电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2016220304031

姓 名 邓巧

（实验） 课程名称 计算机操作系统

理论教师 王佳昊

实验教师 杨珊

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：邓巧 学号：2016220304031 指导教师：王佳昊**

**实验地点：信软楼400 实验时间：2018.5.4**

**一、实验名称：利用管道实现两个进程的通信**

**二、实验学时：4**

**三、实验目的：**

1.实验3 利用管道实现两个进程的通信

1. 熟悉Linux下的应用程序开发
2. 熟悉Linux的进程控制原语的使用
3. 掌握Linux操作系统的进程间通信机制管道的使用。
4. 掌握Linux操作系统中父进程与子进程的同步。

2.实验4 利用消息队列实现进程间的通信

通过对进程间通信（消息队列）的设计，深入理解进程之间是如何通过消息队列进行通信的。

**四、实验原理：**

**1.实验3 利用管道实现两个进程的通信**

首先创建两个子进程，注意Linux下使用fork()函数创建进程的方法。父进程和两个子进程间需要同步，使用waitpid()函数实现父进程等待子进程运行完毕后从管道中读取数据并打印，只有子进程将数据写入管道后，父进程才能够执行打开管道操作。由于fork函数让子进程完整地拷贝了父进程的整个地址空间，所以子进程都有管道的读端和写端。所以在相关进程中最好关掉不用的那一端。根据要求，“父进程先接收子进程P1发来的消息，然后再接收子进程P2发来的消息。”存在两个同步问题，两个子进程和父进程之间（先子写后父读）同步、子进程1和子进程2之间（先1写，再2写）

**2.实验4 利用消息队列实现进程间的通信**

消息队列就是一个消息的链表。可以把消息看作一个记录，具有特定的格式以及特定的优先级。对消息队列有写权限的进程可以向中按照一定的规则添加新消息；对消息队列有读权限的进程则可以从消息队列中读走消息。

**五、实验内容：**

1.实验3

在Linux系统中使用系统调用fork()创建两个子进程，使用系统调用pipe()建立一个管道，两个子进程分别向管道各写一句话：

Child process 1 is sending a message!

Child process 2 is sending a message!

而父进程则从管道中读出来自于两个子进程的信息，显示在屏幕上。然后分别结束两个子进程的运行。

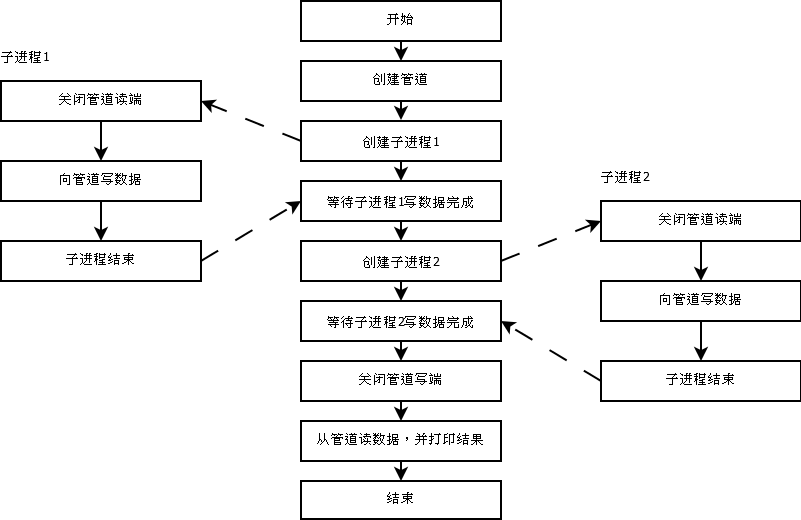
2.实验4

编程实现两个进程通过消息队列进行通信，一个Server进程，一个client进程。Server进程向client进程发送其进程ID，client进程同时也向Server进程发送其进程ID，双方接收到消息后，将所接收到的进程ID输出到屏幕上。

