

Dalton Driscoll

February 11th, 2025

Shem Louisy

EECE 4811/5811

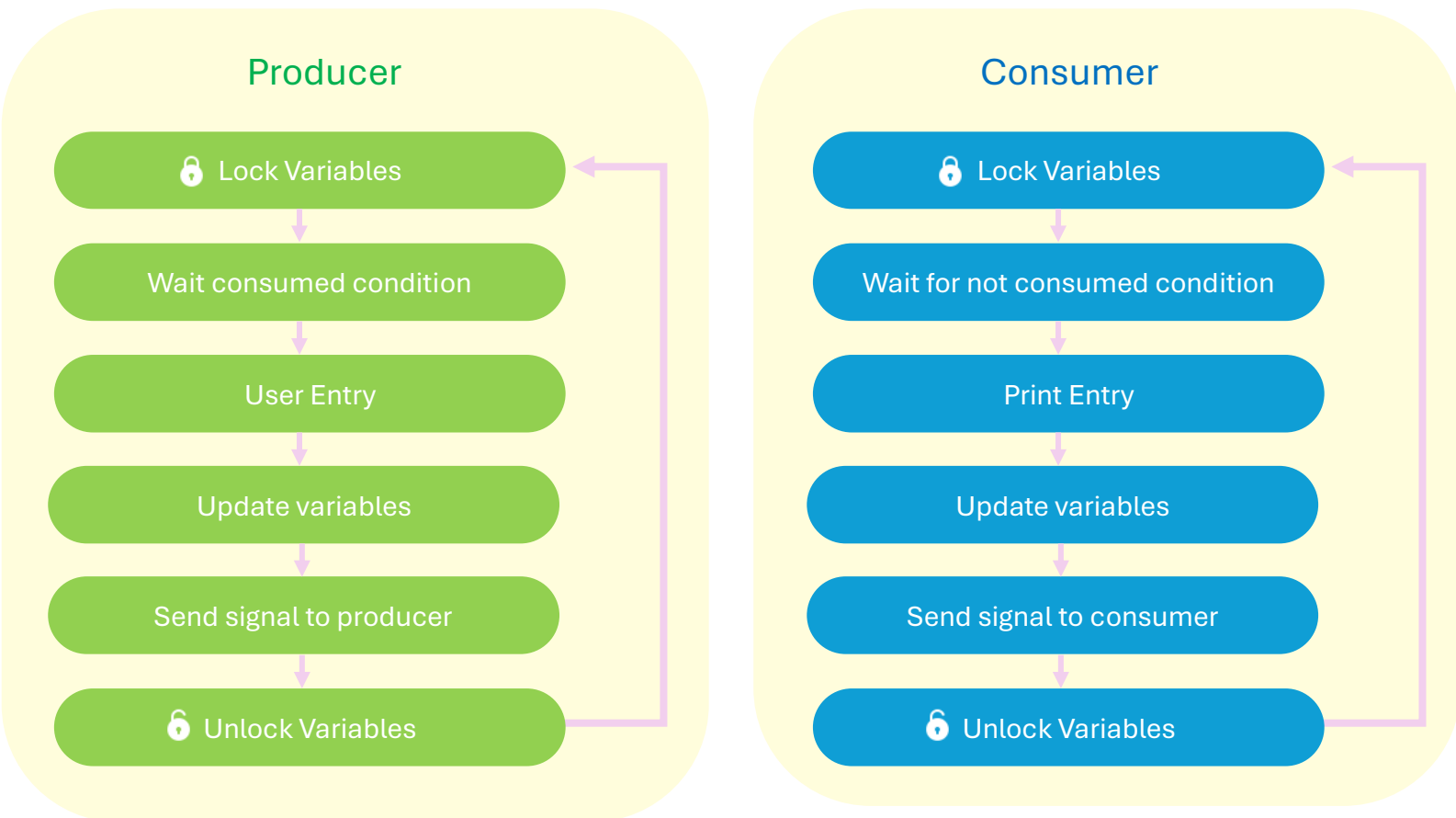
Abumere Okhihan

Professor Tseng

Goroutines vs Threads

Goroutines and threads are both used to attain concurrency, but they are different in the way that they are executed and designed. **[4]** The tour of Go defines a goroutine as “a lightweight thread managed by the Go runtime.” **[5]** Whereas Wikipedia defines a thread as “the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system. In many cases, a thread is a component of a process”. All Goroutines and threads share the commonality of running in the same address space, therefore access to shared memory must be synchronized for both. One of the biggest advantages of a goroutine is their efficiency, consuming far less resources than a thread would. **[6]** This allows thousands of goroutines to run concurrently with very little overhead. Unlike threads, which the OS schedules preemptively, goroutines use cooperative scheduling, meaning they voluntarily yield control. This approach minimizes context-switching overhead and, thus, improves efficiency. Goroutines also have growable segmented stacks, dynamically adjusting their memory usage. **[7]** Conversely, threads have a fixed stack size which can often be ineffective. Goroutines also have an easy communication medium known as channel, which allows safe and efficient data transferring between concurrently executing goroutines. Threads lack