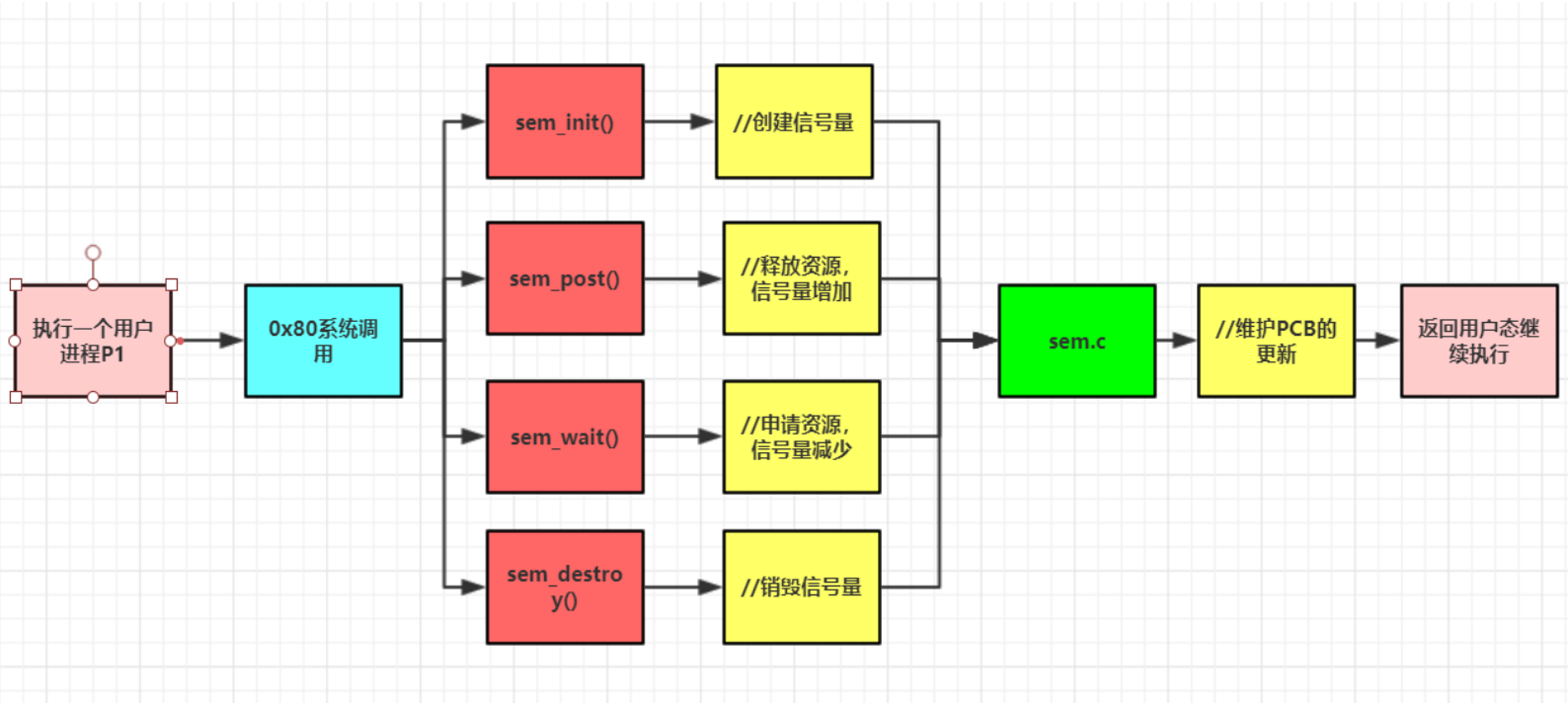
lab4 实验报告

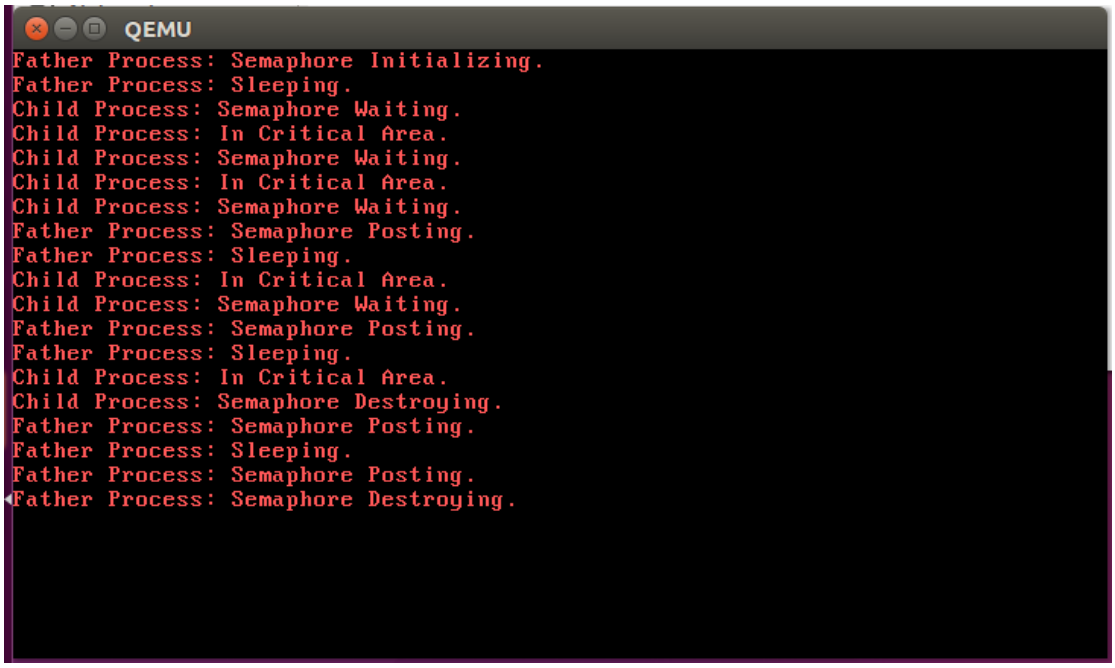
161220124 王明

1.1 代码流程与实验结果

代码流程图解：



运行结果如下：



1.2.实验流程

信号量队列的设计：

事实上完成这次实验只需要一个信号量就可以了，但是为了模拟得更加实际一点，我采用了一个队列（或者说一个数组）叫做 sem_queue[MAX_SEM_NUM]，给的长度是 20，也就是可以维护 20 个信号量（20 种资源），信号量的结构体维护了三个变量，一个是信号量的值、一个是因为信号量申请不成功而阻塞的进程（采用指针的格式维护一个链表）、一个是标记该信号量是否正在投入使用，结构如下：

```
5 struct Semaphore sem_queue[MAX_SEM_NUM];
```

```
52 struct Semaphore {
53     int value;
54     struct ProcessTable *list;
55     int used;
56 };
```

信号量创建：

遍历信号量队列，找到第一个没有被用过的信号量，将 sem 指向它，并且将它的值初始化为 value，维护的队列指针初始化为 NULL，将使用过的标记变量置为 1，代码如下：

```
64 int init_sem(int *sem, int value)
65 {
66         int flag = 0;
67         int i = 0;
68         for(int i = 0; i < MAX_SEM_NUM; i++)
69         {
70                 if(sem_queue[i].used == 0)
71                 {
72                         flag = 1;
73                         break;
74                 }
75         }
76         if(flag == 0)
77         {
78                 return -1;
79         }
80         else
81         {
82                 *sem = i;
83                 sem_queue[i].used = 1;
84                 sem_queue[i].value = value;
85                 return i;
86         }
87 }
```

信号量释放：

这个操作其实就是 V 操作，先将指定的信号量加 1，然后如果信号量的值小于等于 0，说明还有进程需要占用该信号量（也就是存在信号量维护的阻塞进程指针队列），这时通过指针找到队尾的 PCB 块，将它的状态设置为就绪态，移出阻塞进程队列，从而就可以使用

刚刚释放的资源去运行了，代码如下：

```
89 int post_sem(int *sem)
90 {
91         return V(&sem_queue[*sem]);
92 }

28 int V(struct Semaphore *sem)
29 {
30         sem->value++;
31         if(sem -> value <= 0)
32         {
33                 if(sem->list == NULL)
34                         return 0;
