北京航空航天大學
BEIHANG UNIVERSITY

操作系统 Lab1 - 2 课上测试

Lab1-2测试说明

测试说明

- **考试时间 14:00 ~ 16：00**
- 每次课上测试题目分为**基础测试**和**附加测试**(选做)两部分
- 每题单独评分，满分都是100分
- **请注意，Lab得分为：Lab基础分值 * (课下成绩 * 0.6 + 课上exam成绩 * 0.4) / 100。**
- **附加测试加分为：通过（>=60分）课上测试Extra题目**所给予的加分。
- 可以登录课程MOOC网站https://os.buaa.edu.cn下载指导书进行参考。

Lab1-2课上基础题

Step1:创建lab1-2-exam分支

- cd ~/学号/
- git checkout lab1
- git add .
- git commit -m "xxxxx"(填写修改或可提示自己的信息)
- git checkout -b lab1-2-exam (有参数-b)

Step2:完成lab1-2基础题代码编写

Step3:提交更改

- cd ~/学号/
- git add .
- git commit -m "xxxxx"(填写修改或可提示自己的信息)
- git push origin lab1-2-exam:lab1-2-exam

Step4:提交结果

```
remote: End build at Tue Mar 15 19:42:37 CST 2022
remote: [ PASSED:22 ]
remote: [ TOTAL:22 ]
remote: [ You have passed all testcases of exercise printf. ]
```

此为课下强测部分；

```
remote: [ PASSED:34 ]
remote: [ TOTAL:34 ]
remote: [ You have passed all testcases of exam printf. ]
remote: [ You got 100 (of 100) this time. Tue Mar 15 19:42:59 CST 2022 ]
```

正确完成实验要求后提交代码，可以看到如上图所示评测结果，得到100分。

题目背景

在C语言中，可以使用结构体（Struct）来存放一组不同类型的数据。如右图所示。

当我们定义了一个struct结构时，只是指定了一个结构体类型，系统对之不分配实际的内存单元。只有在定义结构体后定义相应类型的变量，才会给这个变量分配内存空间，存储具体的数据。

```
struct SIMPLE
{
    int a;
    char b;
    double c;
};
```

Lab1-2-exam测试题目

定义struct my_struct结构为：

```
struct my_struct {
    int size;
    char c;
    int array[SIZE_C];
};
```

提示：

- 保证测试时的结构与此结构一定相同；
- 我们定义SIZE_C为某一大于零的常量，且保证测试时构造my_struct结构变量时设置的size的值一定与SIZE_C相同；
- 在不同的测试用例中， SIZE_C常量的值可能不同。

课上测试基础题

Lab1-2-exam测试题目

printf 的格式串原型为：%[*flags*][*width*][.*precision*][*length*]*specifier*
现要求在Specifier类型中新增一个如下：

Specifier	输出	例子
T	结构体的各个域，用大括号括起来，逗号分隔。数组按下标顺序列出。	{2,a,0,1}

请修改 print.c 中的 lp_Print() 函数相应部分，以实现 printf 函数对新增Specifier的支持。

注意：printf原功能需要继续保留，flags、width等副格式符(sub-specifier)的效果作用在结构体每个域的输出上，格式串为%T时，printf传入参数为结构体变量的首地址。

课上测试基础题

Lab1-2-exam测试题目

参考样例： 对于以下两个变量

```
struct my_struct1 t1;
struct my_struct2 t2;
t1.size=3; t1.c='b'; t1.array[0]=0; t1.array[1]=1; t1.array[2]=2;
t2.size=2; t2.c='Q'; t2.array[0]=-1; t2.array[1]=-2;
```

我们实现的printf要支持的功能类似于：

printf("%T",&t1); 结果为：{3,b,0,1,2}

printf("%T",&t2); 结果为：{2,Q,-1,-2}

printf("%04T",&t1); 结果为：{0003, b,0000,0001,0002} (注意b前有3个空格)

printf("%04T",&t2); 结果为：{0002, Q,-001,-002} (注意Q前有3个空格)

