RW778 Concurrent Programming

Lecture 1: Introduction

Outline

1. What is concurrent programming?
2. Examples of concurrent programs
3. Parallel & distributed systems
4. Synchronization
5. Safety & liveness
6. Program properties
7. Formal logical systems
8. Predicates & states
9. Programming logic
10. Axioms
11. Inference rules
12. Weakest preconditions

What is concurrent programming?

Concurrent programming is the activity of constructing a program containing multiple processes that execute in parallel and cooperate to perform a task.

Examples of concurrent programs

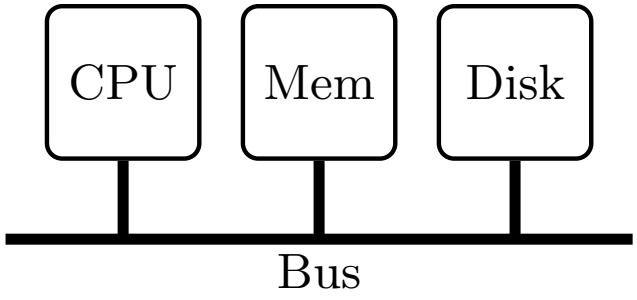
- Operating systems,
- window systems on personal computers and workstations,
- transaction processing in multiuser database systems,
- file servers in a network,
- scientific computations that manipulate large arrays of data,
- aircraft control systems,
- electronic vehicle systems,
- SETI@home,
- and others???

Parallel & distributed systems

In this course we'll distinguish two main classes of concurrent programs:

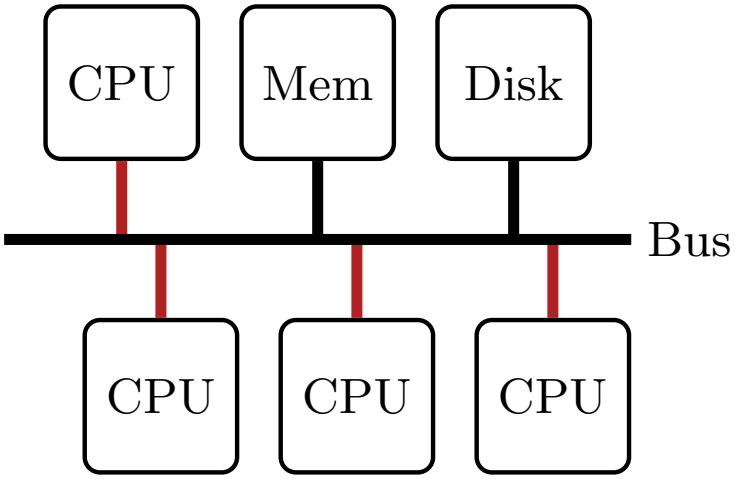
- *parallel systems* typically execute on a (single) multiprocessor machine and share (parts of) the same memory space or communicate via a high-speed bus;
- *distributed programs* typically execute on separate machines (that are capable of operating on their own) and communicate via a (relatively) low-speed data network.

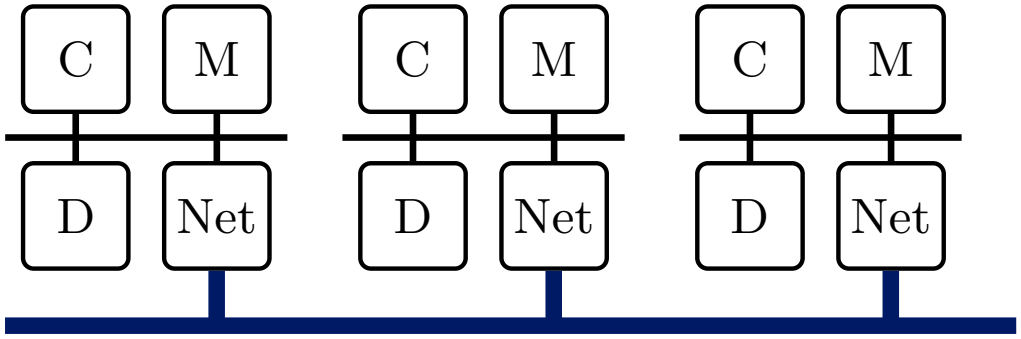
Our classification of parallel and distributed systems is not perfect. It is not always clear when a system is parallel and when it is distributed. It can even be both at the same time.



