

# **操作系统 实验 4**

## **进程同步**

**白晋斌**

**171860607**

**810594956@qq.com**

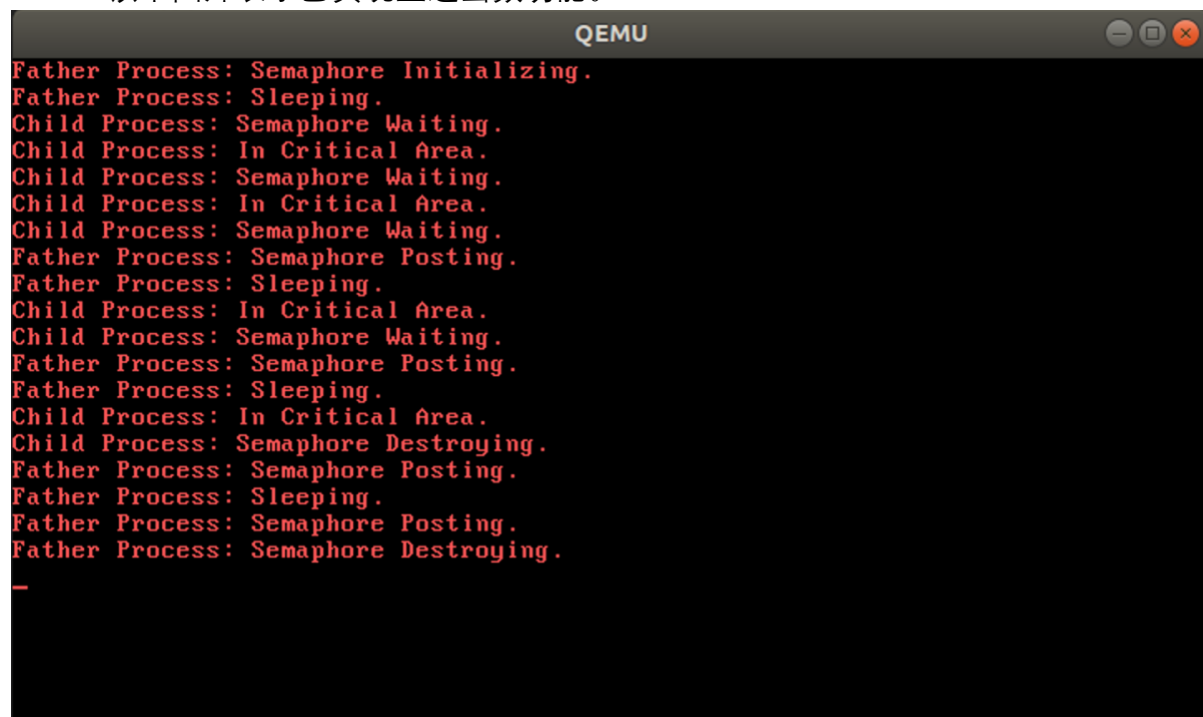
## 目录

一、重点，精华都在这里.....	3
1.1 实现 SEM_INIT、SEM_POST、SEM_WAIT、SEM_DESTROY 系统调用 .....	3
1.2 多进程进阶.....	3
1.3 信号量进程同步进阶 .....	3
二、实验要求.....	4
1.1. 实现 SEM_INIT、SEM_POST、SEM_WAIT、SEM_DESTROY 系统调用 .....	4
1.2. 多进程进阶.....	4
1.3. 信号量进程同步进阶 .....	4
三、实验过程.....	5
0.阅读框架代码.....	5
1. 实现 SEM_POST、SEM_WAIT、SEM_DESTROY 系统调用.....	5
1.1 补充完成 SEM_POST 函数.....	5
1.2 补充完成 SEM_WAIT 函数 .....	5
1.3 补充完成 SEM_DESTROY 函数 .....	5
1.4 输出 .....	5
2. 多进程进阶.....	6
3. 信号量进程同步进阶 .....	6
3.1 实现 getpid 系统调用 .....	6
3.2 编写测试用例 .....	6
3.3 输出 .....	6
四、拓展功能.....	7
1.对信号量的 PV 操作在终端做了回显 .....	7
2.提高程序鲁棒性 .....	7
3.供需均衡 .....	7
五、DEBUG 史 .....	7
六、友情鸣谢.....	7

