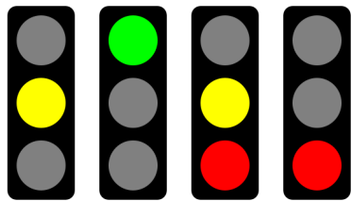


Not that blocking!

Øyvind Teig
www.teigfam.net/oyvind/home

@CPA 2015 fringe http://www.wotug.org/cpa2015/

)

Autronica Fire and Security AS, Trondheim, Norway. Autronica is a part of UTC Building & Industrial Systems, a unit of United Technologies Corporation

Maybe the other blocking?

So that this blocking isn’t?

Based on the blog note "Not so blocking after all" by Øyvind Teig.
See http://www.teigfam.net/oyvind/home/technology/092-not-so-blocking-after-all/

The blog note also contains a thread with comments from Tony Gore, Roger Shepherd, Matt Pedersen, Jon Kerridge, Marc Smith, Larry Dickson, David May, Chris Jones, Rick Beton and Ian East,

some of which I have incompletely pasted from here:

Abstract

1. Communicating to fellow programmers that the concept of "*blocking*" in process-oriented design is perfectly acceptable, while using a word with basically negative connotations, is difficult.
2. The bare need to do it *often* means that misunderstanding this concept is harmful.
3. The first contender on a "*blocking channel*" has also correctly been said (by several people) to "*wait*" on the channel.
4. A better correspondence between the negative meaning and the semantics is when "blocking" describes serious side effects on the temporal properties of other concurrent components.
5. This is the correctly feared blocking.
6. This *fringe* presentation will explore this further and invite discussion.

Historically (1/2)

Chris Jones

I suspect the term has been lifted from the ***telecoms industry*** *where telephone exchange equipment was considered to be* ***non-blocking*** ***when a caller was guaranteed always to be able to get an immediate connection to another non-busy user on a fully functioning, non-blocking exchange.*** *In the UK, local exchanges were non-blocking while trunk connections were not. I would have thought the usage referring to the possibility of sending a commu- nication was still pertinent.*

Roger Shepherd

This sounds likely and the difference between this and what happens in a (for example) occam program is interesting. ***In an occam program the communication channel is always available, however the "other party" is not. So, in the telephone sense the communication is non-blocking – it's just that the other party might not be there, and if you are making a call (output) you have have to hang on the line until the other party answers.***

Historically (2/2)

Chris Jones

When the early telephone exchanges were being devised, they were very concerned about blocking, not just from the point of view of convenience but ***mechanically****. These were* ***electro-mechanical switches*** *with the real possibility of latching up or locking mechanically in unwanted ways like typewriters when several keys were pressed too close together in time and they arms would lock together preventing further typing until they were manually released.*

Roger Shepherd

The issue in ***telephones*** *is precisely whether the switch allows another call to be established or whether some calls may be "****blocked****" because a circuit cannot be established.*

Discussion (1/5)

Tony Gore

*I think it was a **common term in use by April 1986 when I joined Inmos**. I think some terms in common use were used a bit loosely. A channel that was waiting for the other end of the communication to become ready "blocked" the process from proceeding. Thus my recollection is that "**blocking on a channel" was commonly used** to describe a process that couldn't proceed until the communication could proceed.*

Roger Shepherd

*I can't say that I like "block" – but it usage is certainly old and is common for multitask systems where **the ability to create an extra task/thread/whatever to do communication is considered to be advantageous – hence "non-blocking communication".***

*On the subject of language, I think the term **"synchronous" is plain wrong when used to describe (occam) channel communication.** The processes are "synchronised" by the communication; the communication is "asynchronous" – there is no clock which causes the communication to happen.*

Discussion (2/5)

Jon Kerridge

*Of course if we go back even further there were **semaphores** to which Dijkstra gave the names **P and V** for the operations on semaphores and as I understand it P and V were the first letters of the Dutch words for **wait** and **signal**.*

Marc Smith

*I think you have hit upon a **significant language barrier when we discuss channel communications outside of our CSP community.** Even though I have heard the phrase **"to block on a channel" I never really thought of it as blocking.***

The term blocking never bothered me because I just understood it in the sense of synch'ing.

Discussion (3/5)

David May

There seems to be **widespread confusion about 'blocking'.** The problem is that a common technique in shared memory concurrency involves 'blocking' on access to shared state and **various techniques have been devised to avoid or reduce the need for this**. Although these work for some algorithms, there is **no general technique to eliminate 'blocking'**, and many algorithms rely on it – if a process depends on (for example) the sum of the results of 1000 others (a reducing operation) it will have to 'block' until they have all completed.

All of these ***'non-blocking' techniques are more difficult to understand and verify than the blocking equivalents; also they use indivisible instructions (such as compare and swap) which have to access the shared (deep and high latency) parts of the memory hierarchy; also the hardware has to 'block' in order to implement the indivisible instructions.*** *So these techniques are probably not as efficient as they may seem*.

