Reading Assignment

- This lecture: 3.1.2
- Next lecture: 3.1.2, 3.2
Homework 1

- Due: 5:00pm 02/11 Chicago time
- Submit on Blackboard.
Outline

Process-Based Models

Kahn Process Network (KPN)

Synchronous Data Flow (SDF)
Process-Based Models

- A set of concurrent processes
  - There is no guarantee on the speed of the processes. Some may run very fast, while some may simply stop there for no apparent reason.

- Processes may communicate with each other to exchange data and establish dependencies among behaviors.
  - Semantics of communication differs for different process-based models.
  - If we need to distinguish to exchange data and to establish dependencies, we call the former communication and the later synchronization.
Process-Based Models

- A set of concurrent processes
  - There is no guarantee on the speed of the processes. Some may run very fast, while some may simply stop there for no apparent reason.
- Processes may communicate with each other to exchange data and establish dependencies among behaviors.
  - Semantics of communication differs for different process-based models.
  - If we need to distinguish to exchange data and to establish dependencies, we call the former communication and the later synchronization.
Specifying and Implementing Processes

▶ Usually, processes are specified as sequential programs.
  ▶ These programs run parallelly and independently.
▶ Implementing processes means to schedule them on processors (CPU) or processing elements (PE)
  ▶ Scheduling a single process on a single CPU/PE is straight-forward.
  ▶ OS usually provides support to schedule multiple processes on a single CPU/PE.
  ▶ Otherwise, we need to provide our own scheduling, which will be introduced later.
Specifying and Implementing Processes

- Usually, processes are specified as sequential programs.
  - These programs run parallelly and independently.

- Implementing processes means to schedule them on processors (CPU) or processing elements (PE)
  - Scheduling a single process on a single CPU/PE is straight-forward.
  - OS usually provides support to schedule multiple processes on a single CPU/PE.
  - Otherwise, we need to provide our own scheduling, which will be introduced later.
Specifying and Implementing Communications

- Two typical communication models
  - Message passing
  - Shared memory

- Communication via message passing
  - Synchronization is implied by the message.
  - Synchronous/rendezvous style: send/recv
  - Asynchronous/queue-based style
  - E.g. the Message Passing Interface (MPI)

- Communication via shared memory
  - Data are shared. Synchronization is the critical issue.
  - Synchronization primitives, e.g. mutex and semaphore, are usually supported by OS through Inter-Process Communication (IPC) mechanisms.
Two typical communication models
- Message passing
- Shared memory

Communication via message passing
- Synchronization is implied by the message.
- Synchronous/rendezvous style: send/recv
- Asynchronous/queue-based style
- E.g. the Message Passing Interface (MPI)

Communication via shared memory
- Data are shared. Synchronization is the critical issue.
- Synchronization primitives, e.g. mutex and semaphore, are usually supported by OS through Inter-Process Communication (IPC) mechanisms.
Two typical communication models
- Message passing
- Shared memory

Communication via message passing
- Synchronization is implied by the message.
- Synchronous/rendezvous style: send/recv
- Asynchronous/queue-based style
- E.g. the Message Passing Interface (MPI)

Communication via shared memory
- Data are shared. Synchronization is the critical issue.
- Synchronization primitives, e.g. mutex and semaphore, are usually supported by OS through Inter-Process Communication (IPC) mechanisms.
Deadlocks

- Process A holds resource a while waiting for resource b.
- Process B holds resource b while waiting for resource a.
- The system deadlocks.
  - No process can proceed and make progress.
- Deadlocks arise if there is a circular dependency on who is owning the resource and who is requesting the resource.
- Deadlocks can be avoided by either statically preventing such dependency to happen at design time or dynamically breaking them at runtime.
  - Still quite challenging for real systems.
Deadlocks

- Process A holds resource a while waiting for resource b.
- Process B holds resource b while waiting for resource a.
- The system deadlocks.
  - No process can proceed and make progress.
- Deadlocks arise if there is a circular dependency on who is owning the resource and who is requesting the resource.
- Deadlocks can be avoided by either statically preventing such dependency to happen at design time or dynamically breaking them at runtime.
  - Still quite challenging for real systems.
Deadlocks

- Process A holds resource a while waiting for resource b.
- Process B holds resource b while waiting for resource a.
- The system deadlocks.
  - No process can proceed and make progress.
- Deadlocks arise if there is a circular dependency on who is owning the resource and who is requesting the resource.
- Deadlocks can be avoided by either statically preventing such dependency to happen at design time or dynamically breaking them at runtime.
  - Still quite challenging for real systems.
Deadlocks

