

# BÀI LÀM PROGRAMMING-EX-2

Huỳnh Tiến Dũng - 21020007

## 1. Chạy chương trình ở Slide 17 - NLHĐH-Bai-2

Mã nguồn nằm trong file myfork.c

Output:

```
^ hynduf ^ ~  
→ ./a.out  
  
Before fork()  
CHILD: value = 20 Before fork()  
PARENT: value = 5%
```

Lí do “Before fork()” bị lặp lại là do chưa flush output ra khỏi buffer. Nếu thêm `fflush(stdout);` hoặc thêm `\n` vào cuối Before fork() trong `printf` thì sẽ chỉ còn một “Before fork()” trong output.

## 2. Chapter 3 - Project 1

Mã nguồn nằm trong file chapter3-project1.c

```
^ hynduf ~
→ gcc simple_shell.c

^ hynduf ~
→ ./a.out
osh>echo 1
1
osh>echo 2
2
osh>echo 3
3
osh>echo 4
4
osh>pwd
/home/hynduf
osh>history
5 pwd

4 echo 4
3 echo 3
2 echo 2
1 echo 1

osh>!!
/home/hynduf
osh>! 1
1
osh>! 4
4
osh>! 5
/home/hynduf
```

### 3. Chapter 3 - Project 2 - Part 1

Mã nguồn nằm trong file chapter3-project2-part1.c

Phần chính:

```
for_each_process(task)
{
    printk(KERN_INFO "%s\t\t%ld\t%d\n", task->comm, (long)task->__state, task->pid);
}
```

task->comm là name của process

task->\_\_state là trạng thái của process

task->pid là pid của process

Output dmesg (vài chỗ bị lệch do tên của các process dài hơn 1 \t):

```
[ 115.082564] Loading Module 21020007 Huynh Tien Dung
[ 115.082571] Name          State    PID
[ 115.082574] systemd             1      1
[ 115.082579] kthreadd             1      2
[ 115.082582] pool_workqueue_      1      3
[ 115.082586] kworker/R-rcu_g      1026   4
[ 115.082590] kworker/R-rcu_p      1026   5
[ 115.082593] kworker/R-slub_      1026   6
[ 115.082597] kworker/R-netns      1026   7
[ 115.082600] kworker/0:0          1026   8
[ 115.082604] kworker/0:1          1026   9
[ 115.082607] kworker/0:0H         1026  10
[ 115.082610] kworker/u32:0         1026  11
[ 115.082614] kworker/u32:1         1026  12
[ 115.082617] kworker/R-mm_pe      1026  13
```

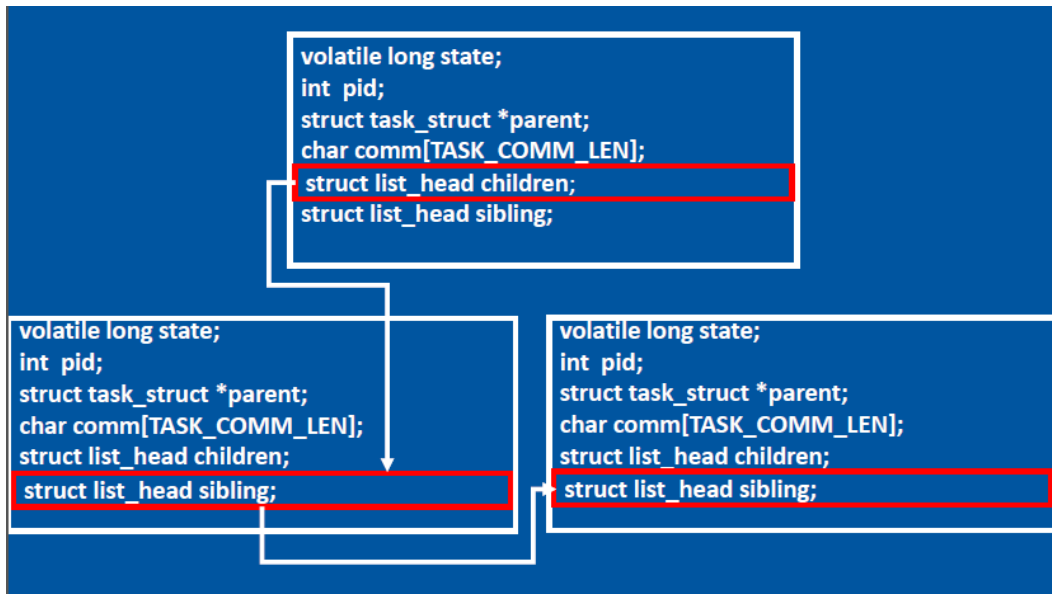
## 4. Chapter 3 - Project 2 - Part 2

Mã nguồn nằm trong file chapter3-project2-part2.c

Phần chính:

```
void depth_first_search_tasks(struct list_head *list, struct list_head *next)
{
    list_for_each(list, next)
    {
        task = list_entry(list, struct task_struct, sibling);
        printk(KERN_INFO "%s\t\t\t%ld\t%d\n", task->comm, (long) task->__state,
task->pid);
        depth_first_search_tasks(list, &task->children);
    }
}
```

Cấu trúc con trỏ children và sibling trong 1 task\_struct:



Đây giải thích cho dòng code:

```
task = list_entry(list, struct task_struct, sibling);
```

(list\_entry() sẽ lấy ra được task\_struct mà có biến 'sibling' chính là con trỏ 'list', 'list' ban đầu khi for sẽ từ children để trở tới sibling đầu tiên ở dưới và lần lượt đi hết các siblings)

Output dmesg (vài chỗ bị lệch do tên của các process dài hơn 1 \t):

```

[ 1373.888911] Removing Module
[ 1396.244064] Loading Module 21020007 Huynh Tien Dung
[ 1396.244067] Name                State    PID
[ 1396.244068] systemd                        1        1
[ 1396.244070] systemd-journal                1       306
[ 1396.244071] systemd-udevd                  1       346
[ 1396.244073] bluetoothd                     1       538
[ 1396.244074] dbus-daemon                    1       539
[ 1396.244075] dhcpcd                         1       540
[ 1396.244076] dhcpcd                         1       542
[ 1396.244077] dhcpcd                         1       687
[ 1396.244079] dhcpcd                         1       543
[ 1396.244080] dhcpcd                         1       544
[ 1396.244081] systemd-logind                 1       545

```

## 5. Bài tập 3.12

i = 0. Sau fork sẽ có 2 process (1 cha, 1 con)

i = 1. Sau fork sẽ có 4 process

i = 2. Sau fork sẽ có 8 process

i = 3. Sau fork sẽ có 16 process

Vậy tổng cộng có 16 process

## 6. Bài tập 3.14

A: pid = 0

B: pid = 2603

C: pid = 2603

D: pid = 2600