

# 实验报告 4

221275207 喻思文

[221275027@smail.nju.edu.cn](mailto:221275027@smail.nju.edu.cn)

## 一、实验进度

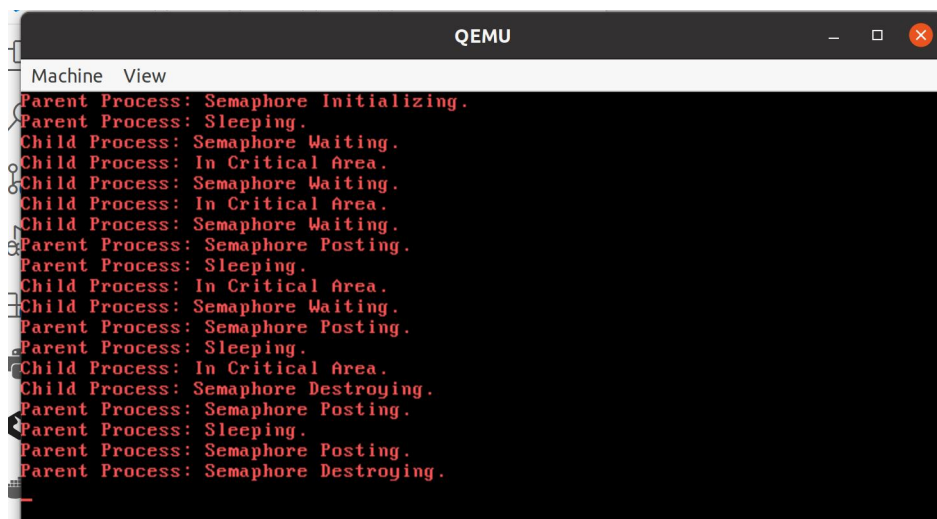
格式化输入（scanf）、信号量（semaphore 的 Init、wait、post、destroy）、共享变量（sharedVar 的 create、destroy、read、write）、基于信号量的进程同步问题（实际实现生产者-消费者、读者-写者）均已实现，未实现 XCHG 指令的自旋锁。

