Linux 作業系統

作業二

授課教授：許富皓 老師

組別: 第23組

成員: 資工碩一 104522010 施宇謙

資工碩一 104522040 張瑞慶

資工碩一 104522065 張翔珳

**作業內容:**

1. **測試環境和版本**
2. **方法一Part I 將process送進自定義wait queue**
3. **方法一Part II將process根據PID把Process喚醒**
4. **方法一實驗結果**
5. **方法一Code**
6. **方法二Part I將process送進自定義wait queue**
7. **方法二Part II將process根據PID把Process喚醒**
8. **方法二實驗結果**
9. **方法二Code**

**檔案路徑說明:**

**Code 資料夾: go\_to\_sleep.c、wake\_up\_my\_process.c**

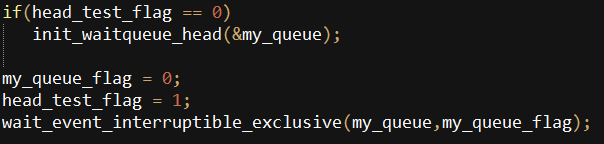
**Testbench 資料夾: tb1.c、tb2.c、tb1執行檔、tb2執行檔。**

**0.測試環境和版本**

|  |  |
| --- | --- |
| **VM** | **Virtual Box** |
| **安裝linux發行版本** | **Ubuntu 14.04(32-bit)** |
| **編譯Kernel版本** | **linux-3.10.77** |

Ubuntu 12.10 的apt-get更新list無法下載 libncurses-dev，需額外新增sudo gedit /etc/resolv.conf底下的nameserver 8.8.8.8或新增repository ，所以改用Ubuntu 14.04。

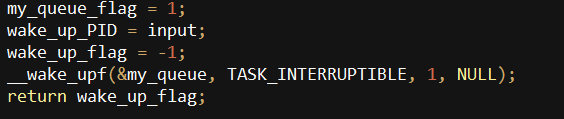
1. **方法一Part I 將process送進自定義wait queue**

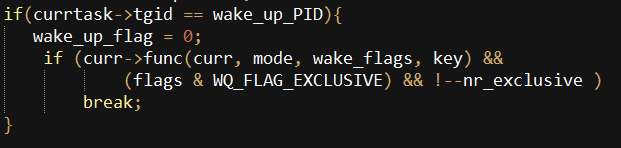
定義自己的成全域變數Wait queue、Condition，然後初始化Wait queue，接著定義喚醒Process的Condition，並送進自定義Wait queue。

這裡設定是使用wait event類型的，並且是exclusive，加入時是從尾端加入。

1. **方法一Part II 將process根據PID把Process喚醒**

第一步先讓Condition成立，在透過wake\_up巨集將process喚醒，根據原始碼我們將整個wake up function trace後，根據wake up原function修改加入pid的判斷來決定喚醒條件。





1. **方法一實驗結果**

Run第一個testbench來達到加入waitqueue的效果，可以看見剛剛run的tb1的pid是2658。





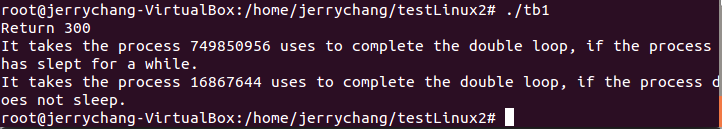
接著再開啟一個Terminal Run下一個tb1，PID是2666



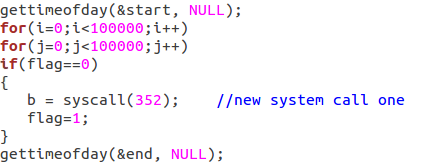
接著Run 第二個testbench，根據PID讓第一個Run的process醒來PID是2658。

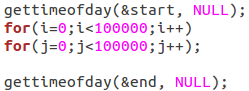


然後可以看見process1 PID2658已經run起來，並結束的畫面。

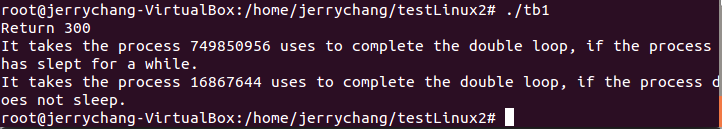


根據testbench可以看出，這個程式跑兩個雙層迴圈，並分別在兩個雙層迴圈外面下時間測試錨點。





而第一個迴圈的第一圈就把此process透過system call送去wait queue，而第二個迴圈是單純的跑完雙重迴圈，透過兩個時間錨點所測出來的時間，紅色框唯有進入wait queue的時間，綠色框為空迴圈，可以看見兩個的時間差，單位為us，相減後為732983312us，所以約為733秒，而空迴圈是17秒，所以證明此process有進入wait queue。



