

ယခု Article မှာတော့ Semaphore နဲ့ Mutex ကွာခြားချက်ကို ဥပမာ လေးနဲ့ ရှင်းပြ ပေးမှာ ဖြစ်ပါတယ်။

Semaphore နဲ့ Mutex နှစ်ခု မှာ တူညီတဲ့ အချက်ကတော့ synchronization primitive ဖြစ်ကြပြီးတော့ single resources ဒါမှမဟုတ် multi resources ဘဲ ဖြစ်ဖြစ် resources တွေကို တစ်ချိန်တည်း / တစ်ပြိုင်တည်း အလုပ်လုပ်နိုင်အောင် သူ့နည်း သူ့ဟန်နဲ့ လုပ်ဆောင်ပေးပါတယ်။

ဆိုပါစို့ :

Toilet တစ်ခု ရှိပါတယ်။ Toilet ကို ဖွင့်ရန် key တစ်ချောင်းဘဲ ရှိပါတယ်။ Toilet ကို အသုံးပြုရန် လူသုံး ယောက်ရှိတယ်လို့ ယူဆကြည့်ရ အောင် ...



Let's assume toilet has a key and person who is having key can enter into the toilet.



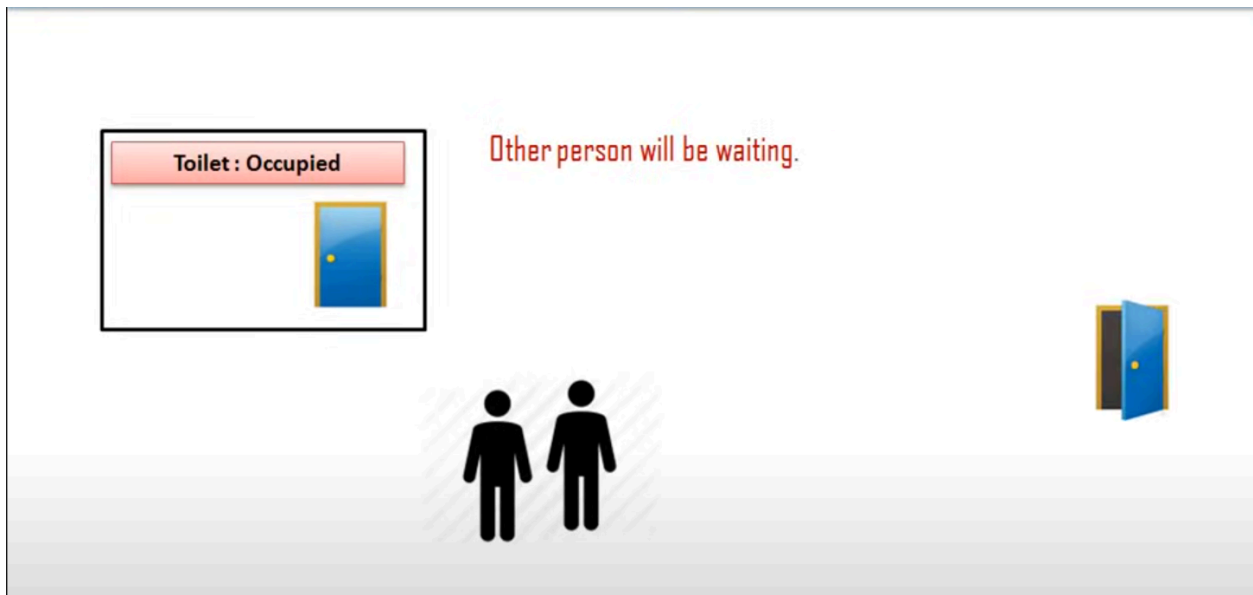
တစ်ချိန်မှာ toilet က key တစ်ခုဘဲ ရှိလို့ လူတစ်ယောက်ဘဲ ဝင်လို့ ရပါတယ်။



Person will unlock the door and then enter into toilet and close it.



