1. (1) **Pros of polling:** Make the process of getting a device to do something simple and working.

   Cons of polling: It is inefficient because it wastes much CPU time to wait for the device to complete its activity.

   Pros of interrupt-based I/O: It can lower CPU overhead by issuing a request, putting the calling process to sleep, and context switch to another task. Then as a hardware interrupt is raised, the previously process waiting for the I/O will then proceed.

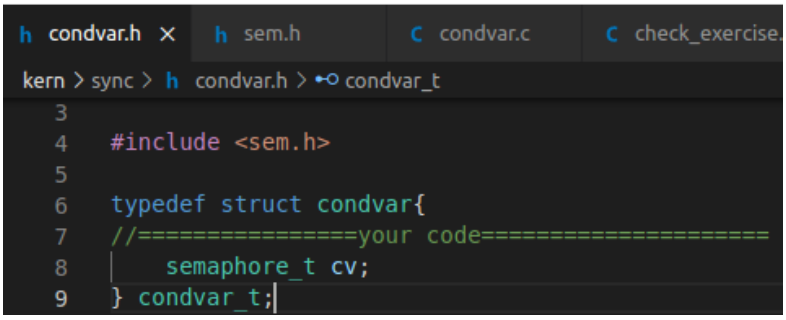
   Cons of interrupt based I/O: For a fast device, an interrupt will slow down the system compared to polling, because polling in a fast device can return the result in its first poll. Also, in networks it is better not to use interrupts because it may cause livelock to the system, making the system unable to service the requests but only process the enormous interrupts.

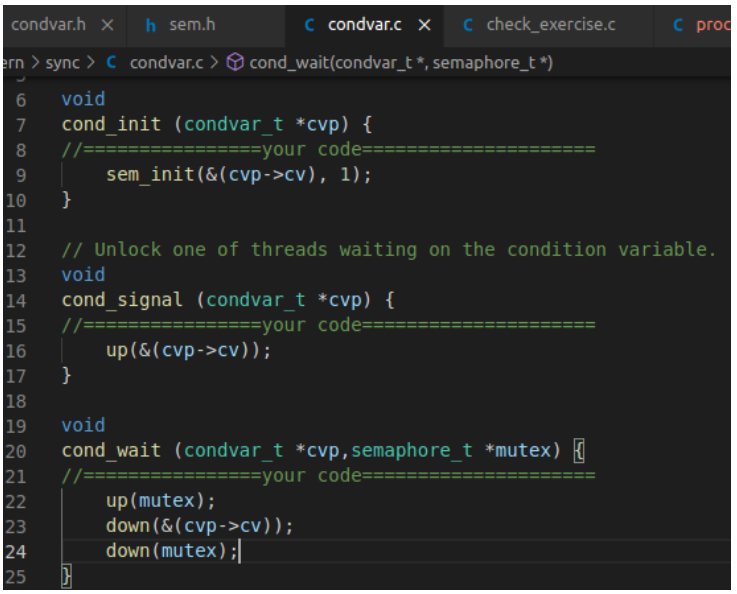
   (2) PIO involves the CPU to deal with data movement, but DMA involves a special DMA engine, a specific device within a system, to deal with data movement. Generally speaking, using DMA makes data movement more efficient because it allows the CPU to run other tasks while DMA focuses on copying data.

   (3) Explicit I/O: Execute privileged instructions to give the device orders, so that the OS is the only entity that is allowed to directly interact with devices, excluding abuse from user process.

   Memory-mapped I/O: Device registers are used as memory locations, mapping them to the device instead of main memory. Therefore only the OS can read or write the address instead of malicious user process.

