Contents

BASIC CONCEPTS IN CONCURRENCY **3**
Definitions . . . 3
A sequence of actions . . . 3
process . . . 3
Threads, . . . 3
in specific languages . . . 3
Some authors . . . 3
Execution Order . . . 3
atomically, . . . 4
nondeterminism. . . . 4
A program is deterministic if, . . . 4
a program is nondeterministic if . . . 4
makes it difficult . . . 4
mechanisms . . . 4
communication between processes, . . . 4
coordination between processes, . . . 4
atomicity, . . . 4
message-passing mechanisms: . . . 4
Buffering. . . . 4
unbufferedcommunication,adata . . . 4
Synchronicity. . . . 5
With asynchronous communication, . . . 5
Message Order. . . . 5
If a mechanism preserves transmission order, . . . 5
An action is atomic . . . 5
A nonatomic action . . . 5
any concurrent programming language . . . 5
Mutual Exclusion . . . 5
for certain operations, . . . 5
critical section. . . . 5
mechanisms. . . . 6
Mutual exclusion: . . . 6
Progress: . . . 6
Bounded waiting: . . . 6
A general approach . . . 6
Using a loop to wait for some condition is called . . . 6
Deadlock . . . 6
technique that prevents deadlock . . . 6
two-phase locking, . . . 6
Semaphores . . . 7
A standard semaphore . . . 7
is represented . . . 7
Initially, . . . 7
The maximum . . . 7

If a program contains several shared resources, ... 7
When a process waits on a semaphore, ... 7
If the value is greater than 0, ... 7
If a wait is executed on a semaphore whose integer value is 0, ... 7
Suspending a process 7
When a process leaves a critical section, ... 7
If the queue is not empty, ... 7
If no process is waiting, ... 7
Monitors 8
Monitors, ... 8
A monitor consists of 8
local data 8
In traditional terminology, ... 8
In modern terminology, ... 8

THE ACTOR MODEL **8**
Each actor 8
three basic actions that an actor may perform: 8
Actor computation 8
an actor program 9
An actor is dormant until 9
When an actor receives a message, ... 9
After executing its script, ... 9
The replacement behavior 9
In any computation, ... 9
messages 9
the mail system, ... 9
Every message must be sent to 9
When an actor A specifies a replacement behavior, ... 9
A message from one actor to another 9
A task has three parts: 10
a behavior may be defined as a function of parameters, which are called 10
actor-specific concept 10
forwarder. 10
General aspects of actors 10
An actor changes state 10
The motivation for using asynchronous communication 10
problems 10

CONCURRENT ML **10**
Threads and Channels 10
A CML process is called 10
When a CML program begins, ... 10
When spawn f is evaluated, ... 10
The return value of the function 11

The thread that evaluates spawn f is called 11
the thread running f() is called 11
the parent–child relation 11
Another CML primitive, ... 11
is forever. 11
channel. 11
operations on channels 11
CML message passing 11
Channels are created by 11
Functions Using Threads and Channels 11
A guarded command, ... 12
The intent 12
First-Class Synchronous Operations: Events 12
we can decompose send into two parts: 12
Selective Communication with Events 12
The select operator 12
The function 12

BASIC CONCEPTS IN CONCURRENCY

Definitions

A sequence of actions

may be called a process, thread, or task.

process

is often used to refer to an operating system process, which generally runs in its own address space.

Threads,

in specific languages

like Concurrent ML and Java, may run under control of the language run-time system, sharing the same operating system address space.

Some authors

define thread to mean **"lightweight process,"** which means a process that does not run in a separate operating system address space.

Execution Order

Concurrent programs may have many possible execution orders.

atomically,

nondeterminism.

A program is deterministic if,

for each sequence of program inputs, there one sequence of program actions and resulting outputs.

a program is nondeterministic if

there is more than one possible sequence of actions corresponding to a single input sequence.

makes it difficult

to design and debug programs.

It is difficult to think carefully about millions of possible execution orders

mechanisms

communication between processes,

achieved by mechanisms such as buffered or synchronous communication channels, broadcast, or shared variables or objects,

coordination between processes,

which may explicitly or implicitly cause one process to wait for another process before continuing,

atomicity,

which affects both interaction between processes and the handling of error conditions.

message-passing mechanisms:

Buffering.

If communication is buffered, then every data item that is sent remains availableuntil it is received.In

unbufferedcommunication,adata

itemsentbefore the receiver is ready to accept that it may be lost.

Synchronicity.

In synchronous communication, the sender cannot transmit a data item unless the receiver is ready to receive it.

With asynchronous communication,

the sending process may transmit a data item and continue executing even if the receiver is not ready to receive the data.

Message Order.

A communication mechanism may preserve transmission order or it may not.

If a mechanism preserves transmission order,
then a sequence of messages will be received in the order that they are sent.

An action is atomic

if every execution will either complete successfully or terminate in a state that is indistinguishable from the state in which the action was initiated.

A nonatomic action

may involve intermediate states that are observable by other processes. A nonatomic action may also halt in error before the entire action is complete, leaving some trace of its partial progress.

any concurrent programming language

must provide some atomic actions, because, without some guarantee of atomicity, it is extremely difficult to predict the behavior of any program.

Mutual Exclusion

for certain operations,

it is important to restrict concurrency so that only one process proceeds at a time.

critical section.

A critical section is a section of a program that accesses shared resources. Because a process in a critical section may disrupt other processes, it may be important to allow only one process into a critical section at a time.

mechanisms.

Mutual exclusion:
Only one process at a time may be in its critical section.

Progress:

If no processes are in their critical section and some process wants to enter a critical section, it becomes possible for one waiting process to enter its critical section.

