管理进程状态

进程，线程和协程是操作系统中经常出现的名词，它们都是操作系统中的抽象概念，有联系和共同的地方，但也有区别。计算机的核心是CPU，它承担了基本上所有的计算任务；而操作系统是计算机的管理者，它可以以进程，线程和协程为基本的管理和调度单位来使用CPU执行具体的程序逻辑。

从历史角度上看，它们依次出现的顺序是进程、线程和协程。在还没有进程抽象的早期操作系统中，计算机科学家把程序在计算机上的一次执行过程称为一个任务（task）或一个工作（job），其特点是任务和工作在其整个的执行过程中，不会被切换。这样其他任务必须等待一个任务结束后，才能执行，这样系统的效率会比较低。

在引入面向CPU的分时切换机制和面向内存的虚拟内存机制后，进程的概念就被提出了，进程成为 CPU 调度和分派的对象，各个进程间以时间片为单位轮流使用 CPU，且每个进程有各自独立的一块内存，使得各个进程之间内存地址相互隔离。这时，操作系统通过进程这个抽象来完成对应用程序在CPU和内存使用上的管理。

随着计算机的发展，对计算机系统性能的要求越来越高，而进程之间的切换开销相对较大，于是计算机科学家就提出了线程。线程是程序执行中一个单一的顺序控制流程，线程是进程的一部分，一个进程可以包含一个或多个线程。各个线程之间共享进程的地址空间，但线程要有自己独立的栈（用于函数访问，局部变量等）和独立的控制流。且线程是处理器调度和分派的基本单位。对于线程的调度和管理，可以在操作系统层面完成，也可以在用户态的线程库中完成。用户态线程也称为绿色线程（GreenThread）。如果是在用户态的线程库中完成，操作系统是“看不到”这样的线程的，也就谈不上对这样线程的管理了。

协程（coroutines），也是程序执行中一个单一的顺序控制流程，建立在线程之上（即一个线程上可以有多个协程），但又比线程更加轻量级的处理器调度对象。协程一般是由用户态的协程管理库来进行管理和调度，这样操作系统是看不到协程的。而且多个协程共享同一线程的栈，这样协程在时间和空间的管理开销上，相对于线程又有很大的改善。在具体实现上，协程可以在用户态运行时库这一层面通过函数调用来实现；也可在语言级支持协程，比如 Rust 语言引入的 `async` 、 `wait`

关键字等，通过编译器和运行时库二者配合来简化程序员编程的负担并提高整体的性能。

观察进程状态

本次实验的首要任务是获取系统当前时刻的进程快照。请在终端中执行：

```
ps aux
```

该指令将输出系统中所有用户的所有进程信息，且不以终端归属进行筛选。

在滚动的输出流中，请重点关注表头的 PID（进程标识符）、%CPU（处理器占用率）以及 STAT（进程状态）字段。为了验证对进程状态代码的理解，请在输出列表中寻找状态列标记为 `R` 或 `R+` 的进程，这代表该进程当前正处于可运行状态。

随后，请执行：

```
top
```

指令以进入动态监控视图，实时显示正在运行的进程。在实时刷新的界面中，可尝试按下 `P` 键，观察进程列表是否依据 CPU 负载进行了重新排序。确认无误后，按下 `q` 键退出监控界面。

ps 指令的常用选项有：

选项	说明
x	显示机器上的所有进程，不以终端来区分
a	显示终端上的所有进程，不以用户来区分
w	提供详细的宽范围输出
u	显示面向用户的格式
f	以进程树格式列出进程

其中最常用的是 `ps -aux` 命令。

ps 运行结果的各字段含义：

字段	说明
UID	用户ID
PID	进程ID
PPID	父进程ID
TTY	控制终端ID
PRI	优先级编号（编号越低，分配给此进程的计算机时间越多）
NI(nice)	影响动态优先级调整
STAT	当前的进程状态
TIME	使用的CPU时间
COMMAND	命令的名称

