卡通画

描述已自动生成

**操作系统实验报告**

**实验题目**  进程的状态与进程调度

**学生姓名**  侯腾跃

**学 号**  2022217477

**专业班级**  计科22-4班

**指导教师**  田卫东

**完成日期**  2024.11.25

**合肥工业大学 计算机与信息学院**

1. **实验目的和任务要求**

调试进程在各种状态间的转换过程，熟悉进程的状态和转换。

 通过对进程运行轨迹的跟踪来形象化进程的状态和调度。

 掌握Linux下的多进程编程技术。

1. **实验原理**

该实验内容包括使用Linux 0.11应用程序调用fork函数创建多个子进程，并编写一个可以创建多个子进程的Linux 0.11应用程序；通过数据可视化工具观察父进程与子进程的运行轨迹。

1. **实验内容**

**在Linux 0.11应用程序中调用fork函数创建多个子进程**

**编写一个可以创建多个子进程的Linux 0.11应用程序**

请读者按照下面的步骤使用fork函数编写一个可以创建多个子进程的Linux 0.11应用程序，一方面可以学习Linux的多进程编程技术，另一方面，通过观察多个进程的运行过程，可以对进程的调度过程有一个初步认识。

1. 使用VSCode打开之前克隆到本地的Linux 0.11应用程序项目。

2. 打开linuxapp.c文件main函数，让父进程新建三个子进程，并分别输出子进程id及其父进程id。代码如下：

int main( int argc, char \* argv[] )

{

if( 0 == fork() )

{

printf("child process pid=%d ppid=%d line=%d\n",

getpid(), getppid(), \_\_LINE\_\_);

}

else if( 0 == fork() )

{

printf("child process pid=%d ppid=%d line=%d\n",

getpid(), getppid(), \_\_LINE\_\_);

}

else if( 0 == fork() )

{

printf("child process pid=%d ppid=%d line=%d\n",

getpid(), getppid(), \_\_LINE\_\_);

}

else

{

wait( NULL );

printf("parent process pid=%d ppid=%d\n",getpid(), getppid());

}

return 0;