- Process A holds resource a while waiting for resource b.
- Process B holds resource b while waiting for resource a.
- The system deadlocks.
  - No process can proceed and make progress.
- Deadlocks arise if there is a circular dependency on who is owning the resource and who is requesting the resource.
- Deadlocks can be avoided by either statically preventing such dependency to happen at design time or dynamically breaking them at runtime.
  - Still quite challenging for real systems.
Deadlocks

- Process A holds resource a while waiting for resource b.
- Process B holds resource b while waiting for resource a.
- The system deadlocks.
  - No process can proceed and make progress.
- Deadlocks arise if there is a circular dependency on who is owning the resource and who is requesting the resource.
- Deadlocks can be avoided by either statically preventing such dependency to happen at design time or dynamically breaking them at runtime.
  - Still quite challenging for real systems.
Deterministic Behavior

- A deterministic model will always generate the same output given the same input.
  - Input/output behavior does not depend on the performance of processes and communication channels.
  - Determinism makes debugging and validation easier.
  - Otherwise, it is difficult to tell whether the model is correct.

- Additional determinism may be introduced to the model to specify the order the processes are executed.
  - For most systems, this may lead to overspecification, impacting the performance of the implementations, since otherwise the non-deterministic behavior may be exploited for better performance.
A deterministic model will always generate the same output given the same input.

- Input/output behavior does not depend on the performance of processes and communication channels.
- Determinism makes debugging and validation easier.
- Otherwise, it is difficult to tell whether the model is correct.

Additional determinism may be introduced to the model to specify the order the processes are executed.

- For most systems, this may lead to overspecification, impacting the performance of the implementations, since otherwise the non-deterministic behavior may be exploited for better performance.
Considerations for Process-Based Models

- How communications are modeled?
  - Can we prevent deadlocks?
  - What kind of determinism is guaranteed?
Considerations for Process-Based Models

- How communications are modeled?
- Can we prevent deadlocks?
- What kind of determinism is guaranteed?
Considerations for Process-Based Models

- How communications are modeled?
- Can we prevent deadlocks?
- What kind of determinism is guaranteed?
Outline

Process-Based Models

Kahn Process Network (KPN)

Synchronous Data Flow (SDF)
Kahn Process Network (KPN)

- Processes are represented by nodes.
- Communications are represented by (directed) arcs.
  - Usually known as channels
  - Unidirectional
  - Point to point
- Communication via message passing
  - Unit of data/message: a token
  - Assume each arc has an unbounded first-in-first-out (FIFO) queue to hold unprocessed tokens.
Kahn Process Network (KPN)

- Processes are represented by nodes.
- Communications are represented by (directed) arcs.
  - Usually known as *channels*
  - Unidirectional
  - Point to point
- Communication via message passing
  - Unit of data/message: a *token*
  - Assume each arc has an unbounded first-in-first-out (FIFO) queue to hold unprocessed tokens.
Kahn Process Network (KPN)

- Processes are represented by nodes.
- Communications are represented by (directed) arcs.
  - Usually known as *channels*
  - Unidirectional
  - Point to point
- Communication via message passing
  - Unit of data/message: a *token*
  - Assume each arc has an unbounded first-in-first-out (FIFO) queue to hold unprocessed tokens.
KPN Example

FIGURE 3.1 Kahn Process Network (KPN) example

(Gajski et al., 2009)
Interactions between Processes and Channels

- A process sending a token to a channel will never block.
  - Since the FIFO queue will take the token immediately.
- A process always blocks on reading a channel.
  - The process waits until a token is available and will retrieve the token immediately.
- The behavior of a process is deterministic.
  - In the sense that the process cannot peek into a channel.
  - The process must decide which channel to read next and then block on that channel until a token arrives.
Interactions between Processes and Channels

- A process sending a token to a channel will never block.
  - Since the FIFO queue will take the token immediately.
- A process always blocks on reading a channel.
  - The process waits until a token is available and will retrieve the token immediately.
- The behavior of a process is deterministic.
  - In the sense that the process cannot peek into a channel.
  - The process must decide which channel to read next and then block on that channel until a token arrives.
Interactions between Processes and Channels

- A process sending a token to a channel will never block.
  - Since the FIFO queue will take the token immediately.
- A process always blocks on reading a channel.
  - The process waits until a token is available and will retrieve the token immediately.
- The behavior of a process is deterministic.
  - In the sense that the process cannot peek into a channel.
  - The process must decide which channel to read next and then block on that channel until a token arrives.
Inputs can be modeled as processes without incoming arc. 
- So they simply generate tokens to be further processed.

