《Linux应用程序开发实验报告》

题目: 进程间通信

班级:\_\_ \_\_ \_\_17计科3班 \_

学号: 10417307 \_\_\_

姓名: \_\_薛冬冬\_ \_\_\_

东南大学成贤学院计算机系

2020年 4月 13日

**进程间通信**

1. **实验目的**

1.掌握linux中进程通信机制；

2．掌握管道通信机制。

1. **实验内容**

1. 完成有关管道和信号等问题。

1. **实验具体要求**

**1、有关管道的问题**

（1）有关pipe()函数的问题：如果将waitpid(pc,NULL,0);注释掉，程序执行会发生什么结果，为什么？

（2）改写标准流管道程序standard\_pipe.c，将popen函数中的r改成w，再做相应调整使得该程序可以正确运行？

（3）管道、标准流管道、有名管道各自的特征是什么？它们之间的区别何在？

**答：**

1. 注释后结果：



waitpid(pc,NULL,0);的作用是查看子进程是否退出，阻塞父进程让子进程执行即收集子进程退出信息。

将waitpid(pc,NULL,0);这句注释后，父进程执行完后就会立刻执行exit(0);语句，即父进程结束进程。而此时的子进程由于休眠时间比父进程长，还未结束休眠，在休眠结束后在屏幕上打印：17 bytes read from the pipe is ' Pipe Test Program'（子进程从管道中读取到的内容）。

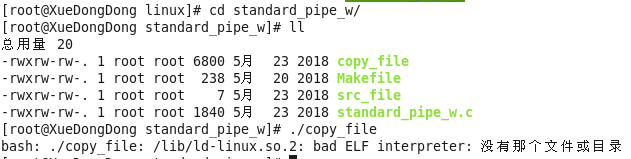
1. “w”：文件指针连接到command的标准输入，即该命令的结果产生输入。

cmd指向命令原本是ps -ef，可以结合第六周学习的内容，来进行解答。第六周有个实例是“从一个文件中读取最后10kb数据并到另一个文件中”，这个实例实现了将一个文件中的数据输入到另一个文件中去。因此我们可以将cmd指向命令改为./copy\_file。这样就可以实现在写端输入。

创建一个src\_file文件，其中放入数字：123456。如果管道建立运行成功，执行./standard\_pipe后应该会生成一个dest\_file文件，文件内容与src\_file一样。

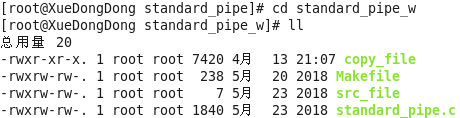
具体操作如下：

●准备

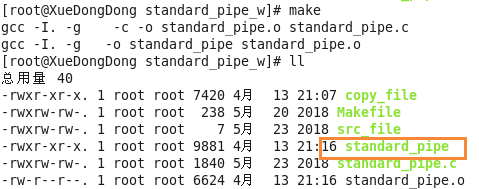


我直接复制了第六周老师给的copy\_file文件和src\_file文件（文件内容为123456），然后接着进行了以下操作。

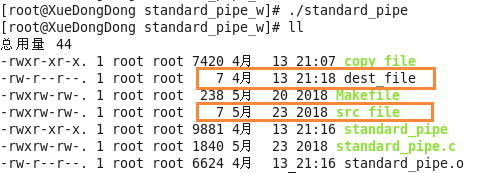
●查看文件夹文件



●使用make编译后

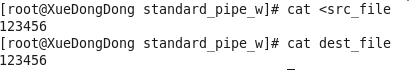


●运行./ standard\_pipe



./copy\_file将src\_file文件内容拷贝到dest\_file中。

●查看src\_file文件和dest\_file文件内容



由此可见程序得到了正确运行。

1. **特点：**

●管道

①它只能用于具有亲缘关系的进程之间的通信（也就是父子进程或者兄弟进程之间）。

②它是一个半双工的通信模式，具有固定的读端和写端。

③管道也可以看成是一种特殊的文件，对于它的读写也可以使用普通的 read()和 write()等函数。但是它不是普通的文件，并不属于其他任何文件系统，并且只存在于内核的内存空间中。