}

其中，getppid函数可以在子进程中获取父进程的id。“\_\_LINE\_\_”是GCC编译器提供的一个预定义宏，它的值是其在源代码文件中的行号。

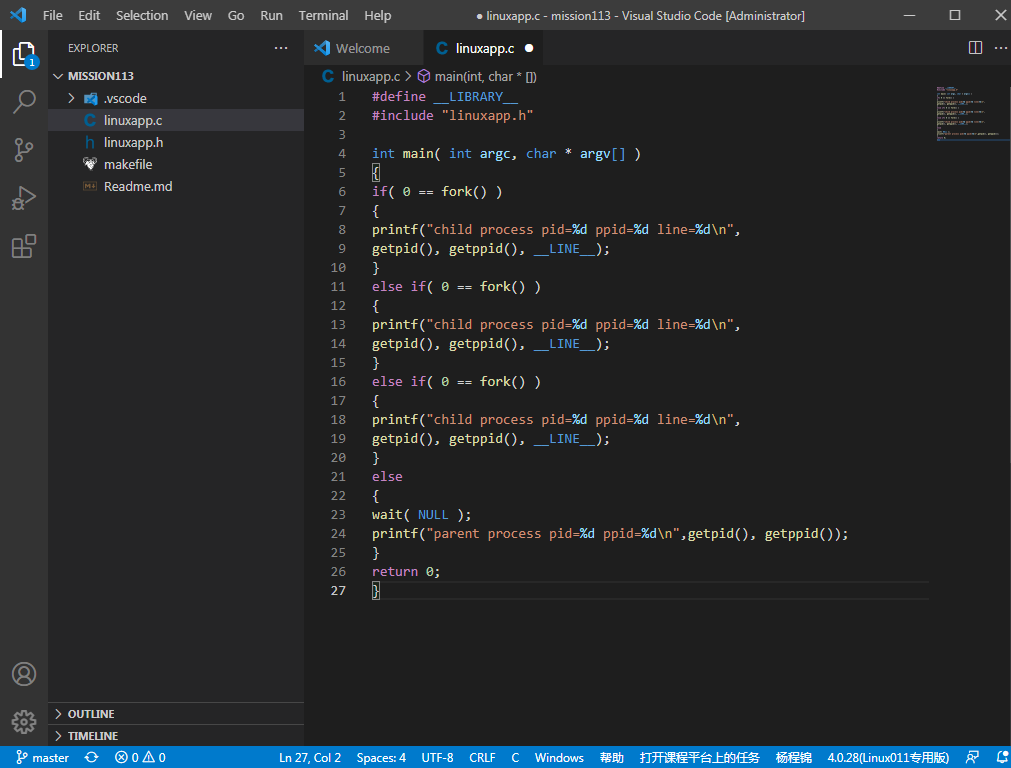


图 1 linuxapp.c文件

3. 不能在VSCode中按F5启动调试Linux 0.11应用程序。只能在VSCode的“Terminal”菜单中选择“Run Build Task...”，会在VSCode的顶部中间位置弹出一个可以执行的Task列表，选择其中的“Bochs 运行(不调试)”。

4. 待Linux 0.11启动完毕后，将生成的可执行文件从软盘B拷贝到硬盘，命令如下：

mcopy b:linuxapp.exe app

5. 为app文件添加可执行权限，命令如下：

chmod +x app

6. 执行“sync”命令确保app文件写入硬盘。

7. 使用命令./app运行可执行文件app，观察各个进程开始执行的顺序和结束执行的顺序，理解进程在其生命周期中状态的转换过程和进程调度过程。

文本

描述已自动生成

图 2 运行app可执行文件

8. 结束调试，关闭Bochs虚拟机。

**查看多个子进程的运行轨迹**

在前面的练习中，读者可以通过父进程和子进程在屏幕上打印输出的信息来间接的判断出父进程和子进程运行的轨迹，但是并不直观，得到的信息也不全面。接下来，读者可以使用数据可视化窗口，更加直观、形象的观察到父进程与子进程的运行轨迹。步骤如下：

1. 使用VSCode的“File”菜单中的“Open Folder”打开之前克隆到本地的Linux 0.11内核项目文件夹。

2. 在VSCode的“Terminal”菜单中选择“Run Build Task...”，会在VSCode的顶部中间位置弹出一个可以执行的Task列表，选择其中的“生成项目”。

3. 待Linux 0.11内核项目生成成功后，使用Windows资源管理器分别打开之前克隆到本地的Linux 0.11应用程序项目和Linux 0.11内核项目所在的文件夹。

4. 用Linux 0.11应用程序项目文件夹中的硬盘镜像文件harddisk.img覆盖Linux 0.11内核项目文件夹中的harddisk.img文件，这样就可以在Linux 0.11内核项目的硬盘中使用之前生成的app文件了。

5. 使用VSCode的“File”菜单中的“Open Folder”打开之前克隆到本地的Linux 0.11内核项目文件夹。

6. 为了方便观察app应用程序中的父进程和子进程的运行轨迹，需要在父进程结束的位置添加一个条件断点。请读者在kernel/exit.c文件的第166行（进程结束后触发进程调度的位置）添加一个条件断点，条件为“current->pid == 6”。

