Lesson 11

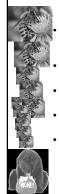
Practical Examples

(Ch 5-9 [BenA 06])

Example Problem
Problem Features
System Features
Various Concurrency Solutions

7.12.2009

Copyright Teemu Kerola 2009



A Bear, Honey Pot and Bees

- Friendly bees are feeding a trapped bear by collecting honey for it. The life of the trapped bear is just eating and sleeping.
- There are N bees and one bear. The size of the pot is H portions.
- The bees carry honey to a pot, one portion each bee each time until the pot is full. Or maybe more?
- When the pot is full, the bee that brought the last portion wakes up the bear.
 - The bear starts eating and the bees pause filling the pot until the bear has eaten all the honey and the pot is empty again. Then the bear starts sleeping and bees start depositing honey again.

[Andrews 2000, Problem 4.36]

7.12.2009

Copyright Teemu Kerola 2009

Problem Features

- · Thousands or millions of bees (N bees), one bear
 - Collecting honey (1 portion) may take very long time
 - Eating a pot of honey (H portions) may take some time
 - Filling up the pot with one portion of honey is fast
 - Same solution ok with N=1000 or N=100 000 000 ?
 - $-\;$ Same solution ok with H=100 or H=1 000 000 ?
 - Same solution ok for wide range of N & H values? Unspecified/not well defined feature
 - Could (should) one separate <u>permission to fill</u> the pot, actually <u>filling</u> the pot, and possibly <u>signalling</u> the bear
 - If (one bee) filling the pot is real fast, this may not matter
 - If (one bee) filling the pot takes time, then this may be crucial for performance
- Can pot be filled from far away?
- What if more than one bears?

7.12.2009

Copyright Teemu Kerola 2009

Maximize Parallelism

- All bees concurrently active, no unnecessary blocking
- · Bees compete only when filling up the pot
 - Must wake up bear when H portions of honey in pot
 - Must fill up the pot one bee at a time
 - Is this important or could we modify specs?
 - How big is the mouth of the pot?
 - Competing just to update the counter would be more efficient?



- Is waking up the bear part of critical section?
- What is the real critical section?

7.12.2009

Copyright Teemu Kerola 2009

Maximize Parallelism (contd)

- Bear wakes up only to eat and only when pot is full
- Bees blocked (to fill the pot) only
 - When bear is eating
 - When waiting for their turn to fill the pot
 - · Or to synchronize with other bees

7.12.2009

Copyright Teemu Kerola 2009

Concurrency Needs

- When is <u>mutex</u> (critical section) needed?
 - A bee is filling the pot or the bear is eating
- When is synchronization needed?
 - Bees wait for earlier bee to fill the pot
 - Each bee may wait before filling the pot
 - Bees wake up the bear to eat
 - Last (Hth) bee wakes up bear after filling the pot
 - Bear lets all bees to resume filling the pot
 - Bear allows it <u>after emptying</u> the pot
- When is <u>communication</u> needed?
 - Must know when pot is full? Nr portions in pot now?
 - What if "honey" would be information in buffer?

7.12.2009

Copyright Teemu Kerola 2009

Lecture 11: Practical Examples

Environment

- Computational object level
 - Bees and bear are threads in one application?
 - Threads managed by programming language?
 - Threads managed by operating system?
 - Bees and bear are <u>processes</u>?
 - · Communication with progr. language utilities?
 - · Communication with oper. system utilities?
- System structure
 - Shared memory uniprocessor/multiprocessor?
 - Distributed system?
 - Networked system?

7.12.2009

Copyright Teemu Kerola 2009

Busy Wait or Suspended Wait

- Bear waits a long time for full pot?
 - Suspended wait would be better (unless <u>lots</u> of processors)
- Bees wait for their turn to fill the pot?
 - Waiting for turn takes relatively long time
 - Earlier bees fill the pot
 - · Bear eats the honey
 - Suspended wait ok
- Bees wait for their turn only to update counters?
 - Relatively long time to wait for turn
 - Suspended wait ok
 - If mutex is <u>only</u> for updating counters (not for honey fill-up turn, or bear eating), busy wait might be ok

7.12.2009 Copyright Teemu Kerola 2009

Evaluate Solutions

- · Does it work correctly?
 - Mutex ok, no deadlock, no starvation
- · Does it allow for maximum parallelism?
 - Minimally small critical sections
 - Could bees fill up the jar in parallel?
- · Is this optimal solution?
 - Overall processing time? Overall communication time?
 - Processor utilization? Memory usage?
 - Response time? Investments/return ratio?
- · Is this solution good for current problem/environment?
 - Bees and bear are threads in Java application in 4-processor system running Linux?
 - There are 20000 bees, collecting honey takes 15 min, depositing one portion in pot takes 10 sec, 5000 portions fill the pot, and bear eats the honey in pot in 10 minutes?

