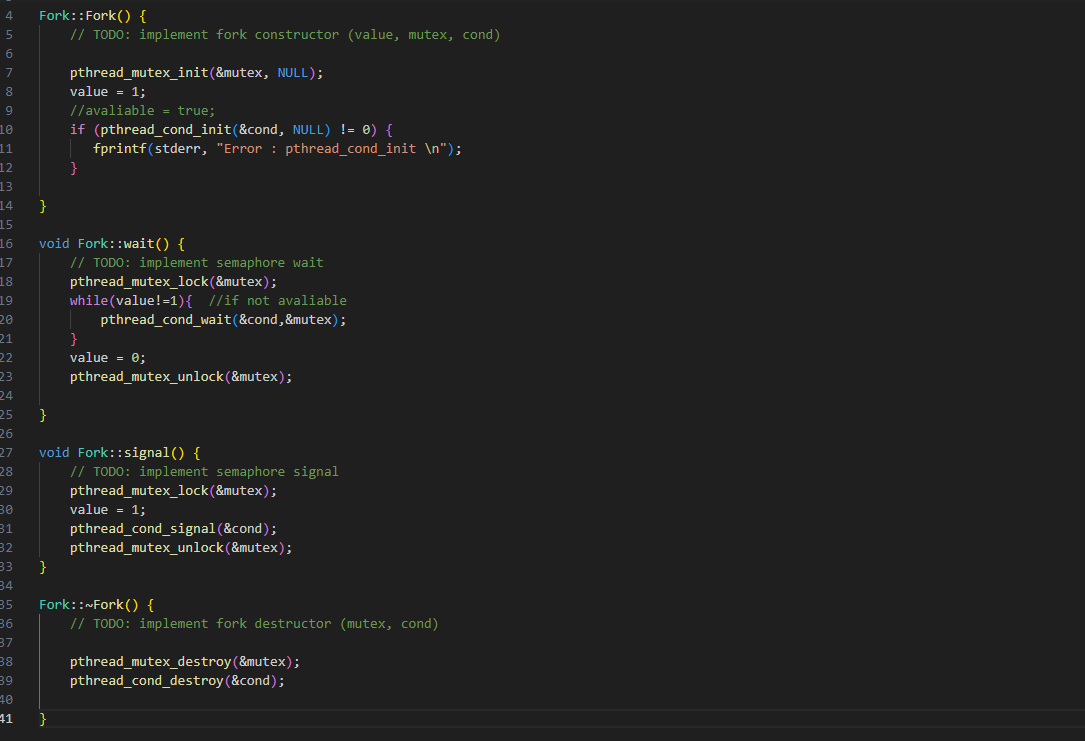
2.

Fork



首先宣告mutex和condition variable,value設成1

Value只會在1和0徘徊,如果有人使用了這個fork會把他設成0.

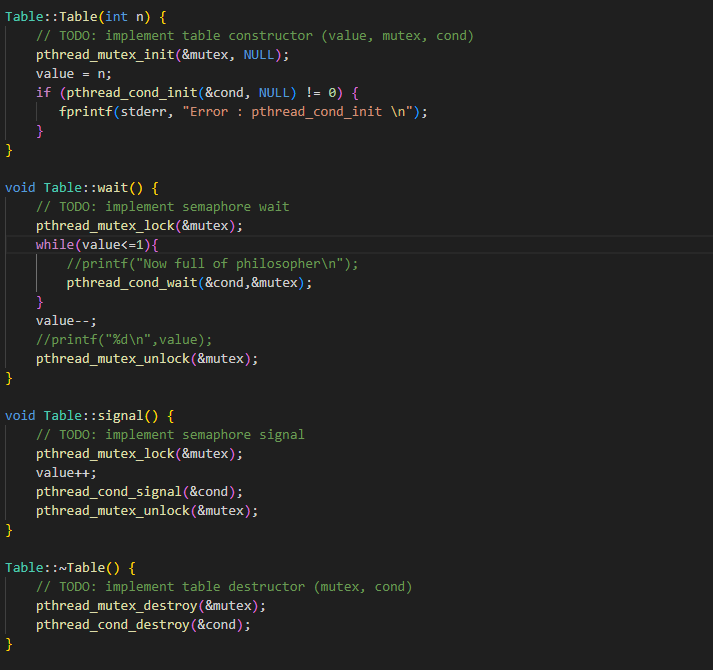
Wait():如果要使用這個fork,會call這個函式,把value設成0,並且使用pthread\_cond\_wait將這個fork put to sleep.

