LISS
信息安全研究所

操作系统设计及实践

《操作系统原理》配套实验

信安系操作系统课程组

2024年11月

操作系统设计实验系列（十）

进程间通信

武漢大学

一、实验目标

- 了解微内核架构和宏内核架构的差异
- 理解微内核架构中IPC的实现机理
- 掌握微内核架构中IPC的实现技巧

二、本次实验基本内容

1. 验证IPC的实现机理
2. 学习分析IPC实现的技巧与细节

三、本次实验要解决的问题

1. 阅读8.1节资料回答以下问题：
   - ① 微内核与宏内核在系统调用角度差异是什么？
   - ② 我们之前的实验实现，更类似哪种架构？
   - ③ 调研一下，目前的主流桌面OS，如windows, linux, mac OS都是怎样的内核架构

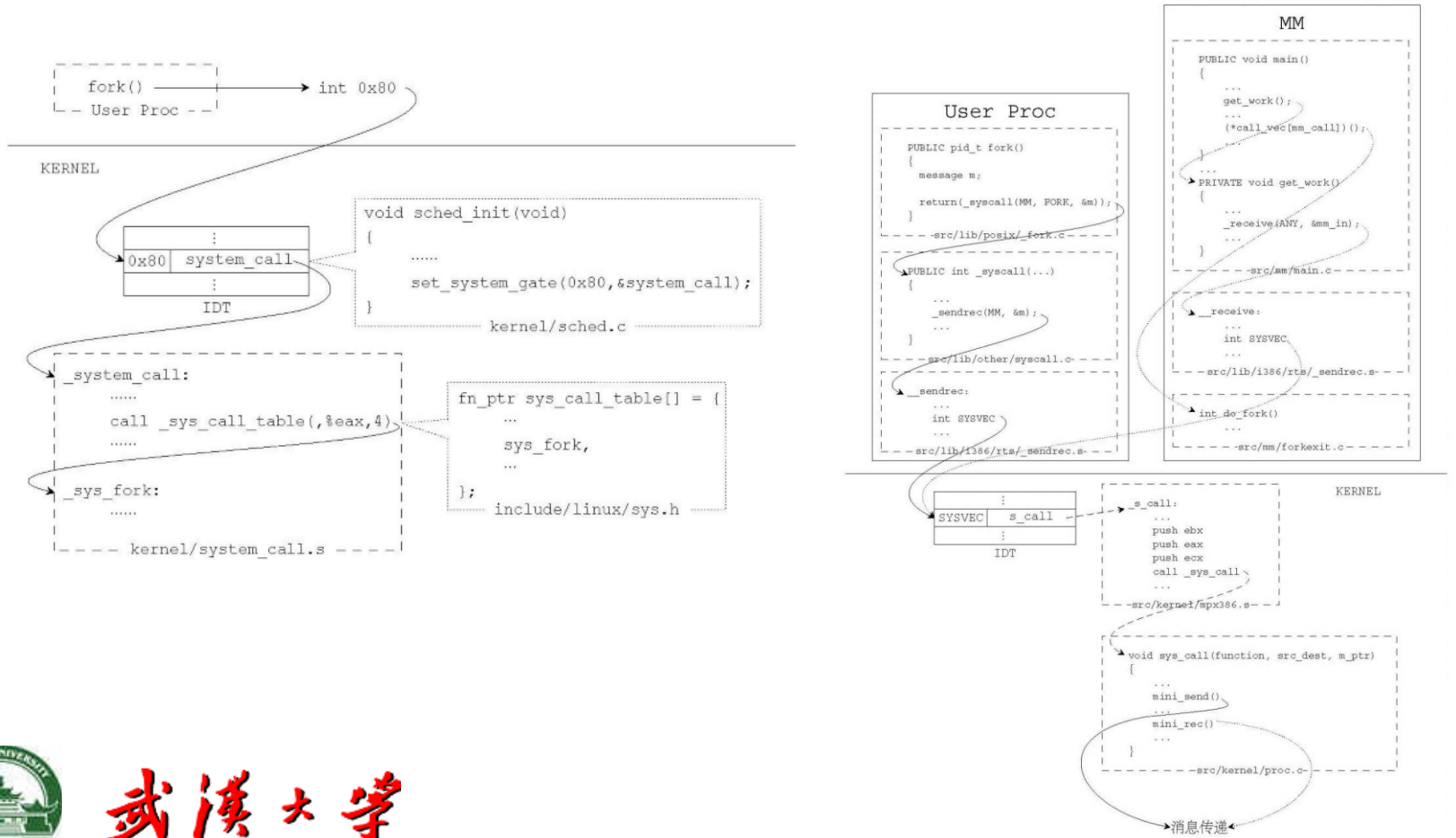
三、本次实验要解决的问题

2. 阅读8.2–8.5，并对代码目录a进行分析，完成以下任务：
   - ① 画出一个逻辑关系图，描述本章实验中IPC的实现框架机理，并加以文字解释，特别注意：处理器状态的切换，信息的流向
   - ② 简要描述该处涉及系统调用的流程与作用
   - ③ 在该代码中，当涉及程序与中断事件的并发时，是如何施加保护的？
   - ④ 解释一下assert、Panic的实现过程（含涉及的系统调用机理），撰写几个小例子验证其作用。
   - ⑤ 在本部分的消息机制中，如何实现通信的？对进程如何调度管理的？
   - ⑥ 死锁问题是如何解决的？是否存在问题，若有改进之，若无说明验证其正确性。
   - ⑦ 简要分析基于IPC，如何扩展get_ticks的方法

3. 针对上学期学习的经典的同步互斥问题，试着用IPC解决一例。

四、需要了解的一些知识

- Linux和Minix中的系统调用



四、需要了解的一些知识

- 如何具体实现：

```
 syscall.asm
INT_VECTOR_SYS_CALL equ 0x90
_NR_printx      equ 0
_NR_sendrec     equ 1

; ========================================================================================
;                  sendrec(int function, int src_dest, MESSAGE* msg);
; ========================================================================================
; Never call sendrec() directly, call send_recv() instead.
sendrec:
    mov eax, _NR_sendrec
    mov ebx, [esp + 4]  ; function
    mov ecx, [esp + 8]  ; src_dest
    mov edx, [esp + 12] ; p_msg
    int INT_VECTOR_SYS_CALL
    ret


global.c
PUBLIC  system_call sys_call_table[NR_SYS_CALL] = {sys_printx, sys_sendrec};
```

武漢大学

四、需要了解的一些知识

- 向上找调用者：
  - 见proc.c中send_recv

```
PUBLIC int send_recv(int function, int src_dest, MESSAGE* msg)