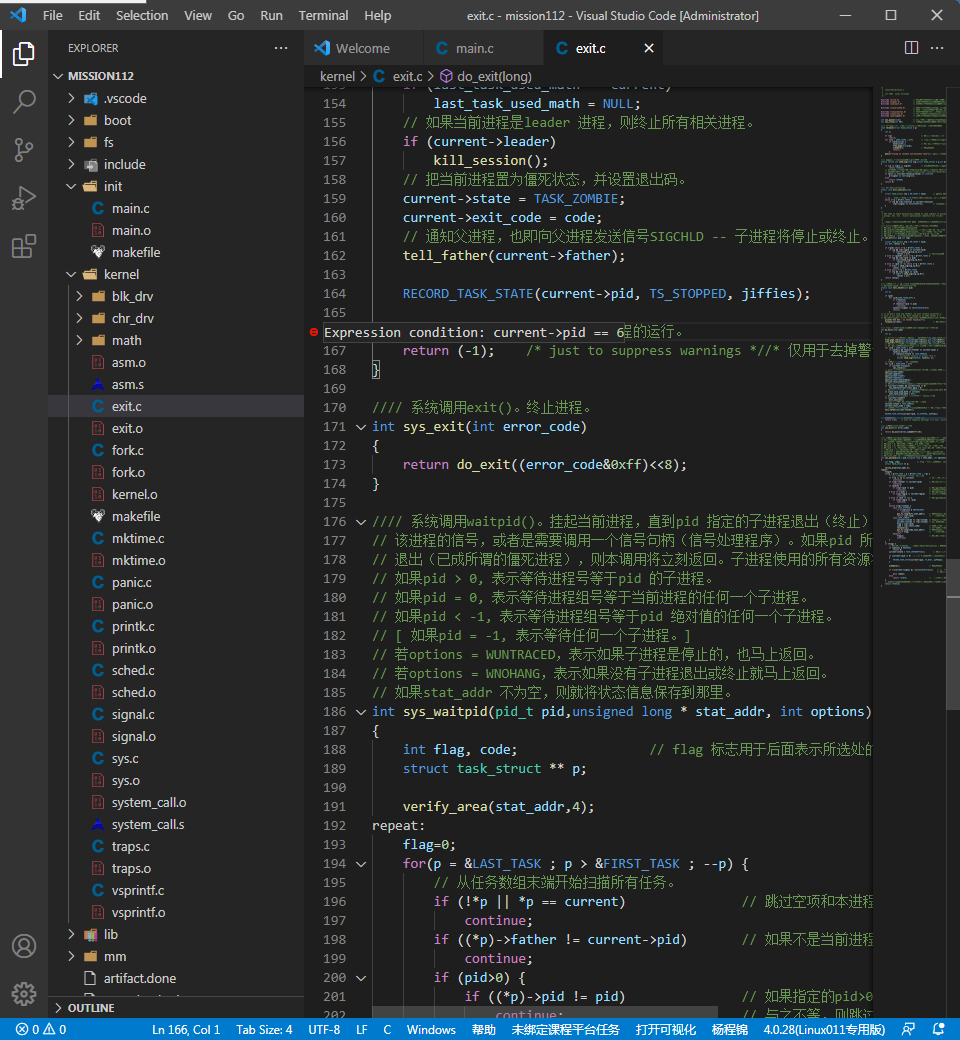


图 3 添加条件断点

7. 按F5启动调试。输入命令./app后，会命中刚刚添加的条件断点。

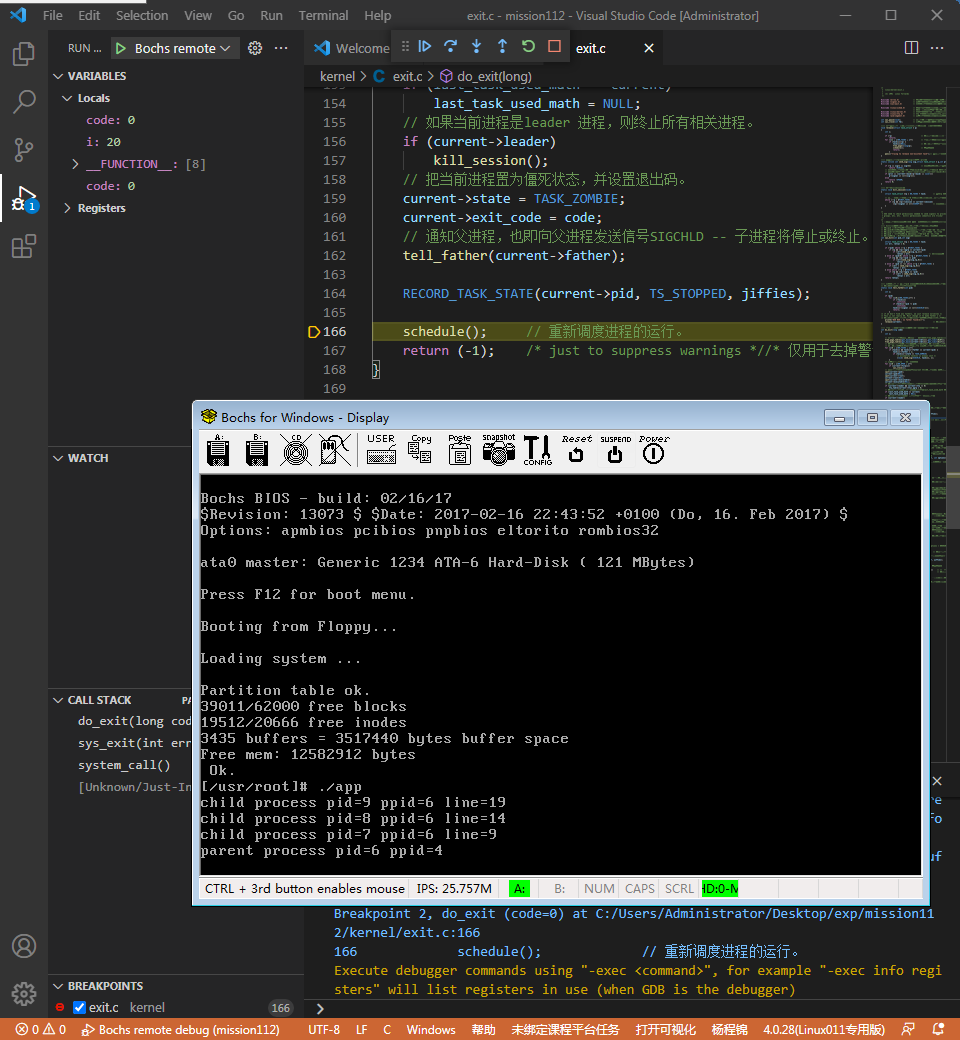


图 4 调试并命中断点

8. 在VSCode的“View”菜单中选择“Command Palette...”，会在VSCode的顶部中间位置显示命令面板，输入“Lab: New Visualizer View”命令后，VSCode会在其右侧弹出一个窗口让读者查看可视化视图。在右侧可视化视图顶部的编辑框中输入命令“#sched”后按回车（需要等待较长时间完成刷新），就可以查看进程运行的轨迹了，如5所示。

图表

描述已自动生成

图 5 进程运行轨迹

在进程运行轨迹图示的最左侧是jiffies，表示每一次进程状态转换发生

**3.2 自行编写代码记录进程的运行轨迹**

跟踪进程的运行轨迹，本质上就是跟踪并记录进程在其生命周期中的状态转换过程和调度过程。进程 在生命周期中状态的转换是由操作系统的调度程序实现的。所以要实现对进程运行轨迹的跟踪，不仅要对 进程在其生命周期的各个状态有一个全面的了解，而且还要对进程的调度有一个透彻的理解。

读者通过前面的练习应该已经对进程的调度过程有了一个初步的认识，然而“知而不行，非真知也”， 下面的练习需要读者自行编写代码将进程运行的轨迹记录在一个日志文件中，然后将日志文件中的结果与 图6-2进行比对，确保正确无误。

读者一次就能编写出正确的代码，记录下正确的轨迹，还是有一定难度的。如果出现了错误也没关系， 读者只需仔细研究一下调用宏函数RECORD\_TASK\_STATE的各处代码，应该能够解决遇到的各种问题。

