SWEN-220
Mathematical Models of Software

Channels
Topics

• Channels
  – Senders
  – Receivers
  – ACKs

• Example
Channels

• Promela/SPIN can be used to model distributed systems

• Nodes are modeled as processes and communication networks as channels
Channels

• Generally, channels are not associated with a process. Any process can send or receive a message on a channel

• A channel is a data type with two operations, send and receive
Declaring Channels

• Each channel is associated with a message type

chan ch = [capacity] of {type, …, type}

• Channels with 0 capacity are called **rendezvous channels**
• Channels with >0 capacity are called **buffered channels**
Rendezvous Channels

• Channels with zero capacity

• Transfer from sender to receiver is synchronous and is performed as a single atomic operation

• Even if there were other processes, no interleaving occurs between the send and receive statements
Send statement

ch ! var, .., var

• The data types of the sequence must match the message types of the channel
Receive statement

rch ? var, .., var

• The data types of the sequence must match the message types of the channel
Send and Receive Statements*

• Expressions in the send statement are evaluated and transferred through the channel

• Receive statement assigns these values to the variables listed

• Receive statements block until a message becomes available
Channel Variables

- The type of all channels is `chan`

- Channel variables hold a reference to the channel itself

- A channel can be sent in a message and received by a process
Additional Channel Topics

• Checking if a channel is full or empty
• Checking the number of messages in a channel
• Copying message values (peek)
• Simulating buffer overflow
• Random receive (??)
• Sorted send (!!)
• Rendezvous vs Buffered Channels
Checking if a channel is full or empty

• Design issue:
  – Clients sending messages to a buffered channel will block if the buffer is full.
  – Servers reading messages from a buffered channel will block if the buffer is empty.
  – We’d like to free up the clients/servers to do other work if they could instead of blocking.

• Promela:
  – full(ch), empty(ch)
  – nfull(ch), nempty(ch)
  – ( do not use !full or !nempty)
Checking the number of messages in a channel

• Design issue:
  – Dynamically allocating/deallocating servers (workers) based on the number of incoming message requests.
  – Common approach to load-balancing

• Promela: len(ch)

Example: load > 75%, add server, load <25% remove server

```promela
if
:: len(ch) > (3*N/4)  // allocate a new server
:: len(ch) < (N/4)    // deallocate an existing server
:: else
fi
```
Copying message value (peek)

• Design issue:
  – Receiver would like to do a non-destructive read (peek at the top of the queue without removing message)

• Promela :
  
  ch ? < value >    // note angle brackets!

  Copies channel content to local variable value, but does not remove message from channel.
Random Reads

• Design issue:
  – Receiver pulls messages from channel in a random

• Promela :
  ch ?? variable
  Receive does a random (not FIFO) read of the channel.
Sorted Sends

• Design issue:
  – Message queue maintained based on sorted content. Example: priority queue

• Promela:
  ch !! variable
  Variable inserted into the message buffer based on sorted (integer) value.
Buffer Overflow

• In some cases it may not be possible for the sending process to detect if the buffer is full. It simply continues to send messages without blocking and messages may become “lost”.

• SPIN can simulate this behavior by running SPIN with the “-m” argument (not an option in JSpin). Sending process will NOT block if channel buffer is full.