Outputs can be modeled as processes without outgoing arc. 
- So they will remove tokens from the system.
Modeling System Inputs and Outputs

- Inputs can be modeled as processes without incoming arc.
  - So they simply generate tokens to be further processed.
- Outputs can be modeled as processes without outgoing arc.
  - So they will remove tokens from the system.
Deadlocks and Determinism

- Certain KPNs may have deadlocks.
  - There is generally no way to decide if a KPN will have a deadlock or not.
- KPNs are deterministic.
  - The deterministic behavior of a process leads to the deterministic behavior of the system since it is independent of when a token arrives.
Deadlocks and Determinism

- Certain KPNs may have deadlocks.
  - There is generally no way to decide if a KPN will have a deadlock or not.
- KPNs are deterministic.
  - The deterministic behavior of a process leads to the deterministic behavior of the system since it is independent of when a token arrives.
Implementation Considerations

- Unbounded queue cannot be implemented.
- Sizes must be imposed on all queues.
  - And we want the sizes to be as small as possible to save memory for queues.
  - However, smaller sizes may lead to artificial deadlocks.
- Scheduling of processes may impact sizes.
- Determinism makes it possible for some algorithms to determine a set of queues with small sizes for certain schedules.
  - However, it is not always possible to find such sizes/schedule.
Implementation Considerations

- Unbounded queue cannot be implemented.
- Sizes must be imposed on all queues.
  - And we want the sizes to be as small as possible to save memory for queues.
  - However, smaller sizes may lead to artificial deadlocks.
- Scheduling of processes may impact sizes.
- Determinism makes it possible for some algorithms to determine a set of queues with small sizes for certain schedules.
  - However, it is not always possible to find such sizes/schedule.
Implementation Considerations

- Unbounded queue cannot be implemented.
- Sizes must be imposed on all queues.
  - And we want the sizes to be as small as possible to save memory for queues.
  - However, smaller sizes may lead to artificial deadlocks.
- Scheduling of processes may impact sizes.
  - Determinism makes it possible for some algorithms to determine a set of queues with small sizes for certain schedules.
  - However, it is not always possible to find such sizes/schedule.
Implementation Considerations

- Unbounded queue cannot be implemented.
- Sizes must be imposed on all queues.
  - And we want the sizes to be as small as possible to save memory for queues.
  - However, smaller sizes may lead to artificial deadlocks.
- Scheduling of processes may impact sizes.
- Determinism makes it possible for some algorithms to determine a set of queues with small sizes for certain schedules.
  - However, it is not always possible to find such sizes/schedule.
Outline

Process-Based Models

Kahn Process Network (KPN)

Synchronous Data Flow (SDF)
The difficulty in deciding a small bound and deadlocks for KPN is in the flexibility of its processes.

- The processes are deterministic but can read/write channels rather arbitrarily.

Restrict how processes execute
- Break a process into atomic blocks of execution, called actors.
- Fire (execute) an actor only when all input tokens are available.
- Require an actor to generate tokens on all its outputs.

Synchronous Data Flow (SDF)
- Actors for the same process consume and produce a constant number of tokens per channel, per firing.
- The numbers could be different for different channels, but remain fixed for a particular channel and read/write operation.
Synchronous Data Flow (SDF)

- The difficulty in deciding a small bound and deadlocks for KPN is in the flexibility of its processes.
  - The processes are deterministic but can read/write channels rather arbitrarily.
- Restrict how processes execute
  - Break a process into atomic blocks of execution, called *actors*.
  - Fire (execute) an actor only when all input tokens are available.
  - Require an actor to generate tokens on all its outputs.
- Synchronous Data Flow (SDF)
  - Actors for the same process consume and produce a constant number of tokens per channel, per firing.
  - The numbers could be different for different channels, but remain fixed for a particular channel and read/write operation.
The difficulty in deciding a small bound and deadlocks for KPN is in the flexibility of its processes.

- The processes are deterministic but can read/write channels rather arbitrarily.

Restrict how processes execute

- Break a process into atomic blocks of execution, called *actors*.
- Fire (execute) an actor only when all input tokens are available.
- Require an actor to generate tokens on all its outputs.

