Parallella Processer 3/9

Klassiska problem

Översikt

- Klassiska problem
  - Dining Philosophers
  - Producer/Consumer
  - Readers/Writers
- Lösningar till dessa

Klassiska problem

- ”Typproblem” i PP
  - Samarbete
  - Tävlan om resurser
  - Global ordning, etc
- Kritiska sektionsproblemet ett exempel
- Fungerande lösningar finns
- Tricket är att känna igen problemen

Dining Philosophers

- Filosofer
  - En typ av process
  - Processerna växlar mellan kritisk sektion (äta) och icke-kritisk (tänka)
- Runt bord
  - Alla har två grannar
- Delar gaffel med granne
  - Båda gafflarna behövs för att äta

Dining philosohers, forts.

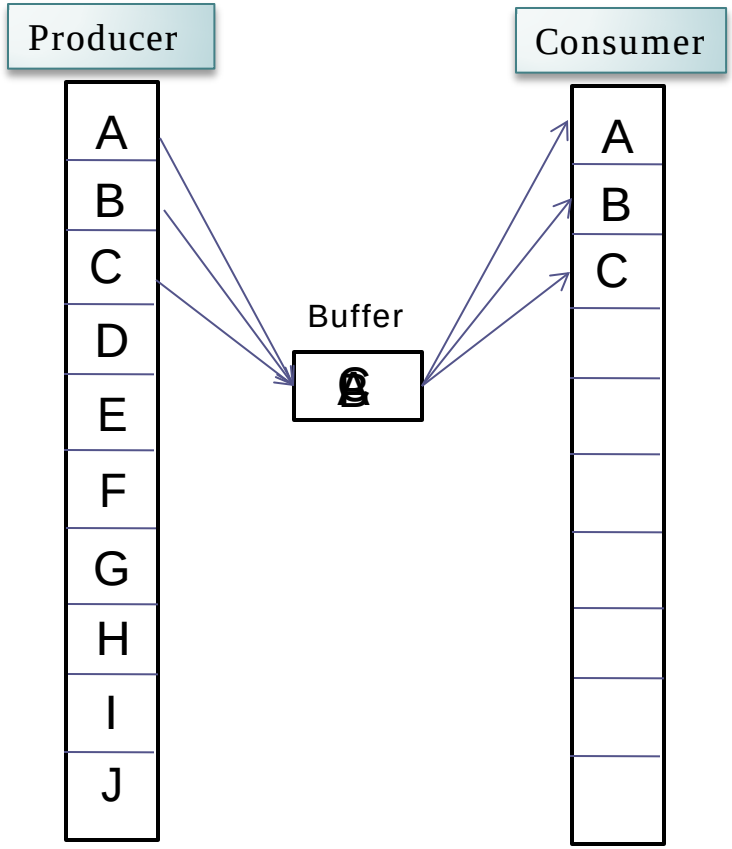
- Mutual exclusion
  - En process kan aldrig äta, dvs vara i sin kritiska sektion, samtidigt som sin bordsgranne
  - Flera processer kan vara i sin kritiska sektion samtidigt **om de inte är grannar**
- Kritisk sektion definieras kring de delade resurserna

Drinking Philosophers

- Generalisering av dining philosophers
- Godtyckligt nätverk
- Godtyckligt antal grannar
- Process kan inte vara i kritisk sektion samtidigt som någon granne
- Global invariant
  - Sann i **alla** möjliga tillstånd
  - $\forall i , j \, (\, (\, \text{eat i ng}[\, i \,] \wedge \text{nei ghbor} [\, i \, , \, j \,] \,) \rightarrow (\, \neg \text{eat i ng}[\, j \,] \,) \,)$

Producer / Consumer

- Två sorters processer
  - Delad buffer
  - Producenten lägger in data
  - Konsumenten hämtar data
- Producenten får inte skriva över ett värde som ännu inte lästs
- Konsumenten får inte läsa förrän värdet uppdaterats
- Enkel buffer implicerar att de måste turas om



Global invariant – Producer/Consumer

- Mutual exclusion

  $\forall i, j \quad \neg(inDeposit[i] \wedge inFetch[j])$
- Producenten skriver inte över

  $\forall i \; (inDeposit[i] \rightarrow counterProducer = counterConsumer)$
- Konsumenten läser inte samma värde två ggr

  $\forall i \; (inFetch[i] \rightarrow counterConsumer < counterProducer)$

Producer/Consumer, forts.