this built in functionality, they rely on tools like mutexes and condition variables, which can lead to potential overhead like latency. Despite all these benefits, goroutines do have some disadvantages over threads. **[8]** They lack the same level of control that a thread possesses, as Go does not provide thread-local storage or IDs for goroutines. Since goroutines use cooperative scheduling, a goroutine that refuses to yield can result in blocked execution and lead to a negative effect on performance. Goroutines also rely on Go’s garbage collection system, which can introduce pauses in execution, leading to further inefficiency.

Producer & Consumer Program Using Multithreading

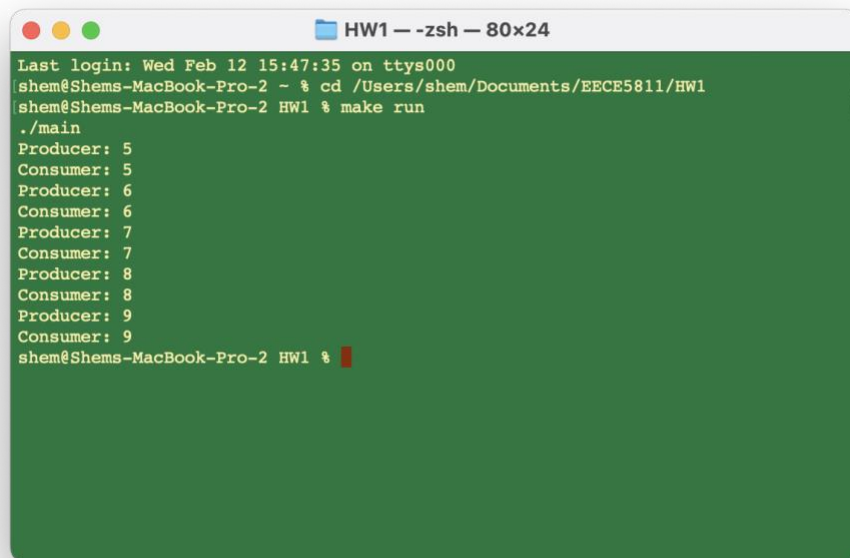


[1] The producer and consumer threads run as a while loop that ends when the **MAX_ENTRIES** are exceeded. [2] A mutex lock is used to prevent both threads from accessing the shared variable (**food, consumed**). [3] To prevent deadlock, a condition is used to wait for the consumed condition. The producer releases the lock to the consumer until the **consumed** variable is true, whereas the consumer releases the lock to the producer until the **consumed** variable is false. After updating the shared variables, a signal is sent to the respective thread through the thread condition letting them know to proceed. A **last_food** variable is used to keep track of the previous food value to prevent the loop from printing the value multiple times.

