《操作系统》实验报告

实验题目	异常与中断

一、实验目的

1. 了解操作系统开发实验环境(Docker)；
2. 熟悉命令行方式的编译、调试工程；
3. 掌握基于硬件模拟器的调试技术；
4. 了解 LoongArch32 中有哪些异常以及例外处理操作。

二、实验项目内容

1. 结合 LoongArch32 的文档，列出 LoongArch32 有哪些例外？以及这些例外有哪两个例外向量入口？

2.请编程完善 kern/driver/clock.c 中的时钟中断处理函数 clock_int_handler，在对时钟中断进行处理的部分填写 trap 函数中处理时钟中断的部分，使操作系统每遇到 100 次时钟中断后，调用 kprintf，向屏幕上打印一行文字”100 ticks”。

3. 请编程完善 kern/driver/console.c 中的串口中断处理函数 serial_int_handler，在接收到一个字符后读取该字符，并调用 kprintf 输出该字符。

三、实验过程或算法（源程序）

1. 获取实验资料包：

```
root@DESKTOP-ELVOHAA:/mnt/c/Users/Jackson1125/lab1# git clone https://github.com/cyyself/ucore-loongarch32.git -b cqu2022_noanswer
Cloning into 'ucore-loongarch32'...
remote: Enumerating objects: 1089, done.
remote: Counting objects: 100% (202/202), done.
remote: Compressing objects: 100% (129/129), done.
remote: Total 1089 (delta 113), reused 133 (delta 73), pack-reused 887
Receiving objects: 100% (1089/1089), 1.26 MiB | 444.00 KiB/s, done.
Resolving deltas: 100% (642/642), done.
```

资料包在终端所在文件夹下，执行前需要 cd 进入；

2. 修改 Makefile 文档：

```
LAB1    := -DLAB1_EX2 -DLAB1_EX3 -D_SHOW_100_TICKS -D_SHOW_SERIAL_INPUT
# LAB2  := -DLAB2_EX1 -DLAB2_EX2 -DLAB2_EX3
# LAB3  := -DLAB3_EX1 -DLAB3_EX2
# LAB4  := -DLAB4_EX1 -DLAB4_EX2
```

需要加上 D_SHOW_SERIAL_INPUT 选项，以实现在终端打印键盘的输入；

3. 完成练习 2 代码，完善 kern/driver/clock.c 中的时钟中断处理函数 clock_int_handler，在对时钟中断进行处理的部分填写 trap 函数中处理时钟中断的部分，实现在终端每 10 毫秒打印一行″100 ticks″：

```c
int clock_int_handler(void * data)
{
#ifdef LAB1_EX2
  // LAB1 EXERCISE2: YOUR CODE
  // (1) count ticks here
  ticks ++;
#ifdef _SHOW_100_TICKS
  // (2) if ticks % 100 == 0 then call kprintf to print "100 ticks"
  if (ticks % 100 == 0) {
    kprintf("100 ticks\n");
  }
#endif
#endif
```

其中通过 ticks++计数，对时钟中断进行处理，使得操作系统没 100 次时钟中断后，调用 kprintf，向屏幕打印“100ticks”。代码执行过程如下，可以看到，终端不断打印“100ticks”：

```
root@DESKTOP-ELVOHAA:/mnt/c/Users/Jackson1125# cd lab1/ucore-loongarch32
root@DESKTOP-ELVOHAA:/mnt/c/Users/Jackson1125/lab1/ucore-loongarch32# make clean
rm -rf dep
rm -rf boot/loader.o boot/loader boot/loader.bin
rm -rf obj
root@DESKTOP-ELVOHAA:/mnt/c/Users/Jackson1125/lab1/ucore-loongarch32# make -j 16
DEP kern/fs/devs/dev_stdout.c
```

```
root@DESKTOP-ELVOHAA:/mnt/c/Users/Jackson1125/lab1/ucore-loongarch32# make qemu
loongson32_init: num_nodes 1
loongson32_init: node 0 mem 0x2000000
++setup timer interrupts
(THU.CST) os is loading ...

Special kernel symbols:
  entry  0xA0000120 (phys)
  etext 0xA0021000 (phys)
  edata 0xA017A570 (phys)
  end   0xA017D850 (phys)
Kernel executable memory footprint: 1395KB
LAB1 Check - Please press your keyboard manually and see what happend.
100 ticks
100 ticks
```