强调1：传入printf的参数为结构体变量的地址（例如：&t1），而不是变量本身(t1)。

强调2：请自行编写符合题目要求的结构体定义进行本地测试，不要直接复制粘贴本页代码！

课上测试基础题

Lab1-2-exam测试题目

本地测试方法：

在本地的init/main.c内定义题目所述的结构体（SIZE_C必须为确定的常量），在main函数内创建这类结构体的实例，给结构体的size字段正确赋值，之后给其它字段合理赋值后，使用printf()函数输出你的结构体内容。

建议多测试%T与不同副格式符之间的搭配情况。

评测方法：

评测机会先进行课下 printf 内容的强测。若测试通过，则会提示“You have passed all testcases of exercise printf.”课下强测占20分。
然后会进行课上内容的测试。测试通过会提示“You have passed all testcases of exam printf.”课上测试占80分。

Lab1-2课上附加题

Step5:创建附加题分支（选做）

- git checkout lab1 (回到lab1分支下)
- git add .
- git commit -m "xxxxx"(填写修改或可提示自己的信息)
- git checkout -b lab1-2-Extra (有参数-b)

Step6:完成lab1-2附加题代码编写（选做）

Step7:提交更改（选做）

- cd ~/学号/
- git add .
- git commit -m "xxxxx"(填写修改或可提示自己的信息)
- git push origin lab1-2-Extra:lab1-2-Extra

Step8:提交结果（选做）

```
remote: [ PASSED:2 ]
remote: [ TOTAL:2 ]
remote: [ You have passed testcase 1/3. ]
```

```
remote: [ PASSED:2 ]
remote: [ TOTAL:2 ]
remote: [ You have passed testcase 2/3. ]
```

```
remote: [ PASSED:2 ]
remote: [ TOTAL:2 ]
remote: [ You have passed testcase 3/3. ]
remote: [ You got 100 (of 100) this time. Mon Mar 28 23:38:44 CST 2022 ]
```

正确完成实验要求后提交代码，可以看到如上图所示评测结果，得到100分。

此时lab1-2-Extra测试通过（大于等于60即为通过），可以获得lab1-2课上测试的额外加分。

测试说明

题目背景

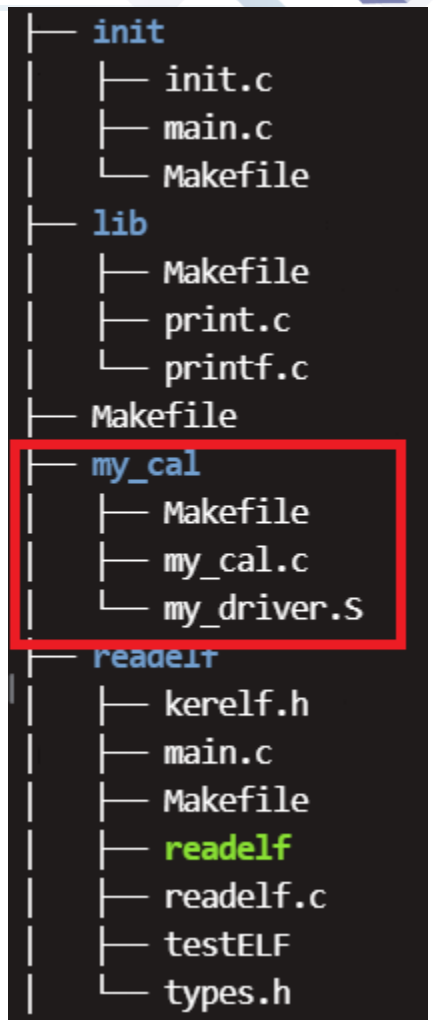
计算机CPU通过读写特定物理地址的寄存器实现与外部设备的沟通。
已知外部设备的物理地址起始地址为0x10000000，偏移信息如下：