7.12.2009

Copyright Teemu Kerola 2009

7.12.2009 Copyright Teemu Kerola 2009 10

Solution with Locks

- Can use locks both for mutex and for synchronization
 - Problem: busy wait for bear
 - Bear waits a long time for full honey pot (some bears do not like waiting!)

```
Int portions = 0; # portions in the pot
Lock_var D = 0 = "open"; # mutex to deposit honey in pot
E = 1 = "closed"; # permission to eat honey

implem. dependent:
Lock_var D = 1; #open
E = 0; #locked
```

Copyright Teemu Kerola 2009

```
Solution with Locks (contd)
                                       portions = 0; # portions in the pot
                             Int
                             Lock_var D = 0; # mutex to deposit honey in pot
process bee [i=1 to N] () { while (true) {
                                       E = 1; # permission to eat honey
    lock (D); # only one bee advances at a time
    portions++;
fill_pot();
    if (portions == H) unlock (E); # wakeup bear, keep lock
       else unlock (D) # let next bee deposit honey
                               process bear () {
                                while (true) {
                                                 # busy-wait, hopefully OK?
                                  lock (E):
                                  eat_honey();
                                  portions = 0
                                   unlock (D); # let next bee deposit honey
 7.12.2009
                            Copyright Teemu Kerola 2009
```

7.12.2009

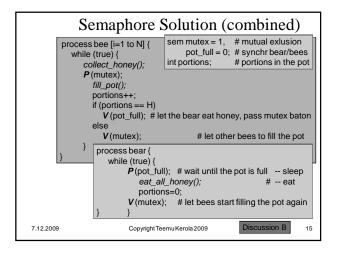
```
Semaphore Solution

process bee[i=1 to N] {
    while (true) {
        collect_honey();
        into_pot(); # deposit one honey portion into the honey pot
    }
}

process bear {
    while (true) {
        sleep(); # wait until the pot is full empty_pot(); # eat all the honey
    }
}

7.12.2009 Copyright TeemuKerola 2009 [Liisa Marttinen] 13
```

```
Semaphore Solution (contd)
        sem mutex = 1, # mutual exlusion
             pot_full = 0; # is the pot full of honey?
        int portions;
                        # portions in the pot
        procedure into_pot() { # bee deposits one honey portion
         P (mutex);
         fill_pot(); portions++
         if (portions == H) V (pot_full); # let bear eat honey, pass baton
         else V(mutex); # let other bees fill the pot
                                                  procedure sleep () {
                                                   P(pot_full);
        procedure empty pot() { # bear eats all honey from the pot
          eat_all_honey();
          portions=0;
          V (mutex); # let bees fill the pot again
7.12.2009
                                                                       14
                         Copyright Teemu Kerola 2009
```



```
Monitor Solution
       • Use monitor only for mutex and synchronization

    Automatic mutex

           - Use of monitor condition variables for synchronization
             solution for bees and bear

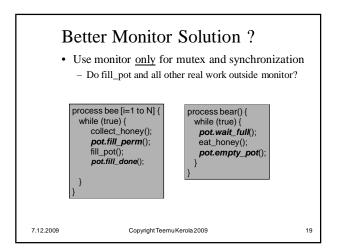
    What type of signalling semantics is <u>in use</u>?

          -E < S < W, i.e., IRR? Assume now no-IRR.
                                        process bear() {
           while (true) {
                                          while (true) {
                                           pot.wait_full();
              collect_honey();
              pot.into_pot();
                                            eat_honey();
                                            pot.empty_pot();
                                                  [Auvo Häkkinen]
7.12.2009
                       Copyright Teemu Kerola 2009
```

```
Monitor Solution (contd)
     monitor pot {
      int portions=0; cond pot_full, pot_empty;
        while (portions == H) waitC (pot_empty); Would "if" work?

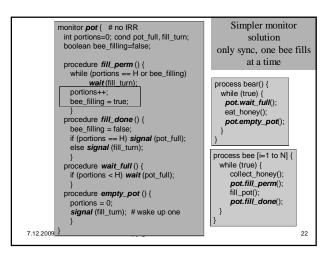
portions++; fill_pot(): #denosit h
       procedure into_pot () {
         if (portions == H) signalC (pot_full);
      procedure wait_full () {
  if (portions < H) waitC (pot_full);</pre>
                                                         Why "if" and not "while"?
                                                         Would "while" work?
                                                        What if some other
       procedure empty_pot() {
                                                       signalling semantics?
        portions = 0:
         signal_allC (pot_empty) # wake up all waiting bees
7.12.2009
                            Copyright Teemu Kerola 2009
                                                                                 17
```