## 一、重点，精华都在这里

### 1.1 实现 SEM\_INIT、SEM\_POST、SEM\_WAIT、SEM\_DESTROY 系统调用

以下图片表示已实现上述函数功能。



```
QEMU
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.
```

### 1.2 多进程进阶

修改 GDT 的 size 即可。

```
#define NR_SEGMENTS 18
```

### 1.3 信号量进程同步进阶

```
1  #include "lib.h"
2  #include "types.h"
3
4  #define mytest
5  //#define originaltest
6  int data = 0;
7
8  int uEntry(void) {
9
10 #ifdef mytest...
68 #endif
69
70 #ifdef originaltest...
107 #endif
108 }
109
```

这里的测试代码我们用上图所示方式控制。若使用框架初始代码则开启 `define originaltest`，若使用第三部分任务代码则开启 `define mytest`。提交版本默认为上图所示状态。

以下图片表示已实现上述函数功能。

```
QEMU
pid 2, producer 2, unlock
pid 4, consumer 2, try lock
pid 4, consumer 2, locked
pid 4, consumer 2, try consume, product 4
pid 4, consumer 2, consumed, product 4
pid 4, consumer 2, unlock
pid 1, producer 1, try lock
pid 1, producer 1, locked
pid 1, producer 1, produce, product 8
pid 1, producer 1, unlock
pid 5, consumer 3, try lock
pid 5, consumer 3, locked
pid 5, consumer 3, try consume, product 4
pid 5, consumer 3, consumed, product 4
pid 5, consumer 3, unlock
pid 2, producer 2, try lock
pid 2, producer 2, locked
pid 2, producer 2, produce, product 8
pid 2, producer 2, unlock
pid 6, consumer 4, try lock
pid 6, consumer 4, locked
pid 6, consumer 4, try consume, product 4
pid 6, consumer 4, consumed, product 4
pid 6, consumer 4, unlock
```

## 二、实验要求

本实验通过实现一个简单的生产者消费者程序，介绍基于信号量的进程同步机制。

1.1. 实现 SEM\_INIT、SEM\_POST、SEM\_WAIT、SEM\_DESTROY 系统调用

1.2. 多进程进阶

调整框架代码中的 gdt、pcb 等进程相关的数据和代码，使得你实现的操作系统最多支持到 8 个进程(包括内核 idle 进程)。

1.3. 信号量进程同步进阶

理解 1.1 节中的测试程序，实现两个生产者、四个消费者的生产者消费者问题，不需要考虑进程调度，公平调度就行。

1.以函数形式实现生产者和消费者，生产者和消费者的第一个参数为 int 类型，用于区分不同的生产者和消费者

2.每个生产者生产 8 个产品，每个消费者消费 4 个产品

3.先 fork 出 6 个进程，通过 getpid 获取当前进程的 pid，简单粗暴的以 pid 小的进程为生产者

4.信号量模拟 buffer 中的产品数量

5.保证 buffer 的互斥访问

6.每一行输出需要表达 pid i, producer/consumer j, operation(, product k)

7.operation 包括 try lock、locked、unlock、produce、try consume、consumed

8.除了 lock、unlock 的其他操作需要包含括号中的内容

9.i 表示进程号，j 表示生产者/消费者编号，k 表示生产者/消费者的生产/消费的第几个产品

### 三、实验过程

#### 0. 阅读框架代码

先学习研究 irqHandle.c 中的其他函数。

#### 1. 实现 SEM\_POST、SEM\_WAIT、SEM\_DESTROY 系统调用

##### 1.1 补充完成 SEM\_POST 函数

SEM\_POST 函数即 V 操作，我们所要做的即通过 sf->edx 取得信号量的编号，判断该编号是否合法，若不合法则设置返回值-1；合法则设置返回值 0，并将 sem 指向的信号量的 value 增一，若 value 取值不大于 0，则释放一个阻塞在该信号量上进程（即将该进程设置为就绪态），这里用到了双向链表作为队列，即将 sem[j].pcb.prev 所指向的进程设置为就绪态，并将其从链表中移出。通过 asm volatile("int \$0x20");陷入时钟中断。