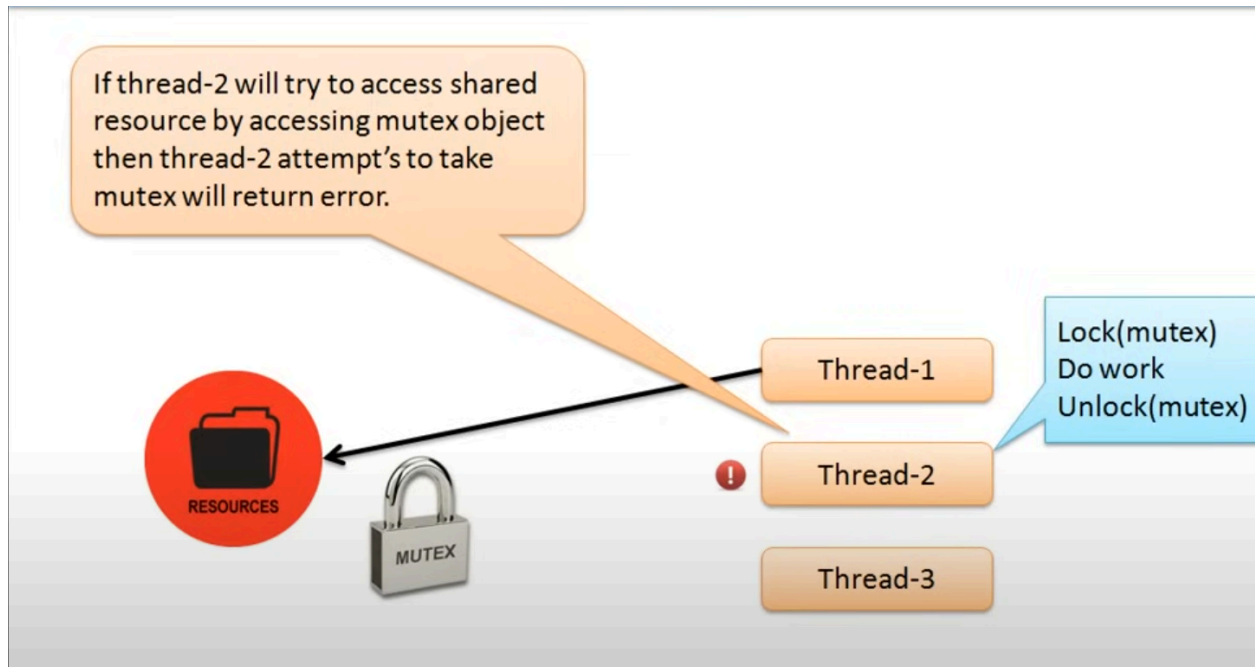
ပထမလူ အသုံးပြုနေတဲ့ အချိန် ကျန်လူ များ Wait State အနေ နဲ့ နေရပါတယ်။ တစ်ချိန်တည်းမှာ လူတစ်ယောက်ဘဲ အလုပ်လုပ်ရတဲ့ ပုံစံမျိုးပါ။



ပထမလူ အသုံးပြုပြီးမှ ဒုတိယ လူကို သော့ လက်ဆင့် ကမ်းပြီး တစ်ယောက်ပြီးမှ တစ်ယောက် အသုံးပြုရတဲ့ ပုံစံမျိုးဖြစ်ပါတယ်။