**六、实验器材（设备、元器件）：**

1. 学生每人一台PC，安装WindowsXP/2000操作系统。
2. 局域网络环境。
3. 个人PC安装VMware虚拟机和Ubuntu系统。

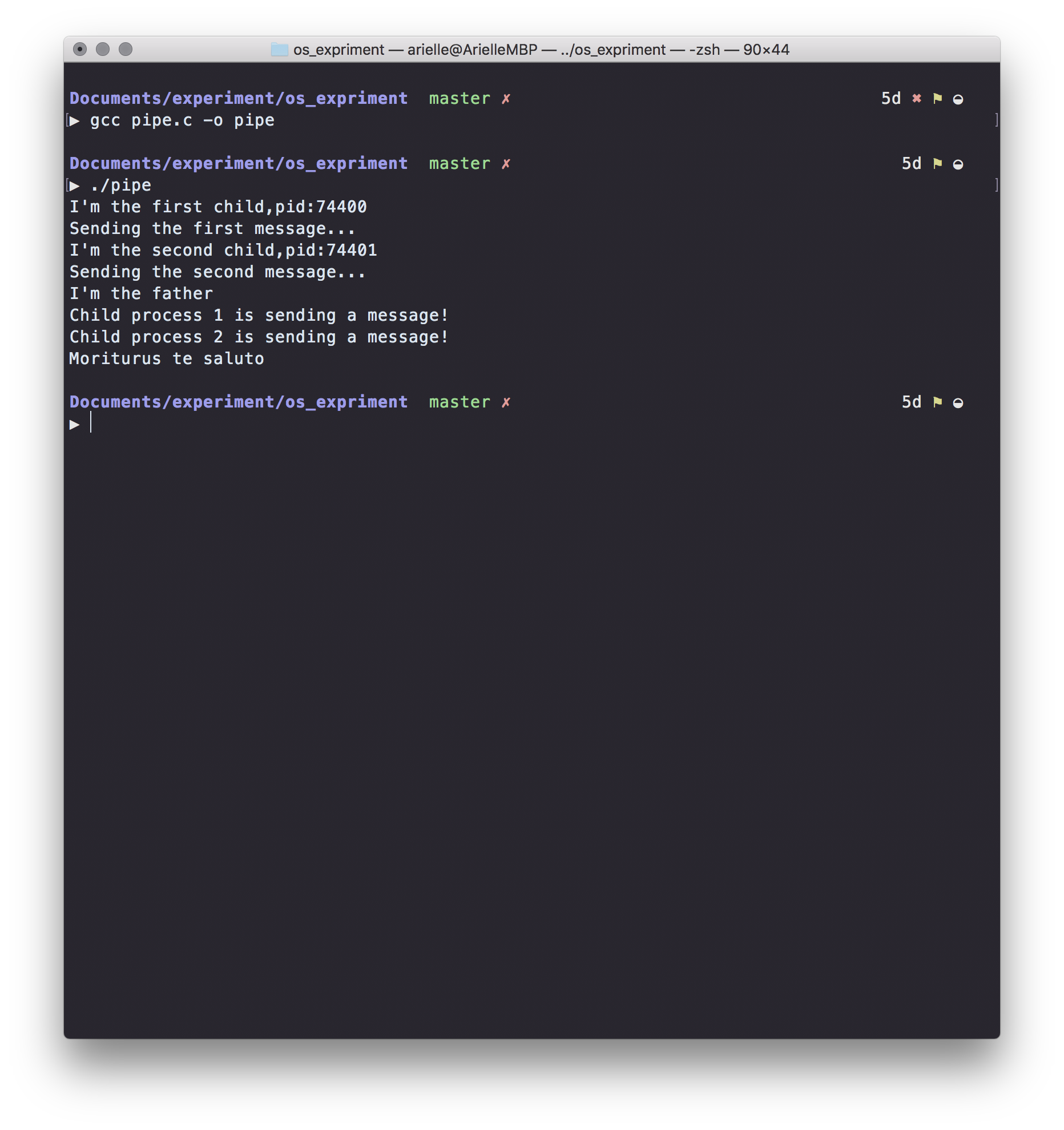
**七、实验步骤：**



**八、实验结果与分析（含重要数据结果分析或核心代码流程分析）**

**实验3 利用管道实现两个进程的通信**

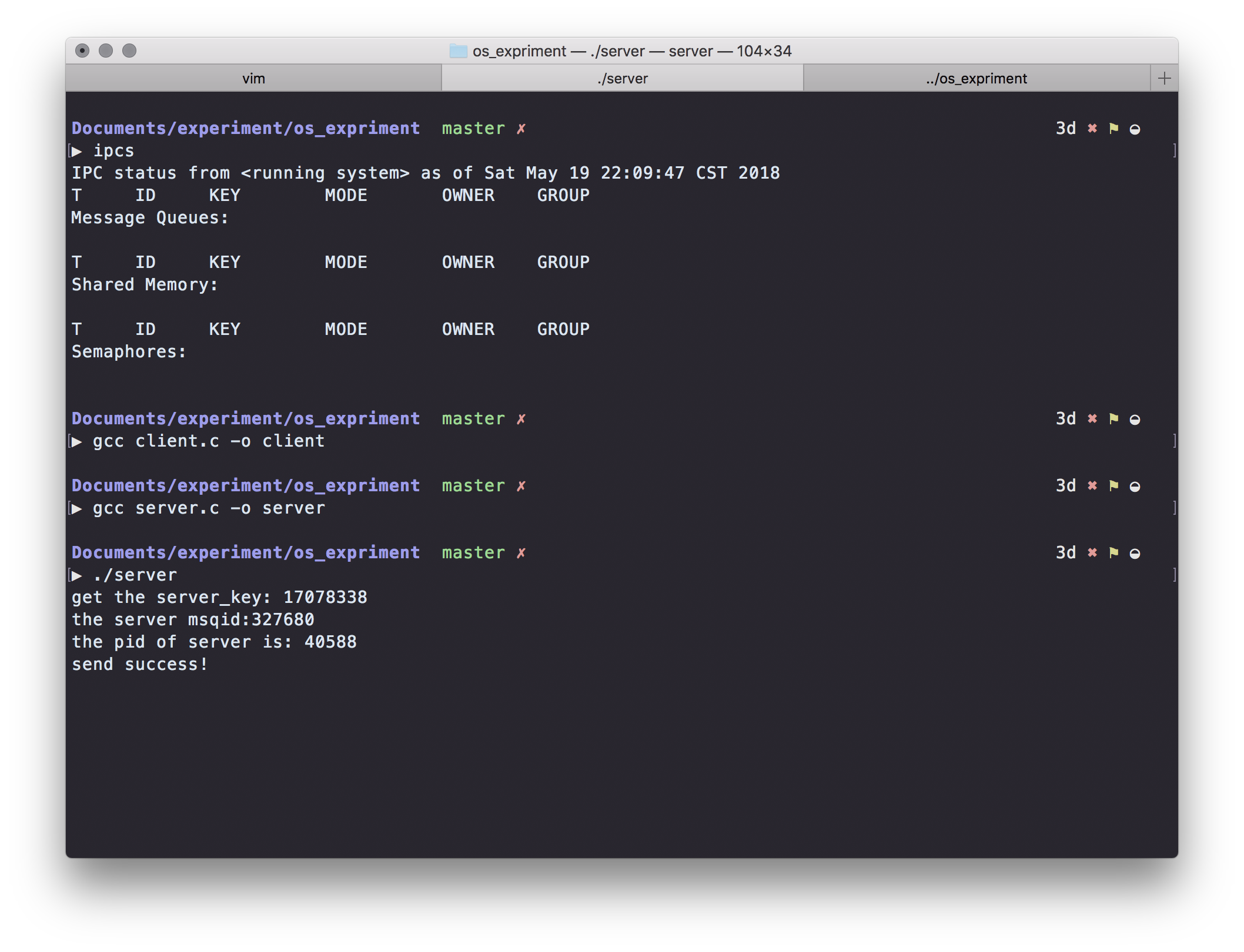
|  |
| --- |
| /\*pipe\*/  /\*author:dengqiao 2016220304031\*/  /date:2018.5.4\*/  #include <unistd.h>  #include <stdio.h>  #include <stdlib.h>  #include <errno.h>  #include <string.h>  #include <sys/types.h>  #include <sys/wait.h>  #define BUFSIZE 1024  int main(){  pid\_t first\_pid,second\_pid;  int fd[2];  char buf[BUFSIZE];  const char \*message1 = "Child process 1 is sending a message!\n";  const char \*message2 = "Child process 2 is sending a message!\n";  if(pipe(fd) < 0){  perror("pipe\n");  exit(errno);  }  if((first\_pid = fork()) < 0 ){  perror("fork\n");  exit(errno);  }  if(first\_pid == 0){  printf("I'm the first child,pid:%d\nSending the first message...\n",getpid());  close(fd[0]);  write(fd[1],(void \*)message1,strlen(message1));  close(fd[1]);  \_exit(EXIT\_SUCCESS);  }  if((second\_pid = fork()) < 0 ){  perror("fork\n");  exit(errno);  }  if(second\_pid == 0){  printf("I'm the second child,pid:%d\nSending the second message...\n",getpid());  close(fd[0]);  write(fd[1],(void \*)message2,strlen(message2));  close(fd[1]);  \_exit(EXIT\_SUCCESS);  }  else{  int status;  close(fd[1]);  waitpid(first\_pid, &status, 0);  waitpid(second\_pid,&status,0);  printf("I'm the father\n");  int len = read(fd[0], buf, sizeof(buf));  write(STDOUT\_FILENO,(void \*)buf,len);  close(fd[0]);  printf("Moriturus te saluto\n");  exit(EXIT\_SUCCESS);  }  return 0;  } |