在开始记录进程的运行轨迹之前，需要先在Linux 0.11内核中添加一个写日志文件的功能，然后使 用此功能将进程状态转换的日志信息写入一个文本文件。

**在系统初始化时打开日志文件process.log**

为了记录操作系统从启动到关机过程中所有进程的运行轨迹，需要在硬盘上维护一个日志文件 /var/process.log，然后添加一个写日志文件的功能，将进程状态转换的日志信息写入该日志文件。为了 能尽早开始记录日志，应当在内核初始化时就打开process.log文件。请读者按照下面的步骤编写源代码， 完成这部分内容。

首先，使用VSCode打开之前克隆到本地的Linux 0.11内核项目并定位到内核的入口函数，即 init/main.c文件中的start函数，其中从第181行开始的一段代码是：

……

move\_to\_user\_mode();

if( !fork() ) { /\*we count on this going ok\*/

init();

}

……

这段代码是在进程0中运行的，其作用是先切换到用户模式，然后第一次调用fork函数创建子进程1， 子进程1接着调用了本文件中的init函数继续进行操作系统的初始化工作。

在init函数中从第227行开始的一段代码是：

……

setup((void \*) &drive\_info);

(void) open("/dev/tty0", O\_RDWR, 0);

(void) dup(0);

(void) dup(0);

……

这段代码在进程1中建立了文件描述符0、1和2，它们分别是stdin、stdout和stderr，即标准输 入、标准输出和标准错误。读者需要在这里添加一行代码把process.log文件关联到文件描述符3。修改 后的init函数如下：

setup((void \*) &drive\_info);