●标准流管道

①管道的操作基于文件流的模式。

②主要是用来创建一个连接到另一个进程的管道，这里的“另一个进程”也就是一个可以进行一定操作的可执行文件。

③标准流管道将一系列的创建过程合并到一个函数popen()中完成。

●有名管道

①它可以使互不相关的两个进程实现彼此通信。

②该管道可以通过路径名来指出，并且在文件系统中是可见的。在建立了管道之后，两个进程就可以把它当做普通文件一样进行读写操作，使用非常方便。

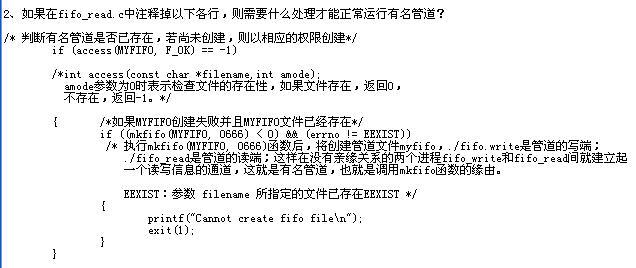
③FIFO严格地遵循先进先出规则，对管道及FIFO的读总是从开始处返回数据，对他们的写则把数据添加到末尾，它们不支持如seek()等文件定位操作。

**区别：**

管道、标准流管道都是只能用于具有亲缘关系的进程之间的通信，而有名管道可以使互不相关的两个进程实现彼此通信。

管道是半双工通信模式，具有固定的读端和写端，而有名管道在建立了管道之后，两个进程就可以把它当做普通文件一样进行读写操作。

管道可以看成是一种特殊的文件，对于它的读写也可以使用普通的 read()和 write()等函数，有名管道不支持如seek()等文件定位操作。



**答：**

该段代码的功能是判断有名管道是否存在，若尚未创建则以相应的权限创建。由源代码中注释得知有名管道文件位置为/tmp/myfifo，因此将上段代码注释即没有使用mkfifo()函数创建有名管道，也就是目录tmp下没有有名管道myfifo。

因此得出结论，只要在tmp目录下创建一个有名管道myfifo即可，创建有名管道可以使用“mknod 管道名 p”方法，本题即“mknod myfifo p”。

具体操作见下：

（1）注释后，在终端1中使用./fifo\_read执行程序报错。



（2）在终端2中进入tmp目录，创建管道文件myfifo





（3）再次进入终端1，执行./fifo\_read，处于阻塞状态一直等待



（4）在终端3中运行./fifo\_write 123



终端1中显示



由上图可见有名管道正常运行。

**3、有关信号的问题：**

（1）如果在int ret=alarm(5);上面再加一条语句休眠10秒，会出现什么结果？ 如果注释掉 pause();会打印底下的语句I have been waken up吗？为什么?

（2）信号是什么？信号集函数组处理信号的步骤是什么？

**答：**

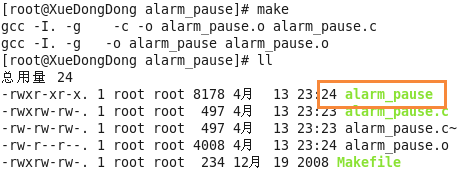
（1）

**■在int ret=alarm(5);上面再加一条语句休眠10秒，会出现什么结果？**

添加语句：



使用make：



结果：



最终运行结果和未加休眠语句的结果一样，都是：闹钟。

**■如果注释掉 pause();会打印底下的语句I have been waken up吗？为什么?**

会出现，见下图：



原因：pause();函数的作用是将进程暂停直至捕捉到信号为止，系统在5s后向进程发送SIGALRM信号，该信号默认动作为终止进程，因此程序执行不到打印消息就已结束。若将pause();函数删除，在执行完alarm(5)后，将会继续往下执行，定时器没有起任何作用。

