利用无名管道实现A|B|C

# 实验目的

利用无名管道实现A|B|C，即多个进程间的通讯。

# 实验内容

* 理解无名管道的使用方法
* 使用fork来常见进程
* 编写三个程序以及一个调用程序实现进程使用无名管道通信

# 实验步骤

## 任务分析

* 首先编写三个程序A、B、C，分别实现随机产生一个小写字符串、字符串右移一位、字符串小写变大写
* 编写一个调用程序，实现创建三个进程，并通过无名管道来实现进程间的通信

## 详细设计

三个子程序很简单，主要是使用无名管道的程序的编写，程序流程图如下：

## 

## 调试分析

编译运行 pipe 文件，可以直接观察到三个进程间的同行以及处理情况。

## 测试结果

x31415@iZ2zea1nnstgr3h5ccfovoZ:~/linux/pipe$ ./pipe   
pid1: 12204  
pid1: 0  
pid2: 12205  
pid2: 0  
A running.............  
B running.............  
The str from A is: jxadqq  
C running.............  
The str from B is: kyberr  
The str from C is: KYBERR

## 使用说明

对文件夹中的四个文件分别编译生成可执行文件，然后运行 ./pipe 即可

# 实验总结

作为最基础的进程间的通信方式，无名管道的运行效率很高，可以从代码中看出，无名管道的作用就是将原本程序的标准输入输出更改流向到管道，建立一条单向的管道，进而实现一个程序的数据可以为另一使用。无名管道使用在具有情缘关系的进程间，进程的建立是按照一个方向建立的。

# 代码

### A

#include<stdio.h>  
#include<stdlib.h>  
#include<time.h>  
  
// 测试程序A，随机产生多个小写字母  
  
int main(int avgc, char \*\*avgs){  
  
 srand(time(NULL));  
 int n = rand() % 20 + 5;  
 while(n--)printf("%c", (char)('a' + rand() % 26));  
 return 0;  
}

### B

#include<stdio.h>  
#include<stdlib.h>  
#include<string.h>  
#include<unistd.h>  
  
// 测试程序B，将每一个字符循环后移一位  
  
int main(int avgc, char \*\*avgs){  
  
 char ch[BUFSIZ];  
 // while(~scanf("%s", ch));  
 // scanf("%s", ch);  
 int len = read(0, ch, sizeof(ch));  
 // int len = strlen(ch);  
 fprintf(stderr, "The str from A is: %s\n", ch);  
 for(int i = 0; i < len; ++i)printf("%c", (char)((ch[i] + 1 - 'a') % 26 + 'a'));  
 return 0;  
}

### C

#include<stdio.h>  
#include<stdlib.h>  
#include<string.h>  
  
// 测试程序C，将每一个字符变为大写  
  
int main(int avgc, char \*\*avgs){  
  
 char ch[BUFSIZ];  
 while(~scanf("%s", ch));  
 // scanf("%s", ch);  
 int len = strlen(ch);  
 fprintf(stderr, "The str from B is: %s\n", ch);  
 printf("The str from C is: ");  
 for(int i = 0; i < len; ++i)printf("%c", (char)(ch[i] - 'a' + 'A'));  
 puts("");  
 return 0;  
}

### pipe

#include<stdio.h>  
#include<stdlib.h>  
#include<sys/types.h>  
#include<unistd.h>  
#include<wait.h>  
  
// 测试无名管道，实现 A|B|C  
// 创建父子孙进程，修改其标准输入输出到管道中即可  
  
int main(void){  
  
 int fd1[2], fd2[2]; // 无名管道  
 int status;  
 pid\_t pid1, pid2;  
 char buf[BUFSIZ];  
 int ret;  
 if(!~pipe(fd1))perror("pipe1....");  
 if(!~pipe(fd2))perror("pipe2....");  
  
 // 创建B进程  
 if((pid1 = fork()) < 0)perror("Fork B failed...");  
 printf("pid1: %d\n", pid1);  
 if(pid1 == 0){  
 // B 进程  
 // 创建 C 进程  
 if((pid2 = fork()) < 0)perror("Fork C failed...");  
 printf("pid2: %d\n", pid2);  
 if(pid2 == 0){  
 // A 进程  
 // 标准写到管道1的写  
  
 dup2(fd2[1], 1);  
 close(fd2[1]);  
  
 close(fd2[0]);  
 fprintf(stderr, "A running.............\n");  
 execlp("./A", "./A", (char\*)0);  
 fprintf(stderr, "A done!!!!!!!!!!!!!!!!!!!!!!\n");  
 exit(EXIT\_SUCCESS);  
 }  
 else{  
 // B 进程  
 // 标准读到管道2读，标准写到管道1的写  
 // close(1);  
 // dup(fd1[1]);  
 sleep(1);  
 dup2(fd1[1], 1);  
 close(fd1[1]);  
  
 close(fd1[0]);  
  
 // close(0);  
 // dup(fd2[0]);  
 dup2(fd2[0], 0);  
 close(fd2[0]);  
  
 close(fd2[1]);  
 fprintf(stderr, "B running.............\n");  
 execlp("./B", "./B", (char\*)0);  
 fprintf(stderr, "B done.........\n");  
 exit(EXIT\_SUCCESS);  
 }  
 }  
 else{  
 // C 进程  
 // 标准读到管道2的读  
 sleep(2);  
 dup2(fd1[0], 0);  
 close(fd1[0]);  
   
 close(fd1[1]);  
 fprintf(stderr, "C running.............\n");  
 execlp("./C", "./C", (char\*)0);  
 exit(EXIT\_SUCCESS);  
 }  
 close(fd1[0]); close(fd1[1]);  
 close(fd2[0]); close(fd2[1]);  
 return 0;  
}