The simplest computer has a CPU and memory connected by a bus. (Example peripheral: disk.)

A four-processor parallel machine with shared memory and peripherals.





A three-processor distributed machine connected by a data communication network.

Synchronization

- Concurrent processes cooperate by communicating.
- They communicate using shared variables or message passing.
- Communication gives rise to the need for *synchronization*.
- Two forms of synchronization occur in concurrent programs:
  1. *mutual exclusion*, and
  2. *condition synchronization*.
- Mutual exclusion ensures that *critical sections* of different programs that access shared variables are not executed at the same time.
- Condition synchronization ensures that a process waits until a given condition is true.

Safety & liveness

- A property of a program is an attribute that is true of every history of that program, and therefore true of all executions of the program.
- A *safety property* asserts that the program never enters a bad state. "Nothing bad will ever happen." Examples?
- A *liveness property* asserts that the program will eventually reach a good state. "Something good will eventually happen." Examples?

Program properties

- *Partial correctness* is an example of a safety property. It asserts that, if the program terminates, the final state is correct, i.e., the right result has been computed.
- *Termination* is an example of a liveness property. It asserts that the program will finally terminate, i.e., every history of the program is finite.
- *Total correctness* is the combination of partial correctness and termination. It asserts that a program always terminates with a correct answer.

Formal logical systems

- A formal logical system consists of
  - a set of *symbols*,
  - a set of *formulas* constructed from the symbols,
  - a set of special formulas called *axioms*, and
  - a set of *inference rules*.
- Inference rules have the form

$$\frac{H_1, H_2, \ldots, H_n}{C}$$

  where H_i are called *hypotheses*, and C is called the *conclusion*.
- A *proof* is a sequence of lines, each is an axiom, or can be derived from previous lines by applying an inference rule.
- A *theorem* is any line in a proof.

Formal logical systems

- By itself a formal logical system is a mathematical abstraction: it is only a collection of symbols and relations between them.
- A logical system becomes interesting when we introduce a mapping between the formulas and true and false. Such a mapping is called an *interpretation*.
- *Soundness*:
  - A logic is *sound* with respect to an interpretation if all its axioms and inference rules are sound.
  - An axiom is sound if it maps to true.
  - An inference rule is sound if its conclusion maps to true, assuming that all its hypotheses map to true.
  - In other words, if a logic is sound, all theorems in the logic are true statements about the domain of discourse.

  In this case the interpretation is called a *model* for the logic.

Formal logical systems

- *Completeness*:

  A logic is *complete* with respect to an interpretation if every formula that is mapped to true, is a theorem.

- If FACTS is the set of true statements that are expressible as formulas, and THEOREMS is the set of theorems of the logic (what we can prove), then

  soundness means THEOREMS $\subseteq$ FACTS; and

  completeness means FACTS $\subseteq$ THEOREMS.

- Gödel's Incompleteness Theorem: if a formal logical system is powerful enough to make statements about arithmetic, it is incomplete. This means that there is a fact $f \in$ FACTS such that $f \notin$ THEOREMS.

- Unfortunately, the domain of concurrent programs includes arithmetic.

- However, a logic that extends another logic can be *relatively complete*, meaning that it doesn't introduce any extra source of incompleteness.