(void) open("/dev/tty0", O\_RDWR, 0);

(void) dup(0); (void) dup(0);

(void) open("/var/process.log", O\_CREAT | O\_TRUNC | O\_WRONLY, 0666);

添加了一行调用open函数打开process.log文件的代码。其中，第二个参数的含义是建立只写文件， 并且如果文件已存在则清空已有内容。第三个参数设置文件权限为所有人可读可写。

这样，文件描述符0、1、2和3就在进程1中建立了，并在进程1的进程控制块中进行了记录。根据 在前面实验中学习的fork函数的原理，fork函数通过将父进程的进程控制块完整复制给子进程，从而允 许子进程从父进程那里继承所有的资源，当然也包括父进程拥有的文件描述符（即父进程打开的文件）。 所以，由进程1创建的子进程就会从进程1中继承这四个文件描述符，而且，由于此后所有的进程都是进 程1的子进程（包括由应用程序创建的进程），当然也都会继承这四个文件描述符了。

**编写fprintk函数用于向process.log文件写入数据**

文件process.log 将会被用来记录进程运行的轨迹，但是，所有进程的状态转换工作都是在内核状态 下进行的，此时write系统调用函数失效（其原理等同于不能在内核状态下调用printf函数，而只能调 用printk 函 数） 。 这就需要在Linux 0.11的内核中编写一个fprintk函数用于向process.log文件写入 数据。编写fprintk函数的难度较大，所以这里直接给出源代码，主要是参考了printk函数和sys\_write 函数而写成的。需要注意在else分支中开始部分的判断条件，首先判断进程控制块表task中下标为4的 项是有效的，这样就避免了记录操作系统初始化进程的运行轨迹，缩小了日志文件的大小；第二个条件语 句是从进程1中获取到process.log文件的描述符，并确保其有效。

**记录进程的运行轨迹**

到现在为止，读者已经让Linux 0.11内核在初始化的时候打开了process.log文件，并解决了向 process.log 文件写入日志信息的问题。接下来需要解决在什么时刻将进程的运行轨迹信息写入 process.log 文件的问题。这就要求读者对进程的状态转换以及进程调度有一个全面的了解。

Linux 0.11 支持四种经典的进程状态的转换过程： 

就绪到运行：通过schedule函数（在文件kernel/sched.c的第190行）完成。 

运行到就绪：通过schedule函数完成。 

运行到睡眠：通过sleep\_on函数（在文件kernel/sched.c的第294行）和 interruptible\_sleep\_on 函数（在文件 kernel/sched.c 的第 328 行）完成。或者通过进程主动 睡眠的系统调用内核函数sys\_pause（在文件kernel/sched.c的第278行）和sys\_waitpid（在 文件kernel/exit.c 的第186行）完成。 

睡眠到就绪：通过wake\_up函数（在文件kernel/sched.c的第367行）完成。 此外还有进程的创建和退出两种情况： 

进程的创建：通过copy\_process函数（在文件kernel/fork.c的第89行）完成。 

进程的退出：通过do\_exit函数（在文件kernel/exit.c的第122行）完成。

所以，只要在以上提到的这些函数的适当位置调用fprintk函数输出日志信息到process.log文件， 就能完成进程轨迹的全面跟踪了。 在调用fprintk函数前，还需要先定义process.log文件中每行日志的格式为： pid state time

读者可以在Linux 0.11内核项目中，使用VSCode提供的“查找”功能，以内核源代码中定义的所有 进程状态名称为关键字进行搜索，就可以找到进程状态发生改变的所有源代码。读者也可以按照下面的步 骤逐步添加调用fprintk函数的语句：

1.为了跟踪进程的创建，可以修改kernel/fork.c文件中的copy\_process函数。在第114行后面 增加语句（一定在子进程控制块的pid和start\_time被赋值之后）: p->start\_time = jiffies; fprintk( 3, "%ld\t%c\t%ld\n", p->pid, 'N', jiffies ); 记录进程在创建后进入了就绪状态。在第168行后面增加语句： fprintk( 3, "%ld\t%c\t%ld\n", p->pid, 'J', jiffies );

