Interprocess communication (IPC)

多線程於一個process 或 多個process間交換資料的一組方法

目的:

交換資訊(網路)

加快計算(分散式計算)

方便(server)

模組化 (micro kernel)

IPC分類

1. Shared memory

優點:快速(不經過OS)

缺點:同步化問題

(用address acess data )

1. Messege passing

優點:實作簡單，比較不怕亂複寫、同步化

缺點:較慢

(用sender,receiver)

1. Socket (IP,port)

交換 bit資料 (不使用結構資料)

1. Remote procedure calls(RPC)

有date type，並使用call function call方式傳

\*shared memory

概念: call system 叫OS 開一塊空間給process，(OS只負責給權限開空間)

使用者處理同步化、禁止同時修改、確定struct..問題

Example:

Consumer & Producer problem

創一個環形buffer跟兩個指標(in 、 out)

In表示下一筆資料寫入的位置 (必須空的)

Out表示下一次要讀取的資料的位置

in == out //表示buffer為空

(in+1)%bufferSize == out //表示buffer為滿 (換言之，留一格為空)

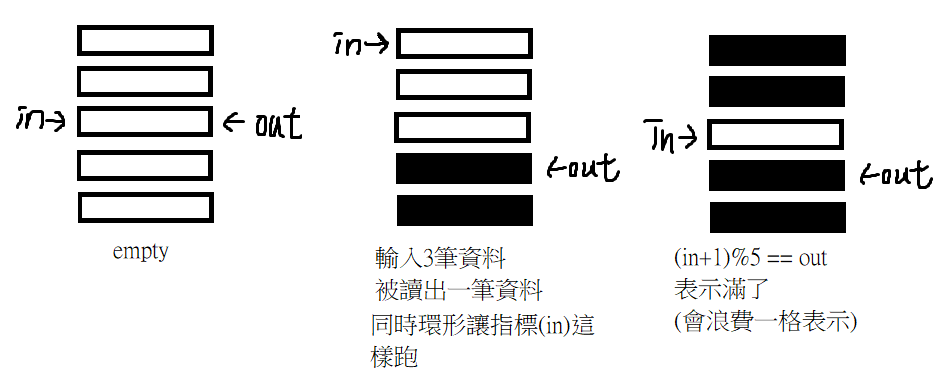
由producer寫入 (生產資料) => 只能動in指標、如果Full則不能執行

由comsumer讀出 (消費資料) => 只能動out指標、如果empty則不能執行

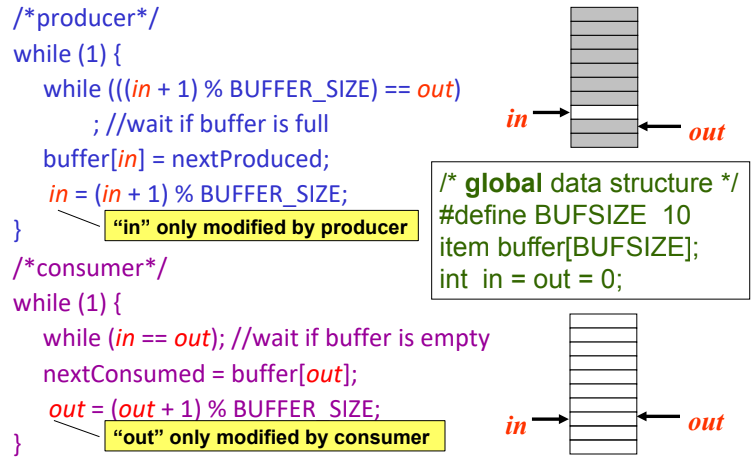
也就是用這種方法處理同步問題

這架構其實很常見，比如Pipe，前程式一直寫入資料，後程式一直讀取

Example



程式範例



Messeging passing

分成sender 、 receiver

又有block、non-block關係 (block:等對方)

需要communication link交換資料

能用硬體(memory)、軟體(send、reicever function)實作

概念

\*direct or indirect

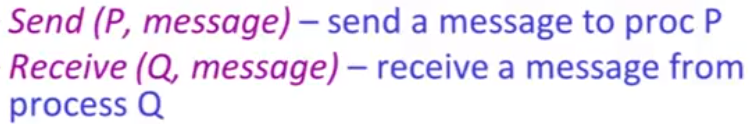
\*Blocking、non-blocking

\*同步非同步

\*copy or pointer

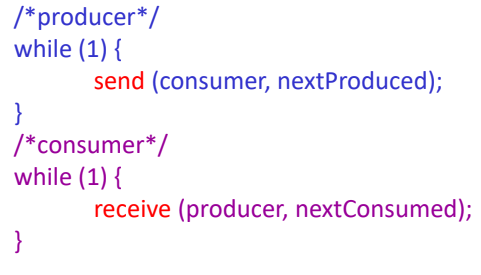
\*固定大小or自訂大小

\*Direct:



