof bulunmaktadır. İşleri **düşünmek** ve **pirinç yemektir**.
- 5 sandalye , 5 tabak ve 5 çubuk bulunmaktadır.
- 1 filozof **2 çubukla** *yemek yiyebilir*.
- Filozoflar düşünürken **birbirlerinden etkilenmezler**.

The Dining-Philosophers Problem (devam…)

- 1 filozof yerken, **en az** bir *komşusunun* çubuğunu kullanır. O arada **komşusu** düşünmek zorundadır.
- İki çubuğu **ele geçiren filozof** *ara vermeden* **yemeğini yer**, **çubukları bırakır** ve düşünmeye devam eder.
- Basit çözümde **her çubuk** semafor ile temsil edilir.
- Bir filozof **çubuk üzerinde** wait operasyonunu işleterek çubuğu yani semaforu *ele geçirir*.
- Aynı şekilde **çubukları signal operasyonu** ile bırakır.
- Hepsi **aynı anda** *sol ellerine sol taraftaki çubukları alırlarsa* hepsi **sonsuza dek aç kalır**. Çözüm için :
  - Aynı anda en fazla 2 filozofun masaya oturmasına izin verilebilir .
  - Bir filozofun yemek yemeye başlamasına sadece her iki çubuk da boş mu diye bakılır. Boşsa izin verilebilir.
  - Tek filozoflara yemek yedirilir. Çiftlere yedirilmez gibi çözümler türetilebilir.

The Dining-Philosophers Problem (devam…)

Senkronizasyon olmadan genel sözde kod….

```
#define N 5                                     /* number of philosophers */

void philosopher(int i)                         /* i: philosopher number, from 0 to 4 */
{
    while (TRUE) {
        think( );                               /* philosopher is thinking */
        take_fork(i);                           /* take left fork */
        take_fork((i+1) % N);                   /* take right fork; % is modulo operator */
        eat( );                                 /* yum-yum, spaghetti */
        put_fork(i);                            /* put left fork back on the table */
        put_fork((i+1) % N);                    /* put right fork back on the table */
    }
}
```

The Dining-Philosophers Problem (devam…)

Semafor çözümü….

```
#define N                5                  /* number of philosophers */
#define LEFT             (i+N–1)%N          /* number of i's left neighbor */
#define RIGHT            (i+1)%N            /* number of i's right neighbor */
#define THINKING         0                  /* philosopher is thinking */
#define HUNGRY           1                  /* philosopher is trying to get forks */
#define EATING           2                  /* philosopher is eating */
typedef int semaphore;                      /* semaphores are a special kind of int */
int state[N];                               /* array to keep track of everyone's state */
semaphore mutex = 1;                        /* mutual exclusion for critical regions */
semaphore s[N];                             /* one semaphore per philosopher */

void philosopher(int i)                     /* i: philosopher number, from 0 to N–1 */
{
      while (TRUE) {                        /* repeat forever */
            think( );                       /* philosopher is thinking */
            take_forks(i);                  /* acquire two forks or block */
            eat( );                         /* yum-yum, spaghetti */
            put_forks(i);                   /* put both forks back on table */
      }
}
```

The Dining-Philosophers Problem (devam…)

Semafor çözümü….

```
void take_forks(int i)                  /* i: philosopher number, from 0 to N–1 */
{
        down(&mutex);                   /* enter critical region */
        state[i] = HUNGRY;              /* record fact that philosopher i is hungry */
        test(i);                        /* try to acquire 2 forks */
        up(&mutex);                     /* exit critical region */
        down(&s[i]);                    /* block if forks were not acquired */
}

void put_forks(i)                       /* i: philosopher number, from 0 to N–1 */
{
        down(&mutex);                   /* enter critical region */
        state[i] = THINKING;            /* philosopher has finished eating */
        test(LEFT);                     /* see if left neighbor can now eat */
        test(RIGHT);                    /* see if right neighbor can now eat */
        up(&mutex);                     /* exit critical region */
}

void test(i)                            /* i: philosopher number, from 0 to N–1 */
{
        if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING) {
                state[i] = EATING;
                up(&s[i]);
        }
}
```

