Australian National University

Computer Science Courses
ANU College of Engineering & Computer Science

Week 10

Students are expected to attempt ALL of the questions in the Multi choice, Short answer and Lab quesitons. They will be discussed and marked at the beginning of the lab.

Multi choice

Question: What memory address does the `trap` instruction take you to? C	**A)** 0x0000 **B)** 0x0001 **C)** 0x0002 **D)** 0x0003 **E)** 0x0100

Question: How often does the timer interrupts trigger when it is enabled? D	**A)** Every 1000 instructions **B)** Every 1000 seconds **C)** Every 1000 milliseconds **D)** Once after 1000 instructions, then stops

Question: An interrupt storm is when _____ B	**A)** someone does a DOS attach on your system. **B)** interrupts are occurring more frequently than the handler can service them. request lost them **C)** interrupts servicing takes over 50% of the CPU time. **D)** rain has made it onto the mother-board causing hardware interrupts.

Short answer

Question: What is the different between the "trap" instruction and the "call 0x0002" instruction in rPeANUt? interrupts will be executed sequentially

Question: What types of interrupts are there in computer systems? software interrupts, hardware interrupts, processor interrupts

Question: What interrupts are there in rPeANUt? How would you characterize them? memory fault - software, IO device - hardware, trap,

Question: Read the man page for "open", "read", "write", and "ioctl". (use "man open", "man 2 read", "man 2 write" and "man 2 ioctl") How are the parameters used in the "read" and "write" functions.

Pointer is used to read, write file.

time - hard ware

Lab questions

Excercise:

Generally a user application program will execute in a non-privileged mode. This is for robustness and security reasons. A non-privileged mode of execution will generally stop the executing program from directly accessing IO devices. Hence, access to IO is done via the Operating System (OS). The aim of this lab is to write part of an OS for rPeANUt that enables character IO via system calls (using the 'trap' instruction).

1. Write an interrupt handler for the trap instruction that outputs to the terminal the string 'trap'. In the main program call this trap in an infinite loop.
2. Modify the handler for the trap instruction such that it implements the system calls 'read' and 'write'. The 'read' system call will read a single character from the terminal (make it blocking), the 'write' system call will write a single character to the terminal. Use register R0 for telling the handler which system call it is (say if the value 0 is in R0 then it is the 'read' system call, and if the value 1 is in register R0 then it is the 'write' system call). Also in the case of 'write' use register R1 to store the character to be written. In the case of 'read' use register R1 for the place the read character is returned.
3. Write a main program that uses these system calls to echo the typed characters. The pseudo code will look something like:

```
void traphandler ()  {
   if (R0 == 0) { // read system call
      R1 = getchar(); // poll the status register of the IO terminal until a char is available then load it into R1
   } else if (R0 == 1) { // write system call
      putchar(R1); // just store R1 to memory location 0xfff0
   }
   enableInterrupts();
}

void main() {
   char c;
   while (1) {
      c = read();  // set R0 to 0 can issue the 'trap' instruction, result will be in R1
      write(c);  // set R0 to 1 and set R1 to 'c' and issue the 'trap' instruction
   }
}
```

In-class Group Task

This weeks group exercise extends on the individual exercise you should have completed prior to the lab group. Complete this in groups of 2 or 3. You should use one of the group's lab work as a starting point. They (the group member that wrote the code) may need to first go over the approach they have taken for their traphandler.

1. The rPeANUt simulator provides buffering of the characters that are typed into the terminal. However, this would generally not be the case so if the program did not consume a character before the next one is typed then a character may be lost. So in this this part of the lab you are required to write an interrupt handler for the IO terminal that will capture characters as they are typed and place them in a buffer. The read system call would then read them out of this buffer rather than directly from the device.
2. This will be done in a two stages, firstly get a single character buffer working, and secondly get a 10 character buffer working. The psudo code for interrupt handler with a single character buffer:

```
void iohandler () {
   if (buffcount == 0) {
      buff = getchar();  // no need to poll on the status register
      buffcount = 1;
   } else {
      getchar(); // no room in the buffer so we lost a character!!!
   }
   enableInterrupts();
}
void traphandler ()  {
   enableInterruptes();
   if (R0 == 0) { // read system call
      while (buffcount == 0) ;
      R1 = buff;
      buffcount = 0;
   } else if (R0 == 1) { // write system call
      putchar(R1); // just store R1 to memory location 0xfff0
   }
}
char buff;
int buffcount;

void main() {
   buffcount = 0;
   enableIOterminalInterrupts();  // involves writing to the control register of the IO terminal device.
```

```
   char c;
   while (1) {
      c = read();
      write(c);
   }
}
```

3. The psudo code for interrupt handler with a 10 character buffer:

```
void iohandler () {
   if ((buffend + 1) % 10 != buffstart) {
      buff[buffend] = getchar();  // no need to poll on the status register
      buffend = (buffend +1) % 10 ;
   } else {
      getchar(); // no room in the buffer so we lost a character!!!
   }
   enableInterupts();
}
void traphandler ()  {
   enableInteruptes();
   if (R0 == 0) { // read system call
      while (buffstart == buffend) ;
      R1 = buff[buffstart];
      buffstart = (buffstart+1) % 10;
   } else if (R0 == 1) { // write system call
      putchar(R1); // just store R1 to memory location 0xfff0
   }
}
char buff[10];
int buffstart;  // the spot to read the next char;
int buffend;    // the spot to write the next char;

void main() {
   buffstart = 0;
   buffend = 0;
   enableIOterminalInterrupts();  // involves writing to the control register of the IO terminal device.
```

```
    char c;
    while (1) {
        c = read();
        write(c);
    }
}
```

Remember your interrupt handlers must leave the registers as they found them. So at the beginning of the handler pushes registers that are used onto the stack and then pops them back into place before the handler returns.

UPDATED: **22 April 2013** / RESPONSIBLE OFFICER: **Head of School** / PAGE CONTACT: **COMP2300 Course Webmaster**