- Cirkulär buffert med flera positioner

  Producent och konsument kan exekvera kritisk sektion samtidigt *om de accessar olika positioner*

  $\forall i, j\ (1 \le i, j \le n \mid (inDeposit[i] \wedge inFetch[j]) \rightarrow front \neq rear)$

  $\forall i\ (inDeposit[i] \rightarrow counterProducer = counterConsumer)$

  $\forall i\ (inFetch[i] \rightarrow counterConsumer < counterProducer)$

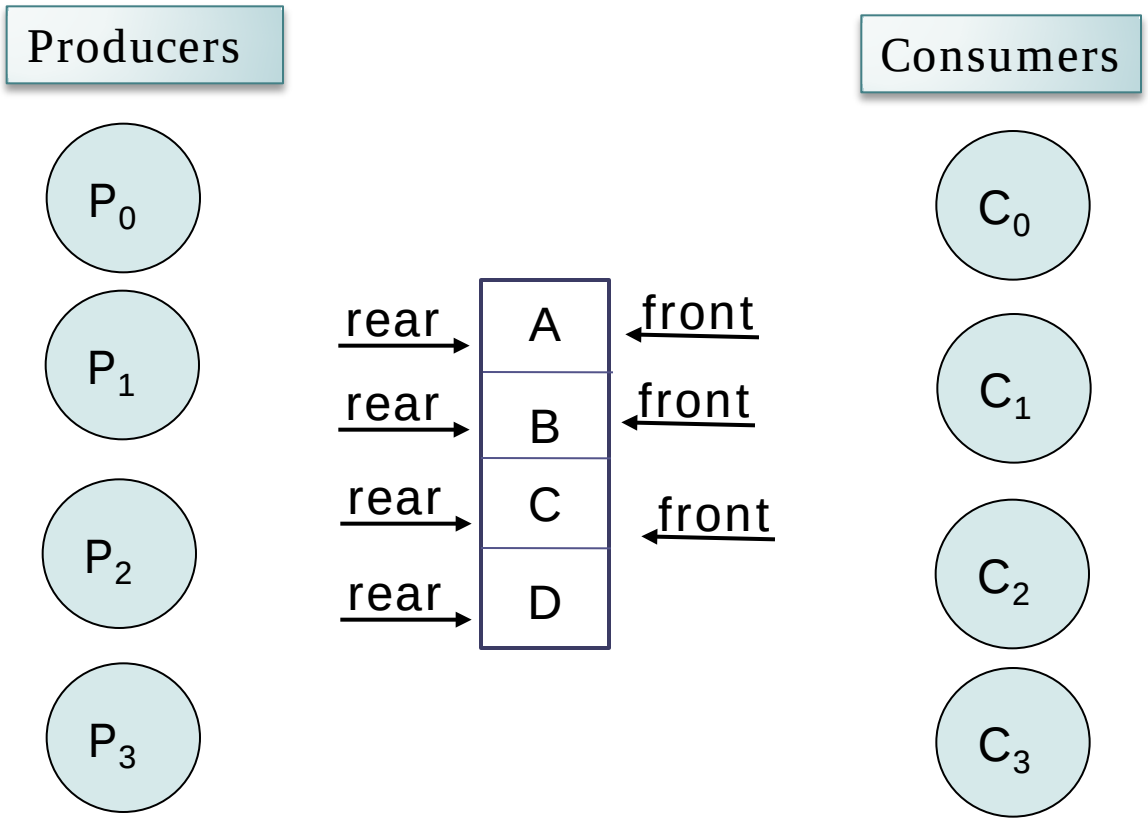
- Godtyckligt antal processer

  Processer *av samma typ* utesluter varandra

  $\forall i, j\ (1 \le i, j \le n \mid (inDeposit[i] \rightarrow \neg inDeposit[j]))$