偏移量	功能
0x00	控制台（键盘）输入字符，读取该物理地址可以获得该字符ASCII值，无输入则获得0 向此物理地址写入字符ASCII值，控制台（屏幕）会输出此字符
0x10	对此物理地址进行任何读写，GXemul会中断仿真执行并退出

测试说明

Lab1-2-Extra测试题目

请在项目根目录下，创建 my_cal 文件夹，并在里面创建 Makefile ，my_driver.S ， my_cal.c 这三个文件，如右图所示。



步骤1：

编写该Makefile文件，使得在当前目录（【./my_cal/】）下执行make可以编译出【my_driver.o】，【my_cal.o】这两个文件。

提示：可以参考boot，init这两个文件夹下的Makefile。

Lab1-2-Extra测试题目

步骤2：

编写刚才创建的【./my_cal/my_driver.S】这个文件，在内部用汇编实现函数【char _my_getchar()】，【void _my_putchar(char ch)】,【void _my_exit()】。

函数功能分别是：

- 【_my_getchar()】：读取控制台的输入的一个字符，并返回读到的字符。
  同时要求：除了能返回读到的字符之外，还必须能够在控制台上回显。也就是说，要可以在控制台上立刻看见输入的字符（包括换行）。
- 【_my_putchar(char ch)】：将字符输出到控制台，即在屏幕上可见输出。
- 【_my_exit()】退出操作系统和GXemul。

测试说明

Lab1-2-Extra测试题目

提示：

- 题目中给出的映射地址，为【物理】地址。在指导书【1.3.3 MIPS 内存布局——寻找内核的正确位置】这一个章节中，提到了如何访问指定外设物理地址的方法（kseg1）；
- 函数的创建请参考boot目录下的start.S，注意包含必要的头文件；
- 在没有输入时，控制台对应寄存器的值为0。因此，需要解决因不及时输入导致读取到0的问题（循环直到读出非0值）；
- 注意每次读取的字节数（lb指令）；
- 由于换行机制的问题，在_my_getchar()读取到‘\r’并回显时，需要额外输出‘\n’以防止”覆盖”之前的内容（此条是为了方便本地测试，实现与否均不影响评测机的评测）；
- 注意mips汇编中，返回值和参数所使用的寄存器，请务必遵守寄存器规范。

