Assignment8

11911839 聂雨荷

Q1

[50 pts] Please read "Three Easy Pieces" Ch36 https://pages.cs.wisc.edu/~remzi/OSTEP/file-devices.pdf, and answer the following questions:

(1) What are the pros and cons of polling and interrupt-based I/O?

(2) What are the differences between PIO and DMA?

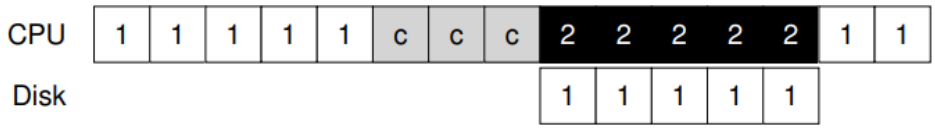
(3) How to protect memory-mapped I/O and explicit I/O instructions from being abused by malicious user process?

(1)

	polling	interrupt-based I/O
pros	being simple and working device is fast: use poll	allow for overlap of computation and I/O, improve utilization device is slow: use interrupt
cons	inefficient, it wastes a great deal of CPU time just waiting for the device to complete its activity, instead of switching to another ready process and thus better utilizing the CPU	interrupts is not always the best solution if the device performs its tasks very quickly, this will slow down the system network problem, when a huge stream of incoming packets each generate an interrupt, it is possible for the OS to livelock, that is, find itself only processing interrupts and never allowing a user-level process to run and actually service the requests

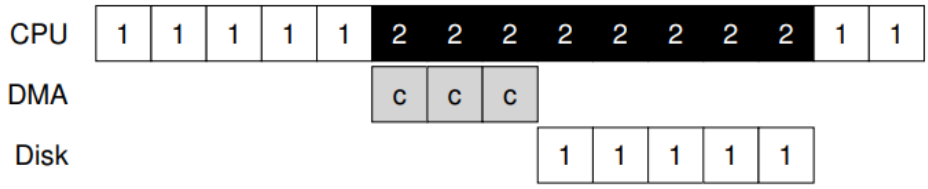
(2)

- PIO (Programmed I/O)



  - When OS sends some data down to the data register, multiple writes would need to take place to transfer a disk block to the device. When the main CPU is involved with the data movement, it is referred to programmed I/O (PIO).

    - With PIO, the CPU spends too much time moving data to and from devices by hand
- DMA (Direct Memory Access)



    - A DMA engine is essentially a very specific device within a system that can orchestrate transfers between devices and main memory without much CPU intervention.
    - DMA works as follows. To transfer data to the device, for example, the OS would program the DMA engine by telling it where the data lives in memory, how much data to copy, and which device to send it to. At that point, the OS is done with the transfer and can proceed with other work. When the DMA is complete, the DMA controller raises an interrupt, and the OS thus knows the transfer is complete.
    - You can see that the copying of data is now handled by the DMA controller

(3)

explicit I/O instructions	memory-mapped I/O
privilege: The OS controls devices, and the OS thus is the only entity allowed to directly communicate with them	access particular register: OS issues a load (to read) or store (to write) the address; the hardware then routes the load/store to the device instead of main memory

Q2

[50 pts] Please implement the condition variable in `ucore` by the already implemented wait queue or semaphore.

Requirements:

1. Please complete the definition of the condition variable in `condvar.h`
2. Please implement the related functions of condition variables in `condvar.c`
3. We have used these functions in `check_milk`, please make sure your implementation can make `check_milk` run in valid order. (Please release annotations of `check_milk` in `init_main` in `proc.c` for testing, and annotate `check_sync`).

Please include your design idea, code(screen-shot) and the running result(screen-shot) in your report

condvar.h

```
typedef struct condvar{
    semaphore_t sem;
} condvar_t;
```

condvar.c

design idea:

- user semaphore to implement condition variable
- semaphore represents the condition

Function	Description
`pthread_cond_wait`	release lock, put thread to sleep until condition is signaled; when thread wakes up again, re-acquire lock before returning
`pthread_cond_signal`	wake up at least one of the threads that are blocked on the specified condition variable; if more than one thread is blocked on a condition variable, the schedule policy shall determine the order in which threads are unblocked

code

```
void cond_init (condvar_t *cvp) {
//================your code====================
    sem_init(&cvp->sem, 0);
}

// unlock one of threads waiting on the condition variable.
void cond_signal (condvar_t *cvp) {
//================your code====================
    up(&cvp->sem);
}

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

running result