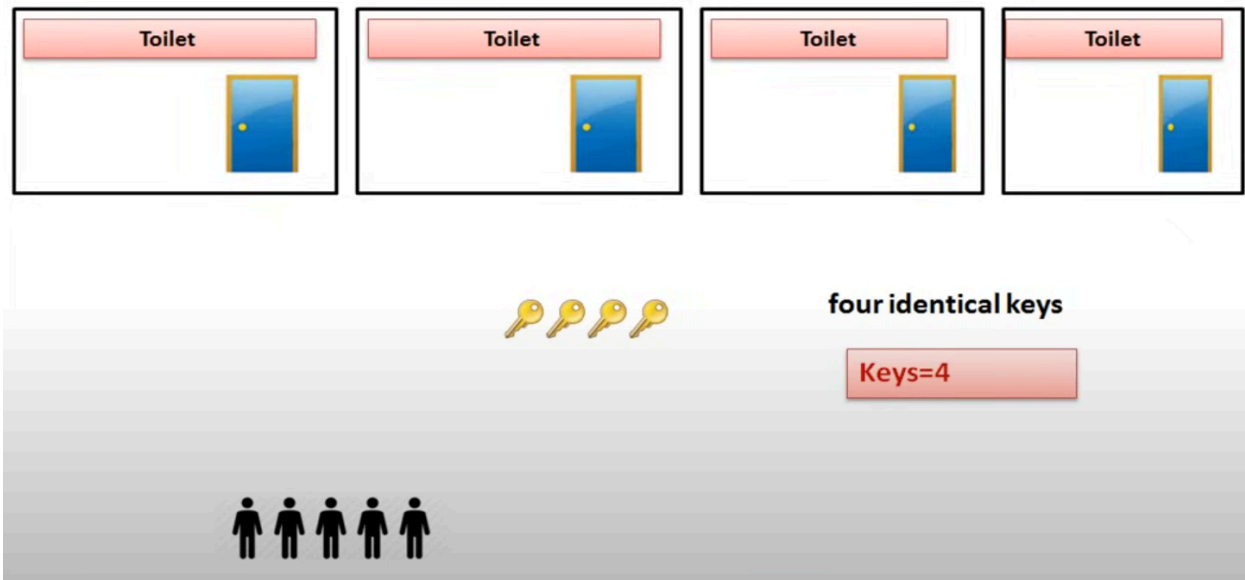
ဒါဟာ Mutex ရဲ့ အကြမ်းဖျင်း အလုပ်လုပ်ပုံ ဖြစ်ပါတယ်။ resource ကို တစ်ချိန်မှတည်းမှာ thread တစ်ခုကဘဲ Mutex ကို Lock လုပ်ပြီး အလုပ်လုပ်ပါတယ်။ အဲ့ဒီ thread ပြီးမှ Mutex ကို unlock လုပ်ပြီး ကျန်တဲ့ thread က ဆက်လက် အလုပ်လုပ်မှာ ဖြစ်ပါတယ်။



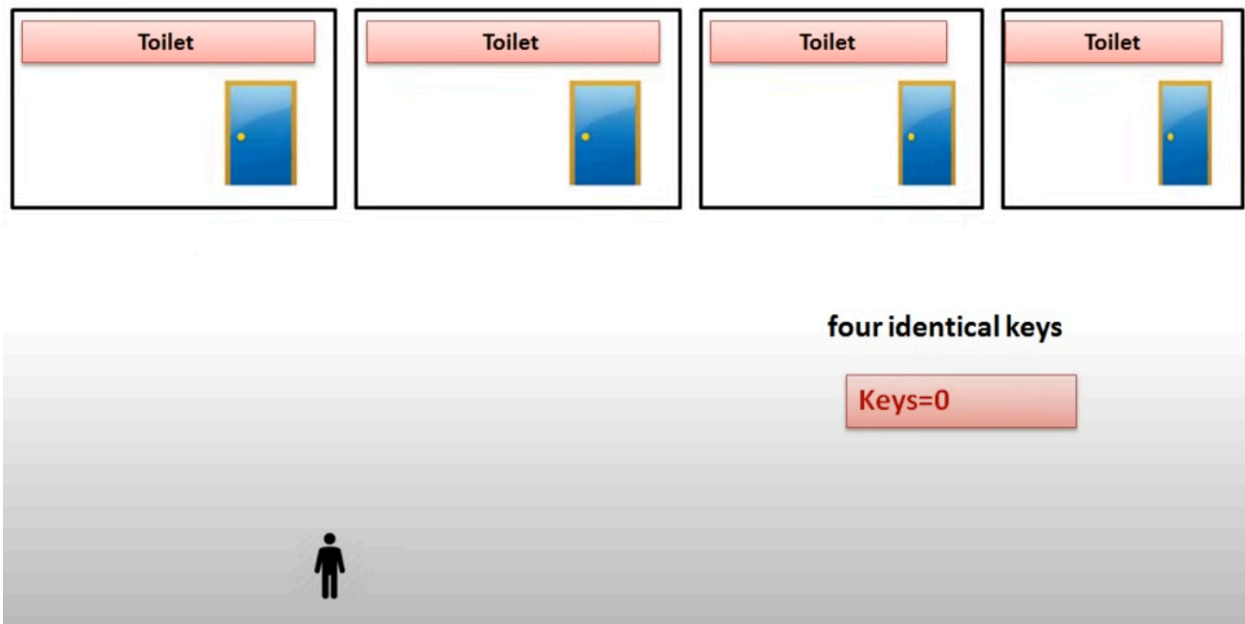
Mutex Lock ဖြစ်နေတဲ့ အချိန် thread တစ်ခုကဘဲ resources ရဲ့ owner ဖြစ်ပြီး access ရမှာ ပါ။ ownership resources access လို့ ပြောလို့ရပါတယ်။ တစ်ချိန်မှာ thread တစ်ခုကဘဲ resources ကို lock လုပ်ပြီး access ရမှာ ဖြစ်တဲ့ အတွက် data conflict ဖြစ်ခြင်းကိုလည်း ကာကွယ်ပြီးသား ဖြစ်ပါတယ်။