测试说明

请注意本地测试时'\n'的输入方式！！！

换行符输入方式：
直接敲回车 或
ctrl + j

ASCII表

（ American Standard Code for Information Interchange 美国标准信息交换代码 ）

高四位 / 低四位		ASCII控制字符												ASCII打印字符											
		0000						0001						0010		0011		0100		0101		0110		0111		
		0						1						2		3		4		5		6		7		
		十进制	字符	Ctrl	代码	转义字符	字符解释	十进制	字符	Ctrl	代码	转义字符	字符解释	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	Ctrl
0000	0	0		^@	NUL	\0	空字符	16	▶	^P	DLE		数据链路转义	32		48	0	64	@	80	P	96	`	112	p	
0001	1	1	☺	^A	SOH		标题开始	17	◀	^Q	DC1		设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q	
0010	2	2	☻	^B	STX		正文开始	18	↕	^R	DC2		设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r	
0011	3	3	♥	^C	ETX		正文结束	19	‼	^S	DC3		设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s	
0100	4	4	♦	^D	EOT		传输结束	20	¶	^T	DC4		设备控制 4	36	$	52	4	68	D	84	T	100	d	116	t	
0101	5	5	♣	^E	ENQ		查询	21	§	^U	NAK		否定应答	37	%	53	5	69	E	85	U	101	e	117	u	
0110	6	6	♠	^F	ACK		肯定应答	22	▬	^V	SYN		同步空闲	38	&	54	6	70	F	86	V	102	f	118	v	
0111	7	7	•	^G	BEL	\a	响铃	23	↨	^W	ETB		传输块结束	39	'	55	7	71	G	87	W	103	g	119	w	
1000	8	8	◘	^H	BS	\b	退格	24	↑	^X	CAN		取消	40	(	56	8	72	H	88	X	104	h	120	x	
1001	9	9	○	^I	HT	\t	横向制表	25	↓	^Y	EM		介质结束	41	)	57	9	73	I	89	Y	105	i	121	y	
1010	A	10	◙	^J	LF	\n	换行	26	→	^Z	SUB		替代	42	*	58	:	74	J	90	Z	106	j	122	z	
1011	B	11	♂	^K	VT	\v	纵向制表	27	←	^[	ESC	\e	溢出	43	+	59	;	75	K	91	[	107	k	123	{	
1100	C	12	♀	^L	FF	\f	换页	28	∟	^\	FS		文件分隔符	44	,	60	<	76	L	92	\	108	l	124	|	
1101	D	13	♪	^M	CR	\r	回车	29	↔	^]	GS		组分隔符	45	-	61	=	77	M	93	]	109	m	125	}	
1110	E	14	♫	^N	SO		移出	30	▲	^^	RS		记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~	
1111	F	15	☼	^O	SI		移入	31	▼	^-	US		单元分隔符	47	/	63	?	79	O	95	_	111	o	127	⌂	^Backspace 代码：DEL

注：表中的ASCII字符可以用“Alt + 小键盘上的数字键”方法输入。 2013/08/08

测试说明

Lab1-2-Extra测试题目

步骤3：

编写刚才创建的【./my_cal/my_cal.c】这个文件，在里面实现【void my_cal()】函数。函数运行之后，**从控制台接受一个十进制非负整数作为输入，以‘\n’（换行符）为结束标志，输出其二进制表示。**

要求：

- 使用_my_getchar()进行字符读取，并使用_my_putchar()进行字符输出。
- 请勿使用printf输出，评测时会对此文件进行printf字符串匹配查询，文件若包含printf字符串则将不会得分。
- my_cal()函数仅执行一次就返回，请勿死循环执行！

Lab1-2-Extra测试题目

关于数据：

- 保证输入的非负整数大小在无符号整数（unsigned int）范围内。
- 该整数可能会有数量不定的前导零，例如：00000012345。
- 输出不允许有前导零。

提示：

- 请在my_cal.c中编写对应的函数原型，以防找不到汇编编写的函数；
- 注意结果按字符输出结果时的顺序。
- 评测时printf及相关函数均无法使用，请务必使用自己编写的汇编函数输出。

测试说明

Lab1-2-Extra测试题目

注意：

- 整个评测过程中，**仅会对./my_cal/目录下的Makefile, my_cal.c, my_driver.S这3个文件进行评测**，其它所有你提交的文件都将被**忽略**并**替换为标准评测版本**；
- _my_exit()的测试将会在评测机的main.c中进行;
- 测试样例如下图所示.

```
GXemul 0.4.6    Copyright (C) 2003-2007  Anders Gavare
Read the source code and/or documentation for other Copyright messages.

Simple setup...
    net: simulating 10.0.0.0/8 (max outgoing: TCP=100, UDP=100)
        simulated gateway: 10.0.0.254 (60:50:40:30:20:10)
            using nameserver 202.112.128.51, domain "s."
    machine "default":
        memory: 64 MB
        cpu0: R3000 (I+D = 4+4 KB)
        machine: MIPS test machine
        loading ./gxemul/vmlinux
        starting cpu0 at 0x80010000
--------------------------------------------------------------------------------

00000012345
11000000111001
```

本地自行测试Tip

1. 修改最外层Makefile文件（加上my_cal目标）
2. 在init/main.c中添加函数调用

北京航空航天大學
BEIHANG UNIVERSITY

下面请同学们开始做题
有问题可以随时提问

祝实验顺利！