Propositional logic

- Symbols:
  - constants: true, false
  - variables: $p, q, r, \ldots$
  - operators: $\neg, \wedge, \vee, \Rightarrow, =$
- Formulas:
  - constant, variable, $\neg P$, $P \wedge Q$, $P \vee Q$, $P \Rightarrow Q$, $P = Q$
- Axioms:
  - true

Propositional logic

Law of Negation:	$P = \neg(\neg P)$	
Law of Excluded Middle:	$P \vee \neg P = \text{true}$	
Law of Contradiction:	$P \wedge \neg P = \text{false}$	
Law of Implication:	$P \Rightarrow Q = \neg P \vee Q$	
Law of Equality:	$(P = Q) = (P \Rightarrow Q) \wedge (Q \Rightarrow P)$	
Laws of Or/And-Simplification:	$P \vee P = P$	$P \wedge P = P$
	$P \vee \text{true} = \text{true}$	$P \wedge \text{true} = P$
	$P \vee \text{false} = P$	$P \wedge \text{false} = \text{false}$
	$P \vee (P \wedge Q) = P$	$P \wedge (P \vee Q) = P$
Commutative Laws:	$(P \wedge Q) = (Q \wedge P)$	
	$(P \vee Q) = (Q \vee P)$	
	$(P = Q) = (Q = P)$	
Associative Laws:	$P \wedge (Q \wedge R) = (P \wedge Q) \wedge R$	
	$P \vee (Q \vee R) = (P \vee Q) \vee R$	
Distributive Laws:	$P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$	
	$P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$	
De Morgan's Laws:	$\neg(P \wedge Q) = \neg P \vee \neg Q$	
	$\neg(P \vee Q) = \neg P \wedge \neg Q$	

Predicates & states

- Propositional logic is not powerful enough to reason accurately about programs. The only propositional constants are true and false, while programming languages of course have additional data types and values such as integers and reals. Predicate logic extends propositional logic to support this more general problem.
- First extension: any expression such as $x < y$ that maps to true or false can be used in the place of a propositional variable.
- Second extension: existential ($\exists$) and universal ($\forall$) quantifiers are provided to characterize sets of values.

Predicates & states

Examples:

- $(\exists i : 1 \leq i \leq n : a[i] = 0)$
- $i = j \wedge (\exists i, k : i, j > 0 : a[i] = a[k])$
- Array $a[\,]$ is sorted?

Predicates & states

Each predicate corresponds to a *set of states* of the program.

But predicates still only allow us to talk about individual states. How can we talk about the relationship between different states?

We need to define a programming logic.

Programming logic

- Symbols:
  - predicates, formulas in predicate logic
  - braces
  - programming language statements
- Formulas: $\{P\}\ S\ \{Q\}$

Interpretation of a triple: Triple $\{P\}\ S\ \{Q\}$ is true if, whenever execution of S is begun in a state satisfying P and execution of S terminates, the resulting state satisfies Q.

Axioms

- **Skip Axiom:** $\{P\}$ **skip** $\{P\}$
- **Assignment Axiom:** $\{P_e^x\}\ x := e\ \{P\}$
- **Swap Axiom:** $\{P_{v2,v1}^{v1,v2}\}\ v1 :=: v2\ \{P\}$

Inference rules

- **Rule of Consequence:** $$\frac{P' \Rightarrow P,\ \{P\}\ S\ \{Q\},\ Q \Rightarrow Q'}{\{P'\}\ S\ \{Q'\}}$$

- **Composition Rule:** $$\frac{\{P\}\ S_1\ \{Q\},\ \{Q\}\ S_2\ \{R\}}{\{P\}\ S_1; S_2\ \{R\}}$$

- **Alternative Rule:** $$\frac{\begin{array}{c} P \wedge \neg(B_1 \vee \cdots \vee B_n) \Rightarrow Q \\ \{P \wedge B_i\}\ S_i\ \{Q\},\ 1 \leq i \leq n \end{array}}{\{P\}\ IF\ \{R\}}$$

- **Iterative Rule:** $$\frac{\{I \wedge B_i\}\ S_i\ \{I\},\ 1 \leq i \leq n}{\{P\}\ DO\ \{R\}}$$

  IF : **if** $B_1 \rightarrow S_1\ []\ \ldots\ []\ B_n \rightarrow S_n$ **fi**

  DO : **do** $B_1 \rightarrow S_1\ []\ \ldots\ []\ B_n \rightarrow S_n$ **od**

Example: Linear Search

- $P : n > 0 \wedge (\exists j : 1 \leq j \leq n : a[j] = x)$
- $LS : a[i] = x \wedge (\forall j : 1 \leq j < i : a[j] \neq x)$
- Implicit property: n, $a[\,]$, and x should not change.
- Theorem: $\{P\}$