##### 1.2 补充完成 SEM\_WAIT 函数

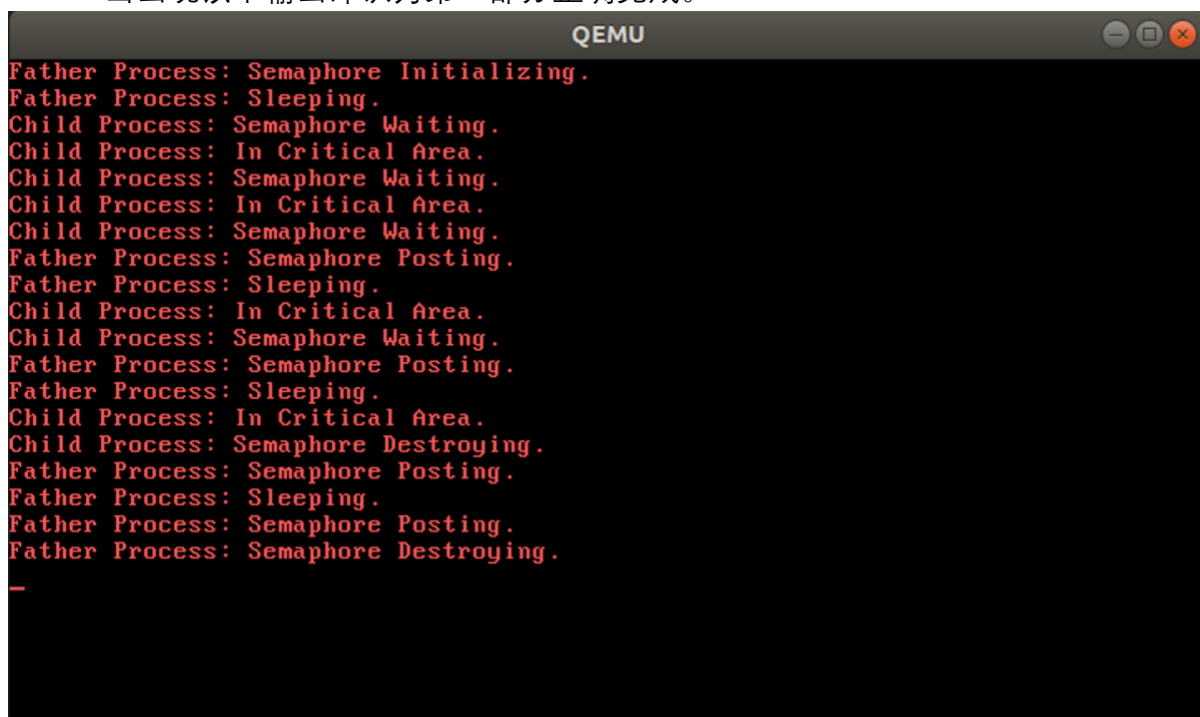
SEM\_WAIT 函数即 P 操作，我们所要做的即通过 sf->edx 取得信号量的编号，判断该编号是否合法，若不合法则设置返回值-1；合法则设置返回值 0，并将 sem 指向的信号量的 value 减一，若 value 取值小于 0，则阻塞自身，否则进程继续执行。阻塞自身的具体实现为将当前进程设置为阻塞态并修改 timeCount 为-1，并将当前进程串联入 sem[j].pcb.next。最后通过 asm volatile("int \$0x20");陷入时钟中断。