一定要指定雙方，並且是1-1

程式碼



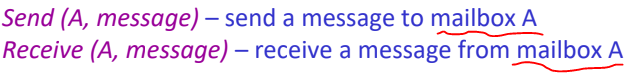
優點:實作簡化

缺點:僵化寫死，如果name改了，必須要重新寫

1-1

\*Indirect

創一個mail box，讓mail box決定送到哪裡



issue:

如何決定receiver?

Sol:

1. 一收一寄只允許兩個process(退化成direct)
2. 一次只允許一隻process call receiver
3. Mail box 決定排程(等到該你再通知)

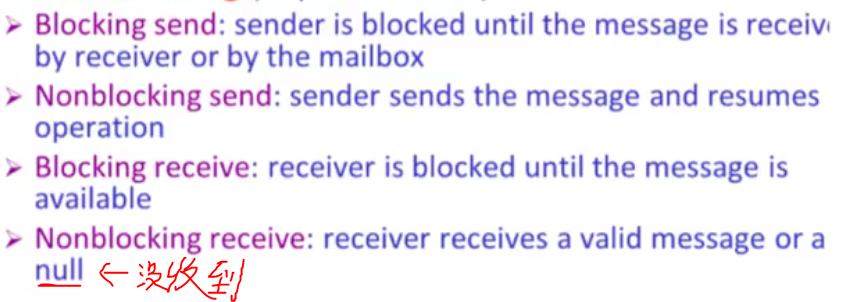
優點:Many-to-many

缺點:較為複雜

\*blocking(同步;等對方)&non-blocking(非同步;不等對方)

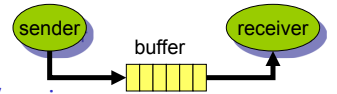
Sender、receiver都可以有兩種選擇

(可以有四種組合)



Non-blocking:不等對方，較為複雜，但有效率

Non-blocking實作方式，使用buffer



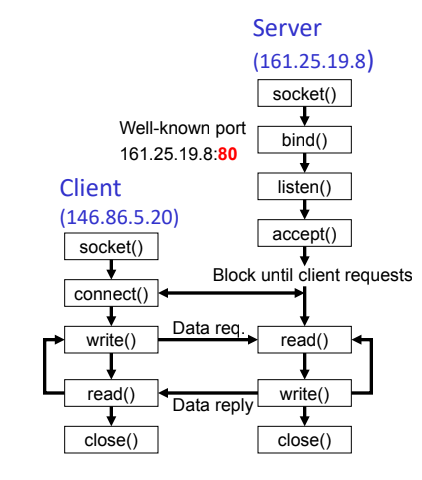
沒有buffer: blocking

Bounded buffer: 滿了才block

Unbounder buffer: 永不block

\*Socket

用IP 與 port溝通

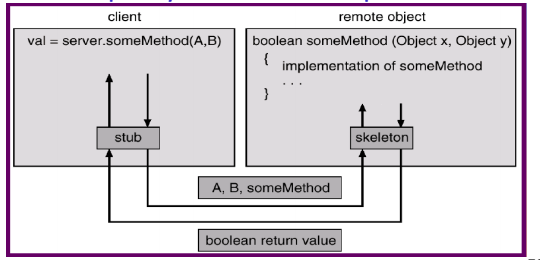
 connect() 是建立一個thread

Socket只負責傳bit

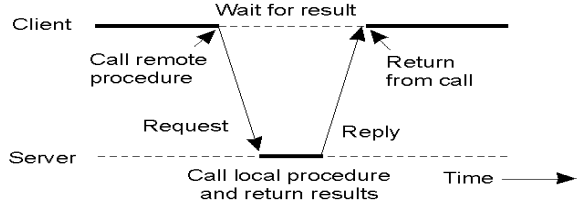
Parser交給高layer

\*RPC

使用在網路系統，抽象化成function call



Stub、skeletion 就是兩隻小程式，用於處理雜務



跨電腦: 管理格式的異同

很顯然地，無法copy address資料(要整塊複製)