Concurrent programs wait faster

Tony Hoare

Waiting for a bus

Let *P* mean: wait for a bus on route nr 9.

The average wait is 28 mins.

Let *Q* mean: wait for a bus on route nr 7.

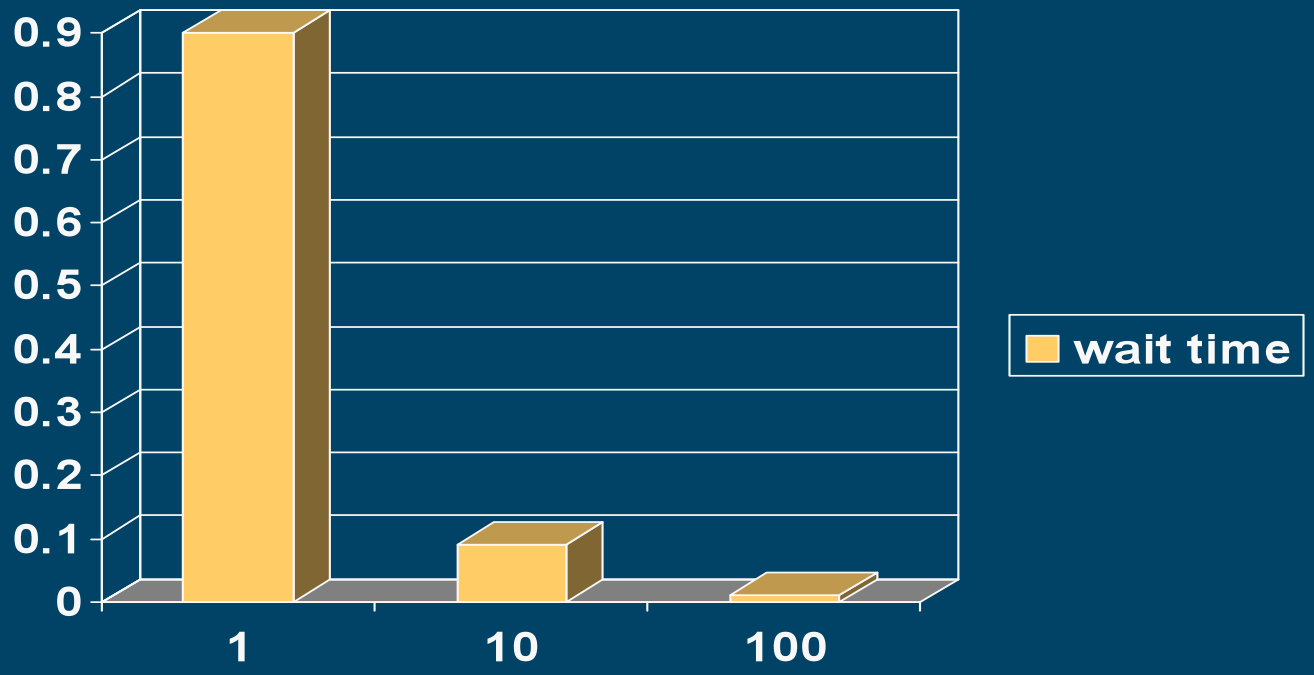
The average wait time is 28 mins

(P first Q) means: wait for the first of the two buses

The average wait reduces to 11 mins

Delay probabilities (say)

