# 操作系统 lab4 进程同步

邱梓豪

141130077

## 实验步骤如下：

```
struct mySemaphore{         // definition of my semaphore
    int value;
    PCB* blocklist;
};
```

在 pcb.h 中定义信号量结构体

```
CFLAGS = -m32 -march=i386 -static \
     -fno-builtin -fno-stack-protector -fno-omit-frame-pointer \
     -Wall -Werror -O2 -I ../lib -I ../kernel/include
```

修改 app 中的 Makefile，然后在 type.h 中定义 sem_t

```
#ifndef __TYPES_H__
#define __TYPES_H__

typedef unsigned int   uint32_t;
typedef          int   int32_t;
typedef unsigned short uint16_t;
typedef          short int16_t;
typedef unsigned char  uint8_t;
typedef          char  int8_t;
typedef unsigned char  boolean;

typedef uint32_t size_t;
typedef int32_t  pid_t;

#include "pcb.h"
typedef struct mySemaphore sem_t;

#endif
```

再在 lib.h 中声明相关的函数

```h
#ifndef __lib_h__
#define __lib_h__

#include "types.h"

void printf(const char *format,...);

int fork();
void sleep(int time);
void exit();

int sem_init(sem_t *sem, uint32_t value);
int sem_post(sem_t *sem);
int sem_wait(sem_t *sem);
int sem_destroy(sem_t *sem);

#endif
```

在 syscall.c 中添加四个系统调用。

```c
int sem_init(sem_t *sem, uint32_t value){
    syscall(5, (uint32_t)sem, value, 0, 0, 0);
    return 0;
}

int sem_post(sem_t *sem){
    syscall(6, (uint32_t)sem, 0, 0, 0, 0);
    return 0;
}

int sem_wait(sem_t *sem){
    syscall(7, (uint32_t)sem, 0, 0, 0, 0);
    return 0;
}

int sem_destroy(sem_t *sem){
    return syscall(8, (uint32_t)sem, 0, 0, 0, 0);
}
```

在 kernel 的 irqHandle 中为系统调用定义相关的操作：

```
case 5: // init
        sem.value = tf->ecx;
        sem.wait = NULL;
        break;
case 7: // wait (P)
        sem.value--;
        if (sem.value<0){
            sem.wait = current;
            pcb_delete(current);
            current=idle;
        }
        break;
case 6: // post (V)
        sem.value++;
        if (sem.value<=0){
            pcb_add(&pcb_ready, sem.wait);
            sem.wait = NULL;
        }
        break;
case 8: sem.value=0;  // sem destory
        sem.wait = NULL;
        break;
```

最后的运行结果如下：

```
Father Process: Semaphore Initializing.
Father Process: Sleeping.
Child Process: Semaphore Waiting.
Child Process: In Critical Area.
Child Process: Semaphore Waiting.
Child Process: In Critical Area.
Child Process: Semaphore Waiting.
Father Process: Semaphore Posting.
Father Process: Sleeping.
Child Process: In Critical Area.
Child Process: Semaphore Waiting.
Father Process: Semaphore Posting.
Father Process: Sleeping.
Child Process: In Critical Area.
Child Process: Semaphore Destroying.
Father Process: Semaphore Posting.
Father Process: Sleeping.
Father Process: Semaphore Posting.
Father Process: Semaphore Destroying.
```