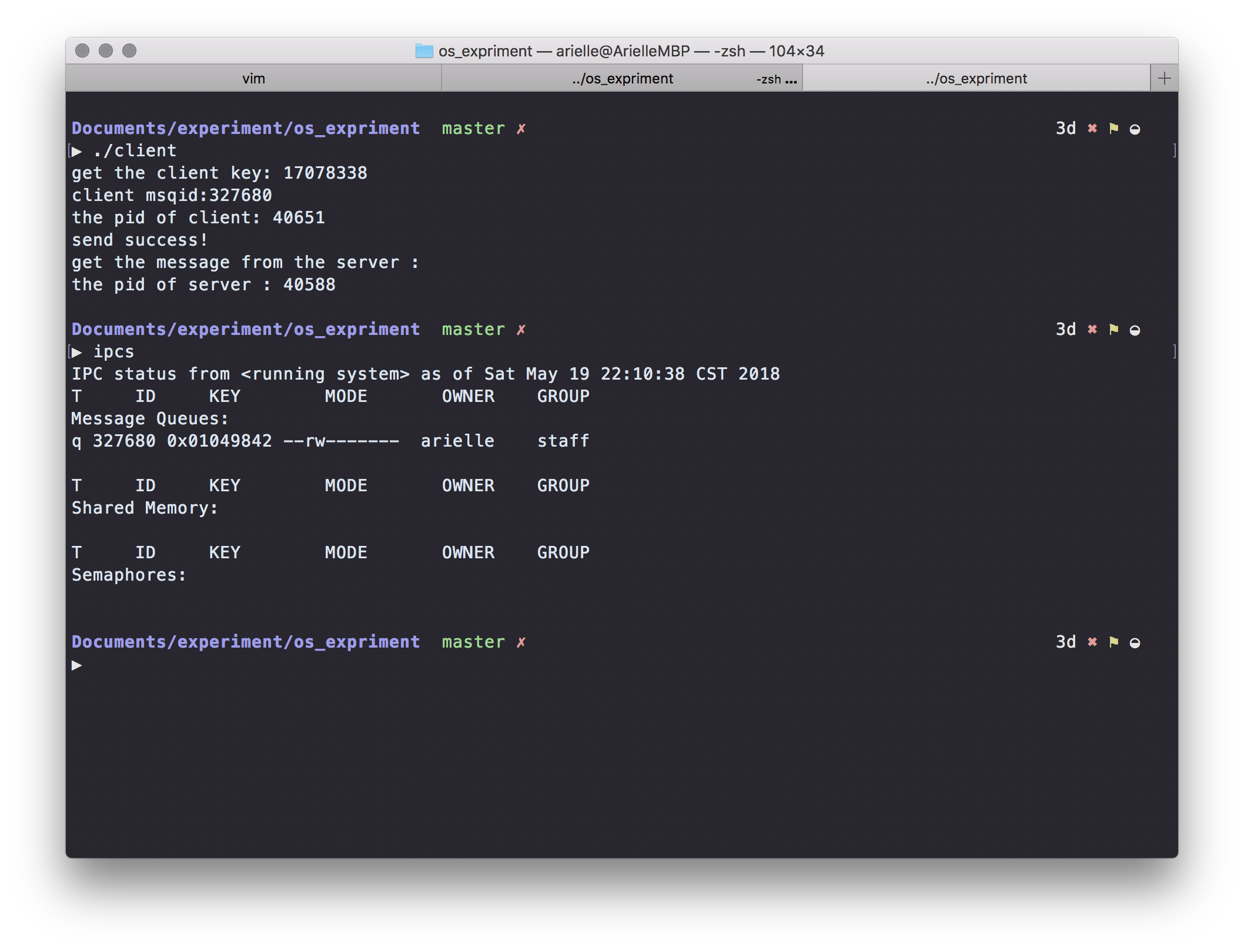
**实验四.利用消息队列实现进程间的通信**

|  |
| --- |
| /\*server.c\*/  /\*author:dengqiao 2016220304031\*/  /date:2018.5.19\*/  #include<stdio.h>  #include<sys/types.h>  #include<sys/msg.h>  #include<sys/ipc.h>  #include<string.h>  #include<errno.h>  #include<unistd.h>  #include<stdlib.h>  #define TEXT\_SIZE 1024  #define SERVER\_TYPE 1  #define CLIENT\_TYPE 2  struct msgbuf{  long type;  int msgtext;  };  int main(){  int msqid,key;  struct msgbuf server\_send,server\_recv;  if((key = ftok("meg.test",1)) < 0 ){  perror("ftok error\n");  exit(EXIT\_FAILURE);  }  printf("get the server\_key: %d\n",key);  if((msqid = (msgget(key,0600|IPC\_CREAT))) < 0){  perror("msgget error!\n");  exit(EXIT\_FAILURE);  }  printf("the server msqid:%d\n",msqid);  server\_send.type = SERVER\_TYPE;  server\_send.msgtext = getpid();  printf("the pid of server is: %d\n",server\_send.msgtext);  if(msgsnd(msqid,&server\_send,(sizeof(struct msgbuf) - sizeof(long)),0) < 0){  perror("send error!\n");  exit(EXIT\_FAILURE);  }  printf("send success!\n");  if(msgrcv(msqid,&server\_recv,(sizeof(struct msgbuf) - sizeof(long)),CLIENT\_TYPE,0) < 0){  perror("receive error!\n");  exit(EXIT\_FAILURE);  }  printf("get the message from the client: %d\n ",server\_recv.msgtext);  return 0;  } |

