计算机系统软件（1）实验报告

组长：2016080042 李相赫

2015080118 崔殷庇

2016011990 靳紫荆

一、 实验目的

- 实现GUI，在GUI的基础上增加应用程序
- 进行系统内部优化。增加多种进程调度算法、实现copy-on-write fork和实现进程间信息传递

二、 实验环境

开发环境：Ubuntu version：14.04 + QEMU

开发语言：C / 汇编语言

测试环境：Ubuntu 14.04 以及Ubuntu 16.04均测试通过

三、 开发思路

1.开发图形界面

使用VESA视频模式中提供的Linear Buffer Frame。它易于管理视频内存，而且屏幕上的点与视频储存地址的关系比较直观。

2.支持灵敏鼠标，完善键盘操作

了解的PS/2接口协议。

3.文本编译器

文本编辑器是基于GUI界面的文字处理程序，通过接受键盘发送的信号基于可视化实现的文档处理系统，实际上是将键盘信号以可视化文字的形式显示在GUI界面中。

4.进程间信息传递

信号(Signal)是Linux系统中用于进程之间通(IPC)的一种机制，信号可以在任何时候发送给某一进程，而无须知道对象进程的状态。 一旦有信号产生，进程可以执行对每种信号的默认操作，也可

以自定义信号处理函数，当信号发生时捕捉信号，执行相应的处理函数。

四、具体实现技术

1. Process scheduling algorithms

修改scheduler函数，实现多种进程调度算法。其包括RR（默认）、FIFO、PRIORITY SCHEDULING和 MULTI-LEVEL FEEDBACK QUEUE。并修改trap函数，控制CPU的谦让。增加SCHED_TYPE，在userinit函数中设置。

核心实现过程：

i. RR：xv6默认调度算法

ii. FIFO：在proc结构体加成员变量unit in_time，记录该进程进入就绪队列的时间。调度时遍历页表找出最早进入就绪队列，即in_time最小的进程执行。FIFO到进行中的进程执行完为止不谦让CPU，因此需要修改trap函数，改为即使发生时间中断也不谦让CPU。

iii. PRIORITY SCHEDULING：在proc结构体加成员变量int priority，表示该进程的优先程度，priority越小优先权越大。调度时遍历页表找出优先权最大的进程，即priority最小的进程执行。具有相同优先权的进程之间用RR法。

iv. MULTI-LEVEL FEEDBACK QUEUE：在proc结构体加成员变量int mlq_level和int tick，分别表示该进程属于的就绪队列等级和执行CPU clock次数（为了不同time quantum计数）。进程的priority越小，所属的mlq_level越小。一个队列的time quantum设成mlq_level*10。根据优先程度，进程会分到不同等级的就绪队列。系统优先执行小等级队列的进程，一个队列里没有优先之分，用RR进行调度。一个time quantum之内若没执行完，则该进程priority加1（mlq_level最大的队列中不再增加priority值，防止越界），进入相应等级的就绪队列。没有真正地利用队列结构体而实现了队列。

修改的文件：proc.c proc.h trap.c foo.c ps.c chpr.c (增加系统调用需修改syscall.c syscall.h sysproc.c user.h usys.S Makefile以后不再列举)

测试文件：foo.c（利用ps和chpr命令看实验结果）

2. copy-on-write fork

xv6原有的copyuvm拷贝父进程的每个页表项的物理页块（physical frame）给孩子进程，而我们实现的cowuvm不及时进行拷贝，而给父进程的每个pte只读标记（以删除可写标记PTE_W来做到），让孩子参考（map）父亲的页块。当其中一个进程尝试写入内存时，由于只读标记CPU抛出page fault中断（T_PGFLT），这时新分配出内存给孩子进程。