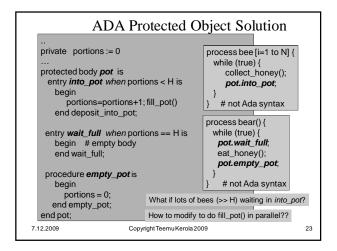
```
All Work Included in Monitor
monitor pot {
                                              process bee [i=1 to N] {
                                                while (true)
 cond pot_full, pot_empty;
                                                   pot.collect_into_pot();
 procedure collect_into_pot() {
   collect_honey();
                                              process bear() {
   while (portions==H) waitC(pot_empty);
                                                while (true)
   portions=portions+1; fill_pot();
                                                  pot.sleep_and_eat();
   if (portions==H) signalC(pot_full);
 procedure sleep_and_eat() {
  if (portions < H) waitC(pot_full);</pre>
                                              Which is better?
   eat_honey();
portions=0;
   signal_allC(pot_empty)
                                              What is the problem?
7.12.2009
                           Copyright Teemu Kerola 2009
                                                                             18
```

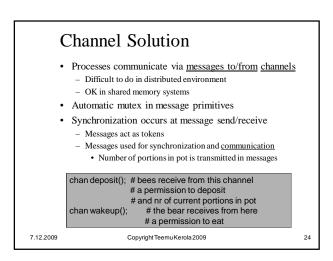


```
Another Monitor Solution
monitor pot { # no IRR
int fill=0, portions=0; cond pot_full, pot_empty;
                                                            (only synchronization,
                                                            many bees can fill at a time)
procedure fill_perm() {
  while (fill+portions == H) waitC (pot_empty);
                                                            process bee [i=1 to N] {
   fill++; # nr of bees with fill permission
                                                             while (true) {
                                                                collect honey();
                                                                pot.fill_perm();
fill_pot();
procedure fill_done() {
  fill--; portions++;
if (portions == H) signalC (pot_full);
                                                                pot.fill_done();
procedure wait_full () {
  if (portions < H) waitC (pot_full);</pre>
                                                               process bear() {
                                                               while (true) {
                                                                  pot.wait_full();
                                                                    eat_honey();
procedure empty_pot() {
                                                                   pot.empty_pot();
   signal_allC (pot_empty) # wake up all
                                    Copyright Teemu Kerola 2009
```

```
nonitor pot { # no IRR
                                                                               Monitor Solution
int fill=0, portions=0; cond pot_full, pot_empty; boolean bee_filling=false; cond fill_turn;
                                                                                    (only sync,
                                                                           one bee fills at a time)
procedure fill_perm () {
   while (fill+portions == H) wait (pot_empty);
   fill++;
                                                                               process bear() {
                                                                                  while (true) {
    pot.wait_full();
   if (bee_filling) wait (fill_turn);
   bee_filling = true;
                                                                                    eat_honey();
frocedure fill_done() {
  fill--; portions++; bee_filling = false;
  if (portions == H) signal (pot_full);
  else signal (fill_turn);
                                                                                   pot.empty_pot();
                                                                              process bee [i=1 to N] {
procedure wait_full() {
    if (portions < H) wait (pot_full);
                                                                                     collect honey();
                                                                                     pot.fill_perm();
fill_pot();
   rocedure empty_pot() {
                                                                                      pot.fill_done();
   portions = 0
    signal_all (pot_empty); # wake up all
                                                                                                            21
```







26

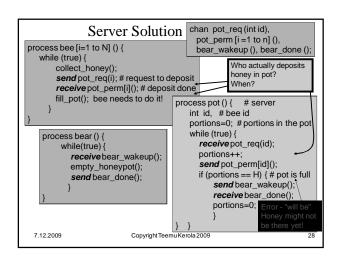
```
Channel Solution
process bee [i=1 to N] () {
  while (true) {
                                 Is it ok to do fill_pot() in distributed fashion?
    collect_honey ();
    receive (deposit_perm, portions); # only one bee advances at a time
    portions++; fill_pot ();
                                               # deposit one portion
    if (portions == H) send (wakeup, dummy); # pot is full, wakeup bear
        else send (deposit_perm, portions); #let next bee deposit honey
                  process bear () {
    send (deposit_perm, 0); # let first bee deposit honey
                   while (true) {
                     receive (wakeup, dummy);
eat_honey ();
                      send (deposit_perm, 0); # let next bee deposit honey
 7.12.2009
                            Copyright Teemu Kerola 2009
```