而現在系統中只剩第二次Run的process2還在系統中。



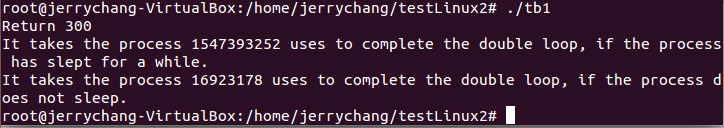
接著在喚醒這個process



等待程式Run完後可以看見系統中已無任何tb2程式在Run



根據此process時間差結果可以看出相差1530秒，約25.5分鐘，因為此process比第一個在wait queue的時間久，所以時間也較長，而空迴圈跑的時間一樣是17秒左右。



1. **方法一Code**

go\_to\_sleep()

|  |
| --- |
| #include <linux/linkage.h>  #include <linux/kernel.h>  #include <linux/sched.h>  #include <linux/wait.h>  wait\_queue\_head\_t my\_queue;  unsigned char my\_queue\_flag;  unsigned char head\_test\_flag = 0;  asmlinkage int sys\_go\_to\_sleep\_now(void) {  if(head\_test\_flag == 0)  init\_waitqueue\_head(&my\_queue);    my\_queue\_flag = 0;  head\_test\_flag = 1;  wait\_event\_interruptible\_exclusive(my\_queue,my\_queue\_flag);    return 300;  } |

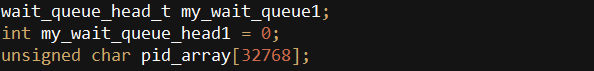
wake\_up\_my\_process

|  |
| --- |
| #include <linux/linkage.h>  #include <linux/kernel.h>  #include <linux/sched.h>  #include <linux/spinlock.h>  #include <linux/list.h>  int wake\_up\_PID = 0;  char wake\_up\_flag = -1;  static void \_\_wake\_up\_commonf(wait\_queue\_head\_t \*q, unsigned int mode,  int nr\_exclusive, int wake\_flags, void \*key)  {  wait\_queue\_t \*curr, \*next;  struct task\_struct \*currtask;  list\_for\_each\_entry\_safe(curr, next, &q->task\_list, task\_list) {  unsigned flags = curr->flags;  currtask = curr->private;  if(currtask->tgid == wake\_up\_PID){  wake\_up\_flag = 0;  if (curr->func(curr, mode, wake\_flags, key) &&  (flags & WQ\_FLAG\_EXCLUSIVE) && !--nr\_exclusive )  break;  }    }  }  void \_\_wake\_upf(wait\_queue\_head\_t \*q, unsigned int mode,  int nr\_exclusive, void \*key)  {  unsigned long flags;  spin\_lock\_irqsave(&q->lock, flags);  \_\_wake\_up\_commonf(q, mode, nr\_exclusive, 0, key);  spin\_unlock\_irqrestore(&q->lock, flags);  }  extern unsigned char my\_queue\_flag;  extern wait\_queue\_head\_t my\_queue;  asmlinkage int sys\_wake\_up\_my\_process(int input) {  my\_queue\_flag = 1;  wake\_up\_PID = input;  wake\_up\_flag = -1;  \_\_wake\_upf(&my\_queue, TASK\_INTERRUPTIBLE, 1, NULL);  return wake\_up\_flag;  } |

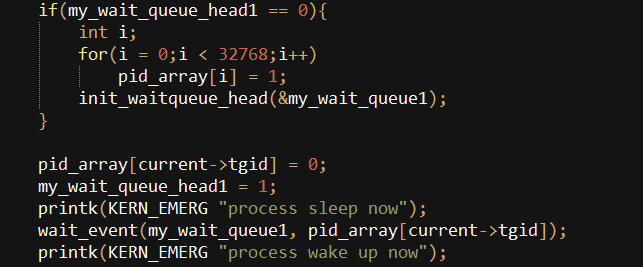
**5.方法二Part I將process送進自定義wait queue**

定義一個陣列儲存每個被送進wait queue內process的flag，以此flag當作wait\_event內的condition，首先將陣列初始為1，如果有process被送入wait queue，則更改flag為0，使process無法break無限for迴圈而進入睡眠。等到wake up時再將相對應pid的flag設為1，使process成功從wait queue被wake up。

