oslab1 - 内核多线程 实验报告

张铭方 161220169

多线程运行原理

> 操作系统中运行若干共享内存的线程，每个线程有共享的代码和数据、独立的堆栈(但在同一地址空间)和寄存器现场。我们通过寄存器现场切换来实现分时的多线程。

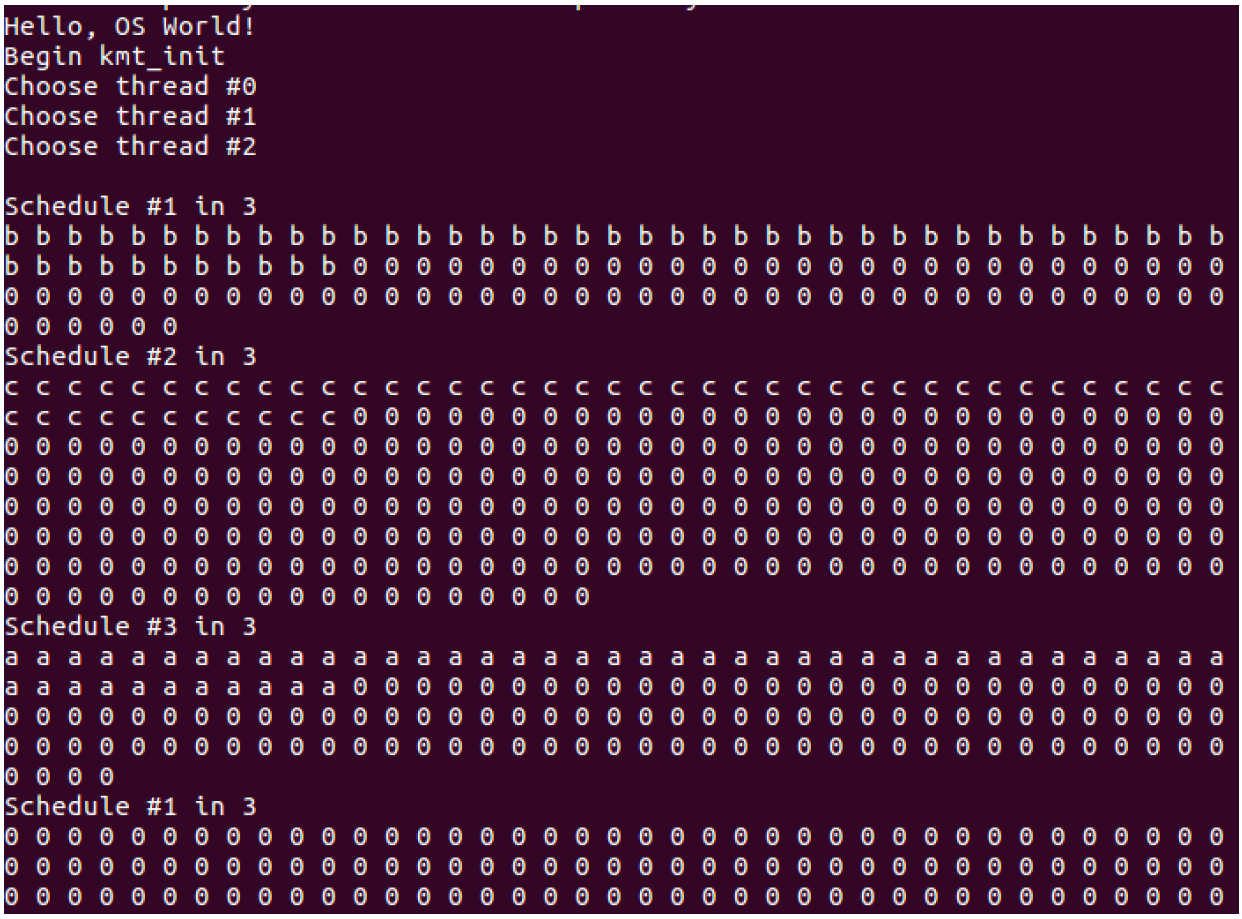
在 `os->run()` 中执行 `kmt->create()` 完成一个新线程的堆栈和寄存器现场创建，之后在 `os->interrupt()` 中每当时钟中断来临，先保存当前运行的寄存器现场，再调用 `kmt->schedule()` 选择下一个该运行的线程，并设置当前的运行寄存器现场为下一个进程的寄存器现场，继续运行。

运行测试

```
static void f(void *arg) {
  kmt->spin_lock(get_lock());

  int i = 0;
  while (i<50) {
    printf("%s ", (char*)arg);
    i++;
  }
  kmt->spin_unlock(get_lock());
  while (1) printf("0 ");
}

static void test_run() {
  kmt->create(new_thread(), f, (void*)"a");
  kmt->create(new_thread(), f, (void*)"b");
  kmt->create(new_thread(), f, (void*)"c");
}
```