{
    int ret = 0;

    if (function == RECEIVE)
        memset(msg, 0, sizeof(MESSAGE));

    switch (function) {
    case BOTH:
        ret = sendrec(SEND, src_dest, msg);
        if (ret == 0)
            ret = sendrec(RECEIVE, src_dest, msg);
        break;
    case SEND:
    case RECEIVE:
        ret = sendrec(function, src_dest, msg);
        break;
    default:
        assert((function == BOTH) ||
               (function == SEND) || (function == RECEIVE));
        break;
    }

    return ret;
}
```

四、需要了解的一些知识

- 向上找调用者：
  - 见main.c中get_ticks

```
PUBLIC int get_ticks()
{
    MESSAGE msg;
    reset_msg(&msg);
    msg.type = GET_TICKS;
    send_recv(BOTH, TASK_SYS, &msg);
    return msg.RETVAL;
}
/*======================================================================*
                               TestA
 *======================================================================*/
void TestA()
{
    while (1)
    {
        printf("<Ticks:%d>", get_ticks());
        milli_delay(200);
    }
}
```

四、需要了解的一些知识

- 向下找系统调用实现：
  - 见proc.c 中sys_sendrec
    - msg_send
      - 检查是否死锁
      - 如果目标程序正在处于接收消息状态，则复制发送者的消息到目标程序，设置目标程序相应状态，唤醒。
      - 否则（目标程序没有处于接收消息），则设置发送者进程为发送中，把消息挂载到发送者proc的发送队列中，阻塞发送进程

四、需要了解的一些知识

- 向下找系统调用实现：
  - 见proc.c 中sys_sendrec
    - msg_receive
      - 如果有一个硬件中断消息，则把消息交给接收者
      - 如果试图接收任意进程的消息，那么就把第一个消息复制给他。
      - 如果试图接收某个进程的消息，则复制消息
      - 如果没有进程发送消息，则调用阻塞自己。

四、需要了解的一些知识

- 实现中几个重要的函数
  - assert与panic
  - block和unblock
    - Block：阻塞进程，检测状态，并转调度
    - Unblock：实际是一个检测状态的空函数，在哪儿改变的呢？
  - Deadlock:检测循环
  - Schedule:会判断当前进程的状态，来进行调度

谢　谢！