The Dining-Philosophers C Çözümü

Global Tanımlar

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

typedef struct {
  int position;
  int count;
  sem_t *forks;
  sem_t *lock;
} params_t;

void initialize_semaphores(sem_t *lock, sem_t *forks, int num_forks);
void run_all_threads(pthread_t *threads, sem_t *forks, sem_t *lock, int

void *philosopher(void *params);
void think(int position);
void eat(int position);
```

The Dining-Philosophers C Çözümü

Hazırlık

```c
int main(int argc, char *args[])
{
  int num_philosophers = 5;
  sem_t lock;
  sem_t forks[num_philosophers];
  pthread_t philosophers[num_philosophers];

  initialize_semaphores(&lock, forks, num_philosophers);
  run_all_threads(philosophers, forks, &lock, num_philosophers);
  pthread_exit(NULL);
}

void initialize_semaphores(sem_t *lock, sem_t *forks, int num_forks)
{
  int i;
  for(i = 0; i < num_forks; i++)
    sem_init(&forks[i], 0, 1);
  sem_init(lock, 0, num_forks - 1);
}

void run_all_threads(pthread_t *threads, sem_t *forks, sem_t *lock, int num_philos
{
  int i;
  for(i = 0; i < num_philosophers; i++) {
    params_t *arg = malloc(sizeof(params_t));
    arg->position = i;
    arg->count = num_philosophers;
    arg->lock = lock;
    arg->forks = forks;

    pthread_create(&threads[i], NULL, philosopher, (void *)arg);
  }
}
```

The Dining-Philosophers C Çözümü

Filizoflar

```
void *philosopher(void *params)
{
  int i;
  params_t self = *(params_t *)params;
  for(i = 0; i < 3; i++) {
    think(self.position);
    sem_wait(self.lock);
    sem_wait(&self.forks[self.position]);
    sem_wait(&self.forks[(self.position + 1) % self.count]);
    eat(self.position);
    sem_post(&self.forks[self.position]);
    sem_post(&self.forks[(self.position + 1) % self.count]);
    sem_post(self.lock);
  }
  think(self.position);
  pthread_exit(NULL);
}

void think(int position)
{
  printf("Philosopher %d thinking...\n", position);
}

void eat(int position)
{
  printf("Philosopher %d eating...\n", position);
}
```

Monitörler

- İzleyici Programlardır.
- Senkronizasyonda **doğru programların yazılmasını** ***kolaylaştırmak adına*** Brinch Hansen (1973) ve Hoare (1974) **yüksek seviyeli senkronizasyon yapıları** geliştirmişlerdir.
- Monitör programları paylaşılan nesneye ulaşmada meydana gelebilecek problemleri ortadan kaldırmaya yönelik geliştirilmiştir.
- Paylaşılan nesneyi oluşturan veri,
  - Bu nesneye ulaşmak için kullanılan **bir grup prosedür**,
  - Nesneyi **başlangıç konumuna getiren** bir program parçasından oluşmaktadır.

Monitörler (devam…)

- Semafor ile senkronizasyon çözümleri, zamana bağlı hatalar nedeni ile yetersiz olabilir.
- Bir prosesin wait() ve signal() sırasını değiştirmesinden dolayı birden fazla proses **aynı anda kritik kısımlarına girerler**.

```
signal(mutex);
    ...
  critical section
    ...
wait(mutex);
```

- Bir proses signal() ve wait() yer değiştirir ise **deadlock oluşur**.

```
wait(mutex);
    ...
  critical section
    ...
wait(mutex);
```

Monitörler (devam…)

- Bu hataların önüne geçmek için *yüksek seviyeli programlama yapıları* geliştirilmiştir.
- Örneğin monitör kilidi herhangi bir C++ sınıfından gerçekleştirilecek bir yordam çağrımıyla otomatik olarak elde edilir.
- Monitor bir kilittir, ve 0 veya daha fazla sayıda koşul değişkeninin ortaklaşa kullanılan veri veya kaynaklara **paralel erişimi** koordine etmek maksadıyla kullanılmaktadır.
- **Semaforlar** monitörlere nazaran **sisteme daha yakın yapılardır** ve karşılıklı dışlama ve senkronizasyon amaçlı çift yönlü kullanılırlar. Bu sebeple semaforlarla **kod yazmak daha fazla detayla uğraşmayı gerektirir**. Yazılan kodun okunması ve doğru olarak ne yapıldığının anlaşılması daha güçtür.

Monitörler (devam…)

- Monitörde **herhangi bir zamanda** sadece tek bir proses *aktif olabilir*.
- Bu da monitörleri, **karşılıklı dışlama probleminde** etkin kılar.
- Monitör programları:
  - wait ve signal işlemleri yanlış gerçekleştirildiğinde oluşan *sistem kilitlenmelerini engeller*.
  - Semaforlarla **işlemler arası kombinasyonu** sağlarlar.
  - wait-signal işlemleri arasındaki kombinasyonu sağlarlar.

Dining Philosophers Monitor Çözümü

