COMP3230 操作系统原理

编程作业一2023 年 10 月 22 日

23:59

共计 11 分

(版本：1.1)

编程练习 - 执行作业提交程序

目标

1. 与 ILO4[实用性]有关的评估任务--"展示应用系统软件和现代操作系统中可用的工具进行软件开发的知识"。
2. 与国际劳工组织 2a 有关的学习活动。
3. 这项编程任务的目标是
   - 进行设计和开发外壳程序的实际操作练习，其中涉及多个流程的创建、管理和协调；
   - 学习如何使用各种重要的 Unix 系统功能
     - 来执行进程创建和程序执行；
   - 使用信号和管道支持进程之间的交互；以及
   - 通过读取 /proc 文件系统来获取进程的运行统计信息。

任务

Shell 程序通常被称为命令解释器，是一个作为操作系统用户界面的程序，可以让系统理解你的请求。例如，当你输入 "**ls -la** "命令时，Shell 会为你执行名为 **ls 的**系统实用程序。shell 程序可以交互式使用，也可以批量处理。

您将执行一个 C 程序，作为***交互式作业提交程序，程序***名为 ***JCshell***。该程序支持以下功能（详细内容将在规范部分介绍）：

1. 程序接受一条命令或由一系列命令组成的任务，这些命令用管道 (|) 连接在一起，并用给定的参数列表执行相应的命令。
2. 该程序可以通过提供绝对路径（以 / 开头）或相对路径（以 ./ 或 ./ 开头），或搜索 $PATH 环境变量下的目录，找到并执行任何有效的程序（即已编译的程序）。
3. 程序应由内置的*退出*命令终止，但不能由 Cltr-c 键或 SIGINT 信号终止。

4. 提交的命令/任务终止后，程序会打印所有终止命令的运行统计信息，并等待用户的下一条命令/任务。

我们建议您将计划的实施分为三个阶段：

- 阶段 1 - 创建 JCshell 程序的第一个版本，该程序(i)接受用户输入行（包含命令及其参数），(ii)创建一个子进程来执行命令，(iii)等待子进程终止，(iv)检查子进程的退出状态，(v)打印运行统计信息和已终止子进程的退出状态。如果进程已终止

信号，它就应该反映在输出中。子进程结束后，JCshell 程序会打印 shell 提示，并等待用户的下一条命令。

- 第 2 阶段 - 修改 JCshell 程序的前一版本，使其能够 (i) 正确处理 SIGINT 信号，(ii) 使用 SIGUSR1 信号控制子进程，以及 (iii) 在用户输入*退出*命令时终止 JCshell 程序。
- 第 3 步 - 修改 JCshell 程序的前一版本，使其能够接受不超过 5 条带/不带参数的命令，并用"|"符号分隔命令。JCshell 程序应先等待所有命令完成，然后再打印每个已终止命令的运行统计信息（按照命令终止的顺序）。

规格

JCshell 程序的行为：

- 与传统的 shell 程序一样，当 JCshell 进程准备好接受输入时，它会显示一个提示，并等待用户输入。提示符应包括 JCshell 的进程 ID。

  ## JCshell [60] ##

  在接受用户的一行输入后，它会解析输入行以提取命令名称和相关参数列表，然后创建子进程来执行命令。我们可以假设命令行的上限为 1024 个字符，最多 30 个字符串（包括 | 符号）。

  当子进程正在运行时，JCshell 应等待子进程结束，然后再显示接受用户下一步输入的提示。

- JCshell 进程应该能够定位并执行任何程序，这些程序可以通过

  -命令行中指定的绝对路径，例如

  ## JCshell [65] ### /home/tmchan/a.out

  -命令行中指定的相对路径，例如

  ## JCshell [65] ### ./a.out

  -搜索环境变量 $PATH 中列出的目录，例如

  ## JCshell [65] ## gcc

  其中 *gcc* 位于 /usr/bin 目录中，而
  $PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

  请参阅编程实验室 1，了解如何定位和执行有效命令或程序。

- 如果目标程序无法定位或执行，JCshell 会显示一条错误信息：

  ## JCshell [65] ### simp
  JCshell: 'simp'： 没有此类文件或目录

  ## JCshell [65] ### /bin/simp
  JCshell： '/bin/simp'： 没有此类文件或目录

- **|** 符号 - 如果每个输入行的命令之间出现 |（管道）符号，JCshell 会尝试创建多个子进程，并将 **|**（管道）之前命令的标准输出 "连接 "到 |**（管道）**之后命令的标准输入。也就是说，每条命令（除了

第一条）读取前一条命令的输出。我们可以假设，用户在输入命令行中输入的管道不会超过 4 个（即 5 条命令）。例如，当用户输入

```
## JCshell [261] ###  cat cpu-mechanisms.txt | grep trap | wc -w
456
```

我们可以认为，将 | 符号作为输入行的第一个或最后一个字符是不正确的。此外，如果两个 | 符号之间没有命令，也是不正确的。例如

```
## JCshell [261] ###   cat cpu-mechanisms.txt | | grep trap JCshell:
should not have two | symbols without in-between command

## JCshell [261] ###   cat cpu-mechanisms.txt || grep trap JCshell:
不应该有两个没有中间命令的 | 符号
```