## 二、运行截图

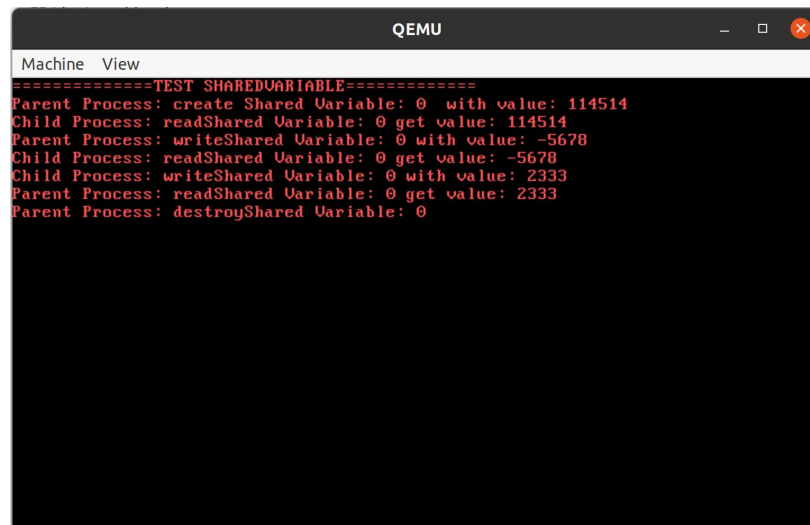
- scanf 功能



- 信号量测试

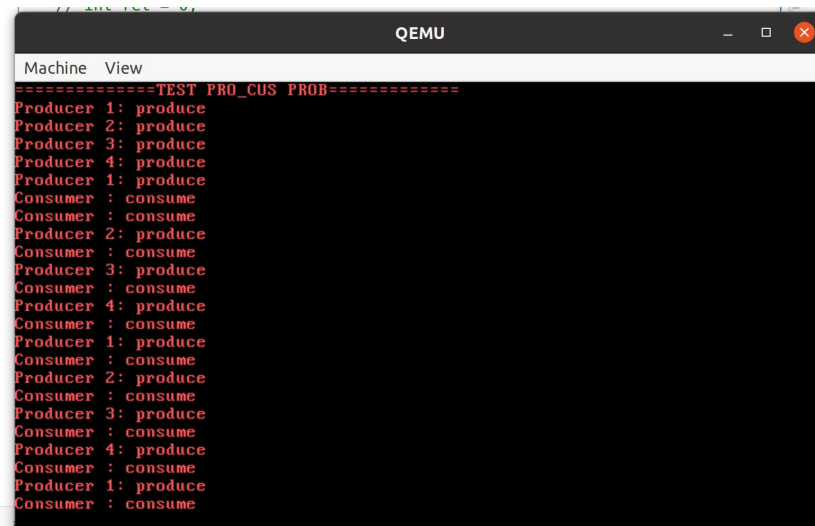


- 共享变量测试



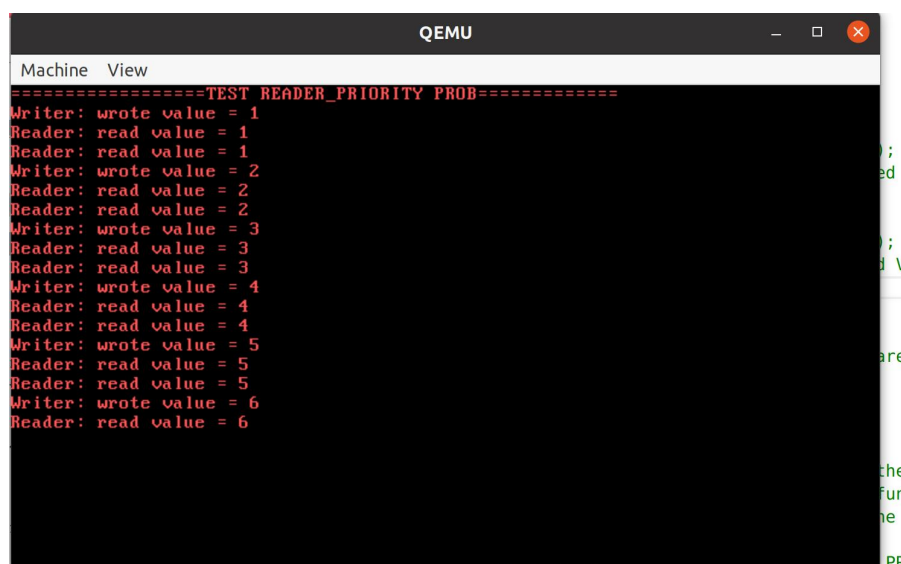
```
Machine View
=====TEST SHARED VARIABLE=====
Parent Process: create Shared Variable: 0 with value: 114514
Child Process: readShared Variable: 0 get value: 114514
Parent Process: writeShared Variable: 0 with value: -5678
Child Process: readShared Variable: 0 get value: -5678
Child Process: writeShared Variable: 0 with value: 2333
Parent Process: readShared Variable: 0 get value: 2333
Parent Process: destroyShared Variable: 0
```

- 生产者-消费者问题



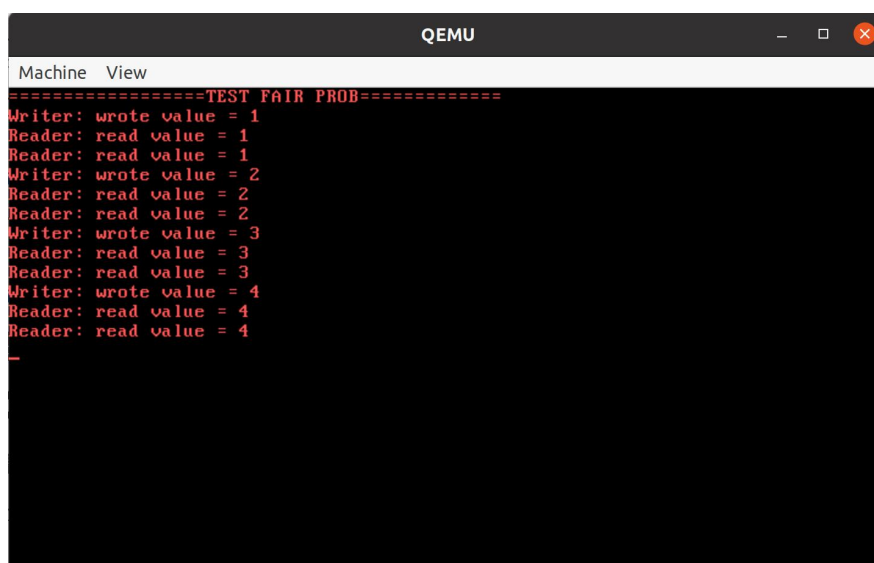
```
Machine View
=====TEST PRO_CUS PROB=====
Producer 1: produce
Producer 2: produce
Producer 3: produce
Producer 4: produce
Producer 1: produce
Consumer : consume
Consumer : consume
Producer 2: produce
Consumer : consume
Producer 3: produce
Consumer : consume
Producer 4: produce
Consumer : consume
Producer 1: produce
Consumer : consume
Producer 2: produce
Consumer : consume
Producer 3: produce
Consumer : consume
Producer 4: produce
Consumer : consume
Producer 1: produce
Consumer : consume
```

