Embedded System Design and Modeling

EE382V, Fall 2008

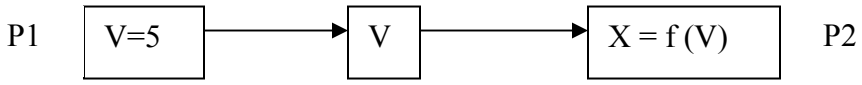
Lecture Notes 2

Models of Computation, Languages

Dates: Sep 2&4, 2008
Scribe: Mahesh Prabhu

Models of Communication:

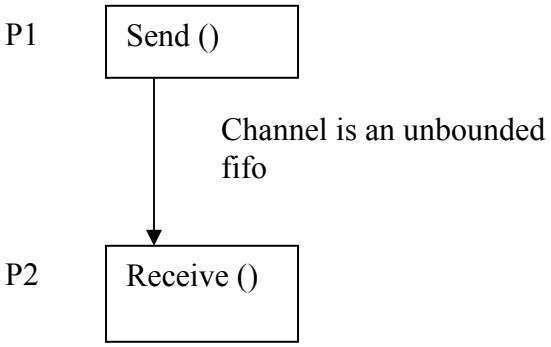
Synchronization is required when 2 processes are accessing the same variable with at least one of the processes is manipulating the variable.



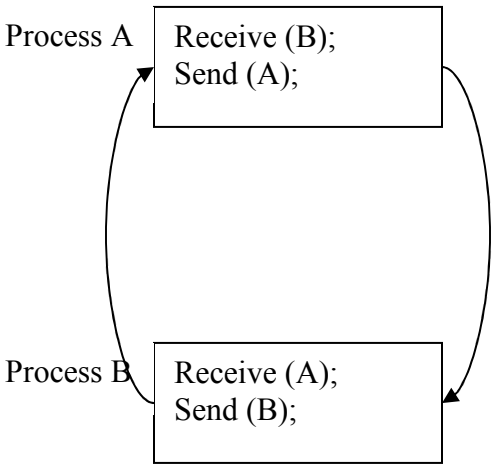
The programming paradigm of critical section is one mechanism used for synchronization. Critical section is a piece of code that accesses a shared resource (data structure or device) that must not be concurrently accessed by more than one thread of execution.

Inter-process Communication:

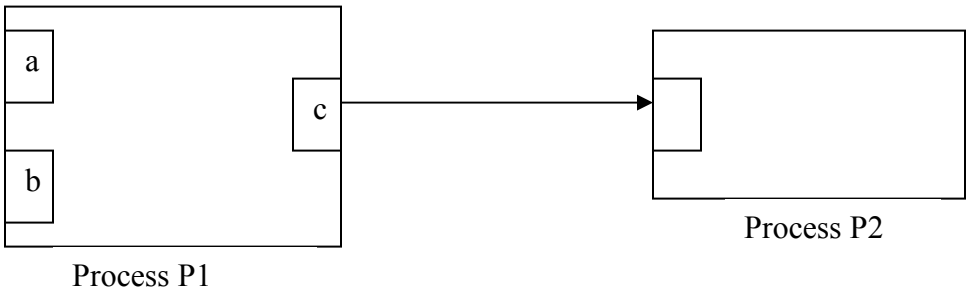
Non-blocking send and receive is achieved through unbounded queues.



Blocking send for message passing might result in deadlocks.

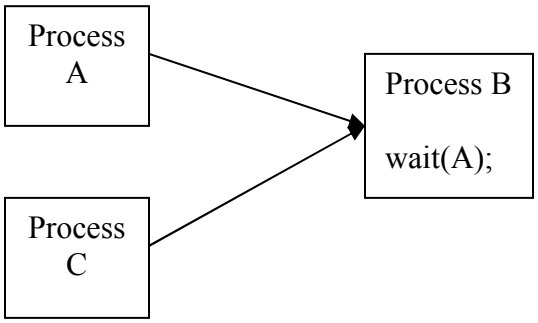


Kahn Process Networks:




```
process P1( in int a, in int b, out int c) {
        int x = 1;
        if ( x )
                wait (a);
        else
                wait (b);
        send (c);
}
```

Three aspects of a communication system schedule:
(1) Termination: under what conditions will the system stop executing.
(2) Boundedness: what are the sizes of the queues required for communication(bounded or unbounded?).
(3) Completeness: will all the processes execute.



In above example a scheduling question would be weather process ‘C’ needs to be executed at all cause process ‘B’ does not use the data sent to it by ‘C’.

A complete schedule might at times need to be unbounded.

Types of Scheduling:

(1) Data Driven Scheduling
(2) Demand(Event) Driven Scheduling

A Kahn Process Network(KPN) terminates when ALL the processes are either waiting or have completed execution i.e a global deadlock or when all processes have exited normally.

A KPN needs context switch mechanism (eg. When we have to wait and resume).

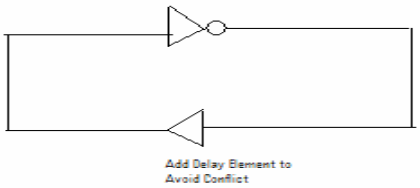
A Data Flow Network(DFN) avoids context switches. A DFN has the following:

- queues are unbounded
- actors "fire" when they have all the required data available at the queues.