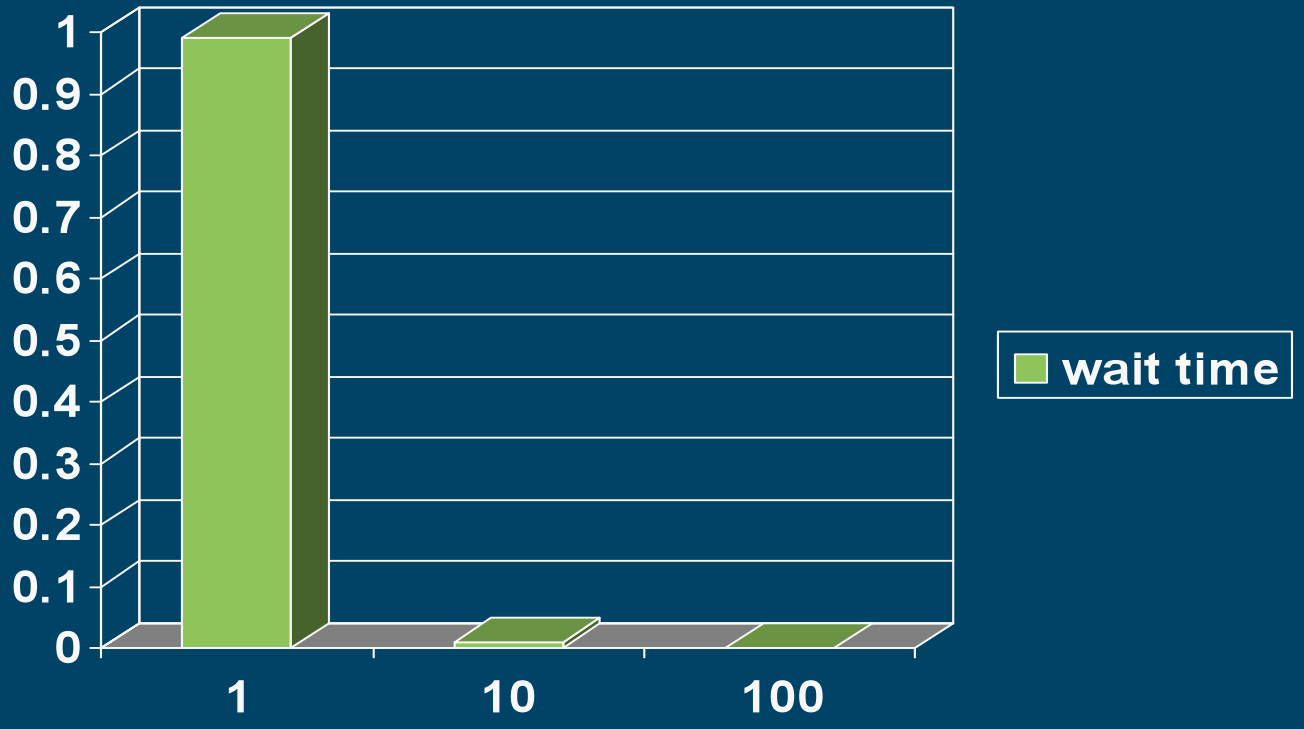
average: **.9 + .9 + 1 = 2.8**



Let *P, Q, R* be so distributed, independently.

Delay of *first*

average: **.99 + .099 + .01 = 1.099**



An event

is an instantaneous or atomic action, that may occur after a delay; it either completes or it fails cleanly without effect (as though it never starts), e.g.,

an input, an output,

an alarm-call, a lock acquisition

…perhaps an RPC or a transaction

Let *P, Q, R* stand for such atomic events.

Examples

```
Sleep (3 * seconds) ;

Scanf (“%d”, &X ) ;

WaitForSingleObject (p, INFINITE) ;
```

Implementation of *first*

```
HANDLE temp [2]  =  { p,q };
if (WAIT_OBJECT_0 ==
WaitForMultipleObjects
          ( 2, temp, false, INFINITE ))
    {/*..code to deal with p..*/}
else {/*..code to deal with q..*/}
```

Sockets implementation

```
fd_set temp;
FD_ZERO(&temp);
FD_SET(p, &temp);
FD_SET(q, &temp);
select(0, &temp, NULL, NULL, NULL);
if (FD_ISSET(p, &temp))
    {/*..read from socket p..*/}
else {/*.. read from socket q..*/}
```

Waiting for both busses

Let *P* mean: wait for a bus on route nr 9.

The average wait is 28 mins.

Let *Q* mean: wait for a bus on route nr 7.