- 读者-写者的读者优先策略



```
Machine View
=====TEST READER_PRIORITY PROB=====
Writer: wrote value = 1
Reader: read value = 1
Reader: read value = 1
Writer: wrote value = 2
Reader: read value = 2
Reader: read value = 2
Writer: wrote value = 3
Reader: read value = 3
Reader: read value = 3
Writer: wrote value = 4
Reader: read value = 4
Reader: read value = 4
Writer: wrote value = 5
Reader: read value = 5
Reader: read value = 5
Writer: wrote value = 6
Reader: read value = 6
```

- 读者-写者的公平策略



```
Machine View
=====TEST FAIR PROB=====
Writer: wrote value = 1
Reader: read value = 1
Reader: read value = 1
Writer: wrote value = 2
Reader: read value = 2
Reader: read value = 2
Writer: wrote value = 3
Reader: read value = 3
Reader: read value = 3
Writer: wrote value = 4
Reader: read value = 4
Reader: read value = 4
```

### 三、实验思路

#### （一）scanf 功能

##### 1. keyboardHandle

当至少有一个进程输入被阻塞，操作如下：

- 1) 将 value 递增，表示“已有一个输入事件被处理”。
- 2) 从阻塞队列中取出进程：

通过链表指针运算找到第一个被阻塞的进程控制块，修改进程状态，将进程设为可运行状态。重置调度相关字段，让进程尽快被调度。

- 3) 调整链表指针，将唤醒的进程从阻塞队列中移除。

##### 2. sysReadStdIn

- 1) 检查标准输入设备的状态值：

- 如果 value < 0，表示设备不可用（如未初始化或错误），直接返回 -1（通过 eax 寄存器传递错误码）。

- 如果 value == 0，表示当前没有输入数据可用：将 value 设为 -1（标记设备为“等待数据”状态）。将当前进程通过双向链表操作加入标准输入设备的阻塞队列。设置进程状态为 STATE\_BLOCKED，并通过 int \$0x20 触发调度器切换进程。

## 2) 读取数据

从 `ds` 寄存器读取用户态数据段选择子，用于访问用户内存；从 `edx` 读取用户缓冲区的起始地址；从 `ebx` 获取缓冲区大小。

## 3) 处理字符

循环从 `keyBuffer` 读取字符，`getChar` 处理原始键盘扫描码到 ASCII 的转换，非空字符通过 `movb` 指令写入用户缓冲区，忽略空字符。

# (二) 信号量

## 1. semInit

1) 从 `edx` 寄存器中获取用户传递的信号量初始值，并存在信号的 `value` 中。

2) 遍历信号量数组，找到一个未被使用的信号量 (`state == 0`)，并初始化，将其状态设置为已被占用。当前进程的 `eax` 寄存器设置为信号量的索引 `i`，直接返回，表示初始化成功。

若未找到可用信号量，则将当前进程的 `eax` 寄存器设置为 `-1`。

## 2. semWait

1) 从 `edx` 寄存器中提取信号量索引，检查信号量 ID 的有效性。如果超出有效范围，将当前进程的 `eax` 寄存器设置为 `-1`，表示操作失败。

如果有效，信号量的值-1，表示申请一个资源，并将 `eax` 设置为 `0`，表示操作成功。

2) 如果信号量值变为负（表示资源不可用），需要阻塞当前进程：

- 将当前进程的 `blocked` 结构插入到信号量的等待队列（双向链表）中，当前进程的 `blocked` 结构被添加到信号量队列的头部。

- 将当前进程状态设置为 `STATE_BLOCKED`（阻塞状态），触发调度器，切换到其他进程。

## 3. semPost

要处理的是合法情况。此时信号量的值+1，表示释放一个资源。接着唤醒等待进程：

如果信号量的值仍小于等于 0，说明有进程在等待该信号量。那么就从信号量的等待队列中取出第一个等待的进程，将状态设为 `STATE_RUNNABLE`，并重置其 `sleepTime` 和 `timeCount`。调整信号量等待队列的指针，将其从队列中移除。