**（2）信号是什么？信号集函数组处理信号的步骤是什么？**

**■**信号是 UNIX 中所使用的进程通信的一种最古老的方法。它是在软件层次上对中断机制的一种模拟，是一种异步通信方式。信号可以直接进行用户空间进程和内核进程之间的交互，内核进程也可以利用它来通知用户空间进程发生了哪些系统事件。它可以在任何时候发给某一进程，而无需知道该进程的状态。

**■**首先使用 sigprocmask()函数检测并更改信号屏蔽字（信号屏蔽字是用来指定当前被阻塞的一组信号，它们不会被进程接收）。

然后使用 sigaction()函数来定义进程接收到特定信号之后的行为。

检测信号是信号处理的后续步骤，因为被阻塞的信号不会传递给进程，所以这些信号就处于“未处理”状态（也就是进程不清楚它的存在）。

sigpending()函数允许进程检测“未处理”信号，并进一步决定对它们作何处理。

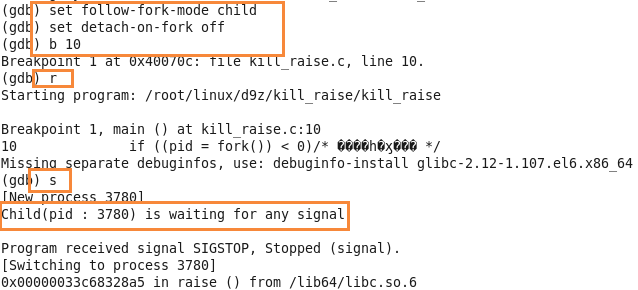


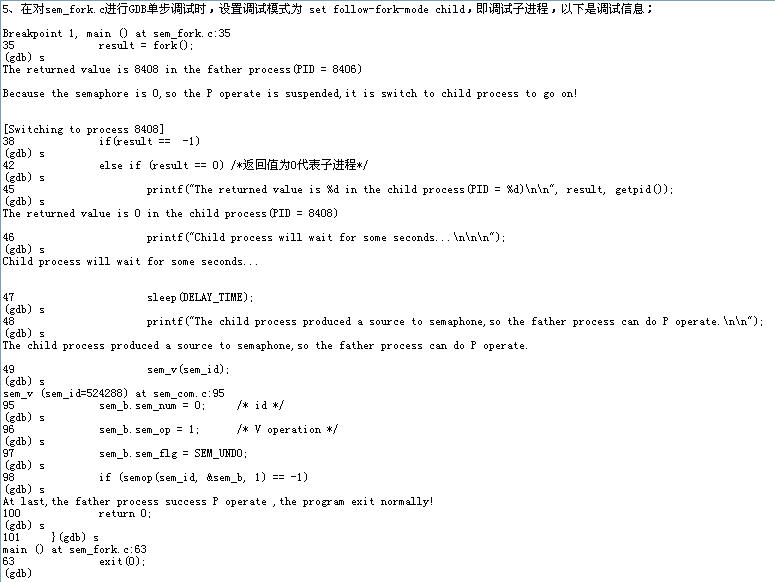
**答：**

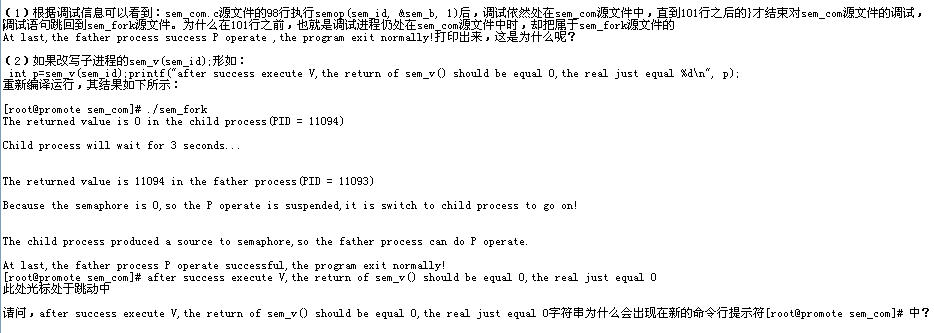
因为调试的时候是要父进程顺序执行，waitpid等不到子进程的信号，从而进入if语句内子进程强行被父进程关掉（杀死），然后调试的时候就会出错