2. Design idea: Use the structure of semaphore to define condition variable. Since condition variable is similar to semaphore, in the init function, I simply initialize a semaphore. In the signal function is the lock process, which is the up function of semaphore. In the wait function is the unlock process, so I use invoke the down function of semaphore. Meanwhile, I unlock the mutex at the beginning of wait function and lock it at the end of wait function, so that only one process unlocks the conditional variable at a time.





```
问题    输出    调试控制台    终端

OS is loading ...

memory management: default_pmm_manager
physcial memory map:
  memory: 0x08800000, [0x80200000, 0x885fffff].
sched class: stride_scheduler
SWAP: manager = fifo swap manager
++ 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.
Dad tell mom and sis to buy milk
sis goes to buy milk...
sis comes back.
sis puts milk in fridge and leaves.
sis checks the fridge.
sis waiting.
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.
Dad eating 20 milk.
Dad checks the fridge.
Dad eating 20 milk.
you checks the fridge.
you tell mom and sis to buy milk
Mom goes to buy milk...
Mon comes back.
Mom puts milk in fridge and leaves.
Mom checks the fridge.
Mom waiting.
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.
you eating 20 milk.
you checks the fridge.
you eating 20 milk.
Dad checks the fridge.
Dad tell mom and sis to buy milk
sis goes to buy milk...
sis comes back.
sis puts milk in fridge and leaves.
sis checks the fridge.
sis waiting.
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.
Dad eating 20 milk.
Dad checks the fridge.
Dad eating 20 milk.
you checks the fridge.
you tell mom and sis to buy milk
Mom goes to buy milk...
Mon comes back.
Mom puts milk in fridge and leaves.
Mom checks the fridge.
Mom waiting.
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.
you eating 20 milk.
you checks the fridge.
you eating 20 milk.
Dad checks the fridge.
Dad tell mom and sis to buy milk
sis goes to buy milk...
sis comes back.
sis puts milk in fridge and leaves.
```

3. Design idea: Use three condition variables, named con1, cond2, cond3. When worker1 receives cond3, he signals cond1 to worker2. When worker2 receives cond1, he signals cond2 to worker3. When worker3 receives cond2, he signals cond3 to worker1. Therefore each worker tells the next one sequentially that he has finished his part so that the next one can begin. Meanwhile, I use do_sleep function to delay some time so that the sequence of each worker can be presented one by one clearly in the terminal. Also, to ensure one person works at a time, I use semaphore as a lock at the beginning and ending of each while loop.


```c
struct proc_struct *pworker1,*pworker2,*pworker3;
condvar_t cond1;
condvar_t cond2;
condvar_t cond3;
semaphore_t mutex;

void worker1(int i)
{
    do_sleep(10);
    while (1)
    {
        down(&mutex);
        cprintf("make a bike rack\n");
        cond_wait(&cond3, &mutex);
        cond_signal(&cond1);
        up(&mutex);
        do_sleep(100);
    }
}


void worker2(int i)
{
    do_sleep(50);
    while (1)
    {
        down(&mutex);
        cprintf("make two wheels\n");
        cond_wait(&cond1, &mutex);
        cond_signal(&cond2);
        up(&mutex);
        do_sleep(100);
    }

}




void worker3(int i){
    do_sleep(100);
    while (1)
    {
        down(&mutex);
        cprintf("assemble a bike\n");
        cond_wait(&cond2, &mutex);
        cond_signal(&cond3);
        up(&mutex);
        do_sleep(100);
    }

}


void check_exercise(void){

    //initial
    sem_init(&(mutex),1);
    cond_init(&cond1);
    cond_init(&cond2);
    cond_init(&cond3);

    int pids[3];
    int i =0;
    pids[0]=kernel_thread(worker1, (void *)i, 0);
    pids[1]=kernel_thread(worker2, (void *)i, 0);
    pids[2]=kernel_thread(worker3, (void *)i, 0);
    pworker1 = find_proc(pids[0]);
    set_proc_name(pworker1, "worker1");
    pworker2 = find_proc(pids[1]);
    set_proc_name(pworker2, "worker2");
    pworker3 = find_proc(pids[2]);
    set_proc_name(pworker3, "worker3");
}
```

```
memory management: default_pmm_manager
physcial memory map:
  memory: 0x08800000, [0x80200000, 0x885fffff].
sched class: stride_scheduler
SWAP: manager = fifo swap manager
++ setup timer interrupts
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
make two wheels
assemble a bike
make a bike rack
```