由於會使用value這個share data,頭尾必須使用lock以及unlock

Signal():當哲學家放下這個fork到餐桌上的時候會call這個函式,把value設成1,並且使用pthread\_cond\_signal wake up這個fork.由於會使用value這個share data,頭尾必須使用lock以及unlock

最後在解構子把mutex以及cond release掉.

Table



首先宣告mutex和condition variable,value設成n

Value會在2~5徘徊(5是哲學家的數量)

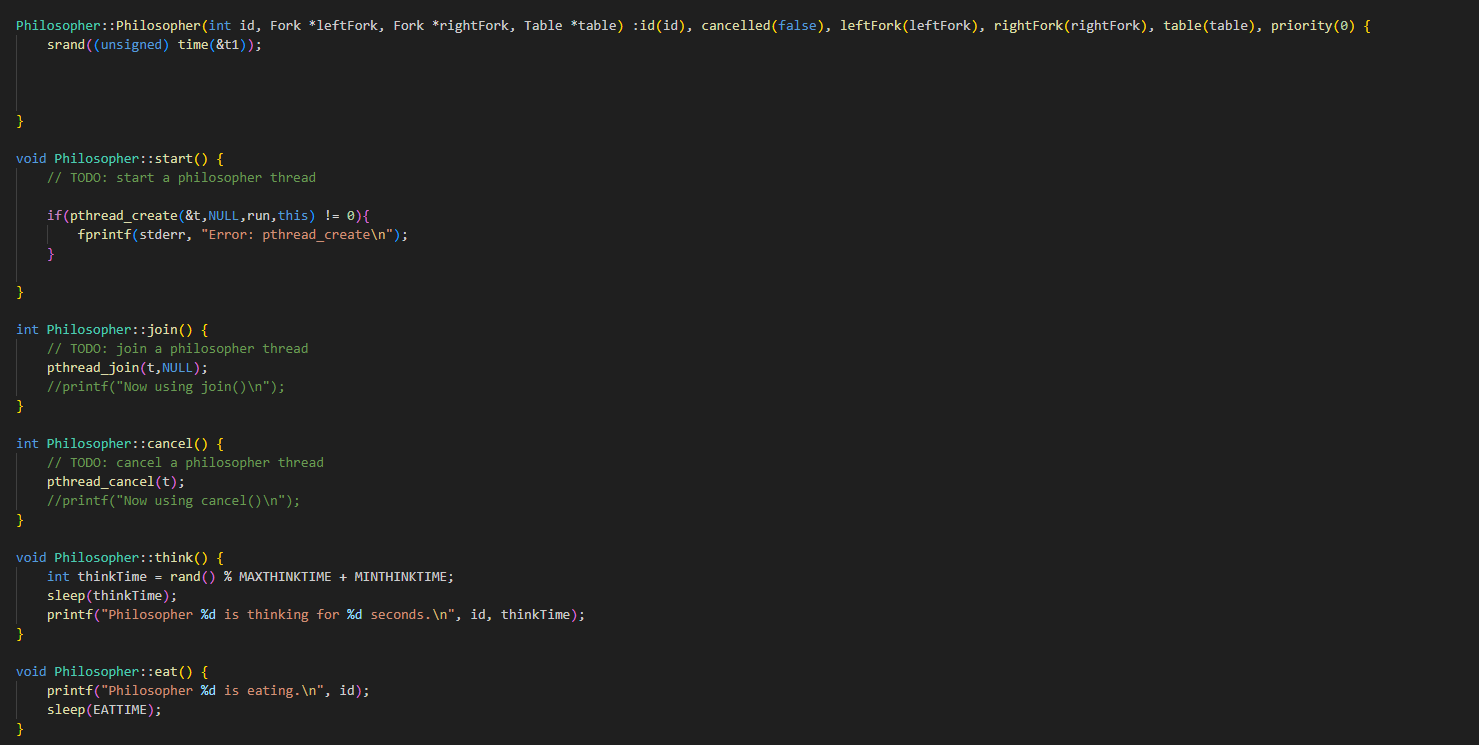
Wait():如果要進入這個table,會call這個函式,把value--,如果value<=1(代表進去4位哲學家了)就使用pthread\_cond\_wait把table put to sleep,確保只有4個哲學家在用餐.

