CS 61C OS & I/O

Summer 2020 Discussion 10: July 22, 2019

1 Pre-Check

This section is designed as a conceptual check for you to determine if you conceptually understand and have any misconceptions about this topic. Please answer true/false to the following questions, and include an explanation:

1.1 Polling and interrupts are only relevant concepts for low level programming.

false

1.2 Memory-mapped IO only works with polling.

false

1.3 Responsibilities of the OS include loading programs, handling services, multiplexing resources, and combining programs together for efficiency.

false Isolation

1.4 The purpose of supervisor mode is to isolate certain instructions and routines from user programs.

True

1.5 User programs call into OS routines using system calls.

True

2 Polling & Interrupts

2.1 Fill out this table that compares polling and interrupts.

Operation	Definition	Pro/Good for	Con
Polling			
Interrupts			

3 Memory Mapped I/O

3.1 For this question, the following addresses correspond to registers in some I/O devices and not regular user memory.

- `0xFFFF0000`—Receiver Control: LSB is the ready bit (in the context of polling), there may be other bits set that we don't need right now.
- `0xFFFF0004`—Receiver Data: Received data stored at lowest byte.
- `0xFFFF0008`—Transmitter Control: LSB is the ready bit (in the context of polling), there may be other bit set that we don't need right now.
- `0xFFFF000C`—Transmitter Data: Transmitted data stored at lowest byte.

Recall that receiver will only have data for us when the corresponding ready bit is 1, and that we can only write data to the transmitter when its ready bit is 1. Write RISC-V code that reads byte from the receiver (busy-waiting if necessary) and writes that byte to the transmitter (busy-waiting if necessary).

4 Forking

One of the many responsibilities of the OS is to load new programs, and in order to do this it creates a new process and loads in the program to execute. In Linux, the system call to create a new process is fork(). fork() creates a new process by duplicating the calling process. The new process is referred to as the child process. The calling process is referred to as the parent process. In the parent process, fork() returns the process ID of the child or -1 if the fork has failed. In the child process, it returns 0.

Use this information to complete the code block below, which creates a child process to change the value of y while the parent process changes the value of x.

```
int x = 10;
int y = 0;
int pid = _________;
if(_________){
    y++
}
else{
    x--;
}
```