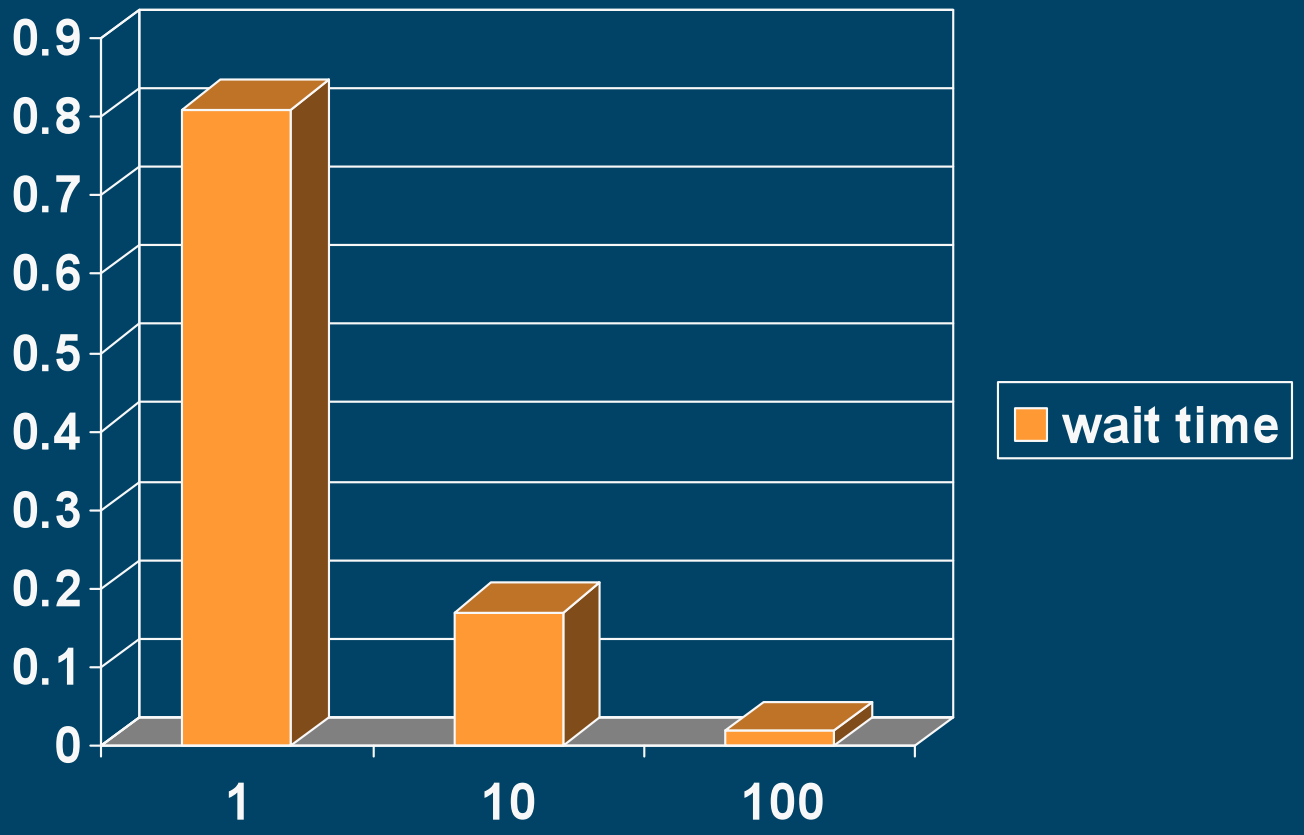
The average wait is 28 mins

(P both Q) means: wait for both busses,
in either order.

The average wait goes up to 45 mins

Delay of *both*

average: .81 + 1.701 + 1.99 = 4.501



Implementation of *both*

```
HANDLE temp [2]  =  { p, q } ;
WaitForMultipleObjects
    ( 2, temp, true, INFINITE) ;
{/*.. code to deal with p and with q..*/}
```

Sockets implementation

```
/* implements  p both q */
  fd_set temp;
  FD_ZERO(&temp);
  FD_SET(p, &temp); FD_SET(q, &temp);
  select(0,&temp, NULL,NULL,NULL);
  select(0,&temp,NULL,NULL,NULL);
  {/*..read from q..*/}
  {/*..read from p..*/}
```

One bus after another

Let *P* mean: wait for a bus on route nr 9.

The average wait is 28 mins.

Let *Q* mean: wait for a bus on route nr 7.

The average wait is 28 mins

(P then Q) means wait for P first,
and then wait for Q.

The average wait goes up to 56 mins

Implementation of *then*

{ P then Q } = P must happen first,
followed by *Q*

…implemented often as *{ P ; Q }*

Commitment

(P then Q) first (R then S)

commits after the first of *P* and *R*.

Bad case:

Atomicity

P both Q = { P then Q } first { Q then P }

because *P* and *Q* are atomic

Implementation

```
/* implements( P then Q ) first ( R then S ) */
HANDLE temp [2] = { p, r };
DWORD whichone  = WaitForMultipleObject
            ( 2, temp, false, INFINITE );
if (whichone == WAIT_OBJECT_0)
      {/*..deal with p..*/
      WaitForSingleObject( q, INFINITE );
      /*..deal with q..*/
      }
else {/*..deal with r..*/
      WaitForSingleObject ( s, INFINITE);
      /*..deal with s..*/
      }
```

Sockets implementation

```
/* implements (P then Q) first (R then S). */
fd_set temp;
FD_ZERO(&temp);
FD_SET(p,&temp); FD_SET(r,&temp);
select(0,&temp,NULL,NULL,NULL);
if(FD_ISSET(p,&temp))
  {FD_ZERO(&temp); FD_SET(q,&temp);
       {/*..read from p..*/}
       select(0,&temp,NULL,NULL,NULL);
       {/*..read from q..*/}
  }
Else {FD_ZERO(&temp);FD_SET(s,&temp);
       {/*..read from r..*/}
       select(0,&temp,NULL,NULL,NULL);
       {/*..read from s..*/}
      }
```

command	wait time
1. *P*	2.8
2. *Q*	2.8
3. *P first Q*	1.1
4. *P both Q*	4.5
5. *P then Q*	5.6

$$5.6 = 2.8 + 2.8 = 4.5 + 1.1$$

$$X + Y = (X \min Y) + (X \max Y)$$

Other distributions

	first	*both*
• Poisson	1.4	4.2
• Uniform	1.9	3.7
• Constant	2.8	2.8

Non-determinism

Let *P* mean: wait for a bus on route nr 9.

The average wait is 28 mins.

Let *Q* mean: wait for a bus on route nr 7.

The average wait is 28 mins

(P any Q) means: let the choice be made (say) by my neighbour

The average wait went up to 40 mins

Conclusion

- Identify wait-points in your program
- Replace them by concurrent waits
- Estimate probabilities for wait durations
- Protect waits by time-out
- Release resources before a wait

.....as far as you can.

Safety

- Allocate each event to a single thread to wait for
- Note what can change during the wait
- Test again the properties that matter
- ASSERT what you hope is unchanged
- …both before and after the wait.

…….May all your waits be fast and safe!