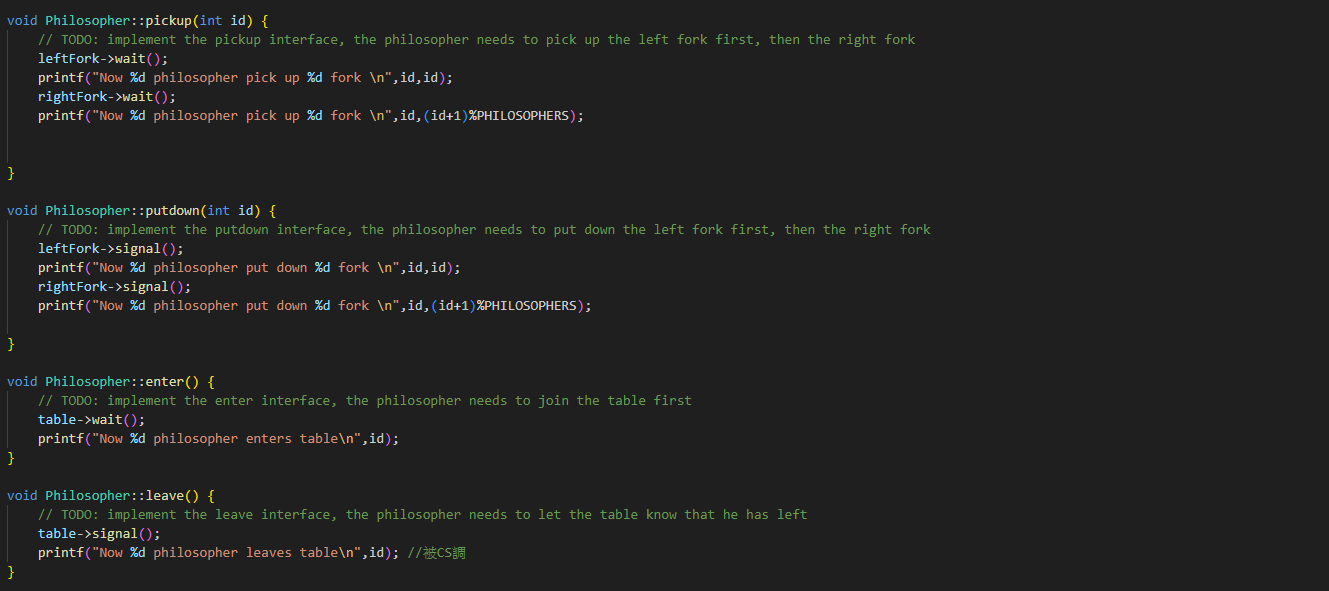
由於會使用value這個share data,頭尾必須使用lock以及unlock