~~使用follow-fork-mode时，只能调试一个进程，不能同时调试父子进程。因此需要设置set follow-fork-mode child和set detach-on-fork off，默认调试子进程一步一步执行，在子进程调试期间父进程没有任何执行。~~

具体操作如下：







**（3）如果调试方式换成系统默认调试，即调试父进程而子进程不受影响继续执行。**

**根据逻辑以及（2）sem\_v(sem\_id)的打印语句：printf("after success execute..."); 子进程首先执行完成，而信号量的值也由0变成1；在接下来调试父进程的过程中，调试环节会跳到sem\_com源程序的sem\_p函数部分，在给sem\_b结构体赋初值后，再键入s单步执行下面的语句if (semop(sem\_id, &sem\_b, 1) == -1)，此时GDB调试就处于暂停状态，再没有任何响应了。**

**根据逻辑，子进程先执行过后，信号量必定变成1，目前有资源可消耗，则父进程的P操作一定可以执行，但是在本次GDB调试中，父进程在执行P操作的semop()函数时，实验现象却显示被阻塞，请问父进程为什么不能执行P操作，也就是不能执行semop()函数？？？**

**答：**

因为6.4版本映像文件在gdb调试过程中一直在报错：

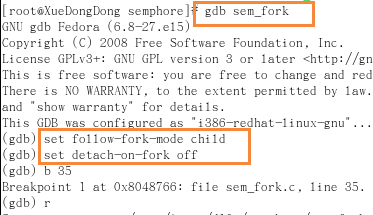


且网上找不到可靠的解决方法，所以以下我加载了5.3版本映像文件重新安装了一个虚拟机，进行了以下调试。

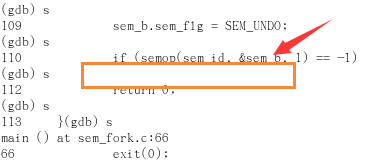
（1）

1. 因为调试的子进程，做了V操作，有了信号量为1，父进程不再挂起，抢占资源执行P操作

~~因为在使用set follow-fork-mode child时，父进程也处于调试过程中，需要将父进程在子进程执行时不执行。需要同时设置set follow-fork-mode child和set detach-on-fork off，默认调试子进程一步一步执行，在子进程调试期间父进程没有任何执行。具体见下图：~~



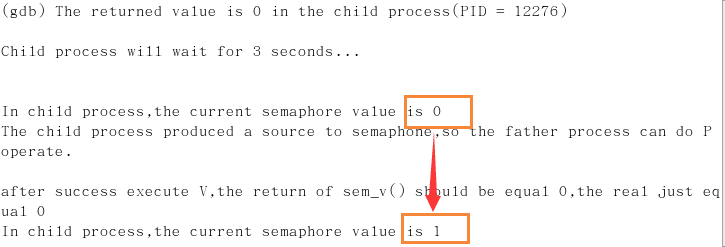
这样设置后，就不存在处于sem\_com源文件中，却打印sem\_fork源文件中父进程调试时内容：