请参阅编程实验室 2，学习如何使用 *pipe()* 和 *dup2()* 系统函数在两个进程间建立管道。

- JCshell 在打印所有已终止子进程的运行统计数据之前，总是会等待所有命令完成。例如，当用户运行 ***ls*** 命令时：

```
## JCshell [60] ## ls
JCshell  JCshell.c  Makefile  loop.c  loopever  loopever.c  segfault
segfault.c  tloop

(PID)61 (CMD)ls ( STATE)Z ( EXCODE)0 ( PPID)60 (USER)0.00 ( SYS)0.00 ( VCTX)8
(NVCTX)0
```

进程的所有统计信息都可以在 /proc/{process id} 目录。**本作业要求利用 /proc 文件系统提取进程的运行统计信息。**请参阅编程实验 1，学习如何从 proc 文件系统中提取进程统计信息。此外，有关 /proc 文件系统的更多详细介绍，请访问 https://man7.org/linux/man-pages/man5/proc.5.html。

以下是输出信息的摘要。

PID	被终止进程的进程 ID
CMD	命令的文件名--从 /proc/{pid}/stat 或 /proc/{pid}/status 获取
国家	进程状态（对于我们的应用程序来说，它始终处于 Zombie (Z) 状态）--从以下地址获取 /proc/{pid}/stat 或 /proc/{pid}/status
EXCODE	已终止进程的退出代码--可从 /proc/{pid}/stat 或 waitpid()
EXSIG	导致进程终止的信号名称
PPID	进程的父进程 ID - 从 /proc/{pid}/stat 或 /proc/{pid}/status 获取
用户	进程处于用户模式的时间（以秒为单位）--从 /proc/{pid}/stat
系统	进程处于内核模式的时间（以秒为单位）--从 /proc/{pid}/stat
VCTX	进程所经历的自愿上下文切换次数--从以下数据中获取 /proc/{pid}/status
NVCTX	进程经历的非自愿上下文切换次数 - 获得 从 /proc/{pid}/status

当进程因接收到信号而终止时，它应反映在输出中。

JCshell [261] ### ./segfault

(PID)309 (CMD)segfault (STATE)Z (EXSIG)Segmentation fault (PPID)261 (USER)0.30 (SYS)0.00 (VCTX)15 (NVCTX)1

如果进程被信号终止，JCshell 应显示导致终止的信号类型。

例如，如果检测到 SIGINT，则打印 "中断"；如果检测到 SIGKILL，则打印 "杀死"。

请参阅编程实验室 2，了解如何处理信号和显示适当的信息。

执行作业（包含多个命令）时，JCshell 应等待所有进程结束后再显示统计数据。输出的顺序应反映这些子进程的终止顺序。下面是一个用两条管道连接三条命令的示例：

JCshell [261] ### cat cpu-mechanisms.txt | grep trap | wc -w
456
(PID)305 (CMD)cat (STATE)Z (EXCODE)0 (PPID)261 (USER)0.00 (SYS)0.00 (VCTX)11 (NVCTX)0
(PID)306 (CMD)grep (STATE)Z (EXCODE)0 (PPID)261 (USER)0.00 (SYS)0.00 (VCTX)4 (NVCTX)0
(PID)307 (CMD)wc (STATE)Z (EXCODE)0 (PPID)261 (USER)0.00 (SYS)0.00 (VCTX)4 (NVCTX)0

下面是另一个示例，显示了一些流程出现错误的情况。

JCshell [261] ### ./tloop 10 | ./tloop 5 | ./tloop 2 运
行时间：2 秒
计划终止。
(PID)330 (CMD)tloop (STATE)Z (EXCODE)2 (PPID)261 (USER)2.00 (SYS)0.00 (VCTX)13 (NVCTX)2
(PID)329 (CMD)tloop (STATE)Z (EXSIG)Broken pipe (PPID)261 (USER)5.00 (SYS)0.00 (VCTX)11 (NVCTX)2
(PID)328 (CMD)tloop (STATE)Z (EXSIG)Broken pipe (PPID)261 (USER)10.00 (SYS)0.00 (VCTX)14 (NVCTX)9

- JCshell 进程应能处理 SIGINT 和 SIGUSR1 信号。应执行相应的信号处理程序。

  -SIGINT信号：JCshell 进程及其子进程必须根据以下准则对 SIGINT 信号（由按下 Ctrl-c 或 kill 命令产生）作出响应：

  JCshell 进程不应被 SIGINT 终止。在 JCshell 进程等待用户输入时，如果用户按下 Ctrl-c，JCshell 应作出反应，显示新的提示。

  ## JCshell [60] ## ^C
  ## JCshell [60] ## ^C
  ## JCshell [60] ## ^C
  ## JCshell [60] ##

  当用户在 JCshell 进程等待其子进程终止时按下 Ctrl-c 时，JCshell 进程不应终止，而子进程应根据子命令的预定义行为响应 SIGINT 信号。例如，如果命令被设置为无需终止即可处

