

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH



Hệ điều hành (CO2017)

Lab 4

Multi-tasking and Scheduler activations

Advisor: Nguyễn Quang Hùng

Students: Lô Hoàng Bảo - 2252066
Lê Nguyễn Nam Khánh - 2252328
Phương Gia Kiệt - 2252407
Nguyễn Thành Phát - 2252604

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 5 NĂM 2024



Table of Contents

1	Crontab	2
2	Problem 1	3
3	Problem 2	4
4	Problem 3	5
5	Video trình bày	6

1 Crontab

Crontab gồm 6 phần được tách với nhau bởi dấu cách space như sau: Trong đó 5 ô đầu là thể

* * * * *	command(s)	output
- - - - -		
- - - -	Day of week (0 - 7) (Sunday=0 or 7) (Mon - Sun)	
- - - - -	Month (1 - 12)	
- - - - -	Day of month (1 - 31)	
- - - - -	Hour (0 - 23)	
- - - - -	Minute (0 - 59)	

Hình 1: Crontab

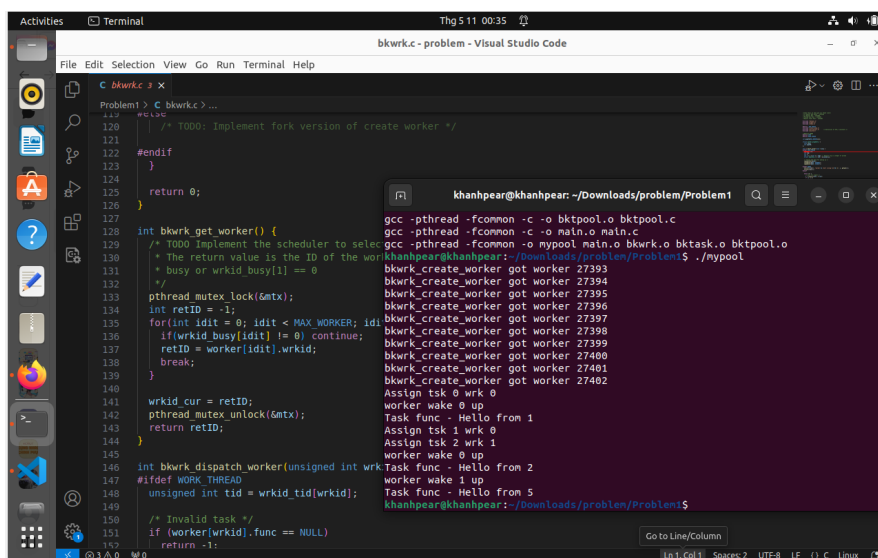
hiện thông tin về thời gian và phần còn lại là lệnh mà mình muốn chạy vào khoảng thời gian đã được cài đặt ở 5 ô trước đó. Và chúng em sẽ chỉ sử dụng 5 ô đầu để thực hiện exercise, practice:

Exercise: Chuyển đổi các khoảng thời gian sau thành bản trình bày crontab:

- Every Monday at 08:30 -> 30 8 * * 1 (Với 1 tương ứng cho mỗi ngày thứ Hai)
- Every workday, every 30 minutes, from 8:15 to 17:45 -> 15,45 8-17 * * 1-5 (do khoảng cách thời gian sẽ đều chuyển từ 15 phút sang 45 phút rồi lại 15 phút ví dụ từ 8:15 thì sau 30 phút sẽ là 8:45 rồi 30 phút kế tiếp sẽ là 9:15 và cứ lặp đi lặp lại ở phần phút nên ở chỗ ô * biểu hiện cho phút sẽ là 15,45 và các ngày làm việc sẽ từ thứ Hai đến thứ Sáu nên sẽ là 1-5)
- Last day of every month at 17:30 ->
 - 30 17 31 1,3,5,7,8,10,12 * (Tháng có 31 ngày)
 - 30 17 30 4,6,9,11 * (Tháng có 30 ngày)
 - 30 17 28 2 * (Tháng 2 năm không nhuận)
 - 30 17 29 2 * (Tháng 2 năm nhuận)

