Project 3: System Call in pi-OS

ENEE 447: Operating Systems — Spring 2017
Assigned Date: Saturday, March 11.
Due Date: **Sunday, March 26th. @11:59 p.m.**

1.Purpose

In project 2, we learned about hardware interrupt by using timer through IRQ request. In project 3, we will learn about software interrupt. In particular, we will look at the concept of software interrupt, differences between hardware interrupt(IRQ) and software interrupt (SWI), the implementation of system call. We'll implement three system calls through software interrupt: 1. printf(), 2. sleep(), and 3. exit().

2.Introduction

2.1 Introduction

Image I have a pseudo program:

```
int main() {
        File f = fopen("in.txt")
        read(f) // read the file
}
```

The program above is trying to read a file called "in.txt" which is stored on the hard-disk. It is currently running under user mode. However, the action accessing file on a hard disk is a privileged operation. Privileged operation should run under kernel mode. But our user program only runs under user mode. So, what we want here is to give user certain capability to execute privileged operation. At the same time, we don't want to give complete control over to user. Here is where **system call** comes in. When we want to do some high privileged job, we request the kernel to do so. Due to separation of tasks, each task could not see any other task in the system. The only way to ask operating system to do something for user task is through SYSTEM CALL. This is because system provides some interface between user task and kernel, and this mechanism is based on software interrupt---when user task makes a system call, CPU will go into some routine, similar to IRQ. We can put kernel function in this routine, serving request from user task.

2.2 System call

System call is a special type of function, which allows kernel to carefully expose some key pieces of functionality to user program. What that means is user program is allowed to execute privileged function via calling system call. You can also think of system call as an API for some privilege functionality.

The user program runs under user mode, and user program is allowed to request privilege operation via calling system call. But, privilege operation, such as reading a file from hard disk actually runs under kernel mode. So there is some MODE switch here, (user task)USER MODE → SUPERVISOR MODE, then do privileged operation → (user task)USER MODE.

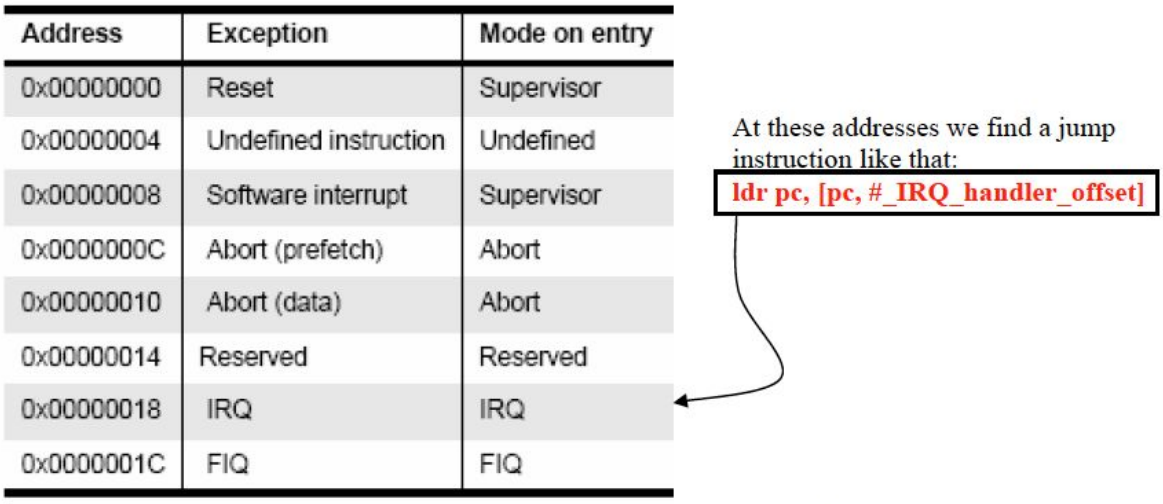
2.3 Software Interrupt

How to trigger a software interrupt for a specific system call:

SWI #1

This makes a system call for exit, since exit number is 1.

Once a software interrupt is generated, hardware (CPU) will save CPSR into SPSR_svc and set PC to SVC entry in **interrupt vector table**. That is CPU will jump to 0x00000008. So we need to put a branch instruction at 0x00000008, to make CPU further jump to our first level handler. There we can do more handling.