4. 完成练习 3 代码，完善 kern/driver/console.c 中的串口中断处理函数 serial_int_handler，在接收到一个字符后读取该字符，并调用 kprintf 输出该字符：

```c
void serial_int_handler(void *opaque)
{
    unsigned char id = inb(COM1+COM_IIR);
    if(id & 0x01)
        return ;
    //int c = serial_proc_data();
    int c = cons_getc();
#if defined(LAB1_EX3) && defined(_SHOW_SERIAL_INPUT)
    // LAB1 EXERCISE3: YOUR CODE
    kprintf("got input %c\n",c);
#endif
```

可以看到，在检测到输入后操作系统会中断打印“100ticks”的进程，读取输入字符并

调用 kprintf 输出该字符。代码执行过程如下，可以看到，终端在打印“100ticks”的间隙输出一些字符，这些字符就是实验过程中键盘键入被输出到终端的：

```
root@DESKTOP-ELVOHAA:/mnt/c/Users/Jackson1125# cd lab1/ucore-loongarch32
root@DESKTOP-ELVOHAA:/mnt/c/Users/Jackson1125/lab1/ucore-loongarch32# make clean
rm -rf dep
rm -rf boot/loader.o boot/loader boot/loader.bin
rm -rf obj
root@DESKTOP-ELVOHAA:/mnt/c/Users/Jackson1125/lab1/ucore-loongarch32# make -j 16
DEP kern/fs/devs/dev_stdout.c
```

```
root@DESKTOP-ELVOHAA:/mnt/c/Users/Jackson1125/lab1/ucore-loongarch32# make qemu
loongson32_init: num_nodes 1
loongson32_init: node 0 mem 0x2000000
++setup timer interrupts
(THU.CST) os is loading ...

Special kernel symbols:
  entry  0xA0000120 (phys)
  etext 0xA0021000 (phys)
  edata 0xA017A580 (phys)
  end   0xA017D860 (phys)
Kernel executable memory footprint: 1395KB
LAB1 Check - Please press your keyboard manually and see what happend.
100 ticks
100 ticks
got input f
100 ticks
```

四、实验结果及分析

1. 实验结果与分析

Q1：结合 LoongArch32 的文档，列出 LoongArch32 有哪些例外？以及这些例外有哪两个例外向量入口？

LoongArch32 有如下例外：

- 中断：当出现需要时，CPU 暂时停止当前程序的执行转而执行处理新情况的程序和执行过程，如本实验在打印 100ticks 过程中中断进程打印输入字符的情况，是最常见的一种异常；
- load 操作页无效：取操作页出现无效情况；
- store 操作页无效：存操作页出现无效情况；
- 取指操作页无效：取指令的操作页发生错误；
- 取指地址错：取指令的地址发生错误，如地址没有对齐；
- 访存指令地址错：访问或存入内存的地址出现错误；
- 系统调用：用户在程序中使用访存指令时调用由操作系统提供的子功能；
- 指令不存在：即所取指令不存在
- 指令特权等级错：指令级别出错；
- TLB 重填：快表需要重写。

这些例外中，**TLB 重填例外**的向量入口为 **CSR.TLBRENTRY**；除此之外所有**普通例外**的向量入口都为 **CSR.EENTRY**。

Q2：编程完善 kern/driver/clock.c 中的时钟中断处理函数 clock_int_handler，在对时钟中断进行处理的部分填写 trap 函数中处理时钟中断的部分，使操作系统每遇到 100 次时钟中断后，调用 kprintf，向屏幕上打印一行文字”100 ticks”。

中断处理函数如下：

```
int clock_int_handler(void * data)
{
#ifdef LAB1_EX2
  // LAB1 EXERCISE2: YOUR CODE
  // (1) count ticks here
  ticks ++;
#ifdef _SHOW_100_TICKS
  // (2) if ticks % 100 == 0 then call kprintf to print "100 ticks"
  if (ticks % 100 == 0) {
    kprintf("100 ticks\n");
  }
#endif
#endif
#ifdef LAB4_EX1
  run_timer_list();
#endif
  reload_timer();
  return 0;
}
```