Message Solution • Processes communicate via messages to/from processes • Bear wakes up with wake-up message to it – Easy, just one bear • Messages used only for synchronization or also for communication? • How to keep track of honey portions – Must use messages • How to send messages to other bees? – Too many receivers, not practical – Need msg server

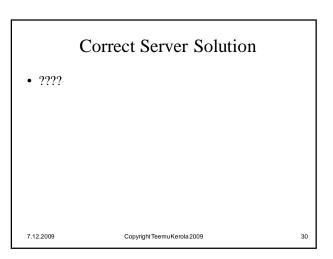
Copyright Teemu Kerola 2009

7.12.2009

Server Solution · All synchronization problems solved by server · Server process pot gives turns to bees and bear · Correct bee must get permission to fill up pot · Centralized solution, like monitor... # request from a bee chan pot req (int id), pot_perm [i =1 to n] (), # permission for each bee bear_wakeup(), # permission to eat for the bear bear done (); # bear finished eating 27 7.12.2009 Copyright Teemu Kerola 2009



Server Solution Comments Who actually deposits honey in pot and when? How to separate permission to deposit honey before honey is deposited and waking up the bear after honey is deposited? chan pot_req (int id), pot_perm[i =1 to n] (), bee_honey_in_pot (), # request from a bee # permission to deposit # deposit done bee_can_proceed [i=1 to n] (), # bee can start # collecting again # permission to eat for bear bear wakeup(), bear_done(); # bear finished eating What if just one request channel and multiple reply - Request for turn to deposit, turn to collect, turn to eat, turn to sleep? - Replies to bees and bear 7.12.2009 Copyright Teemu Kerola 2009 29

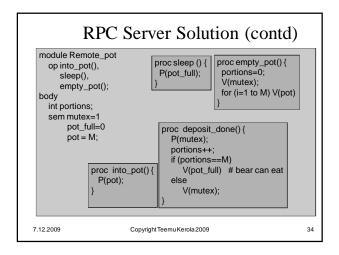



```
Rendezvous Solution (contd)
  module Control_Pot op into_pot(), deposit_pot(), sleep(), empty_pot(); # services
  body
   process Pot {
    int portions = 0, deposits=0;
    while (true)
     in into\_pot () and portions+deposits < MAXSIZE \rightarrow deposits++;
     [] deposit\_done() \rightarrow deposits--; portions++;
        sleep () and portions == MAXSIZE →
        empty\_pot() and portions == MAXSIZE \rightarrow portions=0;
                           Is this part needed?
  end Control_pot

    Solution with Ada similarly

7.12.2009
                         Copyright Teemu Kerola 2009
                                                       Discussion D
```

PRC Server Solution (2) • Distributed system over LAN? process bee [i=1 to N] { while (true) { collect_honey(); call Remote_pot.into_pot(); call Remote_pot.deposit_done(); } process bear { while (true) { call Remote_pot.sleep(); eat_honey(); call Remote_pot.empty_pot(); } } 7.12.2009 Copyright TeemuKerola 2009 33



Evaluate Your Solution • Same problem - many solutions - all correct? Does it work correctly? Does it allow for maximum parallelism? Is this optimal solution? Is this solution good for current problem/environment? 25 000 - 250 000 000 bees, 25 000 - 250 000 000 bees, collecting honey takes 30-60 min, depositing one portion in pot takes 1-3 mins, 10000-100000 portions fill the pot, and bear eats the honey in pot in 5-50 minutes? You might get another bear next year? What if much more bees? What if the pot allows for 100-1000 simultaneous fill-ups? Bees and bear are threads in Java application in 4-processor system running Linux? "Honey" is an 80-byte msg to be used by "bear"? 7.12.2009 Copyright Teemu Kerola 2009 35

Summary • Specify first your requirements • What concurrency tools do you have at your disposal? • Does your solution match your environment? • Will some known solution pattern apply here? - Readers-writers, producers-consumers, bakery? • Does it work? • Is it optimal in time/space? • Does it allow for maximum parallelism? • Does it minimize waiting?