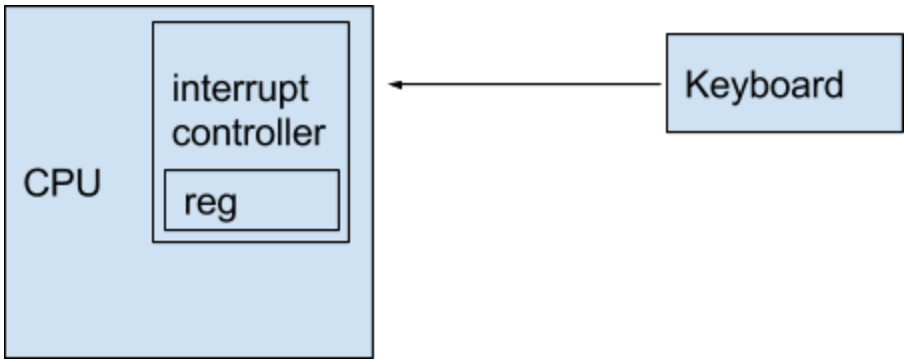


Address	Exception	Mode on entry
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	Reserved	Reserved
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

Figure 2 An exact vector table with the branching instruction[5]

2.4 Hardware Interrupt

This section contains some side notes about hardware interrupt. We just learned that software interrupt is generated by executing an assembly instruction with corresponding system call number, hardware interrupt is generated in a more straight forward manner.



When a key is pressed on a keyboard, it will generate a signal which will propagate through a physical wire and signal is directly sent to CPU. Inside your CPU, there is a physical component called interrupt controller, which is responsible for handling interrupt requests, such as the external peripheral keyboard in this case. The signal generated by the keyboard flip the corresponding bit of the register inside the interrupt handler. Once the corresponding bit change is detected by the interrupt controller, corresponding hard interrupt is generated.

2.5 Software Interrupt v.s. Hardware Interrupt