该函数作用为处理时钟中断，每 1ms 变量 ticks 加一。当 ticks 可以被 100 整除时打印“100ticks”，由此使操作系统每遇到 100 次时钟中断就调用 kprintf 并向屏幕打印“100ticks”。

编译运行后控制台输出如下，可以看到，随着时间控制台不断输出“100ticks”：

```
LAB1 Check - Please press your keyboard manually and see what happend.
100 ticks
```

```
LAB1 Check - Please press your keyboard manually and see what happend.
100 ticks
100 ticks
100 ticks
100 ticks
100 ticks
100 ticks
100 ticks
100 ticks
100 ticks
100 ticks
```

Q3：请编程完善 kern/driver/console.c 中的串口中断处理函数 serial_int_handler，在接收到一个字符后读取该字符，并调用 kprintf 输出该字符。

中断处理函数如下：

```
void serial_int_handler(void *opaque)
{
    unsigned char id = inb(COM1+COM_IIR);
    if(id & 0x01)
        return ;
```

```
    //int c = serial_proc_data();
    int c = cons_getc();
#if defined(LAB1_EX3) && defined(_SHOW_SERIAL_INPUT)
    // LAB1 EXERCISE3: YOUR CODE
    kprintf("got input %c\n",c);
#endif
#ifdef LAB4_EX2
    extern void dev_stdin_write(char c);
    dev_stdin_write(c);
#endif
}
```

该函数作用为处理串口中断，每接收到字符后读取该字符，中断正在打印的“100ticks”并输出该字符，然后将处理器再交由原先进程。

编译运行后控制台输出如下，可以看到，随着时间控制台不断输出“100ticks”，并且中间夹杂着键盘输入的字符：

```
100 ticks
got input d
got input s
got input g
got input v
got input a
got input e
got input f
100 ticks
got input i
got input h
got input n
got input e
got input j
got input p
100 ticks
100 ticks
```

2. 调试过程：

（1）修改完代码编译出错，检查发现忘记清空编译产物：

```
make clean # 清空编译产物
make -j 16 # 编译
make qemu # 运行 qemu
```

（2）编译执行后控制台没有预期输出，检查发现没有添加 D_SHOW_SERIAL_INPUT 选项：

```
LAB1    := -DLAB1_EX2 -DLAB1_EX3 #-D_SHOW_100_TICKS -D_SHOW_SERIAL_INPUT
# LAB2  := -DLAB2_EX1 -DLAB2_EX2 -DLAB2_EX3
# LAB3  := -DLAB3_EX1 -DLAB3_EX2
# LAB4  := -DLAB4_EX1 -DLAB4_EX2
```

3. 实验感悟：

操作系统本质上是一个软件，也需要通过某种机制加载并运行它。对于 LoongArch32 架构的计算机来说，上电复位最初启动的是一个 BIOS 软件（如 PMON），该 BIOS 软件能够支持从网络加载 ELF 格式的操作系统内核，从而开始启动我们已经编译好的 uCore 内核。

而对于 QEMU 虚拟机而言，我们可以直接使用 kernel 内核来指定我们需要加载的内核的 ELF 文件，从而直接完成了内核的载入过程，并直接从 ELF 的入口点开始启动。

操作系统还需要对计算机系统中的各种外设进行管理，因此需要 CPU 和外设能够相互通信。所以需要操作系统和 CPU 能够一起提供某种机制，让外设在需要操作系统处理外设相关事件的时候，能够**打断操作系统和应用的正常执行**，让操作系统**完成外设的相关处理，然后再恢复操作系统和应用的正常执行**。也就是操作系统中的**中断机制**，本实验代码部分的主要内容。

中断机制给操作系统提供了**处理意外情况**的能力，同时也是实现进程或线程抢占式调度的一个重要基石。

通过这个实验，我们了解到了操作系统中异常与中断的类型、处理过程、返回过程以及他们的关系，并亲手实践了操作系统中的串口中断机制。