《操作系统原理》
实 验 报 告 书

学 生 姓 名　范仲昊

学　　号　20009200293

班　　级　2018011

2021 — 2022 学年　第 2 学期

《操作系统原理》实验报告

实验名称	Linux 操作系统实例	实验序号	6
实验日期	2022/6	实验人	范仲昊 20009200293

一、实验题目

- 阅读教材第 18 章（Linux 案例），并在互联网上查阅相关资料，对照操作系统课程中所讲的原理（进程管理，存储管理，文件系统，设备管理），了解 Linux 操作系统实例
- 形成一份专题报告
- 可以是全面综述性报告
- 可以是侧重某一方面的报告（进程调度，进程间通信，存储管理，文件系统，安全）

本题选择研究 Linux 系统有关于进程调度和进程间通信的操作实例。

二、相关原理与知识

进程的结构

Linux 下一个进程在内存里有三部分的数据，就是"代码段"、"堆栈段"和"数据段"。一般的 CPU 都有上述三种段寄存器，以方便操作系统的运行。这三个部分也是构成一个完整的执行序列的必要的部分。

进程的分类：

（1）I/O 消耗型：
运行的进程如果大部分来进行 I/O 的请求或者等待，这种类型的
进程经常处于可以运行的状态，但是都只是运行一点点时间，绝大多数的
时间都在处于阻塞（睡眠）的状态。
（2）CPU 消耗型：
进程的绝大多数都在使用 CPU 做运算，对于处理器消耗型的进程，
调度策略往往是降低他们的执行频率，延长运行时间。

三、实验过程

1. 阅读教材，在最基础的资料里获取并理解基础的知识 。

2. 在通读课程教材的情况下确定选题，最好是自己理解并且与课程高度关联。

3. 自主查阅资料，包括从网上获取资料，也包括课外书籍和更多课程资料，获取更多信息，加深理解。

4. 将获取到的资料进行整理，筛选自己需要的资料，并进一步确定自己在报告中要用到那些。

5. 根据整理过的资料撰写报告，并修改润色。

四、实验结果与分析

LINUX 系统下进程的结构

Linux 下一个进程在内存里有三部分的数据，就是"代码段"、"堆栈段"和"数据段"。一般的 CPU 都有上述三种段寄存器，以方便操作系统的运行。这三个部分也是构成一个完整的执行序列的必要的部分。

"代码段"，顾名思义就是存放了程序代码的数据，假如机器中有数个进程运行相同的一个程序，那么它们就可以使用相同的代码段。"堆栈段"存放的就是子程序的返回地址、子程序的参数以及程序的局部变量。而数据段则存放程序的全局变量，常数以及动态数据分配的数据空间（比如用 malloc 之类的函数取得的空间）。这其中有许多细节问题，这里限于篇幅就不多介绍了。系统如果同时运行数个相同的程序，它们之间就不能使用同一个堆栈段和数据段。

LINUX 系统下进程的调度

1.进程调度含义：
进程调度决定了将哪个进程进行执行，以及执行的时间。
2.进程调度作用：
操作系统进行合理的进程调度，可以使资源得到最大化的利用。
3.进程调度的任务：
（1）、分配时间给进程
（2）、上下文切换
4.
Linux 调度算法：

（1）相关介绍：
Linux 中有一个总的调度结构，称之为 调度器类（scheduler class），它允许不同的可动态添加的调度算法并存，总调度器根据调度器类的优先顺序，依次去进行调度器类的中的进程进行调度，挑选了调度器类，再在这个调度器内，使用这个调度器类的算法（调度策略）进行内部的调度。
Linux 系统为了提升响应度速度，倾向于优先调度 I/O 消耗型.

（2）调度器的优先级顺序：

Stop_ask > Real_Time > Fair > Idle_Task

（3）进程调度的时机

a.当前进程主动放弃处理器，如正常终止
b.当前进程主动被动放弃处理机，如响应中断

需要注意的是，进程在操作系统内核程序临界区中**不能**进行调度与切换。

（4）进程调度的方式

a.非抢占方式

只允许进程主动放弃处理机。

实现简单，系统开销小但是无法及时处理紧急任务，适合用于早期的批处理系统。

b.抢占方式

当有一个更重要或者更紧迫的进程需要使用处理机时，就会暂停正在进行的进程，然后将处理机分配给更紧急的进程。

可以优先处理更紧急的进程，也可以实现让各进程按时间片轮流执行的功能。适合用于分时操作系统，实时操作系统。

（5）调度算法的评价标准：

a. CPU 利用率
b. 系统吞吐量
c. 周转时间，分为平均周转时间和平均带权周转时间
d. 等待时间
e. 响应时间

（6）调度算法

A.先来先服务（FCFS）

- 算法思想：
  从“公平”的角度考虑
- 算法规则：
  按照进程到达的先后顺序进行服务，用于进程调度，考虑那个进程先到达就绪队列
- 是否可抢占：
  非抢占式的算法
- 优点：
  公平，算法简单
- 缺点：
  排在进程后面的短作业需要等待很长时间，带权周转时间很长。FCFS 对于长作业有利。
- 是否会导致饥饿：
  不会