So far, we learned about two types of interrupts, software interrupt (SWI) and Hardware Interrupt (IRQ). Let's summarize the difference between these two types of interrupts:

1. Hardware interrupt is triggered outside CPU (interrupt controller), different types of hardware can generate different types of interrupt and usually they are more complex. While software interrupt **is triggered inside the** CPU.
2. Hardware interrupt is serving on behalf of **hardware**, while software interrupt is generated on behalf of the **application**. Software interrupt source is codes.

It is straightforward to understand the part for hardware interrupt. For software interrupt, let's go back to the pseudo program presented in the first page. A user program tries to

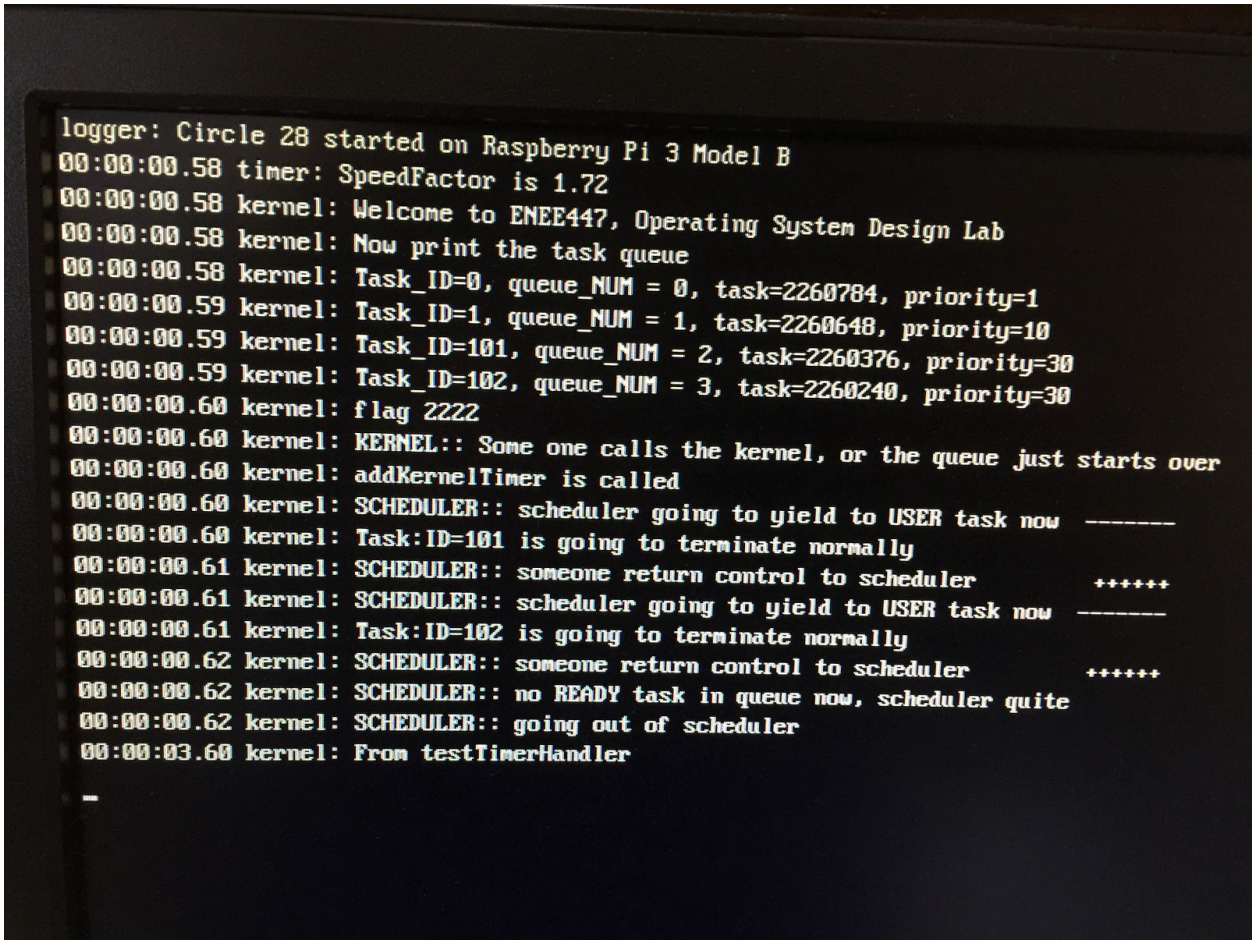
read a file from hard disk, it execute a system call read() to perform the operation. To completed the require operation (read a file from disk), a software interrupt is generated through executing an assembly instruction with corresponding system call number. Then CPU goes to SUPERVISOR mode, running a sequence of handling procedures. After handling, CPU can go back to where just below SVC instruction.