The fact that DFN avoids context switches implies that there is no need for a true OS. A single actor will start executing only when all the data required to it is available and stops only when all the computation is complete.

Each process in KPN can be broken into actors to get a DFN. Essentially a DFN is a special case of a KPN.

Synchronous Data Flow Networks(SDF) also have problem with deadlocks, to avoid this we need to initialize the communication channels.



To get a schedule we need to solve a system of equations obtained from the token requirement of each actor.
Consider the SDF in Lecture-2 slide 14. We can have the following equations based on the data requirements of each process:

$2a = b$
$2b = c$
$b = d$
$2d = c$
$2a = d = 1/2c = b$
$4a = 2d = c = 2b$

Possible Schedules :
(1) adbdbcccc
(2) a(2db)(4c)
(3) adbccdbcc

Each schedule would have different memory requirements. For example the memory requirement for 1st schedule given above is 12.

A schedule in which every function appears exactly once is called a single appearance schedule.
Inconsistent SDFN would have a system of unsolvable equations.

Software tools like LabView, Matlab, Simulink user SDFNs.

Process Calculi:
Consider two processes P1 and P2:-

P1 = a -> Φ , process P1 waits for event 'a' before it completes.

P2 = (a-> Q) | (b -> N), process P2 waits for event 'a' before it proceeds to execute Q, or it executes N if event 'b' occurs.

P1 || P2 = (a -> Φ) || (a-> Q) | (b -> N)
= Q

State Machines:

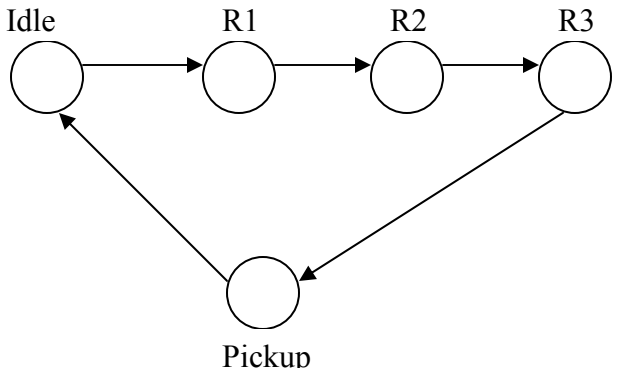
Definition:
S : Set of States, I : Set of Inputs, O : Outputs, f : S x I -> S, h : output function.

Mealy : S x I -> O
Moore : S -> O

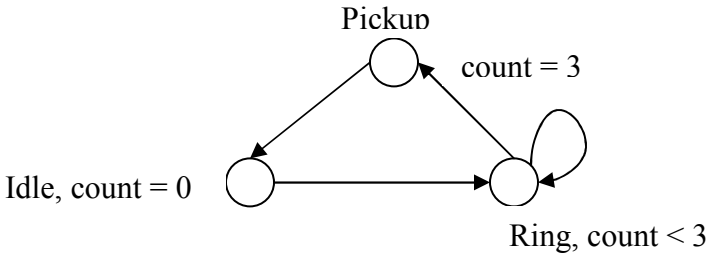
A simple traffic light controller is an example of an FSM.

Enumerating States Vs Super-States

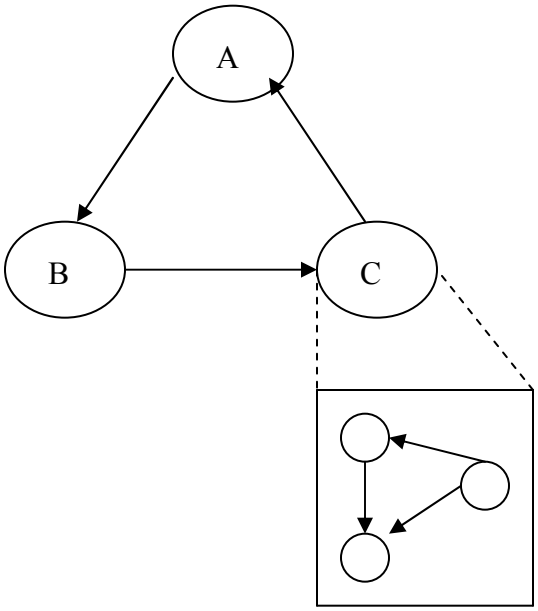
Consider an example of a phone which is allowed to "ring" 3 times before we pick it up. An FSM with all states enumerated will look as follows:-



FSM with a super-state



In the case of Hierarchical FSM's we can have a single state representing an entire FSM.



An FSM can be easily converted into a C program using goto's or the switch-case statements.

eg:

```
switch ( current_state ) {
        case IDLE:
                next_state = RUN;
        ...
}
```

Converting a C-program into an FSM is a much harder task as the states in the program cannot be easily identified.

We can have FSMs with OR hierarchy or AND hierarchy.
In the case of OR hierarchy FSM, there will be a single FSM and the current state of the FSM can be represented with a single state.

In the case of AND hierarchy FSM, we will have 2 or more FSMs running in parallel and the current state of the hierarchy will be represented by a tuple of current states of each of FSMs.