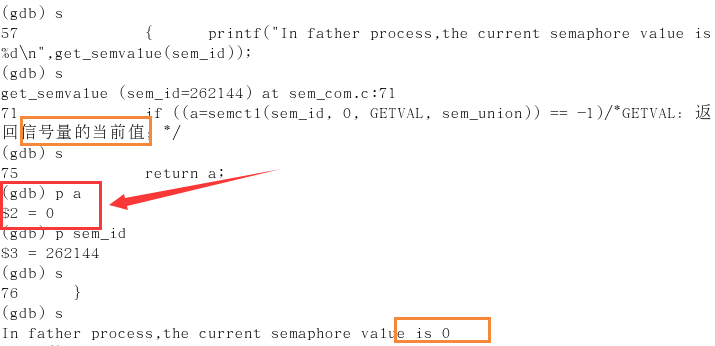
（2）父进程刚开始因为信号量初始值为0，P操作就是-1操作不能执行，所以父进程被阻塞，当子进程执行后，P操作变为1，父进程可以继续执行，在父进程成功执行完P进程后，程序就成功退出了。但是子进程还没有运行结束，所以子进程继续在新的命令提示符中打印字符，又因为子进程中没有执行退出的指令，所以光标一直在跳动等待退出的指令。

（3）分析如下：

子进程首先执行完成，信号量的值由0变1：



但是在刚开始执行父进程时，却显示信号量当前值为0：



当前信号量值为0，并不是子进程执行后信号量的值大小：1，可见父子进程执行过程毫不相关。因为当前信号量值为0，所以父进程无资源消耗，所以在执行P操作的semop()函数时显示父进程阻塞。

根据以上调试信息，可以得出：在调试时，父子进程是依次进行执行、互不相关的。

**6、共享内存是最为高效的进程间通信方式，请写出课后共享内存实验是如何使用信号量实现进程间通信的效果？搞懂生产者消费者程序执行的流程，并用自己的话描述两个源程序执行的关键步骤。**

**答：**

1. 通过使用信号量实现访问共享内存的进程之间的互斥，即同一时刻只能允许一个进程对共享内存进行写操作。
2. 生产者：首先使用函数semget()创建信号量并初始化，然后使用函数shmget()创建共享内存，并使用shmat()函数将共享内存映射到该进程地址空间中，接着执行信号量P操作（信号量初始值为1，P操作可执行），将用户输入的字符串和当前进程号等信息写入到共享内存之中后执行V操作，使用while()语句判断用户是否输入quit，若未输入quit则继续执行PV操作，当输入quit后，先删除信号量再删除共享内存到当前地址空间的映射。

消费者：首先使用函数semget()从生产者那儿获得信号量，用函数shmget()从生产者那儿获得共享内存，然后将共享内存映射到该进程地址空间中，获得共享内存的映射地址后，接着进行P操作（信号量初始值为1，P操作可执行），使用if语句判断收到的字符串是否为quit，若不是quit继续执行，从共享内存中读取数据，然后执行V操作，倘若收到quit后，使用break跳出循环，先删除共享内存到当前进程地址空间中的映射，再删除共享内存。

**7、在消息队列中，消息发送端进程和消息接收端进程之间为什么不需要额外实现进程间同步的机制？**

**答：**

因为发送端将发送的消息类型设置为该进程的进程号，接收端则根据消息类型确定消息发送者的进程号，这就已经实现了进程间的同步，所以不需要额外实现进程间同步。

1. **实验体会**

前半节实验课我们学习了管道和信号的相关实验，使我更透彻的理解了管道、标准流管道和有名管道、信号、信号发送函数、信号集函数组等理论知识，再加上老师的实践展示和代码讲解，使得这些知识变得浅显易懂。尤其值得夸赞的是赵老师源代码里注释，几乎每条语句都加了注释，使我在分析代码作用时更为便利，也顺便复习了理论知识。后半节实验课，我们学习了信号量、共享内存和消息队列，其中信号量和共享内存是重点，其中课后实验中有这两个重点的结合生产者和消费者实验，可见这个实验的重要性，和以往一样，我在结合了老师的视频讲解和注释后，对代码有了更深的理解，方便了掌握其中的知识及函数的应用。

实验整体来说难度中等，赵老师和以往一样给我们发了讲解视频和源代码，在跟着视频里老师一步一步操作后再做实验，瞬间觉得简单了很多，有好几条题目老师在视频里和源代码注释内都做作了详细的解释，只要稍微整理组织一下语言即可得出结论，但还有几条需要自己不断地动手实践才能得出结论和结果。