2. 当进程的状态被设置为TASK\_ZOMBIE，并且已经获取了退出码时，表示进程已经完成了退出操作， 虽然此时进程的控制块还没有被释放（要留待父进程获取子进程的信息）。子进程退出的最后一 步是发送信号通知父进程，目的是唤醒正在等待此事件的父进程，由父进程来释放子进程的进程 控制块。从时序上来说，应该是子进程先退出，父进程才被唤醒。所以，为了跟踪进程的退出， 可以修改kernel/exit.c文件中的do\_exit函数。在第159行将进程状态设置为TASK\_ZOMBIE的后面插入一条语句，如下： current->state = TASK\_ZOMBIE; fprintk( 3, "%ld\t%c\t%ld\n", current->pid, 'E', jiffies );

3. 接下来，修改kernel/sched.c文件中的schedule函数，跟踪进程进入就绪状态或运行状态。注 意，schedule 函数找到的next进程是接下来将要运行的进程。如果next恰好是当前正处于运行 态的进程，switch\_to(next)也会被调用，这种情况相当于当前进程的状态没有改变。在第213 行进程由于收到信号而进入就绪状态，所以需要在此行的后面添加一条语句： fprintk( 3, "%ld\t%c\t%ld\n", (\*p)->pid, 'J', jiffies );

4. 在第271行switch\_to(next)语句前按照下面的要求添加代码。在第251行if判断语句中增加下 面的语句，用于记录当前进程从运行态进入就绪态的情况，因为此时调度算法选择的next进程 与处于运行态的当前进程是不同的，所以当前进程要进入就绪态等待下次调度，而next进程变 为运行态获得处理器： fprintk(3, "%ld\t%c\t%ld\n", current->pid, 'J', jiffies); 在第262行后面增加下面的语句，用于记录进程0（空闲进程）恢复为就绪态的情况，因为当空 闲时是进程0在运行，而此时调度算法选择的next进程不是进程0，说明有新的进程可以运行了， 进程0就需要让出处理器，从而进入就绪状态： fprintk(3, "%ld\t%c\t%ld\n", current->pid, 'J', jiffies); 在第268行后面增加下面的语句，用于记录调度算法选择的next进程进入运行态的情况： fprintk(3, "%ld\t%c\t%ld\n", task[next]->pid, 'R', jiffies);

5. 另外，只要操作系统处于空闲状态，就会让进程0运行，进程0会不停的调用sys\_pause函数， 以激活调度程序，使得调度程序可以随时选择处于就绪态的进程来运行。此时，可以认为进程0 处于等待状态（等待有其它可运行的进程），也可以认为进程0处于运行态，因为它是唯一在CPU 上运行的进程，只不过运行的效果是等待。所以，在sys\_pause函数中记录进程进入阻塞状态的 情况时，应该将进程0排除在外，否则就会记录大量进程0进入阻塞状态（不可中断的等待状态） 的情况，为后续分析日志中的数据代理很大困难。所以，在sys\_pause函数的第288行的后面添 加的跟踪语句应该在if条件语句中： if(current->pid != 0) fprintk( 3, "%ld\t%c\t%ld\n", current->pid, 'W', jiffies );

6. 注意，当进程被唤醒时虽然会将进程设置为TASK\_RUNNING状态，但是进程实际是进入了就绪状 态，而并没有立即开始运行，所以此时在日志记录中应使用标志‘J’。只有在schedule函数的 最后，被选中为next的进程才会进入实际的运行状态，才能在日志记录中使用标志‘R’。也 就 是说，在读者添加的所有调用fprintk函数的语句中，只有在schedule函数的最后这一个地方 使用‘R’标志。余下的跟踪语句请读者在适当的位置自行添加。这里可以给出一点提示，包括 前面提到的几处，总共需要在15处调用fprintk函数（不包含fprintk函数的定义和声明）， 读 者可以将“fprintk”作为“查找”功能的关键字进行查找，确保数量和位置正确。

待读者添加完所有的跟踪语句之后，需要在Linux 0.11中运行在3.1中构建的应用程序app，来记录 进程调度的日志。按照下面的步骤完成实验：

1.使用Windows资源管理器打开3.1中克隆到本地的Linux 0.11应用程序项目的文件夹，复制其 中的harddisk.img 硬盘镜像文件。

2.使用Windows资源管理器打开3.2中克隆到本地的Linux 0.11内核项目的文件夹，将上一步复 制的harddisk.img 文件粘贴覆盖同名的文件。这样就将之前构建的应用程序app的可执行文件 通过硬盘复制到Linux 0.11内核项目中了。