根據/proc/sys/kernel/pid\_max內的值，知道pid最大值32768，並在全域變數定義一個32768的unsigned char陣列，如下圖所示。

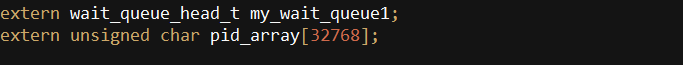


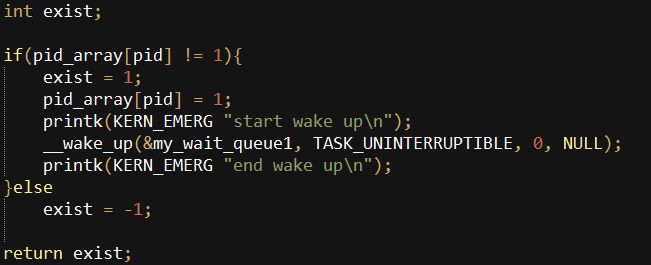
並將陣列在第一次時system call執行時全部初始化為1，再送入wait queue。



**6.方法二Part II將process根據PID把Process喚醒**

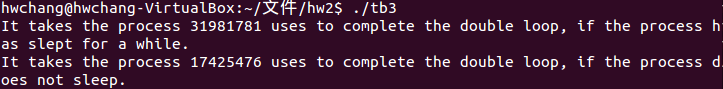
在wake\_up\_my\_process1一開始檢查每個process的flag是否為1(代表沒被送入wait queue)，如果為1回傳-1，testbench便知道該pid的process並不在wait queue，，如果為0回傳1，testbench便知道該pid的process正在wait queue，且即將被wake up。



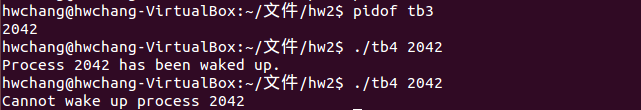


**7.方法二實驗結果**

如方法一實驗步驟，先執行測試程式1進入睡眠，可看到測試程式1確實有進入睡眠。



再執行測試程式2從睡眠喚醒。



**8.方法二Code**

go\_to\_sleep\_now1.c

|  |
| --- |
| #include <linux/linkage.h>  #include <linux/kernel.h>  //#include <linux/wait.h>  #include <linux/sched.h>  wait\_queue\_head\_t my\_wait\_queue1;  int my\_wait\_queue\_head1 = 0;  unsigned char pid\_array[32768];  asmlinkage void sys\_go\_to\_sleep\_now1(void){  if(my\_wait\_queue\_head1 == 0){  int i;  for(i = 0;i < 32768;i++)  pid\_array[i] = 1;  init\_waitqueue\_head(&my\_wait\_queue1);  }    pid\_array[current->tgid] = 0;  my\_wait\_queue\_head1 = 1;  printk(KERN\_EMERG "process sleep now");  wait\_event(my\_wait\_queue1, pid\_array[current->tgid]);  printk(KERN\_EMERG "process wake up now");  } |

wake\_up\_my\_process1.c

|  |
| --- |
| #include <linux/linkage.h>  #include <linux/kernel.h>  #include <linux/sched.h>  #include <linux/spinlock.h>  #include <linux/list.h>  extern wait\_queue\_head\_t my\_wait\_queue1;  extern unsigned char pid\_array[32768];  asmlinkage int sys\_wake\_up\_my\_process1(int pid){  int exist;    if(pid\_array[pid] != 1){  exist = 1;  pid\_array[pid] = 1;  printk(KERN\_EMERG "start wake up\n");  \_\_wake\_up(&my\_wait\_queue1, TASK\_UNINTERRUPTIBLE, 0, NULL);  printk(KERN\_EMERG "end wake up\n");  }else  exist = -1;    return exist;  } |