理 SIGINT，则进程应继续运行；否则，进程应终止。

```
## JCshell [112] ## ./forever
^收到 SIGINT！ ！ 忽略它:)接收 SIGINT!
忽略它）
^收到 SIGINT！ ！ 忽略它:)
^C

## JCshell [112] ## ./tloop 10
^C
(PID)113 (CMD)tloop ( STATE)Z ( EXSIG)Interrupt (PPID)112 (USER)3.01 (SYS)0.00
(VCTX)16 (NVCTX)2
```

-SIGUSR1 信号：该信号可供用户定义自己的活动或事件。使用 JCshell 时，所有子进程不会在创建后立即运行目标命令。我们希望它们在执行命令前等待 JCshell 发送的信号（SIGUSR1）。这种机制可确保在执行子进程之前，JCshell 用于管理进程的所有控制结构都已更新。

- 内置命令：exit - 如果用户输入 ***exit*** 命令，JCshell 将终止，并重新显示标准 shell 提示。例如，如果用户输入

```
## JCshell [76] ## exit JCshell
： 已终止
root@ff860ab794d9:/home/c3230# ps
  PID  TTY          STAT     时间 命令
    1  pts/0        Ss       0:00 敲击
   79  pts/0        R+       0:00 ps
root@ff860ab794d9:/home/c3230#
```

如果***退出***命令有其他参数，JCshell 不会将其视为有效请求，也不会终止。此外，如果***退出命令不***作为命令行的第一个单词出现，JCshell 也不会将其视为***退出***命令。

```
## JCshell [96] ###  不退出
JCshell: 'not'： No such file or
directory ## JCshell [96] ### 现在退出
JCshell： "使用其他参数 "退出
```

资源

为您提供的文件如下。

- `utility.zip` - 该文件包含三个 C 语言文件（forever.c、tloop.c 和 segfault.c），可用于测试 JCshell 程序。

文件

1. 在提交的源代码的头部，明确说明

- 学生姓名和编号
- 开发平台：
- 备注 - 说明您完成了多少工作

2. 内联注释（尽量详细，以便他人轻松理解您的代码）

使用的计算机平台

在这项作业中，您需要在 workbench2 Linux 平台上开发和测试您的程序。程序必须用 C 语言编写，并用 gcc 成功编译。最好在自己的机器（WSL2 或 Ubuntu docker 镜像）上开发和测试程序。在本地完成测试后，将程序上传到 workbench2 服务器进行最终测试。

提交

将您的程序提交到课程 Moodle 网站上的 Programming # One 提交页面。用以下格式命名您的程序 JCshell_StudentNumber.c（将 StudentNumber 替换为您的香港大学学号）。由于 Moodle 网站可能拒绝源代码提交，请将程序压缩为 zip 或 tgz 格式后再上传。

评分标准

1. 您提交的程序将在 workbench2 上进行测试。请确保您的程序在编译时*没有任何错误*。否则，我们将无法测试您提交的程序，您将得到零分。
2. 由于导师将检查您的源代码，因此请在编写程序时保持良好的可读性（即使用良好的代码规范和足够的注释），以免因可能出现的混淆而丢分。

文件	• 包括必要的注释，以清楚说明程序的逻辑（如不包括注释，得 -1 分）。 • 包括计划开始时所需的学生信息（-0.5 如果缺少则扣分）。	
程序的正确性（11 分）	创建和执行流程--（3 分）	• 应能打印 "## JCshell ## "并接受用户输入 • 应能执行用户输入的命令 • 可以定位并执行全路径、相对路径或标准 PATH 下的命令 • 可执行包含任意数量参数的命令 • 能正确处理出错情况，如文件名错误、路径错误等。 • 在接受下一条命令前，应等待命令执行完毕 • 应接受下一条执行命令 • 应删除所有僵尸进程
	流程的创建和执行--"|" 的使用	• 正确使用 | 符号；可报告 | 符号的不当使用 • 可执行多条命令（最多五条），任意管道连接的参数数

	(3分)	
	打印进程的运行统计数据（2.5 分）	• 系统能以正确的格式打印进程的运行统计数据 • 命令/任务完成后，系统应打印信息 • 对于包含多个命令的作业，系统应根据终止顺序输出统计信息 • 程序应从 /proc 文件系统中获取数据（如果没有，-1 分）。

	信号的使用 （2 分）	SIGINT 信号（1 分） • JCshell 进程和子进程处理 SIGINT 信号的正确行为 SIGUSR1 信号（1 分） • 在执行目标命令之前，所有子进程都应等待 SIGUSR1 信号。
	内置命令：退出 (0.5分)	• 正确使用*退出*命令；可报告不当使用情况

剽窃

抄袭是一种非常严重的违法行为。学生应了解什么是抄袭、抄袭的后果以及如何避免抄袭。**请注意，我们可能会要求您向我们解释您的程序是如何运行的，我们也可能会使用软件工具来检测软件抄袭行为。**