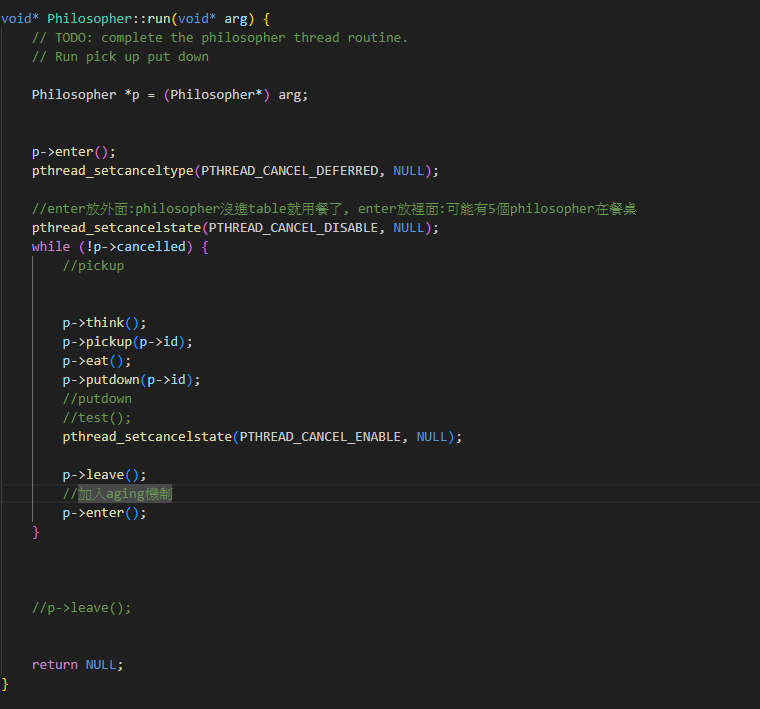
Signal():當哲學家離開餐桌的時候會call這個函式,把value++,並且使用pthread\_cond\_signal wake up這個table,讓其他哲學家可以進來table用餐(餐桌上仍然最多4個人).由於會使用value這個share data,頭尾必須使用lock以及unlock

最後在解構子把mutex以及cond release掉.

Philosopher:







Start():create這個thread

Join():main thread之外的thread要執行完畢

Cancel():把thread取消掉,不使用的話程式會無限執行下去

Think():哲學家在思考一個random時間

Eat():哲學家在eating

Pickup():將左右叉子拿起來,call wait(),將value設成0,並把叉子put to sleep

Putdown():將左右叉子放下來call signal,將value設成1,並wake up 左右叉子

Enter():call wait(),哲學家進入這個table,最多四個哲學家進入

Leave():call signal(),哲學家離開這個table

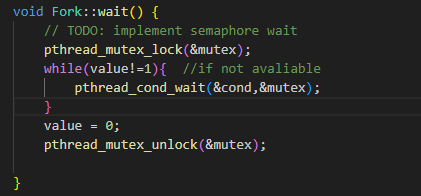
Run():一開始哲學家入座,並且在loop不斷思考 拿筷子 吃飯 放筷子,如果哲學家要離開可以再由其他哲學家進入table

3.

1.由於上下會有mutex\_lock綁住如果call pthread\_cond\_wait()時,該thread會卡在critical section出不去,而這時候由於該thread已經把mutex搶走了,導致其他thread進不去critical section,導致deadlock,所以要在pthread\_cond\_wait()的參數傳入mutex parameter,當thread被put to sleep的時候把mutex釋放出來,讓其他thread 可以進入他們的critical section.

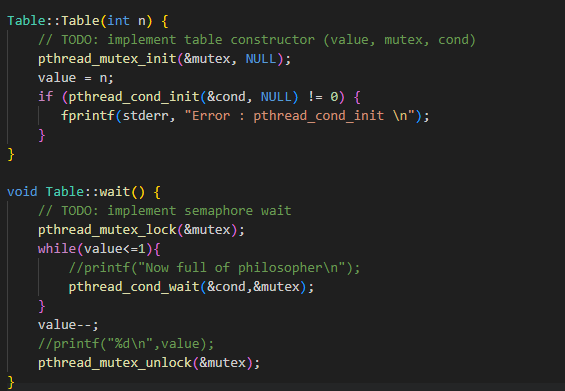
而,pthread\_cond\_signal()僅僅是發送一個訊號通知等待在condition variable的thread,不涉及mutex的操作,因此不需要mutex當作參數.

2.