Instructions

To run program in Mac/Linux Terminal:

1. Navigate to folder with code and makefile
2. Type "make run"



```
HW1 - -zsh - 80x24
Last login: Wed Feb 12 15:47:35 on ttys000
shem@Shems-MacBook-Pro-2 ~ % cd /Users/shem/Documents/EECE5811/HW1
shem@Shems-MacBook-Pro-2 HW1 % make run
./main
Producer: 5
Consumer: 5
Producer: 6
Consumer: 6
Producer: 7
Consumer: 7
Producer: 8
Consumer: 8
Producer: 9
Consumer: 9
shem@Shems-MacBook-Pro-2 HW1 %
```

To run program on Windows:

Compile using g++ compiler in command prompt or run "**main.c**" in an IDE such as Visual Studio.

References

- [1] GeeksforGeeks, "Thread functions in C/C++," *GeeksforGeeks*, <https://www.geeksforgeeks.org/thread-functions-in-c-c/>. Accessed Feb. 8, 2025
- [2] GeeksforGeeks, "Mutex lock for Linux thread synchronization," *GeeksforGeeks*, <https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/>. Accessed Feb. 8, 2025
- [3] GeeksforGeeks, "Condition wait and signal in multi-threading," *GeeksforGeeks*, <https://www.geeksforgeeks.org/condition-wait-signal-multi-threading/>. Accessed Feb. 8, 2025
- [4] Go Team, "Concurrency," The Go Programming Language Tour, 2024 <https://go.dev/tour/concurrency/1>. Accessed Feb. 11, 2025
- [5] Wikipedia contributors, "Thread (computing)," *Wikipedia*, 2025. [https://en.wikipedia.org/wiki/Thread_\(computing\)](https://en.wikipedia.org/wiki/Thread_(computing)). Accessed Feb. 11, 2025.
- [6] GeeksforGeeks, "Golang Goroutine vs Thread," *GeeksforGeeks*, 2025. <https://www.geeksforgeeks.org/golang-goroutine-vs-thread/>. Accessed Feb. 11, 2025
- [7] D. Atal, "Understanding the differences between Go routines and Threads," Medium, 2025. <https://durgeshatal1995.medium.com/understanding-the-differences-between-go-routines-and-threads-b631068d4fdd>. Accessed: Feb. 11, 2025
- [8] S. R. Teja Chintakrindi, "Goroutines and Threads: Exploring Concurrency in Go," Medium, Jan.30, 2020. <https://medium.com/@sairavitejachintakrindi/goroutines-and-threads-exploring-concurrency-in-go-370d609038c>. Accessed Feb. 11, 2025

GeeksforGeeks, "Thread functions in C/C++," "Mutex lock for Linux thread synchronization," "Condition wait and signal in multi-threading," "Golang Goroutine vs Thread":

Credibility: GeeksforGeeks is a well-established online platform that provides high-quality tutorials and resources on a wide range of programming topics. It is widely trusted by developers for learning and improving practical skills, particularly for intermediate-level topics. While not a peer-reviewed academic resource, the site is authored by experts in the field and its content is frequently updated. GeeksforGeeks is a reliable source for foundational knowledge and practical coding examples, especially for programming languages like C/C++, Linux, and Go.

Go Team, "Concurrency" (The Go Programming Language Tour):

Credibility: This official resource from the Go programming language developers is highly authoritative, providing accurate, up-to-date information on concurrency and Go's concurrency model. As a product of the official Go team, this source is considered a definitive guide for Go-related topics and is ideal for understanding the intricacies of Go's concurrency features.

Wikipedia contributors, "Thread (computing)":

Credibility: Wikipedia is a widely used platform that provides a collaborative approach to content creation. While the articles are not peer-reviewed, the platform is regularly updated and maintained by many contributors.

D. Atal, "Understanding the differences between Go routines and Threads" (Medium)

S. R. Teja Chintakrindi, "Goroutines and Threads: Exploring Concurrency in Go" (Medium):

Credibility: Medium hosts insightful and informative content by authors with expertise in their respective fields. Articles published on Medium can be valuable resources for practical learning and offer different perspectives on topics. As a platform, it provides a variety of views and is a helpful resource for understanding complex topics in programming.