2 Problem 1

Đối với vấn đề này ta sử dụng 1 khóa mutex để kiểm tra các trường hợp race condition, ở hàm `get_worker`, nếu `wrkid_busy[i] == 0` sẽ được coi là không có việc gì và ngược lại

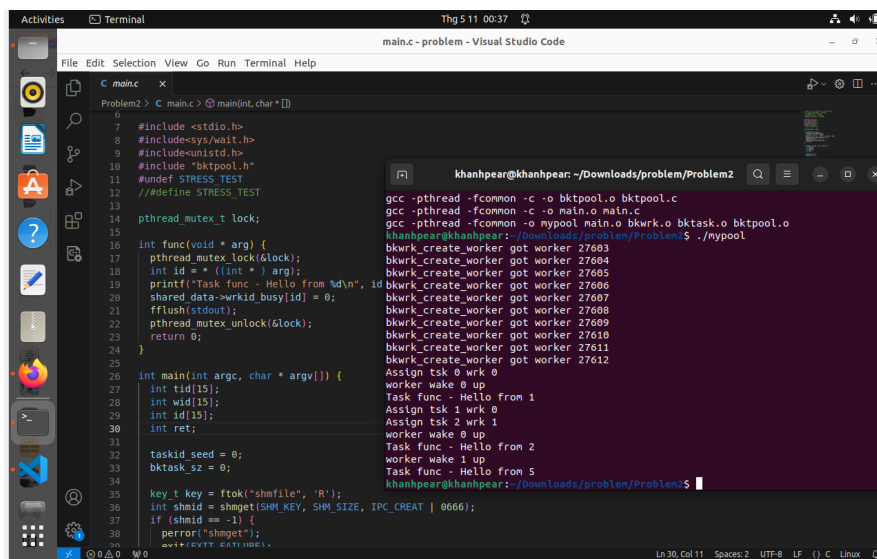


```
gcc -pthread -fcommon -c -o bktpool.o bktpool.c
gcc -pthread -fcommon -c -o main.o main.c
gcc -pthread -fcommon -o nypool main.o bktpool.o bktask.o bktpool.o
khanhpear@khanhpear:~/Downloads/problem/Problem1$ ./nypool
bktwk_create_worker got worker 27393
bktwk_create_worker got worker 27394
bktwk_create_worker got worker 27395
bktwk_create_worker got worker 27396
bktwk_create_worker got worker 27397
bktwk_create_worker got worker 27398
bktwk_create_worker got worker 27399
bktwk_create_worker got worker 27400
bktwk_create_worker got worker 27401
bktwk_create_worker got worker 27402
Assign tsk 0 wrk 0
worker wake 0 up
Task func - Hello from 1
Assign tsk 1 wrk 0
Assign tsk 2 wrk 1
worker wake 0 up
Task func - Hello from 2
worker wake 1 up
Task func - Hello from 5
khanhpear@khanhpear:~/Downloads/problem/Problem1$
```

Hình 2: Output của Problem 1

3 Problem 2

Đối với vấn đề trên, đề bài yêu cầu chúng ta sử dụng fork API để xử lý việc quản lý các tác vụ task sao cho có chức năng như Problem 1, để sử dụng tính năng tương tự như với clone API, ta sẽ sử dụng các chức năng shared memory để chia sẻ các tài nguyên cần thiết cho các tiến trình tương tác qua lại với nhau