```
Monitor dp
{
        enum{thinking, hungry, eating} state[5];
        condition self[5];
        void pickup (int i) {
                state[i]:=hungry;      : aç olduğunu belirtir.
                test(i);
                if (state[i] != eating) : yemek yemiyor ise
                        self[i].wait(); : bekleme durumuna geçer
        }
        void putdown(int i) {
                state[i] = thinking;   :düşünmeye başlar
                test((i+4)%5);
                test((i+1)%5);
        }
        void test (int i) {
                if ((state[(i+4)%5] != eating) &&   : i. Filozof açsa ve komşuları yemiyorsa
                  (state[i]= = hungry) &&
                  (state[(i+1)%5]) != eating)) {
                        state[i]= eating;                :yemek yiyebilir
                        self[i].signal();                : yedikten sonra kendini signal eder
                }
        }
        void init () {
                for (int i = 0; i<5; i+ +)
                        state[i] = thinking;
        }

}
dp.pickup(i);
....
eat
....
dp.putdown (i);
```

Dining Philosophers .NET Monitor Çözümü

Filozof Sınıfı Bölüm 1

```
public class Filozof
{
    private int n;
    private int thinkDelay;
    private int eatDelay;
    private int left, right;
    private FilizofWork philofork;
    5 references
    public Filozof(int n, int thinkDelay, int eatDelay, FilizofWork philofork)
    {
        this.n = n;
        this.thinkDelay = thinkDelay;
        this.eatDelay = eatDelay;
        this.philofork = philofork;
        left = n == 0 ? 4 : n - 1;
        right = (n + 1) % 5;
        new Thread(new ThreadStart(Run)).Start();
    }
```

Dining Philosophers .NET Monitor Çözümü

Filozof Sınıfı Bölüm 2

```
public void Run()
{
    for (; ; )
    {
        try
        {
            Console.WriteLine("Philosopher " + n + " is thinking...");
            Thread.Sleep(thinkDelay);
            philofork.Get(left, right);
            Console.WriteLine("Philosopher " + n + " is eating...");
            Console.ReadLine();
            Thread.Sleep(eatDelay);
            philofork.Put(left, right);
        }
        catch
        {
            return;
        }
    }
}
```

Dining Philosophers .NET Monitor Çözümü

FilozofWork Sınıfı

```
public class FilizofWork
{
    bool[] fork = new bool[5];
    1 reference
    public void Get(int left, int right)
    {
        lock (this)
        {
            while (fork[left] || fork[right])
                Monitor.Wait(this);
            fork[left] = true;
            fork[right] = true;
        }
    }
    1 reference
    public void Put(int left, int right)
    {
        lock (this)
        {
            fork[left] = false;
            fork[right] = false;
            Monitor.PulseAll(this);
        }
    }
}
```

Dining Philosophers .NET Monitor Çözümü

Filozof Test

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        FilizofWork philofork = new FilizofWork();
        new Filozof(0, 10, 50, philofork);
        new Filozof(1, 20, 40, philofork);
        new Filozof(2, 30, 30, philofork);
        new Filozof(3, 40, 20, philofork);
        new Filozof(4, 50, 10, philofork);
    }
}
```

İYİ ÇALIŞMALAR…

Yararlanılan Kaynaklar

- **Ders Kitabı:**
  - Operating System Concepts, Ninth Edition, Abraham Silberschatz, Peter Bear Galvin, Greg Gagne
- **Yardımcı Okumalar:**
  - İşletim Sistemleri, Ali Saatçi
  - Şirin Karadeniz, Ders Notları
  - İbrahim Türkoğlu, Ders Notları
- **M. Ali Akcayol, Gazi Üniversitesi Bilgisayar Mühendisliği Bölümü**
- **http://www.albahari.com/threading/**