核心实现过程：

i. mmu.h中加新的页表标志PTE_COW，表示pte对应的物理页块正在被共享。当父进程创建子进程时给此标记。

ii. 以cowuvm函数替代copyuvm。给父进程的每个pte只读标记，让孩子参考父亲的物理页块，页块的参考次数加1。由于父进程的页表有了变动，最后要清空TLB。

iii. 当父进程或孩子进程尝试写入有PTE_COW标记的（即只读）内存时，CPU会抛出page fault中断（T_PGFLT）。因此需要写page fault handler来处理中断。

iv. page fault handler的工作是：找到发出T_PGFLT中断的虚拟地址和物理地址，分配新的物理页块，把现在参考的（mapped）页块复制到新的页块。旧页块的参考次数减1，使pte指向新的页块并删掉只读标记和PTE_COW标记，从而可以改变内存。因为页表有变动，于是清空TLB。

v. 修改deallocuvm使只有没有被共享的页块才能释放内存，以保证每个进程有指向的内存。

修改的文件：vm.c trap.c proc.c mmu.h defs.h kalloc.c cow_test.c

测试文件：cow_test.c

3. Signals Framework

核心实现过程：

i. 新加函数指针类型的结构体sighandler_t。在proc结构体加成员变量uint signal和sighandler_t sighandlers，表示目前收到的信号和信号处理函数的指针数组。

ii. 写系统调用signal，其功能是自定义进程的信号处理函数（register custom handler）。写系统调用sigsend，其功能是向指定进程发信号。

iii. 写一个函数register_handler，当收到信号时注册相应的信号处理函数（更新esp和eip，esp值减4，eip指向信号处理函数）。由调度算法一个进程被选出来后，该进程得到CPU之前检查是否收到了信号。若收到了，则调用register_handler而注册信号处理函数。该进程得到CPU后，信号处理函数会被执行。

iv. 实现的信号有：SIGINT(id=0)，SIGKILLCHILD(id=1)，SIGCHILDEXIT(id=2).

这些信号的默认处理设为：SIGINT-杀死进程，SIGKILLCHILD-杀死子进程，SIGCHILDEXIT-告诉父进程，它的子进程已被杀死。

修改的文件：proc.c proc.h defs.h types.h sigtest.c

测试文件：sigtest.c

4. 鼠标驱动

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Y overflow	X overflow	Y sign bit	X sign bit	Always 1	Middle Btn	Right Btn	Left Btn
Byte 2	X movement							
Byte 3	Y movement							

如上图，标准PS/2鼠标会把鼠标移动信息分成3个byte传递，分别为鼠标的按钮状态和X轴与Y轴的移动距离。读入鼠标信息并不难，激活鼠标中断之后，对鼠标进行操作就会发生中断，此时读取键盘端口0x60。

最后使用环形队列来管理这些数据。环形队列是一个首尾相连的FIFO的数据结构，采用数组的线性空间,数据组织简单。能很快知道队列是否满为空。能以很快速度的来存取数据。

5. API接口

为了方便在界面上的绘图，实现了一些API接口，包括在窗口上绘制直线等图形，以及打印字符的函数，这些API接口大大简化了记事本以及画图程序的代码。

6. 窗口管理

实现了窗口队列的管理。具体实现如下：把所有工作中的窗口放在一起，按打开的顺序排列。每一次对窗口进行操作(关闭窗口，点击铺盖在下面的窗口等)时，根据排列的顺序重新加载图片与窗口。如果点击了铺盖在另一个窗口下的窗口，那么把该窗口调到最优先位置，按排列重新加载。如果点击了关闭按钮，那么把该窗口从排列中除去，重新按顺序加载。

五、任务分工

李相赫：鼠标驱动与消息队列，窗口管理，API，简单的画图板，报告撰写

靳紫荆：文本编译器以及相关的数据结构，报告撰写

崔殷庇：进程调度算法，copy-on-write fork，进程间信息传递及其相关测试代码，报告撰写

六、总结

由于初次接触操作系统内核代码，遇到汇编代码，理解xv6架构不太容易。但是理解清楚以后，顺利完成各自的任务，成功地优化了xv6。通过本次实验，对操作系统的工作原理有了很深的理解。

七、参考文献

[1] signals framework指导文献 https://www.cs.bgu.ac.il/~os122/wiki.files/Operating%20Syst

ems%20-%20assignment%201.pdf

[2] copy-on-write fork指导文献 https://www.cse.iitb.ac.in/~mythili/teaching/cs347_autumn2016/labs/lab6.pdf

[3] 图片的加载 https://github.com/THSS13/XV6/tree/master/xv6