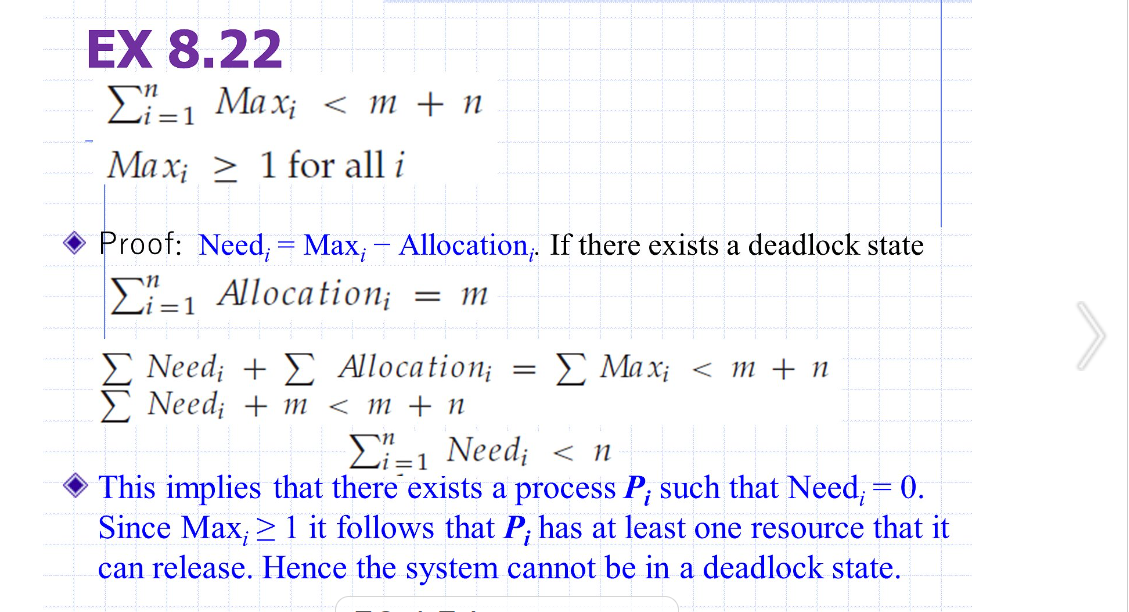
這一個部分.創建fork的時候會把value initialize成1,當call wait()時候會將value設成0,代表這個fork正在被使用,且會被put to sleep,等待其他人call signal(),叫醒這個fork, value只有0跟1,確保一次只有一個哲學家使用.

3.



這部分,由於五個哲學家只取四個哲學家,至少會多出來一個resource可以釋放,一個人一定可以完工並且放掉資源

概念類似這題:



4.並沒有starvation-free,如果A哲學家離開後,馬上回到餐桌,就有可能有一個philosopher完全沒機會拿到fork用餐

5.

在迴圈以前就把state設成disable,這樣做的目的是確保在執行critical section時不被取消,確保在做一系列動作後才可以取消,維持程式的完整性和一致性.

這樣的設計可以控制thread在特定時刻是否可以被取消,並確保系統的正確運行和資源的正確釋放.

4.

Monitor優點:

提供一個簡單且直觀的機制解決同步問題,容易理解和實作,且提供一個high-level的抽象層將share data和相關的同步機制封裝再一個module裡,並且monitor確保同一時間只有一個thread能夠使用share data,避免race condition

Monitor缺點:

1.通常需要使用lock,很多thread在等待lock的釋放,影響整體效能

2.只有單一點入口,當一個thread進入monitor時候,thread必須等待該thread完成操作才能進入,尤其是程式有大量同步需求時候.

3.另外,所有thread都必須進入monitor才能共享data,可擴展性有問題.

不用monitor優點:

1. 可以根據問題需求使用合適的同步機制,如semaphore和mutex
2. 不需要進入和離開monitor的同步成本,提高效率

不用monitor缺點:

需要programmer去做同步的問題,可能會出錯,增加程式的複雜性和錯誤

可讀性降低,因為同步需要寫在程式裡面,會讓程式碼變的冗長難以理解.

5.

如果可以在作業截止前,上一個作業的成績可以出來是最好的~