##### 1.3 补充完成 SEM\_DESTROY 函数

因为要求是“销毁 sem 指向的信号量，销毁成功则返回 0，否则返回-1，若尚有进程阻塞在该信号量上，可带来未知错误”，所以我们只需要通过 sf->edx 取得信号量的编号，判断该编号是否合法，若不合法则设置返回值-1；合法则设置返回值 0，并将当前信号量的 state 设置为 0。

##### 1.4 输出

当出现以下输出即认为第一部分正确完成。



```
QEMU
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.
```

## 2. 多进程进阶

- GDT 共有10项。第0项为空；第9项指向 TSS 段，内核代码段和数据段占了第1、2两项；剩下的6项可以作为3个用户进程的代码段和数据段

注意到 lab3 的 PPT 上有这样一句话，因此我们通过观察框架代码中的宏定义，修改 GDT 的 size 即

```
#define NR_SEGMENTS 18
```

即可。

## 3. 信号量进程同步进阶

### 3.1 实现 getpid 系统调用

在我们自己写的测试样例中要通过 getpid 获取当前进程号，通过模仿 fork 系统调用的实现，在 syscall.c, lib.h, irqHandle.c 等文件中添加相关宏定义、case 分支语句、函数即可。

这里可能出现报错 implicit declaration of function 'x'，其实是某些头文件未对函数声明，或者声明的类型与自己定义的类型不一致，注意一下即可。

### 3.2 编写测试用例

在这里我的实现为进程 ID 1 和 2 为生产者，进程 ID 3、4、5、6 为消费者。通过三个信号量，1 是 buf（模拟 buffer 中的产品数量），2 是 mutex（保证各个进程互斥），3 是 empty（判断 buffer 是否为空，防止消费者误入造成死锁）。之后模仿原测试用例的格式进行代码编写即可。详细代码见 app/main.c。

因为共 2 位生产者，每位生产 8 个产品，共 4 位消费者，每位消费 4 个产品。因此我们的生产者进程每次生产完产品都 sleep(128)，消费者进程每次消费完产品都 sleep(256)。

### 3.3 输出

出现如下输出即认为第三部分完成。

```
QEMU
pid 2, producer 2, unlock
pid 4, consumer 2, try lock
pid 4, consumer 2, locked
pid 4, consumer 2, try consume, product 4
pid 4, consumer 2, consumed, product 4
pid 4, consumer 2, unlock
pid 1, producer 1, try lock
pid 1, producer 1, locked
pid 1, producer 1, produce, product 8
pid 1, producer 1, unlock
pid 5, consumer 3, try lock
pid 5, consumer 3, locked
pid 5, consumer 3, try consume, product 4
pid 5, consumer 3, consumed, product 4
pid 5, consumer 3, unlock
pid 2, producer 2, try lock
pid 2, producer 2, locked
pid 2, producer 2, produce, product 8
pid 2, producer 2, unlock
pid 6, consumer 4, try lock
pid 6, consumer 4, locked
pid 6, consumer 4, try consume, product 4
pid 6, consumer 4, consumed, product 4
pid 6, consumer 4, unlock
```

## 四、拓展功能

### 1.对信号量的 PV 操作在终端做了回显

这样可以通过观察终端输出来分析自己程序的运行过程，从而判断运行逻辑是否正确。

### 2.提高程序鲁棒性

对信号量的 P 操作阻塞当前进程时，额外设置 timeCount 为-1，因为我们的调度函数比较厉害对-1 做了安全性检查，但换一个调度函数，若不设置 timeCount 为-1，则可能出现意料之外的 bug。

### 3.供需均衡

为了使第三部分的程序供需均衡，我们的生产者进程每次生产完产品都 sleep(128)，消费者进程每次消费完产品都 sleep(256)。

## 五、DEBUG 史

在实现 getpid 系统调用的过程中，可能出现报错 implicit declaration of function 'x'，其实是某些头文件未对函数声明，或者声明的类型与自己定义的类型不一致，注意一下即可。

## 六、友情鸣谢

感谢老师助教和各位群友鼎力相助！！！！