HW4 Linux Semaphore

提供檔案: proj3.zip (sem.c prog1.c prog2.c p1.c p2.c p3.c awk_sem.h)

上繳檔案:1.所有需要用到的檔案(包含 source code)

2.README:請解釋你的程式如何編譯及執行

壓縮後命名為「學號-HW4.zip」並上傳至 LMS 系統

執行環境:Linux only

簡介:

此 project 的目的是要讓你熟悉 semaphore 的實際運作,以及如何運用 semaphore 做 process 之間的同步(coordination)。基本上這樣子的同步概念,也 照樣適用於 multitthreads。不同的作業系統都會提供一群同步的指令,但是基本上都大同小異。請注意,本 project 只能在 Linux 的環境底下執行。助教會提供 Linux 相關環境。

在提供的檔案當中,有兩個範例程式 prog1.c 以及 prog2.c 。prog1.c 非常的簡單,開始先建立一個 semaphore。該 semaphore 的名稱由兩部分組成,一個是路徑,一個是 project id。在 prog1.c 中用 create_sem 建立一個 semaphore,其名稱是"."+'S'。而且這個 semaphore 的初始值為0。

重點提示: 一般來說你所建立的 semaphore 只要是系統內其他 processes

(包含他人) 知道這個路徑+名字,就都可以使用它。所以你如果想要好好的進行 測試不希望被別人干擾,你最好取一個別人不會知道的路徑+名字。

在建立了 semaphore 之後,prog1 呼叫 P(semid),也就是一般作業系統課本裡頭的 wait(semid)。當然實際的 Linux semaphore system calls 不會叫 P() 也不是叫做 wait(S)。有興趣的你們可以打開 sem.c 來了解一下。

因為 semid 被初始化為 0,所以 prog1 會 block 在這個系統呼叫。一旦 prog1 進入等待的狀態後,你可以啟動 prog2。 prog2 的功能也非常簡單,它根據名字"."+'S'找到 prog1 所建立的 semaphore,然後執行 V(semid)。也就是課本裡頭的 signal(S)。這個呼叫會把沈睡中的 prog1 叫起來。然後 prog1 和 prog2 各自執行至結束。

執行範例:

要執行 prog1 以及 prog2,請先在 Linux 底下進行編譯

> gcc -o prog1 prog1.c sem.c

> gcc -o prog2 prog2.c sem.c

編譯完成之後,請先用下面的方式執行 prog1 > prog1 &

"&"的作用是告訴你的 shell,你雖然要執行 prog1,但是希望在背景執行,,不要鎖住 shell。所以雖然 prog1 會被 block,但是你的 shell 仍然可以輸入指令。

接著,請繼續執行

> prog2 &

你會發現 prog2 會叫醒 prog1,然後兩個程式各自結束執行。

這時候雖然兩個程式都已經結束執行了,但是 semaphore 其實還繼續存在著。你可以執行指令 *ipcs* ,然後你可以很清楚的看到你擁有一個 semaphore 的 ipc (Inter Process Communication)。若你想要清除該 semaphore,你可以用指令 *ipcrm sem semid*。其中 *semid* 是 *ipcs* 指令列出的 semaphore id。

你也可以試一試下面一連串的指令

- > prog1 &
- > prog1 &
- > prog1 &

這一連串的指令,會分別 block 三個 processes 於這個 semaphore 上面。你可以用指令 *jobs* 來顯示你目前有多少背景程式在執行。接著你可以執行一次 prog2 &。每次執行, prog2 會把最早 block 在這個 semaphore 的 process 按照先後順序叫醒一個。所以你必須執行 prog2 三次,才能把所有都 prog1 都叫醒。prog1 被叫醒的過程中會列印一些訊息,以及它們的 pid,方便你知道是哪一個 prog1 被叫醒。

Project goal:

在提供的檔案中有三個程式 p1.c p2.c 以及 p3.c。這三個程式都非常的簡單。分別印出不同的訊息。假設第一個被執行的檔永遠是 p1.c。也就是說,你可以在 p1.c 建立 semaphore,而 p2 以及 p3 不用負責建立 semaphore。你的目標是練習利用 semaphore 來同步 p1,p2,p3 使得執行順序是 p1 一次,然後 p2 一次,接著 p3 兩次。然後一直循環下去直到迴圈結束。也就是說,假設助教執行 p1 &; p2 &; p3 & 之後,你的輸出應為

P1111111

P2222222

P3333333

P3333333

P1111111

P222222

P3333333

P3333333

.