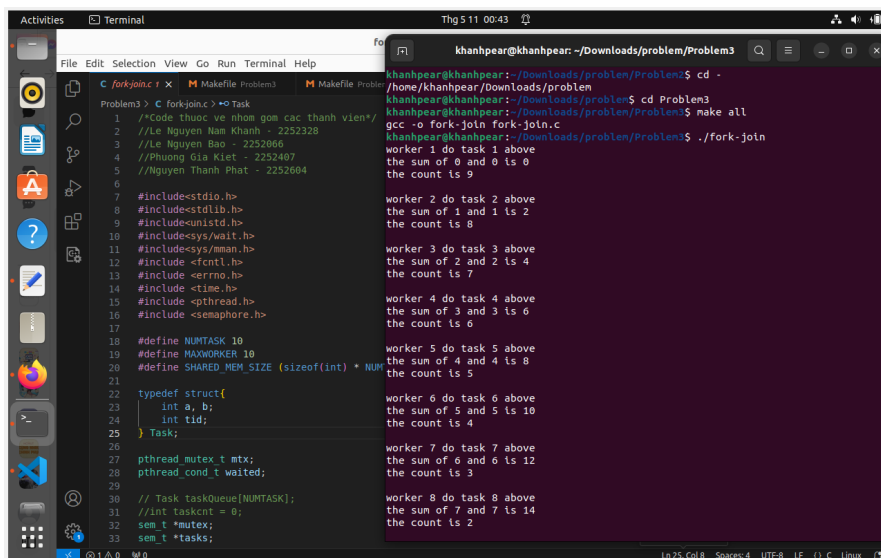


```
gcc -pthread -fcommon -c -o bktpool.o bktpool.c
gcc -pthread -fcommon -c -o main.o main.c
gcc -pthread -fcommon -o mypool main.o bkwrk.o bktask.o bktpool.o
khanhpear@khanhpear:~/Downloads/problem/Problem2$ ./mypool
bkwrk_create_worker got worker 27603
bkwrk_create_worker got worker 27604
bkwrk_create_worker got worker 27605
bkwrk_create_worker got worker 27606
bkwrk_create_worker got worker 27607
bkwrk_create_worker got worker 27608
bkwrk_create_worker got worker 27609
bkwrk_create_worker got worker 27610
bkwrk_create_worker got worker 27611
bkwrk_create_worker got worker 27612
Assign tsk 0 wrk 0
worker wake 0 up
Task func - Hello from 1
Assign tsk 1 wrk 0
Assign tsk 2 wrk 1
worker wake 0 up
Task func - Hello from 2
worker wake 1 up
Task func - Hello from 5
khanhpear@khanhpear:~/Downloads/problem/Problem2$
```

Hình 3: Output của Problem 2 (gần như tương tự Problem 1)

4 Problem 3

Đối với vấn đề này, ta được yêu cầu tạo ra một cơ chế fork-join có chức năng quản lý các tiến trình làm nhiều công việc thay vì chỉ 1 tiến trình, qua đó tương tự như Problem 2, bằng cách đưa các dữ liệu cần thiết vào 1 shared memory được khởi tạo ta có thể truyền các tác vụ cộng 2 đối tượng và tính tổng của 1 số dựa trên các tiến trình riêng biệt đó



```
khanhpear@khanhpear:~/Downloads/problem/Problem3$ cd -  
/home/khanhpear/Downloads/problem  
khanhpear@khanhpear:~/Downloads/problem$ cd Problem3  
khanhpear@khanhpear:~/Downloads/problem/Problem3$ make all  
gcc -o fork-join fork-join.c  
khanhpear@khanhpear:~/Downloads/problem/Problem3$ ./fork-join  
worker 1 do task 1 above  
the sum of 0 and 0 is 0  
the count is 9  
worker 2 do task 2 above  
the sum of 1 and 1 is 2  
the count is 8  
worker 3 do task 3 above  
the sum of 2 and 2 is 4  
the count is 7  
worker 4 do task 4 above  
the sum of 3 and 3 is 6  
the count is 6  
worker 5 do task 5 above  
the sum of 4 and 4 is 8  
the count is 5  
worker 6 do task 6 above  
the sum of 5 and 5 is 10  
the count is 4  
worker 7 do task 7 above  
the sum of 6 and 6 is 12  
the count is 3  
worker 8 do task 8 above  
the sum of 7 and 7 is 14  
the count is 2  
khanhpear@khanhpear:~/Downloads/problem/Problem3$ make all
```

Hình 4: Output sử dụng tiến trình của Problem 3



5 Video trình bày

Link video trình bày ở đây: <https://youtu.be/HvwxHiqHs-Y>