Synchronous Data Flow (SDF)

- Actors for the same process consume and produce a constant number of tokens per channel, per firing.
- The numbers could be different for different channels, but remain fixed for a particular channel and read/write operation.
SDF Example

FIGURE 3.2  Synchronous Data Flow (SDF) example

(Gajski et al., 2009)
Can we prevent deadlock?
- Not for all SDFs, but we can decide if there will be a deadlock.
- Deadlock only happens when there is not enough tokens on certain arcs.
- Assume initially there are enough tokens on each arc.
  - So we may freely fire any actor for certain amount of times.
- If we can find a scheduling to fire the actors such that the number of tokens will remain unchanged afterwards, then we know there is no deadlock if we repeat such scheduling.
  - Also known as the relative execution rates of actors.
Can we prevent deadlock?

- Not for all SDFs, but we can decide if there will be a deadlock.
- Deadlock only happens when there is not enough tokens on certain arcs.

Assume initially there are enough tokens on each arc.

- So we may freely fire any actor for certain amount of times.
- If we can find a scheduling to fire the actors such that the number of tokens will remain unchanged afterwards, then we know there is no deadlock if we repeat such scheduling.

Also known as the relative execution rates of actors.
Can we prevent deadlock?

- Not for all SDFs, but we can decide if there will be a deadlock.
- Deadlock only happens when there is not enough tokens on certain arcs.

Assume initially there are enough tokens on each arc.

- So we may freely fire any actor for certain amount of times.

If we can find a scheduling to fire the actors such that the number of tokens will remain unchanged afterwards, then we know there is no deadlock if we repeat such scheduling.

- Also known as the relative execution rates of actors.
Determine the Relative Execution Rates

- Assigning each process an unknown representing the times the actor should be fired.
  - For each arc, writing down an equation requiring the number of produced tokens to be equal to that of the consumed tokens.
    - Recall we have assumed there will be enough tokens initially.
  - Now you have a system of linear equations.
    - Though there are usually more equations (number of arcs) than unknowns (number of nodes).
    - 0 is a solution but we are looking for a solution other than that.
    - If no such solution exists, we know deadlock or overflow eventually happens.
  - Otherwise, we can find a minimum integer solution as the relative execution rates.
Determine the Relative Execution Rates

- Assigning each process an unknown representing the times the actor should be fired.
- For each arc, writing down an equation requiring the number of produced tokens to be equal to that of the consumed tokens.
  - Recall we have assumed there will be enough tokens initially.
- Now you have a system of linear equations.
  - Though there are usually more equations (number of arcs) than unknowns (number of nodes).
  - 0 is a solution but we are looking for a solution other than that.
  - If no such solution exists, we know deadlock or overflow eventually happens.
  - Otherwise, we can find a minimum integer solution as the relative execution rates.
Determine the Relative Execution Rates

- Assigning each process an unknown representing the times the actor should be fired.
- For each arc, writing down an equation requiring the number of produced tokens to be equal to that of the consumed tokens.
  - Recall we have assumed there will be enough tokens initially.
- Now you have a system of linear equations.
  - Though there are usually more equations (number of arcs) than unknowns (number of nodes).
  - 0 is a solution but we are looking for a solution other than that.
  - If no such solution exists, we know deadlock or overflow eventually happens.
  - Otherwise, we can find a minimum integer solution as the relative execution rates.
Our scheduling as relative execution rates doesn't specify the order to fire those actors.  
  
  ▶ This leaves great flexibility in determining an order of firing.  
  ▶ Different orders may require different resources to complete one round of firings.  
    ▶ Number of initial tokens on each arc  
    ▶ Sizes of the queues  
  ▶ Tools may help you and it could even be possible to share the memory for queues.
Our scheduling as relative execution rates doesn’t specify the order to fire those actors.

- This leaves great flexibility in determining an order of firing.

Different orders may require different resources to complete one round of firings.

- Number of initial tokens on each arc
- Sizes of the queues

Tools may help you and it could even be possible to share the memory for queues.
Implementation Considerations for SDF

- Our scheduling as relative execution rates doesn't specify the order to fire those actors.
  - This leaves great flexibility in determining an order of firing.
- Different orders may require different resources to complete one round of firings.
  - Number of initial tokens on each arc
  - Sizes of the queues
- Tools may help you and it could even be possible to share the memory for queues.
Processes and communications are critical for process-based models.

By restricting how processes execute and how communications happen, we may obtain models that guarantee no-deadlock while still flexible enough for different implementations.