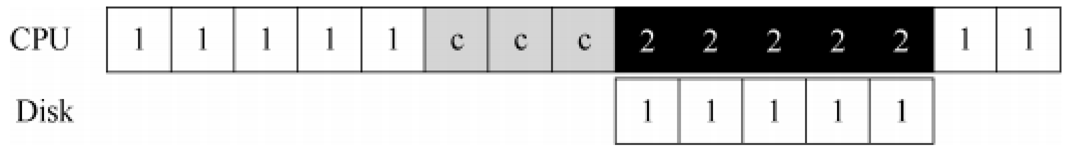
1. pros and cons of polling and interrupt-based I/O

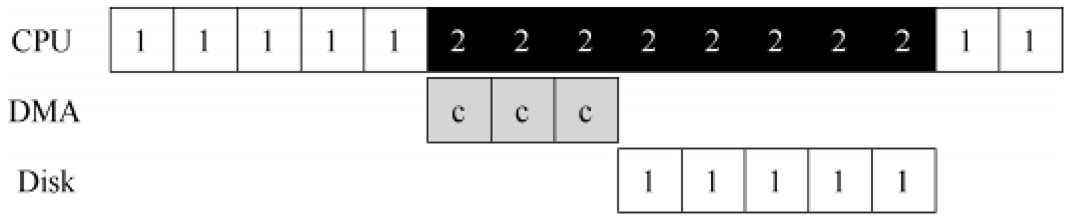
- 轮询(Polling):
  - 优点：简单且有效，对于fast device，polling更快
  - 缺点：轮询过程比较低效，在等待设备执行完成命令时浪费大量 CPU 时间
- 中断(interrupt-based):
  - 优点：CPU不必等待设备完成命令，而是可以让设备对应的进程睡眠，CPU切换执行其他任务，在slow device上，中断效率更高
  - 缺点：Context Switch的代价较高，假设有一个高性能的设备，那么CPU频繁切换进程反而会使效率变低。

2. differences between PIO and DMA?

- 编程的 I/O（programmed I/O，PIO）,指主CPU参与数据的搬移，如将一个磁盘块写入磁盘。它的缺点是，当需要传输的数据很多时，传输过程会阻塞CPU执行其他进程。



- DMA（Direct Memory Access）引擎是系统中的一个特殊设备，它可以协调完成内存和设备间的数据传递，不需要 CPU 介入。操作系统会告诉 DMA引擎数据在内存的位置，要拷贝的大小以及要拷贝到哪个设备。在此之后，操作系统就可以处理其他请求了。当 DMA 的任务完成后，DMA 控制器会抛出一个中断来告诉操作系统自己已经完成数据传输。

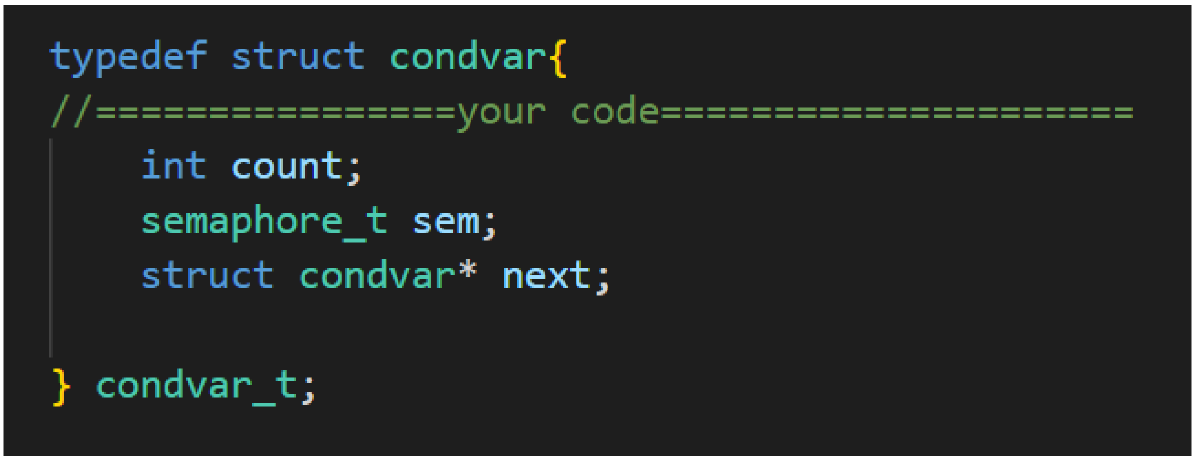


3. protect memory-mapped I/O and explicit I/O instructions from being abused by malicious user process