35                 struct ProcessTable *temp = sem->list;
36                 while(temp->next != NULL)
37                         temp = temp->next;
38                 temp->state = STATE_RUNNABLE;
39                 if(temp == sem->list)
40                         sem->list = NULL;
41                 else
42                 {
43                         struct ProcessTable *tmp = sem->list;
44                         while(tmp->next != temp)
45                                 tmp = tmp->next;
46                         tmp->next = NULL;
47                 }
48                 update_s();
49         }
50         return 0;
51 }
```

信号量申请：

这个操作其实就是 P 操作，先将信号量减 1，此时如果信号量小于 0，那么代表资源不够，申请资源失败，这时就要将当前进程的状态设置为 STATE_BLOCK_SEM，也就是阻塞状态，这时把当前进程的 PCB 放进该信号量维护的阻塞进程队列，再将当前进程置空，代码如下：

```
94 int wait_sem(int *sem)
95 {
96         return P(&sem_queue[*sem]);
97 }

15 int P(struct Semaphore *sem)
16 {
17         sem->value--;
18         if(sem->value < 0)
19         {
20                 now->state = STATE_BLOCK_SEM;
21                 now->next = sem->list;
22                 sem->list = now;
23                 now = NULL;
24         }
25         return 0;
26 }
```

信号量销毁：

首先检查该信号量维护的阻塞进程队列是否为空，如果不为空则不能销毁，发生错误，此时用 assert(0)强行终止程序，如果可以销毁，则将 value 置为-1，并且将该信号量标记为可以使用的状态（即 used = 0），代码如下：

```
 99 int destroy_sem(int *sem)
100 {
101         if(sem_queue[*sem].list != NULL)
102                 assert(0);
103         sem_queue[*sem].value = -1;
104         sem_queue[*sem].used = 0;
105         return 0;
106 }
```

1.3. 实验感想

这次实验相对于上一次来讲显得比较简单，因为就是实现了 4 个系统调用，在上次进程切换的基础上加上了信号量的限制（在我理解就是模拟计算资源的申请和释放）。然后键盘驱动有点没头绪，不知道要怎么做，感觉要留到 lab5 去实现了。还是要多学多学多学多学多学多学多学。