而进程状态 STAT 可为以下状态之一：

状态代码	说明
R（可运行）	可运行的进程
S（休眠）	正在等待外部事件的进程。
D（不可中断的休眠）	类似 S， 区别在于此时不能终止此进程
T（已跟踪或已停止）	进程已被暂停
X （不用）	进程已死
Z（僵进程）	进程已自行终止，但尚未释放

top 的常用命令：

字段	说明
?	帮助

字段	说明
h	帮助
r	将一个新的 nice 值分配给运行中的进程
k	向某个运行中的进程发送终止信号（与 kill 或 killall相同）
N	按进程 ID 排序
P	按 CPU 负载排序

top 运行结果的各字段含义：

字段	说明
PID	进程ID
USER	用户名
PR	优先级
NI	Nice值
VIRT	进程使用的虚拟内存总量（单位为KB）
RES	进程持有的物理内存总量（单位为KB）
SHR	共享内存大小（单位为KB）
S	进程状态
%CPU	CPU使用率
%MEM	内存使用率(RES)
TIME	CPU时间
COMMAND	命令名称/行

分析进程树

Linux 系统中的进程通过父子关系构成了严格的树状结构。为了直观地解析这一关系，请执行：

```
pstree -p
```

该指令将以树形图的方式展示进程间的派生关系，并同时显示每个节点的 PID。

请在输出的树状图中，溯源当前正在使用的 Shell 进程（通常为 bash 或 zsh）。通过追踪连接线，识别出该 Shell 进程的父进程（PPID），并一直向上追溯至根节点（通常为 systemd 或 init），以此构建出当前会话的完整进程家族谱系。

pstree常用选项：

选项	说明
-p	显示进程PID
-u	显示用户ID

管理后台进程

本环节要求手动模拟多任务并发场景。

请首先执行 `sleep 180` ，这将启动一个持续 180 秒的休眠进程。在光标停滞状态下，需按下 `Ctrl+Z` 组合键，向该前台进程发送挂起信号，使其暂停并进入后台。

随后，连续执行 `sleep 240 &` 以及 `sleep 300 &` 。行尾的 `&` 符号指示系统直接在后台启动这些进程。此时，执行 `jobs` 指令，终端应列出上述三个作业及其对应的作业号。

为了演练前后台切换，实验者需执行 `fg 1`（假设第一个作业号为 1），将挂起的进程调回前台，随后立即按下 `Ctrl+C` 终止该进程。接着，执行 `bg 2`，确保第二个作业在后台继续运行而非处于暂停状态。

调整进程优先级

在掌握了作业控制后，接下来练习通过信号直接干预进程生命周期。

首先，利用 `ps` 命令查找剩余 `sleep` 进程的 PID。假设目标进程 PID 为 1234，实验者需执行 `kill -15 1234`（即 SIGTERM 信号），观察进程是否正常退出。

若进程无响应，则需尝试执行 `kill -9 1234`（即 SIGKILL 信号）强制销毁进程。

随后的任务涉及 CPU 调度优先级的调整。首先启动一个新的后台进程： `sleep 1000 &`。使用 `ps -l` 查看该进程当前的 NI（Nice）值，默认值应为 0 或 10。

接下来尝试执行 `renice +5 -p [PID]` 指令，将该进程的 Nice 值增加 5，这意味着降低该进程的调度优先级。随后尝试执行 `renice -5 -p [PID]`，如果当前用户不是 Root 用户，系统将返回“权限拒绝”错误，因为只有 Root 用户具备降低 Nice 值（即提升进程优先级）的权限。

/proc 文件系统

系统中运行的每个进程在 /proc 目录下有一个目录，目录名称为进程的 PID。

利用 /proc 文件系统，选择一个进程，例如 bash 或 shell，使用 `cd` 进入 `/proc/[PID]/` 目录，随后执行 `cat status` 指令。请解释运行结果中的 `Name`、`State` 及 `Pid` 字段。

随后，执行 `cat cmdline` 指令，验证该文件是否准确记录了启动该进程时的完整命令行参数。

/proc/$PID/ 下包含进程的详细信息：

文件	说明
cmdline	进程的命令行

文件	说明
environ	进程的环境信息
fd目录	为每个打开的文件描述符提供一个入口
mem	可通过其访问进程的内存映像
stat	包含进程的大多数信息
status	包含用户可读的进程信息
cwd	指向进程的当前工作目录
exe	指向正在被执行的文件
maps	内存映射区信息
root	进程的根目录
statm	进程对内存的使用情况