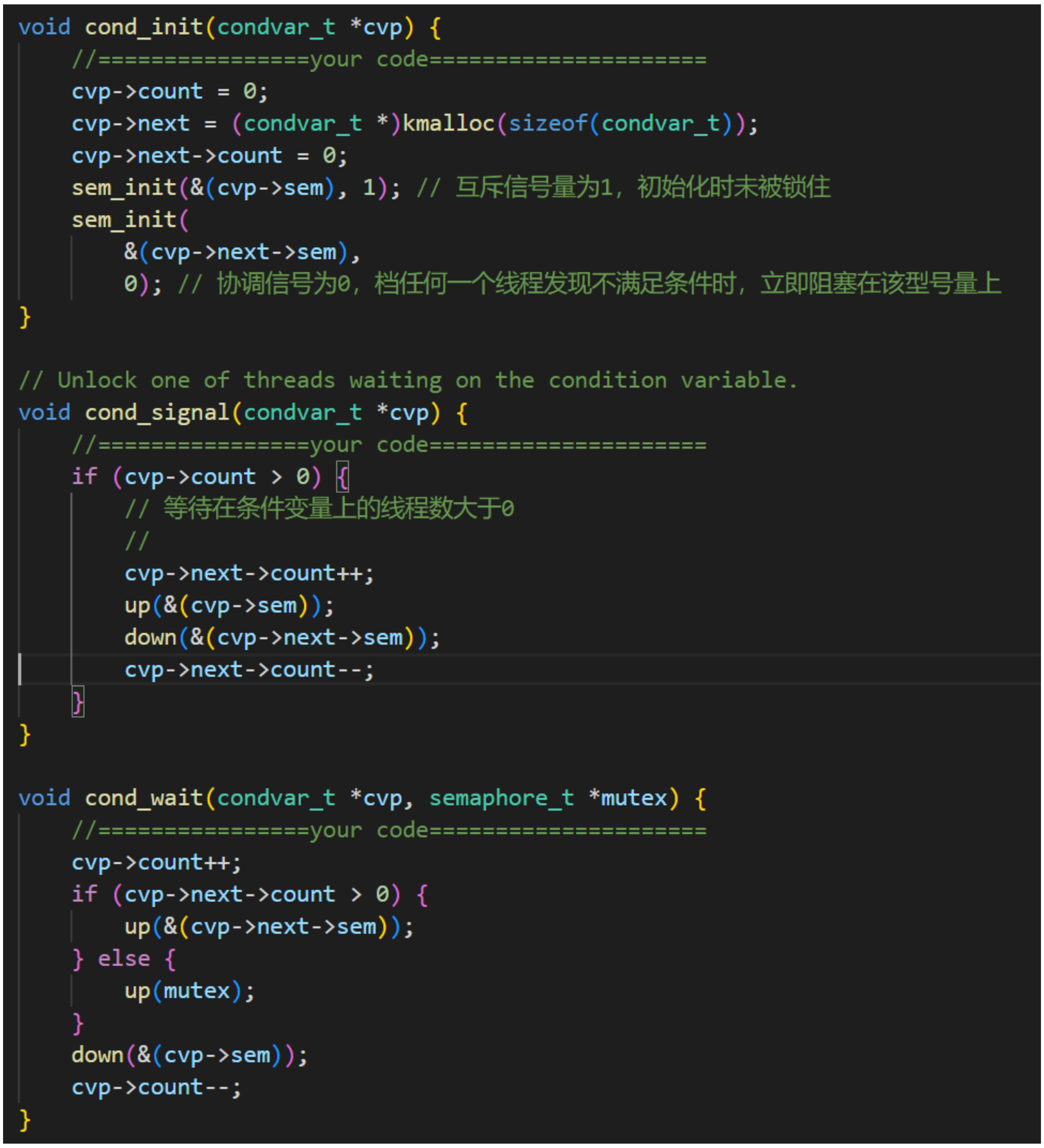
这些I/O指令通常是特权指令（privileged），操作系统是唯一可以直接与设备交互的实体。

4.