3. 使用VSCode打开Linux 0.11内核项目，按F5启动调试。

4. 待Linux 0.11 启动完毕后，在终端输入命令./app执行app可执行文件，然后在纸上记录下打印 输出的父进程和子进程的id，以便下面分析数据时使用。

5. 在Linux 0.11 启动和app运行的过程中，就会将进程调度信息写入日志文件process.log中。 使用下面的命令将日志文件复制到软盘B： mcopy /var/process.log b:log.txt

6. 结束此次调试，关闭Bochs虚拟机。

7. 使用软盘编辑器工具打开floppyb.img文件，将软盘 B中的log.txt文件复制到Windows本地目 录中。

8. 打开log.txt文件（可以将log.txt文件拖动到VSCode窗口中释放）。要求读者在日志文件中能 够找到类似于下面日志的app应用程序的运行轨迹信息，并且应该与图6-2中显示的进程运行轨 迹一致。其中，4是Shell程序的进程id，6是app的主进程id，7,8和9是三个子进程id。结 合应用程序app的源代码及其运行时打印输出的信息分析这些日志，理解进程在生命周期中状态 的转换过程和进程调度过程。

…… 4 R 116

6 N 1386

6 J 1387

4 W 1388

6 R 1388

6 W 1388

6 J 1388

6 R 1388

0 J 1388

6 R 1388

6 W 1389

6 J 1389

7 N 1393

7 J 1394

8 N 1394

8 J 1395

9 N 1395

9 J 1396

6 W 1396

9 R 1396

9 E 1397

6 J 1397

8 R 1397

8 E 1398

7 R 1399

7 E 1400

6 R 1400

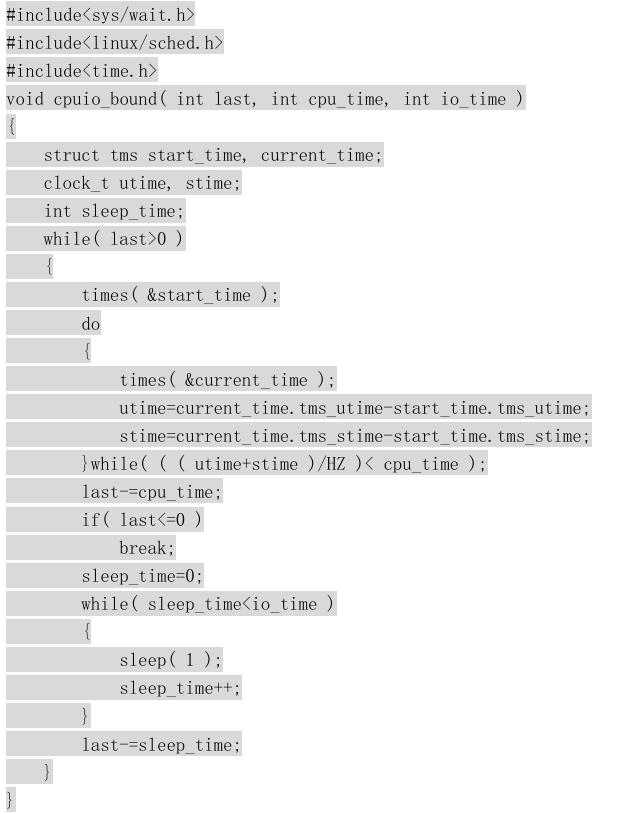
6 E 1401

……

**对进程调度过程进行量化分析**

在3.1中编写的应用程序app中还无法控制各个子进程运行的时间，特别是无法控制子进程实际占用 CPU 的时间和等待I/O操作的时间，也就无法对进程的调度过程进行量化分析。如果在内核中对各种设备 的I/O操作时间进行统计，并在应用程序中对各种I/O设备进行实际访问也不现实。所以，接下来采用在 应用程序中模拟这样一种折中的方式，从而实现对进程调度过程的量化分析。