要達到上述的目標,你應該同步這三個程序,使得 p1 輸出一次,就得輸到 p2 輸出一次。而當 p2 輸出一次之後,才輸到 p3。而且 p3 必須輸出兩次之後 p1 才可以重新另外一個回合。請注意,助教也可能執行另外一種順序如 p1 &; p3 &; p2 & ,而你的結果應該也要一樣。<u>助教在執行你的程式前,會先用 ipcrm 確定清除任何存在的 semaphore。所以為了不要混淆程式執行結果,你最好在測試你</u>的程式之前也先用 ipcrm 清除存在的 semaphores。

重點住要事項

為了達成這樣的同步,請在 p1.c p2.c p3.c 加入適當的 P(),V() 來進行同步。 切記,你只能用 semaphore 來達到目標。例如你不能新增一個 printf 來讓 p3 多輸出一行。在 printf 的前面以及後面,你只能用 semaphore 指令 P(), V()。若你用了額外的指令,取巧達到所需的輸出,助教將以 0 分計算。

- (1)實驗失敗的 process 記得要 Kill 掉,以免嚴重影響系統效能
- (2) 執行你的程式之前,記得用 ipcrm 確定清除你之前的程式配

置的任何 semaphore

會做 HINT:

要完成上述的同步,你要先確認你需要幾個 semaphores 來完成。一但確定之後,用 create_sem() 在 p1.c 的迴圈前建立 semaphores。 然後在 p2.c 以及 p3.c 的迴圈前用 $get_sem()$ 來讀取 p1.c 所建立的 semaphore。

接著,你一定得弄清楚用 create_sem 建立 semaphore 時,它的初始值應該為多少。Semaphore 的初始值一定得大於等於0。你如果想設定初始值為負值,此系統呼叫會失敗,也顯示你的觀念不正確。

初始值設定之後,你接著得想辦法在 printf 的前後,用 P(),V() 來同步。祝好運。

sem.c 功能解說:

真正的 unix semaphore 系統呼叫是 semget, semop, semctl。由於這幾個系統呼叫有著複雜的參數,遠比課本所教的 semaphore 複雜許多。所以在 sem.c 當中,已經為你把這些系統呼叫包裝起來,提供下面的呼叫,讓它們像課本的 wait(S) 及 signal(S)。

create_sem(): 用來建立一個新的 semaphore,以及初始化該 semaphore 的值。如果呼叫成功,它會傳回一個 semid 。如果失敗它會傳回-1。如同前面所說的,前面兩個參數是名字的兩部分。第三個參數則是初始值。請記住,當你呼叫 create_sem 時,若所給定的 semaphore 名字已經有 semaphore 使用,這個呼叫不會幫你建立新的 semaphore,它會傳回已存在的 semaphore 的 id。並且根據你的第三項參數,設定 semaphore 的值。在 prog1 的例子當中,你每次執行 prog1 其實每次都會把 semaphore 的值設定為 0。在這個例子只是剛好不出錯而已。

get_sem(path, projid):根據名字來找到一個已存在的 semaphore id。如果能找到該 semaphore 則傳回它的 semid。若失敗則傳回-1。

P(semid): 把 semid 所指的 semaphore 值減一。若原本 semaphore 的值已經為 0則 block 這個 process

V(semid):把 semid 所指的 semaphore 值加一。並叫醒一個被 blocked 的 process, 如果有的話。

以下兩個functions 只是供你觀看程式的執行與DEBUG。請不要用在你的程式中,否則以0分計算

get_blocked_no(semid): 回傳 block 在 semid 的 semaphore 上的 process 總數。
get_sem_val(semid): 回傳目前 semid 的 semaphore 值。請注意,在 Unix 中,一個 semaphore 的值是不會小於 0 的。也就是說 Unix semaphore 的值永遠大於等於 0。而在課本中,semaphore 的值是可以為負值的。假設課本的 semaphore 值為 T,而真實 unix 的 semaphore 值為 S。它們之間的關係如下

T = S - get blocked no(S)

May force be with you.

~ Yoda , the Master