3. Your mission

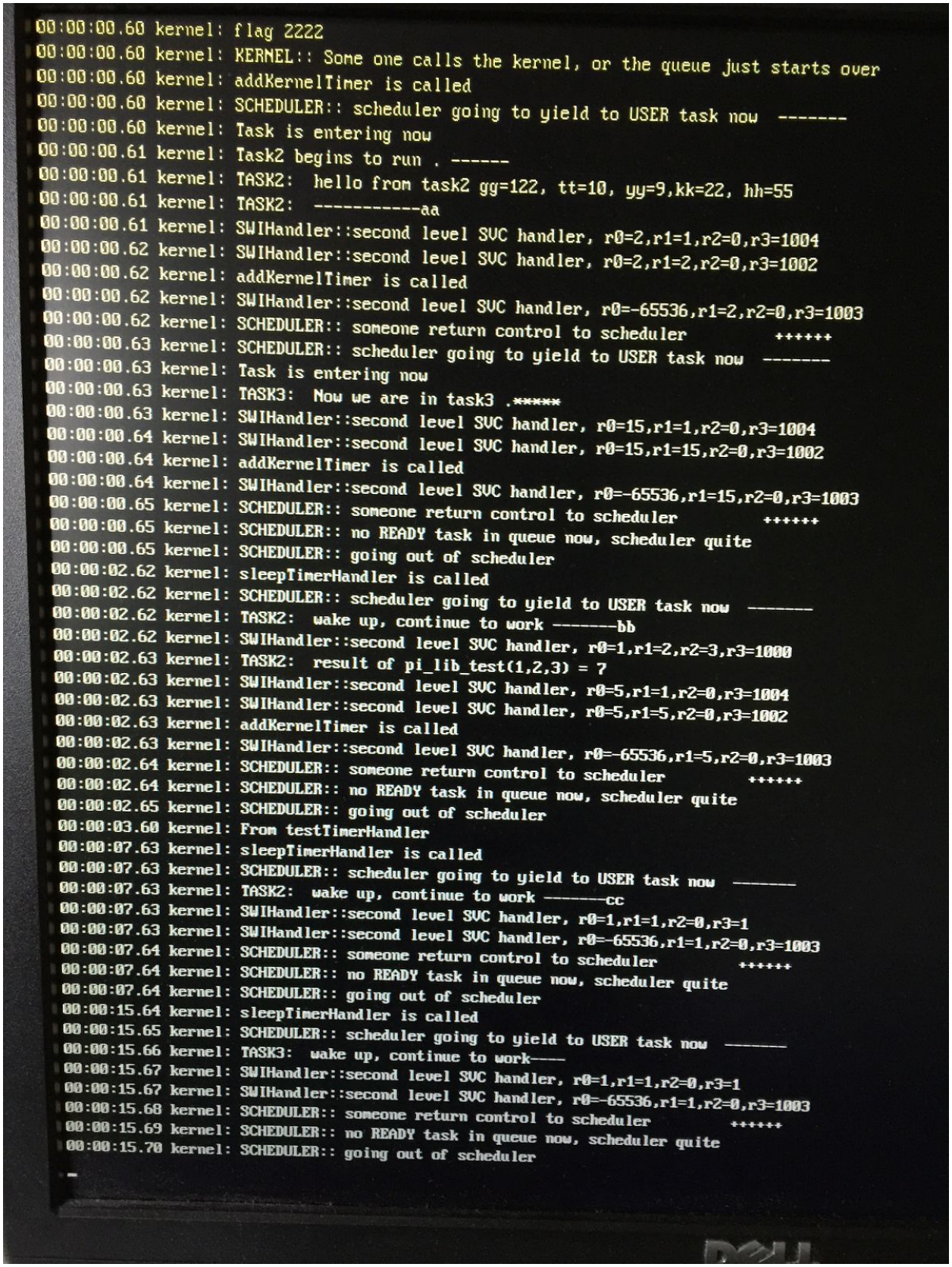
The initial project already defines the behavior of task2 and task 3:

```
61 void task2_run()
62 {
63     printf("Task2 begins to run . ------");
64     int gg = 122;
65     int tt=10;
66     int yy =9;
67     int kk=22;
68     int hh=55;
69     printf("TASK2:  hello from task2 gg=%d, tt=%d, yy=%d,kk=%d, hh=%d", gg, tt, yy,kk,hh);
70     printf("TASK2:  -----------aa");
71     sleep(2);
72     printf("TASK2:  wake up, continue to work -------bb");
73     int result=pi_lib_test(1,2,3);
74     printf("TASK2:  result of pi_lib_test(1,2,3) = %d", result);
75     sleep(5);
76     printf("TASK2:  wake up, continue to work -------cc");
77     exit(1);
78     printf("TASK2:  this message should not appear");
79 }
80
81 void task3_run()
82 {
83     printf("TASK3:  Now we are in task3 .*****");
84     sleep(15);
85     printf("TASK3:  wake up, continue to work---- ");
86     exit(1);
87     printf("TASK3:  this message should not appear");
88 }
89
```