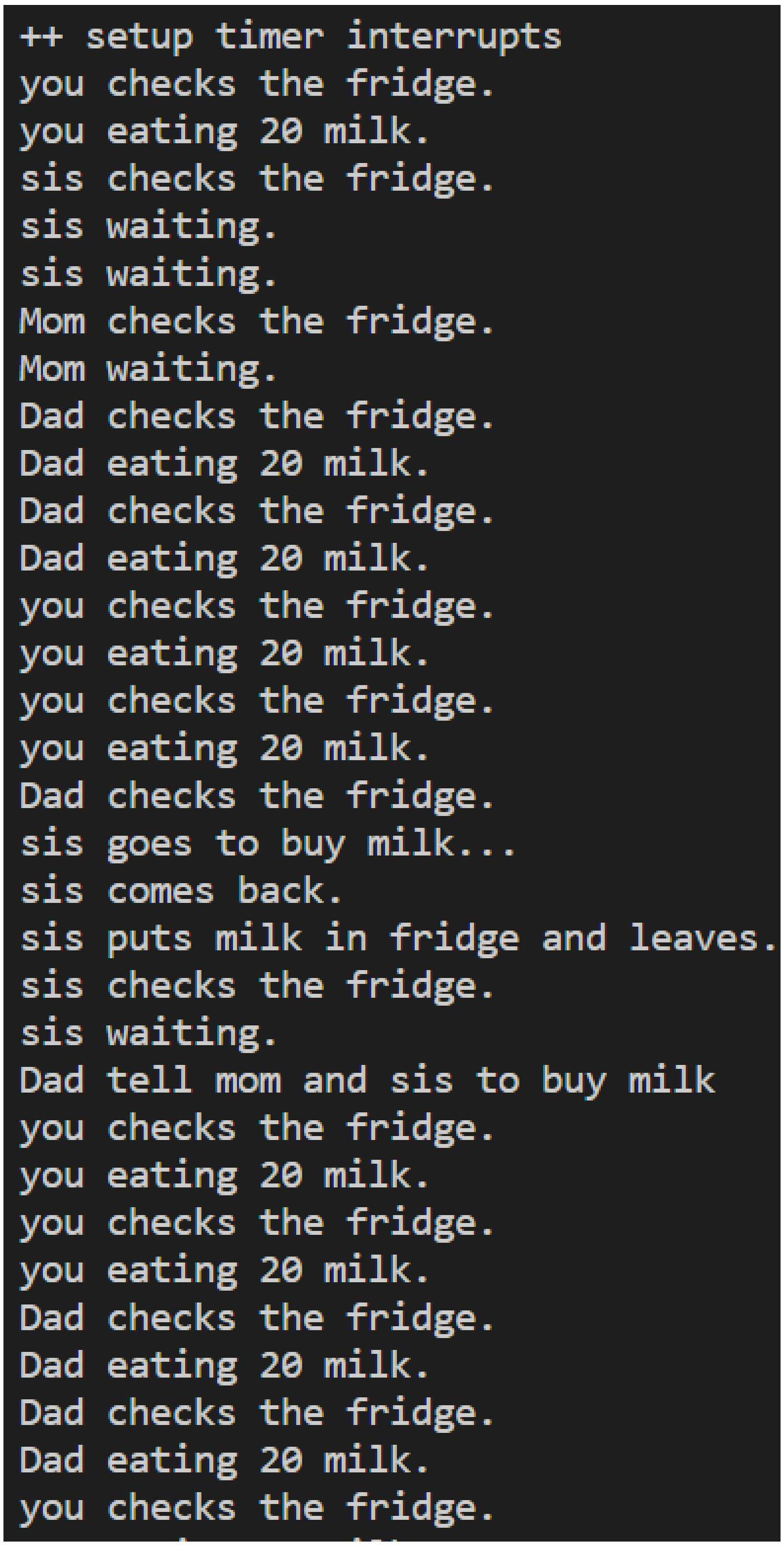
The design idea of my implementation is to use a semaphore to provide mutual exclusion on the condition variable, and another semaphore to coordinate the waiting of threads on the condition variable. Each condition variable has its own semaphore and a count to track the number of threads waiting on it. When a thread signals the condition variable, it wakes up one waiting thread by releasing the coordinating semaphore. When a thread waits on the condition variable, it increments the count and checks if there are any waiting threads already. If there are, it immediately signals the coordinating semaphore to allow one of the waiting threads to acquire the lock. If there are no waiting threads, it releases the lock and waits on the condition variable semaphore. The count is used to ensure proper signaling and waiting behavior when multiple threads are waiting on the condition variable.



```
typedef struct condvar{
//================your code=====================
    int count;
    semaphore_t sem;
    struct condvar* next;

} condvar_t;
```



```
void cond_init(condvar_t *cvp) {
    //================your code=====================
    cvp->count = 0;
    cvp->next = (condvar_t *)kmalloc(sizeof(condvar_t));
    cvp->next->count = 0;
    sem_init(&(cvp->sem), 1); // 互斥信号量为1，初始化时未被锁住
    sem_init(
        &(cvp->next->sem),
        0); // 协调信号为0，档任何一个线程发现不满足条件时，立即阻塞在该型号量上
}

// Unlock one of threads waiting on the condition variable.
void cond_signal(condvar_t *cvp) {
    //================your code=====================
    if (cvp->count > 0) {
        // 等待在条件变量上的线程数大于0
        //
        cvp->next->count++;
        up(&(cvp->sem));
        down(&(cvp->next->sem));
        cvp->next->count--;
    }
}

void cond_wait(condvar_t *cvp, semaphore_t *mutex) {
    //================your code=====================
    cvp->count++;
    if (cvp->next->count > 0) {
        up(&(cvp->next->sem));
    } else {
        up(mutex);
    }
    down(&(cvp->sem));
    cvp->count--;
}
```



```
++ setup timer interrupts
you checks the fridge.
you eating 20 milk.
sis checks the fridge.
sis waiting.
sis waiting.
Mom checks the fridge.
Mom waiting.
Dad checks the fridge.
Dad eating 20 milk.
Dad checks the fridge.
Dad eating 20 milk.
you checks the fridge.
you eating 20 milk.
you checks the fridge.
you eating 20 milk.
Dad checks the fridge.
sis goes to buy milk...
sis comes back.
sis puts milk in fridge and leaves.
sis checks the fridge.
sis waiting.
Dad tell mom and sis to buy milk
you checks the fridge.
you eating 20 milk.
you checks the fridge.
you eating 20 milk.
Dad checks the fridge.
Dad eating 20 milk.
Dad checks the fridge.
Dad eating 20 milk.
you checks the fridge.
```