Bounded waiting:

no waiting process should have to wait indefinitely.

if one process halts in a critical section, other processes will eventually be able to enter the critical section.

A general approach

The main idea is that each process executes some kind of **wait** action before and executes some kind of signal action afterward.

Using a loop to wait for some condition is called

busy waiting.

Deadlock

occurs if a process can never continue because the state of another process. processes are waiting for the lock held by the other process. In this situation, neither process can proceed and deadlock has occurred.

technique that prevents deadlock

two-phase locking.

two-phase locking,

a process is viewed as a sequence of independent tasks. For each task, the process must first acquire all locks that are needed to perform the task. Before proceeding from one task to the next, a process must release all locks.

Semaphores

A standard semaphore

is represented

by an integer variable, an integer maximum, and a queue of waiting processes.

Initially,
the integer variable is set to the maximum.

The maximum
indicates the number of processes that may enter a critical section at the same time; in many situations the maximum is one.

If a program contains several shared resources,
the program may use a separate semaphore for each resource.

When a process waits on a semaphore,

the wait operation checks the integer value of the semaphore.

If the value is greater than 0,
this indicates that a process may proceed. The value is decremented before proceeding to limit the number of processes that are allowed to proceed.

If a wait is executed on a semaphore whose integer value is 0,

then the process is suspended and placed on a queue of waiting processes.

Suspending a process

is an operating system operation that keeps the process from continuing until the process is resumed.

When a process leaves a critical section,

The signal operation checks the semaphore queue to see if any process is suspended.

If the queue is not empty,
one of the suspended waiting processes is allowed to run.

If no process is waiting,
then the integer value of the semaphore is incremented.

Monitors

Monitors,

place the responsibility for synchronization on the operations that access data. Monitors are similar to abstract data types, with all synchronization placed in the operations of the data type. This makes it easier to write correct client code.

A monitor consists of

one or more procedures, an initialization sequence, and local data.

local data
are accessible only by the monitor procedures, which are responsible for all synchronization associated with concurrent access to the data.

In traditional terminology,

a process **enters** the monitor by invoking one of its procedures.

In modern terminology,

a monitor might be called a **synchronized object.**

THE ACTOR MODEL

Each actor

is a form of reactive object, executing some computation in response to a message and sending out a reply when the computation is done. Actors do not have any **shared state,** but use buffered asynchronous message passing for all communication.

three basic actions that an actor may perform:

It may send communication to itself or other actors,

It may create actors,

It may specify a replacement behavior,

which is essentially another actor that takes the place of the actor that creates it for the purpose of responding to later communication.

Actor computation

is reactive, which means that computation is performed only in response to communication.

an actor program

creates some number of actors and sends them messages. All of these actors can react to messages concurrently, but there is no explicit concept of thread.

An actor is dormant until

it receives communication.

When an actor receives a message,

the script of the actor may specify subsequent communication and a set of new actors to create.

After executing its script,

the actor returns to its dormant state.

The replacement behavior

specifies how the actor will behave when it receives another message.

In any computation,

each actor receives a linearly ordered sequence of messages.

messages

are not guaranteed to arrive in the order in which they are sent.

the mail system,

routes and buffers messages between actors.

Every message must be sent to

a mail address; one actor may communicate with another if it knows its mail address.

When an actor A specifies a replacement behavior,

the replacement behavior will be the script for a new actor that receives all messages addressed to the mail address of A.

A message from one actor to another

is called a task.

A task has three parts:

A unique tag, distinguishing it from other tasks in the system,

A target, which is the mail address of the intended receiver,

A communication, which is the data contained in the message.

a behavior may be defined as a function of parameters, which are called

acquaintances:

actor-specific concept

forwarder.

A forwarder actor does nothing itself, except forward all tasks to another actor.

General aspects of actors

An actor changes state

only by becoming another actor after it completes processing input.

The motivation for using asynchronous communication

is that it is easily implemented on a wide-area network.

problems

message order, message delivery, and coordination between sequences of concurrent actions

CONCURRENT ML

Threads and Channels

A CML process is called

a thread.

When a CML program begins,

it consists of a single thread. This thread may create additional threads by using the **spawn** primitive:

When spawn f is evaluated,

the function call f() is executed as a separate thread.

The return value of the function

is discarded, but the function may communicate with other threads by using channels.

The thread that evaluates spawn f is called

the parent thread

the thread running f() is called

the child thread.

the parent–child relation

does not have any impact on thread execution either thread may terminate without affecting the other.

Another CML primitive,

is forever.
This function takes an initial value and a function that can be repeatedly applied to this value:

channel.

For each CML type 'a, there is a type 'a chan of channels that communicate values of type 'a.

operations on channels

recv : 'a chan → 'a → send : ('a chan * 'a) → unit

CML message passing

is synchronous, meaning that communication occurs only when both a sender and a receiver are ready to communicate.

Channels are created by

the channel constructor,

channel : unit → 'a chan

Functions Using Threads and Channels

Because channels are "just another type" in CML, we can define functions that take channel arguments and return channel results.

A guarded command,

has historically been written in the form

Condition $\Rightarrow$ Command

consists of a predicate and an action.

The intent

is that, if the Condition is true, then the Command may be executed.

First-Class Synchronous Operations: Events

we can decompose send into two parts:

the code that will cause a **send** to occur is the **send event,** and doing the send is called **synchronizing on this event.**

Selective Communication with Events

The select operator

for selective communication can be defined from a primitive function on events that chooses an event out of a list.

The function

choose : 'a event list $\rightarrow$ 'a event

takes a list of events and returns one event from the list. choose must return an event that can be synchronized on if there is one in the list.