  $i := 1$

  do $a[i] \neq x \rightarrow i := i + 1$ **od**

  $\{LS\}$

Example: Linear Search

1. $\{P \wedge 1 = 1\}$ by Assignment Axiom
   $i := 1$
   $\{P \wedge i = 1\}$

2. $(P \wedge 1 = 1) = P$ by Predicate Logic

3. $\{P\}$ by Rule of Consequence with 1 & 2
   $i := 1$
   $\{P \wedge i = 1\}$

Example: Linear Search

4. $\{P \wedge (\forall j : 1 \leq j \leq i+1 : a[j] \neq x)\}$ by Assignment Axiom
   $i := i + 1$
   $\{I : P \wedge (\forall j : 1 \leq j \leq i : a[j] \neq x)\}$

5. $(I \wedge a[i] \neq x) =$ by Predicate Logic
   $(P \wedge (\forall j : 1 \leq j \leq i+1 : a[j] \neq x))$

6. $\{I \wedge a[i] \neq x\}$ by Rule of Consequence with 4 & 5
   $i := i + 1$
   $\{I\}$

Example: Linear Search

7. $\{I\}$ by Iterative Rule with 6

   do $a[i] \neq x \rightarrow i := i + 1$ **od**

   $\{I \wedge a[i] = x\}$

8. $(P \wedge i = 1) \Rightarrow I$ by Predicate Logic

9. $\{P\}$ by Composition Rule with 7 & 8

   $i := i$

   do $a[i] \neq x \rightarrow i := i + 1$ **od**

   $\{I \wedge a[i] = x\}$

Example: Linear Search

10. $(I \wedge a[i] = x) \Rightarrow LS$ by Predicate Logic

11. $\{P\}$ by Rule of Consequence with 9 & 10

$i := i$

do $a[i] \neq x \rightarrow i := i + 1$ **od**

$\{LS\}$

Weakest preconditions

Weakest Precondition: The weakest preconsition of a statement S and predicate Q, denoted $wp(S, Q)$, is a predicate characterizing the largest set of states such that, if execution of S is begun in any state satisfying $wp(S, Q)$, then execution is guaranteed to terminate in a state satisfying Q.

- Relationship of wp to our programming logic: if $P \Rightarrow wp(S, Q)$, then $\{P\}S\{Q\}$ is a theorem.
- Law of the excluded Micracle: $wp(S, \text{false}) = \text{false}$
- Distributive Law of Conjunction: $wp(S, Q) \wedge wp(S, R) = wp(S, Q \wedge R)$
- Distributive Law of Disjunction: $wp(S, Q) \vee wp(S, R) \Rightarrow wp(S, Q \vee R)$

Weakest preconditions

- $wp(\mathbf{skip}, Q) = Q$
- $wp(x := e, Q) = Q^x_e$
- $wp(v1 :=: v2, Q) = Q^{v1,v2}_{v2,v1}$
- $wp(S_1; S_2, Q) = wp(S_1, wp(S_2, Q))$
- $wp(IF, Q) = \neg(B_1 \vee \ldots \vee B_n) \Rightarrow Q \wedge$
  $(B_1 \Rightarrow wp(S_1, Q) \wedge \ldots \wedge B_n \Rightarrow wp(S_n, Q))$
- $wp(DO, Q) = (\exists k : 0 \leq k : H_k(Q))$, where
  $H_0(Q) = \neg BB \wedge Q$
  $H_k(Q) = H_0(Q) \vee wp(IF, H_{k-1}(Q))$

  $BB : B_1 \vee \ldots \vee B_n$

  $DO : \mathbf{do}\ BB \rightarrow IF : \mathbf{if}\ B_1 \rightarrow S_1\ []\ \ldots\ []\ B_n \rightarrow S_n\ \mathbf{fi}\ \mathbf{do}$