Discussion (4/5)

Rick Beton

For this reason and for reasons others have mentioned, I feel that it is ***wrong to describe channels with this terminology*** *because it evidently causes nuances and misunderstandings in those not very familiar with a CSP or Occam way of doing things.* ***'Waiting' on a channel communication is helpfully different. Many people would describe a zero-buffer channel as "synchronous" but an infinite-place buffer channel as "asynchronous" because the sender never waits in the latter case. This terminology is flawed in my view;*** *how do you differentiate between an infinite-buffer channel and a one-place buffer channel? The latter could behave "synchronously" or "asynchronously" depending on the current dynamic state; clearly, using synchronous and asynchronous in this way lacks rigour. Alas, it seems to be pervasive though.*

Discussion (5/5)

Ian East

*I'd then distinguish the **three known forms of synchronisation** permitting synchronous communication : **common clock** (as in current digital systems), **handshake** (used in a typical system bus) and **rendezvous**.*

David May

I have just looked through the early drafts and published versions of occam manuals.

***None of them talks about 'blocking'**. It's all about processes being '**ready**' to communicate (sometimes 'ready and waiting') to communicate. On a channel, communication takes place when 'both processes are ready'.*

*Discovered '**hang** on a channel' in an early Inmos-authored magazine article!*

Which blocking do you mean?

= waiting?

=

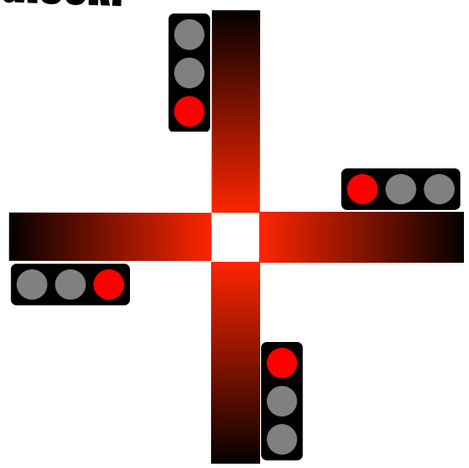
~~yielding?~~

The show goes on with this *blocking*

= blocking?

This *blocking* stops the show

= deadlock!



This *blocking* stops the world

Wikipedia

Wikipedia article on **non-blocking algorithm *was*** that:

«Literature up to the turn of the 21st century used "non-blocking" synonymously with lock-free. However, since 2003, the term has been weakened to only prevent progress-blocking interactions with a preemptive scheduler. In modern usage, therefore, an algorithm is non-blocking if the suspension of one or more threads will not stop the potential progress of the remaining threads. They are designed to avoid requiring a critical section. Often, these algorithms allow multiple processes to make progress on a problem without ever blocking each other. For some operations, these algorithms provide an alternative to locking mechanisms.»

Wikipedia (now)

Wikipedia article on **non-blocking algorithm *is*** that:

*«In computer science, an algorithm is called **non-blocking** if failure or suspension of any thread cannot cause suspension or failure of another thread; for some operations, these algorithms provide a useful alternative to traditional **b**locking operations. A non-blocking algorithm is **lock-free** if there is guaranteed system-wide progress, and **wait-free** if there also is guaranteed per-thread progress»*

Wikipedia ("guaranteed"?)

Wikipedia article on **non-blocking algorithm *is*** that:

*«In computer science, an algorithm is called **non-blocking** if failure or suspension of any thread cannot cause suspension or failure of another thread; for some operations, these algorithms provide a useful alternative to traditional **b**locking operations. A non-blocking algorithm is **lock-free** if there is* **guaranteed** *system-wide progress, and **wait-free** if there also is* **guaranteed** *per-thread progress»*

Does guarantee have to do with fulfilling specification?

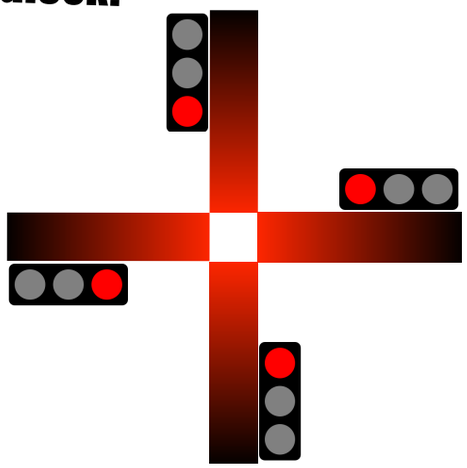
= waiting?

=

~~yielding?~~

= blocking?

= deadlock!



The show goes on with this *blocking*

This *blocking* stops the show

This *blocking* stops the world

?:

Spec of others known OK!

Spec of others not known

Spec of others unknown or known to be pathological

«The word "block" is the correct term»

golang-nuts and golang-dev

I started the thread Yielding instead of blocking on a channel? on golang-nuts on 25Sep2014. There were some interesting comments.

The final say there is of course **Rob 'Commander' Pike**'s:

«The word "block" is the correct term. Please let's leave this alone. «