使用VSCode打开在3.1中克隆到本地的Linux 0.11应用程序。首先在linuxapp.c文件中的main函 数的前面添加一个新函数cpuio\_bound，用来模拟进程在生命周期中占用CPU（运行态）与I/O操作（阻 塞态）的情景，代码为:



参数last表示占用CPU以及I/O操作的总时间，不包括在就绪队列中的时间。参数cpu\_time表示一 次连续占用CPU的时间，必须大于等于0。参数 io\_time表示一次I/O操作占用的时间，必须大于等于0。 如果last > cpu\_time + io\_time,则往复多次占用 CPU 和I/O 操作，且时间单位均为秒。其中struct tms 在include/sys/times.h 中定义，clock\_t 在 include/time.h 中定义。

按照下面步骤操作，得到新的应用程序:

1.不能在VSCode中按F5启动调试Linux 0.11应用程序。只能在VSCode的“Terminal”菜单中选 择“Run Build Task...”，会在 VSCode 的顶部中间位置弹出一个可以执行的Task列表，选择其 中的“Bochs 运行(不调试)”。

2. 待Linux 0.11 启动完成后，在终端输入下面的命令将可执行文件从软盘B拷贝到硬盘 mcopy b:linuxapp.exe app

3. 为app文件添加可执行权限，命令如下： chmod +x app

4. 执行“sync”命令将文件保存到硬盘。

5. 使用命令./app运行app应用程序，确保应用程序可以正常运行。

6. 关闭Bochs虚拟机。

按照下面的步骤得到新的日志文件：

1.使用Linux 0.11应用程序项目文件夹中的硬盘镜像文件harddisk.img覆盖Linux 0.11内核项 目文件夹中的harddisk.img文件，这样就可以在内核项目中使用新生成的app可执行文件了。

2. 使用VSCode打开Linux 0.11内核项目，按F5启动调试。

3. 待Linux 0.11 启动完成后，使用命令./app运行app应用程序，然后在纸上记录下打印输出的父 进程和子进程的id，以便下面分析数据时使用。

4. 将process.log 文件复制到软盘B，命令如下： mcopy /var/process.log b:log.txt

5. 结束调试，关闭Bochs虚拟机。

6. 使用软盘编辑器工具打开floppyb.img文件，将软盘B中的log.txt文件复制到Windows本地目 录中。

7. 打开log.txt文件（可以将log.txt文件拖动到VSCode窗口中释放）。查看日志文件中记录的信 息，根据之前记录的进程id找出app应用程序运行时产生的进程调度轨迹信息。

**调度算法、时间片和优先级**

根据Linux 0.11 的进程调度函数schedule的代码（在文件kernel/sched.c的第190行 ），可知 Linux 0.11 的调度算法是选取counter（时间片）最大的就绪进程占用CPU。处于运行态的进程（即current指 针指向的进程）的counter每当发生时钟中断时就会减1（通过文件kernel/sched.c的do\_timer函数第 550 行if ((--current->counter)>0) return;这行语句完成），所以是一种比较典型的时间片轮转调度算 法。

另外，由schedule函数可以看出，当没有counter大于0的就绪进程时，要对所有进程做 “(\*p)->counter = ((\*p)->counter >> 1) +(\*p)->priority;”操作（kernel/sched.c 第 241 行），效果 是对所有的进程（包括阻塞进程）进行counter的衰减（除2），并累加priority的值，这样对已经阻塞 的进程来说，一个进程在阻塞队列中停留的时间越长，其优先级越大，被分配的时间片就越大（不会大于 优先级的2倍），而对于时间片已经为0的进程来说，其时间片的值会被重置为其优先级的值。所以总的 来说，Linux 0.11 的进程调度是一种综合考虑进程优先级，并能动态反馈调整时间片的轮转调度算法。

1. **总结和感想体会**

这份报告展示了我在进程状态与进程调度方面的实验学习。该实验内容包括使用Linux 0.11应用程序调用fork函数创建多个子进程，并编写一个可以创建多个子进程的Linux 0.11应用程序。该过程包括使用VSCode打开项目，编写代码以创建多个子进程，并通过一系列的步骤来运行和观察这些进程。我记录了每个实验步骤，包括代码编写、程序构建、调试过程，以及如何通过数据可视化工具观察父进程与子进程的运行轨迹。报告中展示了对进程状态和调度的深入理解，以及对Linux多进程编程的实际应用。