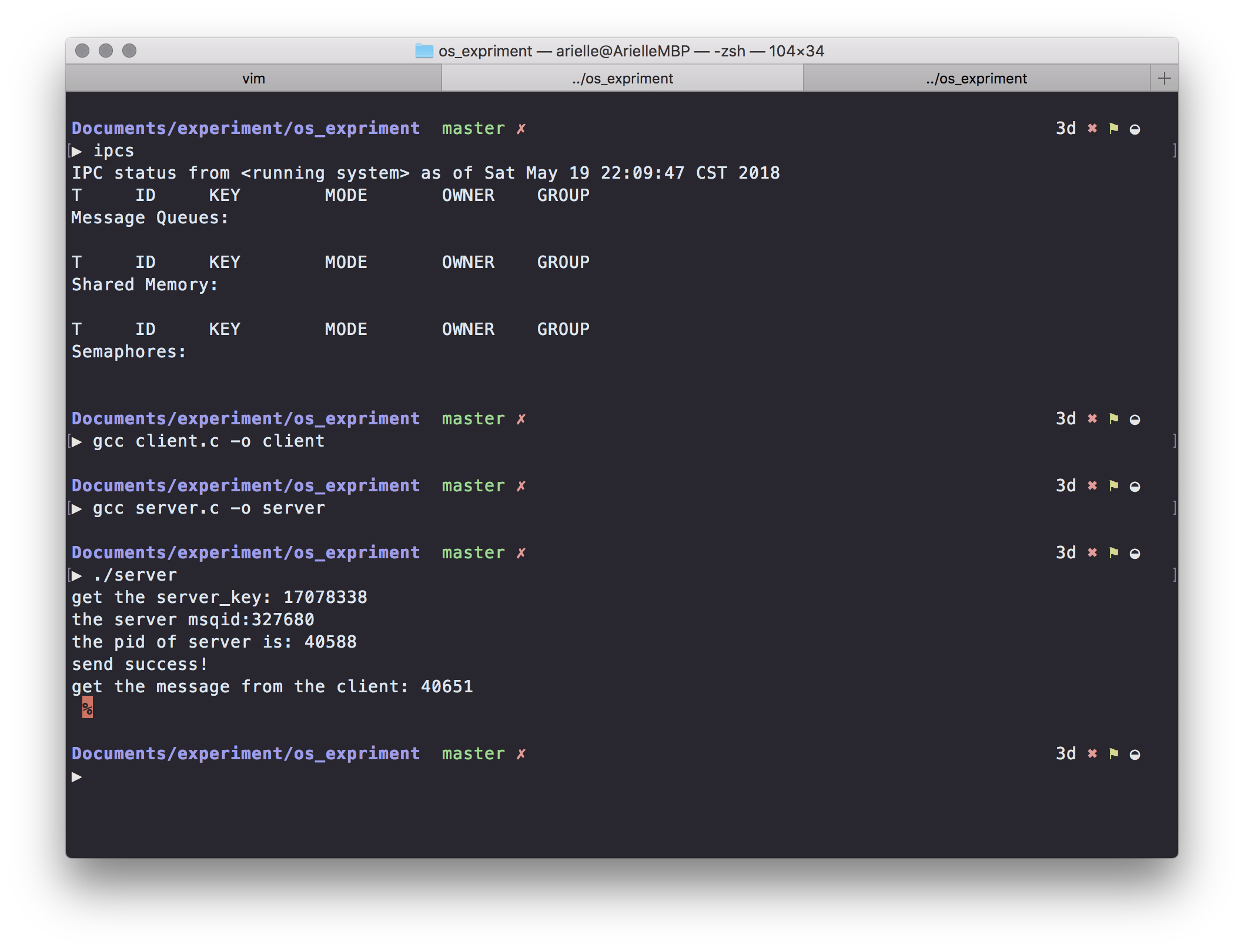
|  |
| --- |
| /\*client.c\*/  /\*author:dengqiao 2016220304031\*/  /date:2018.5.19\*/  #include<stdio.h>  #include<sys/types.h>  #include<sys/msg.h>  #include<sys/ipc.h>  #include<string.h>  #include<errno.h>  #include<unistd.h>  #include<stdlib.h>  #define TEXT\_SIZE 1024  #define SERVER\_TYPE 1  #define CLIENT\_TYPE 2  struct msgbuf{  long type;  int msgtext;  };  int main(){  int msqid,key;  struct msgbuf server\_send,server\_recv;  if((key = ftok("meg.test",1)) < 0 ){  perror("ftok error\n");  exit(EXIT\_FAILURE);  }  printf("get the server\_key: %d\n",key);  if((msqid = (msgget(key,0600|IPC\_CREAT))) < 0){  perror("msgget error!\n");  exit(EXIT\_FAILURE);  }  printf("the server msqid:%d\n",msqid);  server\_send.type = SERVER\_TYPE;  server\_send.msgtext = getpid();  printf("the pid of server is: %d\n",server\_send.msgtext);  if(msgsnd(msqid,&server\_send,(sizeof(struct msgbuf) - sizeof(long)),0) < 0){  perror("send error!\n");  }  printf("send success!\n");  if(msgrcv(msqid,&server\_recv,(sizeof(struct msgbuf) - sizeof(long)),CLIENT\_TYPE,0) < 0){  perror("receive error!\n");  exit(EXIT\_FAILURE);  }  printf("get the message from the client: %d\n ",server\_recv.msgtext);  return 0;  } |

****

运行server.c,向消息队列发送server的pid

****

运行client.c,向消息队列发送client的pid,并接收消息队列中server的pid

****

Server接收消息队列中client的pid

**九、总结及心得体会：**

1.应该尽量使用符合POSIX标准的函数

2.遇到问题可以多搜集资料

**十、对本实验过程及方法、手段的改进建议：**

无

**报告评分：**

**指导教师签字：**