  $\forall i, j\ (1 \le i, j \le n \mid (inFetch[i] \rightarrow \neg inFetch[j]))$

Cirkulär buffer



Readers/Writers

- Två sorters processer
  - Läsare och skrivare
- Delar en resurs, t ex en databas
- Flera kan läsa i databasen samtidigt
- Vid uppdatering får endast en skrivare ha tillgång till databasen
- Selektiv mutual exclusion
  - Läsare utesluter bara skrivare
  - Skrivare utesluter alla andra processer
- Global invariant för readers/writers-problemet
  - RW: $(nw = 0 \lor nr = 0) \land nw \leq 1$

Dining Philosophers

```
sem fork[n] := ([n] 1)
process Philosopher(i := 1 to n)
do true →
    P(fork[i])
    P(fork[(i mod n)+1])
    eat
    V(fork[(i mod n)+1])
    V(fork[i])
    think
od
end
```

Dining Philosophers

```
sem fork[n] := ([n] 1)
process Philosopher(i := 1 to n-1)
  do true →
     P(fork[i])
     P(fork[i+1])
     eat
     V(fork[i+1])
     V(fork[i])
     think
  od
end
```

```
process Philosopher(n)
  do true →
     P(fork[1])
     P(fork[n])
     eat
     V(fork[n])
     V(fork[1])
     think
  od
end
```

Producer/Consumer: enkel buffer

```
sem full  := 0
sem empty := 1
var buf  : int

process Producer::
   var m : int
   do true →
      produce m
      P(empty)
      buf := m
      V(full)
   od
end
```

```
process Consumer::
   var n : int
   do true →
      P(full)
      n := buf
      V(empty)
      consume n
   od
end
```

Split binary semaphores

- Används när processer delar kritisk sektion men blockerar på **olika** villkor
- Invariant
  - $S_1 + S_2 + \ldots + S_n \leq 1$

Producer/Consumer: cirkulär buffer

```
sem full  := 0
sem empty := N
var buf[N]: int
var rear  : int := 1
var front : int := 1

process Producer::
  var m: int
  do true →
    produce m
    P(empty)
    buf[rear] := m
    rear := rear mod N +1
    V(full)
  od
end
```

```
process Consumer::
  var n : int
    do true →
    P(full)
    n := buf[front]
    front := front mod N +1
    V(empty)
    consume n
  od
end
```