#### 4. `semDestroy`

首先依然是检查信号量状态，如果信号量已被销毁或无效，直接返回。否则设置 `eax` 为 0，重置信号量属性。接着重置 PCB 链表，即，将信号量的等待队列（pcb 链表）初始化为自环（`next` 和 `prev` 指向自身），表示无进程等待。最后通过内联汇编触发软中断。

### （三）共享变量

#### 1. `kvm.c/initSharedVariable`

类似已给出的 `initSem` 函数，而且这个是全球数组，只需要循环遍历初始化 `state=0` 和 `value=0` 即可。

#### 2. `sysSVarCreate`

遍历数组，如果找到空闲槽位，将其 `state` 设为 1，将 `ebx` 的值赋给该共享变量的 `value` 字段，将槽位索引 `i` 通过 `sf->eax` 返回。

#### 3. `sysSVarDestroy`

遍历寻找待销毁的变量，若变量不存在或者未被使用则返回。否则将共享变量的状态标记为未使用，重置共享变量的值为 0。

#### 4. `sysSVarRead`

仍然遍历查找。如果共享变量有效，将其值存入 `eax` 寄存器，作为返回值传递给用户空间。

#### 5. `sysSVarWrite`

与 `read` 相反，如果查找存在，将 `ebx` 寄存器中的值写入对应共享变量的 `value`。

### （四）生产者-消费者问题

#### 1. 信号量

`empty`: 初始值为 5，表示缓冲区中可用的空位数量

**full:** 初始值为 0，表示缓冲区中已填充的项目数量

**mutex:** 初始值为 1，作为互斥锁，保护临界区

## 2. 子进程处理

循环 4 次，创建 4 个子进程，只有父进程 (`id == 0`) 才会继续调用 `fork()`，子进程 (`id > 0`) 会跳出循环，最终会有 1 个父进程和 4 个子进程。

3. 消费者部分（父进程）：等待缓冲区中有项目可消费，获取互斥锁，进入临界区，进行消费。然后释放互斥锁，增加空位计数

4. 生产者部分（子进程）：等待缓冲区有空位，获取互斥锁，进入临界区，进行生产。然后释放互斥锁，增加已填充项目计数

## （五）读者-写者问题

### 1. 读者优先

只要有一个读者正在读，后续读者可以直接进入，而写者必须等待所有读者完成。

#### 1) 信号量

- **mutex:** 保护 `read_count` 变量的互斥访问（二进制信号量，初始值 1）。
- **wrt:** 控制写者访问共享变量的互斥信号量（初始值 1）。
- **read\_count:** 记录当前活跃的读者数量。

#### 2) 共享变量:

**shared\_var:** 通过 `createSharedVariable` 初始化的共享变量（初始值 0）。

#### 3) 读者流程（子进程）

• **进入临界区:** 通过 `P(mutex)` 获取 `read_count` 的访问权。增加 `read_count`，若这是第一个读者，则通过 `P(wrt)` 阻塞写者。释放 `mutex`，允许其他读者进入。

• **读操作:** 读取 `shared_var` 的值并打印。

• **离开临界区:** 再次获取 `mutex`，减少 `read_count`。若没有活跃读者，通过 `V(wrt)` 唤醒可能的阻塞写者。释放 `mutex`。

#### 4) 写者流程（父进程）

- **获取写权限:** 通过 `P(wrt)` 等待 `wrt` 信号量。

- 写操作：读取 `shared_var` 的值，递增后写回。
- 释放权限：通过 `V(wrt)` 允许其他写者或读者（若没有活跃读者）。

#### 5) 读者优先的体现：

只要有一个活跃读者，后续读者可以直接进入（通过 `read_count` 计数），而写者必须等待所有读者完成（`read_count == 0` 时才会释放 `wrt`）。

但是如果读者持续到来，写者可能长期阻塞，导致写者饥饿。

## 2. 读写公平

### 1) 信号量

- `mutex`: 保护 `readcount` 变量的互斥信号量
- `rw`: 读写互斥信号量，确保写操作独占访问
- `w`: 公平性信号量，防止读者或写者饥饿
- `readcount`: 记录当前活跃的读者数量

### 2) `shared_var`: 共享变量，将被读写

3) 读者流程：先获取公平锁 `w`，防止写者饥饿；更新读者计数，如果是第一个读者则获取读写锁；执行读取（共享变量）操作；更新读者计数，如果是最后一个读者则释放读写锁。

4) 写者流程：获取公平锁 `w` 和读写锁 `rw`；执行写操作（对共享变量读取、递增、写入）；释放锁。

## 四、思考题

本次实验暂无思考题