The initial project screen look like :



You need to implement three system calls, following designing document provided to you. Your user task would like to use printf, sleep, exit through software interrupt. They should not be accessed directly through pSystemCall structure like in project 2. **You need to implement system call through ARM SVC(SuperVisor Call).** Make sure your final submission look like this:


00:00:00.60 kernel: flag 2222
00:00:00.60 kernel: KERNEL:: Some one calls the kernel, or the queue just starts over
00:00:00.60 kernel: addKernelTimer is called
00:00:00.60 kernel: SCHEDULER:: scheduler going to yield to USER task now -------
00:00:00.60 kernel: Task is entering now
00:00:00.61 kernel: Task2 begins to run . ------
00:00:00.61 kernel: TASK2: hello from task2 gg=122, tt=10, yy=9,kk=22, hh=55
00:00:00.61 kernel: TASK2: ----------aa
00:00:00.61 kernel: SWIHandler::second level SVC handler, r0=2,r1=1,r2=0,r3=1004
00:00:00.62 kernel: SWIHandler::second level SVC handler, r0=2,r1=2,r2=0,r3=1002
00:00:00.62 kernel: addKernelTimer is called
00:00:00.62 kernel: SWIHandler::second level SVC handler, r0=-65536,r1=2,r2=0,r3=1003
00:00:00.62 kernel: SCHEDULER:: someone return control to scheduler ++++++
00:00:00.63 kernel: SCHEDULER:: scheduler going to yield to USER task now -------
00:00:00.63 kernel: Task is entering now
00:00:00.63 kernel: TASK3: Now we are in task3 .*****
00:00:00.63 kernel: SWIHandler::second level SVC handler, r0=15,r1=1,r2=0,r3=1004
00:00:00.64 kernel: SWIHandler::second level SVC handler, r0=15,r1=15,r2=0,r3=1002
00:00:00.64 kernel: addKernelTimer is called
00:00:00.64 kernel: SWIHandler::second level SVC handler, r0=-65536,r1=15,r2=0,r3=1003
00:00:00.65 kernel: SCHEDULER:: someone return control to scheduler ++++++
00:00:00.65 kernel: SCHEDULER:: no READY task in queue now, scheduler quite
00:00:00.65 kernel: SCHEDULER:: going out of scheduler
00:00:02.62 kernel: sleepTimerHandler is called
00:00:02.62 kernel: SCHEDULER:: scheduler going to yield to USER task now -------
00:00:02.62 kernel: TASK2: wake up, continue to work -------bb
00:00:02.62 kernel: SWIHandler::second level SVC handler, r0=1,r1=2,r2=3,r3=1000
00:00:02.63 kernel: TASK2: result of pi_lib_test(1,2,3) = 7
00:00:02.63 kernel: SWIHandler::second level SVC handler, r0=5,r1=1,r2=0,r3=1004
00:00:02.63 kernel: SWIHandler::second level SVC handler, r0=5,r1=5,r2=0,r3=1002
00:00:02.63 kernel: addKernelTimer is called
00:00:02.63 kernel: SWIHandler::second level SVC handler, r0=-65536,r1=5,r2=0,r3=1003
00:00:02.64 kernel: SCHEDULER:: someone return control to scheduler ++++++
00:00:02.64 kernel: SCHEDULER:: no READY task in queue now, scheduler quite
00:00:02.65 kernel: SCHEDULER:: going out of scheduler
00:00:03.60 kernel: From testTimerHandler
00:00:07.63 kernel: sleepTimerHandler is called
00:00:07.63 kernel: SCHEDULER:: scheduler going to yield to USER task now -------
00:00:07.63 kernel: TASK2: wake up, continue to work -------cc
00:00:07.63 kernel: SWIHandler::second level SVC handler, r0=1,r1=1,r2=0,r3=1
00:00:07.63 kernel: SWIHandler::second level SVC handler, r0=-65536,r1=1,r2=0,r3=1003
00:00:07.64 kernel: SCHEDULER:: someone return control to scheduler ++++++
00:00:07.64 kernel: SCHEDULER:: no READY task in queue now, scheduler quite
00:00:07.64 kernel: SCHEDULER:: going out of scheduler
00:00:15.64 kernel: sleepTimerHandler is called
00:00:15.65 kernel: SCHEDULER:: scheduler going to yield to USER task now -------
00:00:15.66 kernel: TASK3: wake up, continue to work----
00:00:15.67 kernel: SWIHandler::second level SVC handler, r0=1,r1=1,r2=0,r3=1
00:00:15.67 kernel: SWIHandler::second level SVC handler, r0=-65536,r1=1,r2=0,r3=1003
00:00:15.68 kernel: SCHEDULER:: someone return control to scheduler ++++++
00:00:15.69 kernel: SCHEDULER:: no READY task in queue now, scheduler quite
00:00:15.70 kernel: SCHEDULER:: going out of scheduler

4. Implementation guide:

Please read ***System Call Design and Implementation.pdf*** thoroughly. You need to look through a lot of references. You need to be familiar with project 2 Docs and codes. Analysis existing codes in project 3. Follow the steps carefully.

You need to modify at least 7 files to accomplish project 3:

5. Submission

Plan your time accordingly, **no extension**, any late submission (after **Sunday, March 26th 11:59pm**) will result in penalty. If you submit one day late, 10% will be deducted from your project score. If you submit two days late, 20% will be deducted, etc.

What is inside your zip submission:

- Design/Documentation Report in PDF format. (include **screenshot** for working system call implementation with task 2 & task 3)
- All Project files, make sure it is compliable, any compile error automatically 10% off.

Demo:

- Make sure your code is working before submission, TA will download your code and compile there, if your code did **not compile**, and result of **demo not matching** our output. at least **10% project will be deducted**.
- Any reschedule for demo should be completed within the **same week** of demo.
- You need to answer questions during the demonstration. You can find example questions for project 3 on Piazza before demonstration. Failure of the demonstration would get you 15% deduction of score, and you would need to show again...

7. Honor Code

Never show your code to your classmates. Copying code and idea is considering cheating and will be reported to Student Honor Council.

The University of Maryland, College Park has a nationally recognized Code of Academic Integrity. This Code sets standards for academic integrity at Maryland for all undergraduate and graduate students. As a student you are responsible for upholding these standards for this course. It is very important for you to be aware of the consequences of cheating, fabrication, facilitation, and plagiarism. For more information on the Code of Academic Integrity or the Student Honor Council, please visit http://www.shc.umd.edu