遇到的问题：在重新调度同一线程时重新执行该线程

解决：通过 `os->interrupt()` 接受的 `regs` 参数保存运行的当前环境

通过以上截图可看出问题已经解决

生产者消费者问题

通过 P、V 信号量解决

运行测试

```
sem_t empty, fill;
int left = 0;
int right = 0;
static void producer(void *arg) {
  while (1) {
    kmt->sem_wait(&empty);
    printf("%s", (char*)arg); left++;
    printf("%d", left-right);
    if (left - right < 0) _halt(1);
```

```
    kmt->sem_signal(&fill);
  }
}
static void consumer(void *arg) {
  while (1) {
    kmt->sem_wait(&fill);
    printf("%s", (char*)arg); right++;
    printf("%d", left-right);
    if (left - right < 0) _halt(1);
    kmt->sem_signal(&empty);
  }
}
static void test_run() {
  kmt->sem_init(&empty, "empty", BUF_SIZE);
  kmt->sem_init(&fill, "fill", 0);
  kmt->create(new_thread(), producer, (void*)"(");
  kmt->create(new_thread(), consumer, (void*)")");
}
```

设置 BUF_SIZE 为 200

得到结果

```
Schedule #1 in 2
55)198)197)196)195)194)193)192)191)190)189)188)187)186)185)184)183)182)181)180
)179)178)177)176)175)174)173)172)171)170)169)168)167)166)165)164)163)162)161)1
60)159)158)157)156)155)154)153)152)151)150)149)148)147)146)145)144)143)142)141
)140)139)138)137)136)135)134)133)132)131)130)129)128)127)126)125)124)123)122)1
21)120)119)118)117)116)115)114)113)112)111)110)109)108)107)106)105)104)103)102
)101)100)99)98)97)96)95)94)93)92)91)90)89)88)87)86)85)84)83)82)81)80)79)78)77)
76)75)74)73)72)71)70)69)68)67)
Schedule #2 in 2
(68(69(70(71(72(73(74(75(76(77(78(79(80(81(82(83(84(85(86(87(88(89(90(91(92(93
(94(95(96(97(98(99(100(101(102(103(104(105(106(107(108(109(110(111(112(113(114
(115(116(117(118(119(120(121(122(123(124(125(126(127(128(129(130(131(132(133(1
34(135(136(137(138(139(140(141(142(143(144(145(146(147(148(149(150(151(152(153
(154(155(156(157(158(159(160(161(162(163(164(165(166(167(168(169(170(171(172(1
73(174(175(176(177(178(179(180(181(182(183(184(185(186(187(188(189(190(191(192
(193(194(195(196(197(198(199(200
Schedule #1 in 2
199)198)197)196)195)194)193)192)191)190)189)188)187)186)185)184)183)182)181)18
0)179)178)177)176)175)174)173)172)171)170)169)168)167)166)165)164)163)162)161)
160)159)158)157)156)155)154)153)152)151)150)149)148)147)146)145)144)143)142)14
1)140)139)138)137)136)135)134)133)132)131)130)129)128)127)126)125)124)123)122)
121)120)119)118)117)116)115)114)113)112)111)110)109)108)107)106)105)104)103)10
2)101)100)99)98)97)96)95)94)93)92)91)90)89)88)87)86)85)84)83)82)81)80)79)78)77
)76)75)74)73)72)71)70)69)68)67)66)65)64)63)62)61)60)59)58)57)56)55)54)53)52)51
)50)49)48)47)46)45)44)43)42)41)40)39)38)37)36)35)34)33)32)31)30)29)28)27)26)25
)24)23)22)21)20)19)18)17)16)15)14)13)12)11)10)9)8)7)6)5)4)3)2)1)0
Schedule #2 in 2
(1(2(3(4(5(6(7(8(9(10(11(12(13(14(15(16(17(18(19(20(21(22(23(24(25(26(27(28(29
(30(31(32(33(34(35(36(37(38(39(40(41(42(43(44(45(46(47(48(49(50(51(52(53(54(55
(56(57(58(59(60(61(62(63(64(65(66(67(68(69(70(71(72(73(74(75(76(77(78(79(80(81
(82(83(84(85(86(87(88(89(90(91(92(93(94(95(96(97(98(99(100(101(102(103(104(105
(106(107(108(109(110(111(112(113(114(115(116(117(118(119(120(121(122(123(124(1
25(126(127(128(129(130(131(132(133(134(135(136(137(138(139(140(141(142(143(144
(145(146(147(148(149(150(151(152(153(154(155(156(157(158(159(160(161(162(163(1
64(165(166(167(168(169(170(171(172(173(174(175(176(177(178(179(180(181(182(183
```

物理内存的分配和释放

实现同 `malloc` 实验，除了要实现 `brk()` 和 `sbrk()`

根据 manual 实现：

> brk() sets the end of the data segment to the value specified by addr, when that value is reasonable, the system has enough memory, and the process does not exceed its maximum data size (see setr−limit(2)).
>
> sbrk() increments the program's data space by increment bytes. Calling sbrk() with an increment of 0 can be used to find the current location of the program break.

遇到的问题：

难以测试实现是否正确