

北京交通大学操作系统实验报告

姓名：程维森

学号：21231264

一．实验内容

这个项目的目标是练习各种进程间通信方法（用于数据传递和同步），并学习Map-Reduce（并行计算）。这两者是工业界经常使用的非常重要的技术。

详细内容：  
 这个项目由三个独立的子项目组成，每个子项目执行相同的任务：给定一个文本文件，程序输出包含特定单词的行。例如，对于一行"Hello World!"，假设感兴趣的单词是"world"，那么这行应该被输出。（请注意，如果一行是"Hello worlds"，则该行不应该被输出）。你的程序（即父进程）将创建一个子进程。父进程可以打开文本文件，读取内容，并使用以下其中一种方法将其传递给子进程，但不能检查单词。而子进程不能打开该文件但可以检查单词。最终，父进程（而不是子进程）应该按字母顺序输出这些行。。

要求：

1. 使用管道作为传递文件内容和结果的方法。

1. 使用Unix域套接字作为传递文件内容和结果的方法。
2. 使用共享内存作为传递文件内容和结果的方法。此外，子进程创建4个线程， 每个线程充当一个Mapper；子进程的主线程充当单一的Reducer。 Map-Reduce仅适用于这个子项目。不允许使用Hadoop作为Map-Reduce 基础设施,相反，您必须使用Posix线程编程来实现Map-Reduce
3. Use Pipe as the method of passing the file content and the result

程序分析：程序的目标是是在一个文本文件中查找特定单词，如果找到，就将包含该单词的行和行号输出到一个文件中。它使用了管道进行进程间通信，并且包括了一个父进程和一个子进程。

1. 父进程：

   - 创建了一个管道，用于和子进程进行通信。

   - 打开了一个文本文件（big.txt），逐行读取文件内容。

   - 对于每一行，它检查是否包含特定的单词（在这里是"call"）。如果找到，父进程将该行的内容写入管道，供子进程读取。

   - 如果文件中没有找到特定单词，父进程向管道写入消息"Word not found"。

   - 等待子进程结束。

2. 子进程：

   - 从管道中读取数据，这些数据是父进程写入的包含特定单词的行。

   - 将读取到的数据写入一个输出文件(output.txt)中，同时加上"Child Process Output: "前缀。

   - 当读取结束后，关闭文件和管道，完成任务。

父进程分析：



父进程：

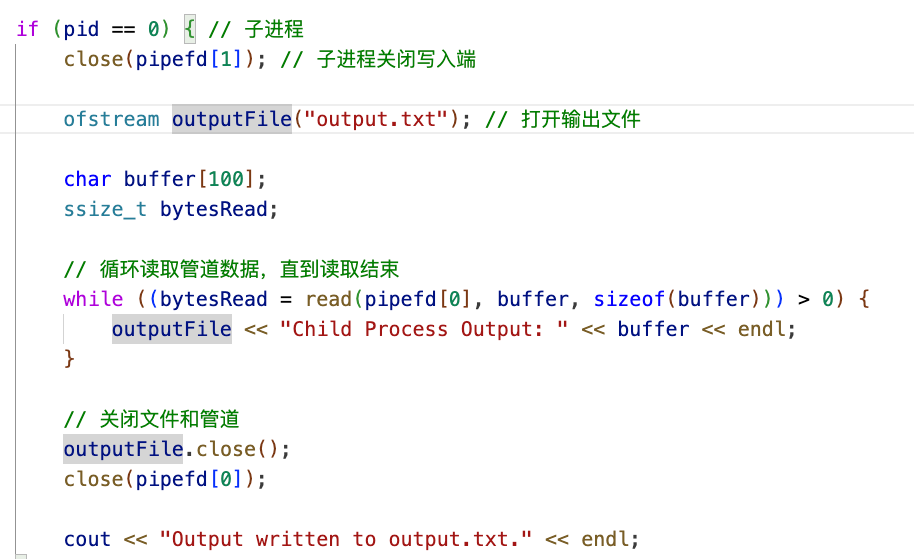
创建管道：在父进程开始执行时，它创建了一个管道（pipefd[2]），用于和子进程进行通信。

打开文件和逐行读取： 父进程打开了一个文本文件（big.txt）并逐行读取文件内容。

查找特定单词： 对于每一行，父进程检查是否包含特定的单词（"call"）。如果找到，它将 该行的内容写入管道，供子进程读取。如果文件中没有找到特定单词，父进程向管道写入消息"Word not found"。