b.短作业优先（SJF）

- 算法思想：
  追求最少的平均等待时间，最少的平均周转时间，最少的平均带权周期。
- 算法规则：
  服务时间最短的进程优先得到服务
- 用于进程调度：
  也可以称为“短进程”算法
- 是否可抢占：
  非抢占式的算法。但可以进行修改变成抢占式调度算法，最短剩余时间优先算法。
- 优点：
  “最短的”平均等待时间、平均周转时间
- 缺点：
  对短作业有利，不利于长作业。可能产生饥饿现象。
- 是否会导致饥饿 ：
  会。如果一直有短作业进来长作业就一直没有办法得到资源进行服务。

c.高响应比优先（HRRN）

- 算法思想:
  综合考虑进程的等待时间和要求服务的时间
- 算法规则:
  在每次调度时先计算每个进程的响应比（用等待时间与要求时间的和除以要求服务时间），为响应比最高的进程进行服务。
- 是否可抢占 ：
  非抢占式的算法。
- 优点:
  综合考虑，结合了上面两种算法的优点。
- 是否会导致饥饿
  不会

（7）基于内核的 CFS 算法

- CFS 算法迎来了完全公平调度程序。
- CFS 引入了一种新的调度算法， 称为公平调度，消除了传统意义上的时间片。
- CFS 依赖于两个可配置变量，**目标延迟**就是每个可运行任务运行一 次的时

间间隔，**最小粒度**是每个进程分的的最小时间长度。

- CFS 除了静态时间片的概念。每个进程都接受到一定比例的处 理器时间，分配的时间长度取决于现有多少其他可运行进程。CFS 算法在交互
- 工作负载上表现良好，而且不会影响大型服务器的吞吐性能

1. LINUX 系统下进程的通信

fork（）和 exec（）进程模式

传统的 UNIX 进程管理的基本原理是把创建进程与运行一个新进程这两个截然不同的操作分开，一个新进程由系统调用 fork（）产生，而新的一个程序通过调用 exec（）来运行，这是两个完全不同的函数。由 fork（）创建的新进程不需要运行新程序——新创建的子进程仅仅是继续执行与父进程同样的进程。同样，运行一个新程序不需要创建新进程：任何进程可以随时调用 exec（）。当前运行的程序立刻被中止，新的程序作为已经存在进程的内容开始执行。

Linux 的进程控制和传统的 Unix 进程控制基本一致，只在一些细节的地方有些区别，例如在 Linux 系统中调用 vfork 和 fork 完全相同，而在有些版本的 Unix 系统中，vfork 调用有不同的功能。由于这些差别几乎不影响我们大多数的编程，在这里我们不予考虑。

一个程序一调用 fork 函数，系统就为一个新的进程准备了前述三个段，首先，系统让新的进程与旧的进程使用同一个代码段，因为它们的程序还是相同的，对于数据段和堆栈段，系统则复制一份给新的进程，这样，父进程的所有数据都可以留给子进程，但是，子进程一旦开始运行，虽然它继承了父进程的一切数据，但实际上数据却已经分开，相互之间不再有影响了，也就是说，它们之间不再共享任何数据了。而如果两个进程要共享什么数据的话，就要使用另一套函数（shmget，shmat，shmdt 等）来操作。现在，已经是两个进程了，对于父进程，fork 函数返回了子程序的进程号，而对于子程序，fork 函数则返回零，这样，对于程序，只要判断 fork 函数的返回值，就知道自己是处于父进程还是子进程中。事实上，目前大多数的 unix 系统在实现上并没有作真正的 copy。一般的，CPU 都是以“页”为单位分配空间的，象 INTEL 的 CPU，其一页在通常情况下是 4K 字节大小，而无论是数据段还是堆栈段都是由许多“页”构成的，fork 函数复制这两个段，只是“逻辑”上的，并非“物理”上的，也就是说，实际执行 fork 时，物理空间上两个进程的数据段和堆栈段都还是共享着的，当有一个进程写了某个数据时，这时两个进程之间的数据才有了区别，系统就将有区别的“页”从物理上也分开。系统在空间上的开销就可以达到最小。

一个进程一旦调用 exec 类函数，它本身就“死亡”了，系统把代码段替换成新的程序的代码，废弃原有的数据段和堆栈段，并为新程序分配新的数据段与堆栈段，唯一留下的，就是进程号，也就是说，对系统而言，还是同一个进程，不过已经是另一个程序了。不过 exec 类函数中有的还允许继承环境变量之类的信息，这个通过 exec 系列函数中的一部分函数的参数可以得到。

在 LINUX 系统下，可以通过 fork()和 exec()函数来进行进程管理，并且进程间通信方式有很多种，而且每一种都可以通过 fork()函数来通过实例实现。