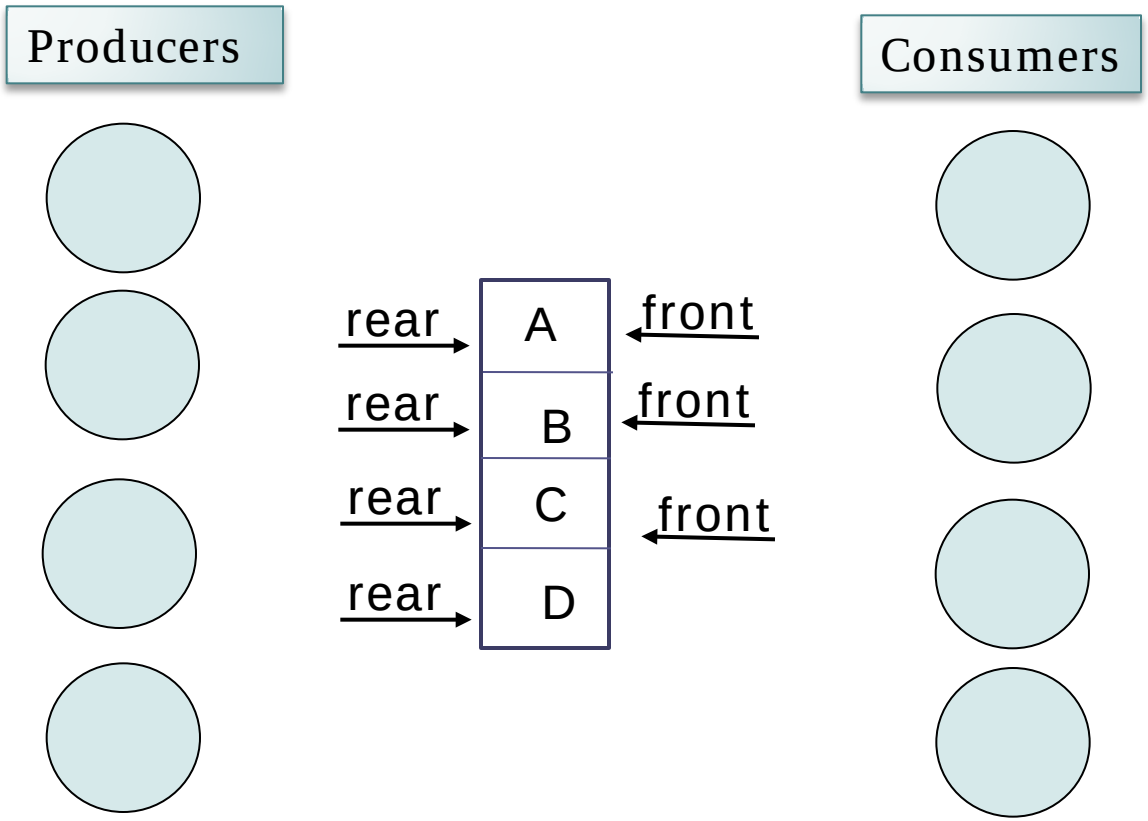
Producer/Consumer: Flera processer

```
sem full := 0
sem empty := N
sem mutexP := 1
sem mutexC := 1
var buf[N]: int
var front : int := 1
var rear : int := 1
```

```
process Producer(i:=1 to M)
  var m : int
  do true →
    produce m
    P(mutexP)
    P(empty)
    buf[rear] := m
    rear := rear mod N +1
    V(full)
    V(mutexP)
  od
end
```

```
process Consumer(j:=1 to M)
  var n : int
  do true →
    P(mutexC)
    P(full)
    n := buf[front]
    front := front mod N +1
    V(empty)
    V(mutexC)
    consume n
  od
end
```

Cirkulär buffer



Readers/Writers

```
var nr, nw : int := 0
var dr, dw : int := 0
sem mutex := 1
sem read := 0
sem write := 0

process reader(i := 1 to N)
 do true →
    P(mutex)
    if (nw > 0) →
        dr ++
        V(mutex)
        P(read)
    fi
    nr++
    signal()
    #do some reading
    P(mutex)
    nr- -
    signal()
    #do something else
 od
end
```

```
process writer(i := 1 to M)
 do true →
    P(mutex)
    if (nw > 0 or nr > 0) →
        dw ++
        V(mutex)
        P(write)
    fi
    nw++
    signal()
    #do some writing
    P(mutex)
    nw- -
    signal()
    #do something else
 od
end

Procedure signal()
 if (nw = 0 and dr > 0) →
    dr - -
    V(read)
 [] (nr = 0  and nw = 0 and dw > 0) →
    dw- -
    V(write)
 [] (nw > 0 or dr = 0)  and (nr > 0 or nw > 0 or dw = 0) →
    V(mutex)
 fi
end
```

Partyproblemet

- Två typer av processer, värd och gäst
- Godtyckligt antal gäster
- Värdens uppgift är att fylla på förfriskningar i en bålskål
- Gästerna dricker och umgås omväxlande
- Hur garantera att gästerna inte dricker från en tom skål?
- Hur garantera att värden fyller på vid rätt tidpunkt?
- Hur garantera att ingen törstig gäst blir utan?

Partyproblemet, forts.

- Vi måste kontrollera innehållet i bålen #glas kvar
  - Kräver mutual exclusion
- Vi måste meddela värden när bålen är slut
  - Kräver synkronisering
- Gästerna måste vänta tills bålen är påfylld
  - Kräver synkronisering