Part 23: Buffered Channels and Worker Pools

Wednesday, December 31, 2025 by Raymond Hazen

What are buffered channels?

Up to now, all channels we've been using have been unbuffered. Unbuffered channels are the default for channels, but there's also a buffered channel, which will receive an unbuffered channel on blocking.

It's possible to create a channel with a buffer, similar to a queue. Sending to a buffered channel will block only when the buffer is full, and receiving from a buffered channel will block only when the buffer is empty.

Buffered channels can be created by passing an additional capacity parameter to `make`, as. Interface which specifies the size of the buffer.

```
ch := make(chan int, capacity)
```

capacity is the desire amount should be greater (fixed to 1) a channel to have a buffer. The capacity of an unbuffered channel is 0 by default and cannot be updated. Buffered channels are used when sending channels to the worker pattern.

Can write some code and create a buffered channel.

Example

```
package main

func main()
{
}

func main()
{
    fmt.Printf("%s", "Hello")
    fmt.Printf("%s", "World")
    fmt.Printf("%s", "!")
}
```

Run programs in playground

In this program above, in line 14 we create a buffered channel with a capacity of 2. Line 15 and line 16 are capacity of 2, it is possible to write 2 strings into the channel without blocking. Line 17 prints the strings received from the channel in the order in which they were sent out. Note. We read the 2 strings written in line two, 1 and 2 respectively. Run program above.

```
Hello
World
!
```

Another Example

Let's write another example of buffered channel to send the number from 1 to 10 and receive and do a concurrent Iteration and read from the main function. Write the code and run it. Note the underlyed other within in a buffered channel buffer.

```
package main

import "fmt"

func main()
{
    fmt.Printf("%s", "Hello")
    fmt.Printf("%s", "World")
    fmt.Printf("%s", "!")
}

func main()
{
    ch := make(chan int, 2)
    ch <- 1
    ch <- 2
    fmt.Printf("%d", <-ch)
    fmt.Printf("%d", <-ch)
}
```

Run programs in playground

In the program above, a buffered channel `ch` of capacity 10 is created. Line 10 and line 11 write 10 integers passed to the `ch` channel in line 10. Line 12 prints the value Generator Range 10 1 seconds. During this time, the main function waits for the generator to write to the channel line 14. Line 15 reads and prints 10 integers from the channel. Line 16 prints the value received from the channel `ch`. Line 17 and line 18 read from the channel `ch` and prints the value received. Line 19 prints the value received from the channel. Since there are no more values to be sent to the channel, the program will print the following. Line unexpectedly.

```
unexpectedly recv 1 in ch
unexpectedly recv 2 in ch
```

After printing the above two lines, the write to the `ch` channel in the `ch <- 3` statement are blocked until someone receives from the channel. The main function waits for the channel. Anyone will ever print anything for the main function. The program will print anything for the main function. The program will print anything for the main function. The program will