Mutex ဟာ synchronization primitive ဖြစ်ပေမယ့် အရေးကြီးတဲ့ critical section resources အပိုင်းထဲ ကို thread တစ်ခုဘဲ ဝင်ရောက်ခွင့်ပြုပါတယ်။

Semaphore ရဲ့ အလုပ်လုပ်ပုံ ကိုကြည့်ရအောင်

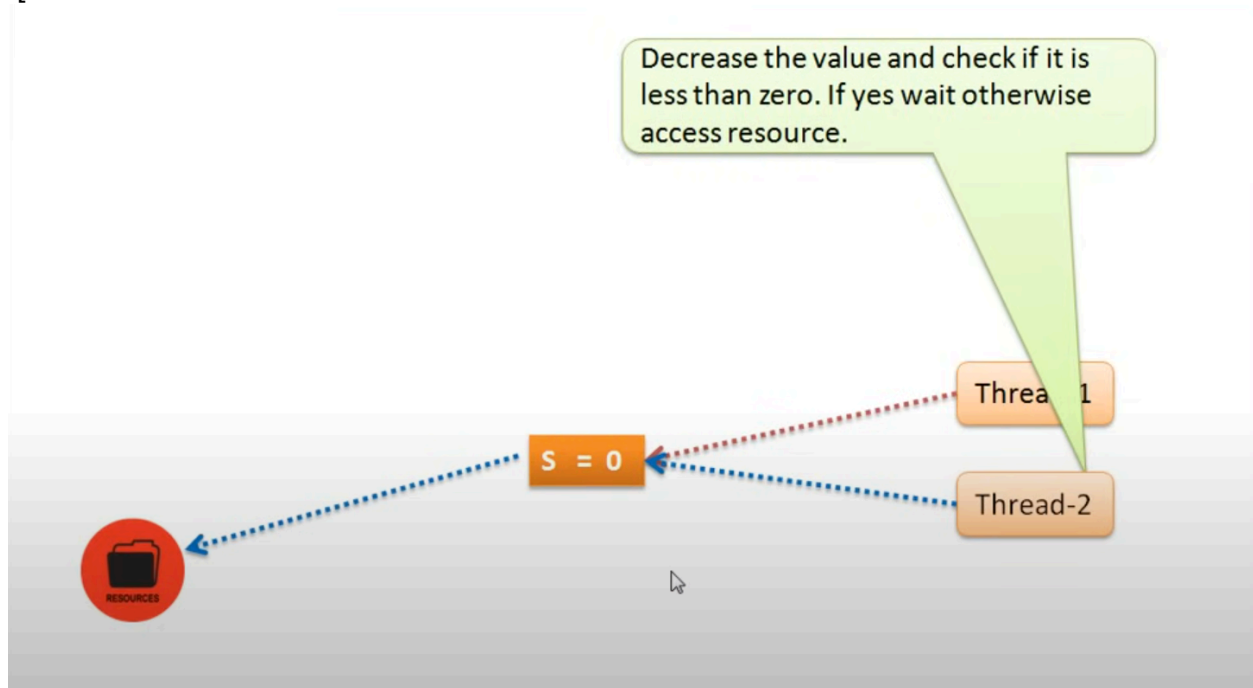


Toilet အသုံးပြုမယ့် လူ ငါးယောက် ရှိပါတယ်။ Toilet လေးခုရှိပါတယ်။ Toilet တစ်ခု ကို key တစ်ခု စီနဲ့ key လေးခုရှိပါတယ်။ လူ လေးယောက်က toilet လေးခု ကို တစ်ချိန်တည်း / တစ်ပြိုင်တည်းကို အသုံးပြုနိုင်မှာ ဖြစ်ပါတယ်။ ဒါပေမယ့် ကျန်တဲ့ တစ်ယောက်ကတော့ Wait State အနေဖြင့် စောင့်ဆိုင်းရမှာ ဖြစ်ပါတယ်။

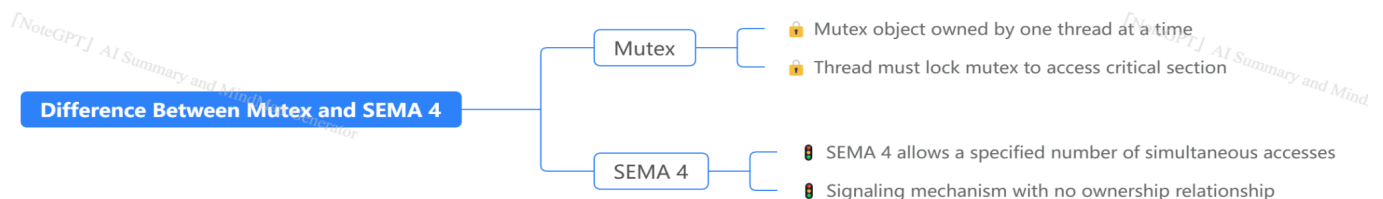


လူလေးယောက်မှ တစ်ယောက် အားမှ ကျန်တဲ့ လူမှ ဆက်လက် အလုပ်လုပ်နိုင်မှာ ဖြစ်ပါတယ်။

အကြမ်းဖျင်းအားဖြင့် Semaphore ကို Resources ကို share သုံးတဲ့ signaling mechanism လို့ ပြောလို့ရပါတယ်။



Semaphore ကတော့ resources contention (Resources အငြင်းပွားမှု) ကို တစ်ချိန် ထဲ/တစ်ပြိုင်တည်းမှာ resources တွေကို ဝင်ရောက်သုံးပြုမယ့် thread access ကို ကန့်သတ်ထားခြင်းဖြင့် ကာကွယ်ထားပါတယ်။





Summary

Mutex and Semaphore (Sema 4) differ in ownership and signaling mechanisms. Mutex allows one thread to own a resource at a time, while Semaphore controls simultaneous access with a set limit.

Highlights


Mutex: Ownership and resource access 🔒


Mutex: One thread owns the resource at a time 


Semaphore (Sema 4): Signaling mechanism and access control 

Semaphore (Sema 4): Limits simultaneous access 

Key Insights

Mutex ensures exclusive ownership of a resource by allowing only one thread to access it at a time. This prevents conflicts and ensures data integrity. 

Semaphore (Sema 4), on the other hand, acts as a signaling mechanism to control simultaneous access to shared resources. It sets a limit on how many threads can access the resource at the same time, preventing resource contention. 

Mutex provides a clear ownership model where a thread must lock and unlock the mutex to access the critical section, ensuring orderly access to shared resources. This helps in avoiding race conditions and data corruption. 

Semaphore (Sema 4) offers a flexible mechanism for managing resource access by allowing a specified number of threads to access a shared resource concurrently. This helps in optimizing resource utilization and improving system efficiency. 