等待子进程结束： 父进程在完成文件读取后，关闭了管道的写入端，并且等待子进程结束。

子进程：



子进程：

关闭写入端：子进程在创建后，关闭了管道的写入端，表示它只从管道中读取数据。

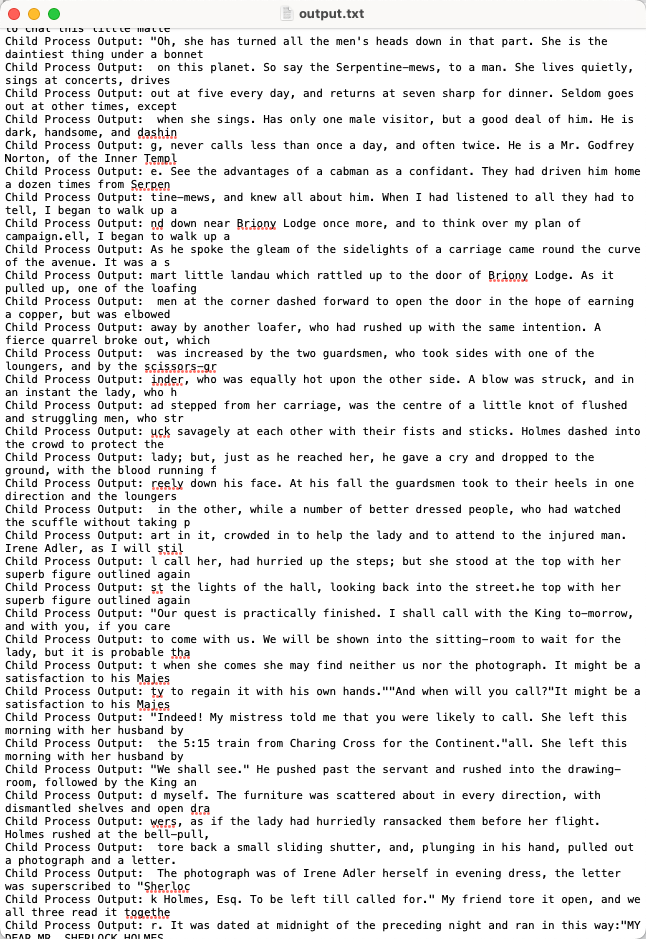
从管道读取数据：子进程从管道中读取数据，这些数据是父进程写入的包含特定单词的行或者"Word not found"消息。

将数据写入文件：子进程将读取到的数据写入一个输出文件(output.txt)中，每行前面加上"Child Process Output: "前缀。

关闭资源：当读取结束后，子进程关闭了输出文件和管道的读取端。

实验结果展示：

仅仅展示部分结果，全部的实验结果请参考附件的’output.txt’文件



实验总结：

这个管道（pipe）实验是一个简单的进程间通信示例，用于演示父子进程之间的数据传递。以下是这个实验的一些总结要点：

1. 目标： 这个实验的目标是演示如何使用管道进行进程间通信。它创建了一个父进程和一个子进程，父进程从一个文本文件中查找特定单词，并将匹配的行写入管道，而子进程从管道中读取这些数据，并将它们写入输出文件。

2. 父子进程：父进程和子进程是两个独立的进程，它们之间可以通过管道传递数据。父进程负责文件操作和数据的写入，而子进程负责读取管道中的数据并将其写入输出文件。

3. 管道创建：通过调用 `pipe` 函数，创建了一个匿名管道，用于在父子进程之间传递数据。

4. 进程间通信： 父进程逐行读取文本文件内容，查找指定单词，并将匹配的行写入管道。子进程从管道中读取数据，将其写入输出文件。

5. 父子进程协作： 父进程在完成文件读取后关闭管道的写入端，子进程在创建后关闭了管道的写入端，以确保不再有数据写入管道。

6. 资源管理： 在父子进程结束后，需要关闭文件和管道，以释放资源。

7. 问题： 这个示例存在一些问题，包括无法处理多个匹配行的情况，以及进程间同步问题。实际应用中，可能需要更复杂的逻辑来处理这些情况。

总的来说，这个实验有助于理解管道的基本概念和如何使用它在不同进程之间传递数据。然而，实际应用中，通常需要更复杂的逻辑来解决更具挑战性的问题。

2. Use Shared Memory as the method of passing the file content and the result.

程序分析：这段代码实现了一个使用共享内存进行进程间通信的程序，主要逻辑如下：

1. 共享内存创建：程序首先创建了一个共享内存段，大小为4MB，使用`shmget`函数。如果创建失败，程序输出错误信息并退出。

2. 创建子进程：接着，程序使用`fork`函数创建了一个子进程。如果创建失败，程序同样输出错误信息并退出。

3. 子进程逻辑：

   - 子进程通过`shmat`函数将共享内存附加到自身的内存空间，得到一个指向共享内存的指针。

   - 子进程尝试打开文件"big.txt"，如果文件不存在，输出错误信息并退出。然后，它读取文件内容到共享内存，并在末尾加上null终止符，以确保字符串结束。

   - 子进程查找共享内存中包含特定单词（"call"）的所有行，将这些行的内容和行号输出到文件"output.txt"中。

   - 子进程完成后，分离共享内存（`shmdt`函数）并删除共享内存段（`shmctl`函数）。

4. 父进程逻辑：

   - 父进程等待子进程执行完毕（`wait`函数）。

总的来说，该程序的目标是在一个大文件中查找指定的单词（"call"），使用共享内存实现了父子进程间的数据传递。父进程负责等待子进程的完成。这个程序假设共享内存足够大，能够容纳整个文件的内容。

子进程代码

实验结果展示：

具体请参考实验附录:



实验总结：  
优点：

共享内存提供了高效的进程间通信方式，适用于大量数据的传输。

使用共享内存可以避免数据的复制，提高了程序执行效率。

缺点：

共享内存需要进行适当的同步，以免多个进程同时修改共享数据导致不一致性。

对共享内存的操作需要谨慎，否则可能引发内存访问错误。

经验教训：

在使用共享内存时，确保正确处理同步问题，避免出现竞态条件。

在操作共享内存时，确保不会越界访问或者访问已经被分离的共享内存。

1. 运行时间对比与分析
2. 关于Map-Reduce
3. 关于Hadoop
4. Linux提供的不同进程间通信方法与何时使用哪种方法
5. 代码与仓库