贵州大学 计算机科学与技术学院 实验报告

院（系）名称	示范性软件学院	**班级**	软工 206	**课程名称**	Linux 系统
实验名称	实验五	**日期**	11 月 18 日	**指导教师**	王老师
学号、姓名	2000770081 秦小龙	**成绩**(10 分制)：	批改日期：		

1. 实验名称

   进程通信

2. 实验目的

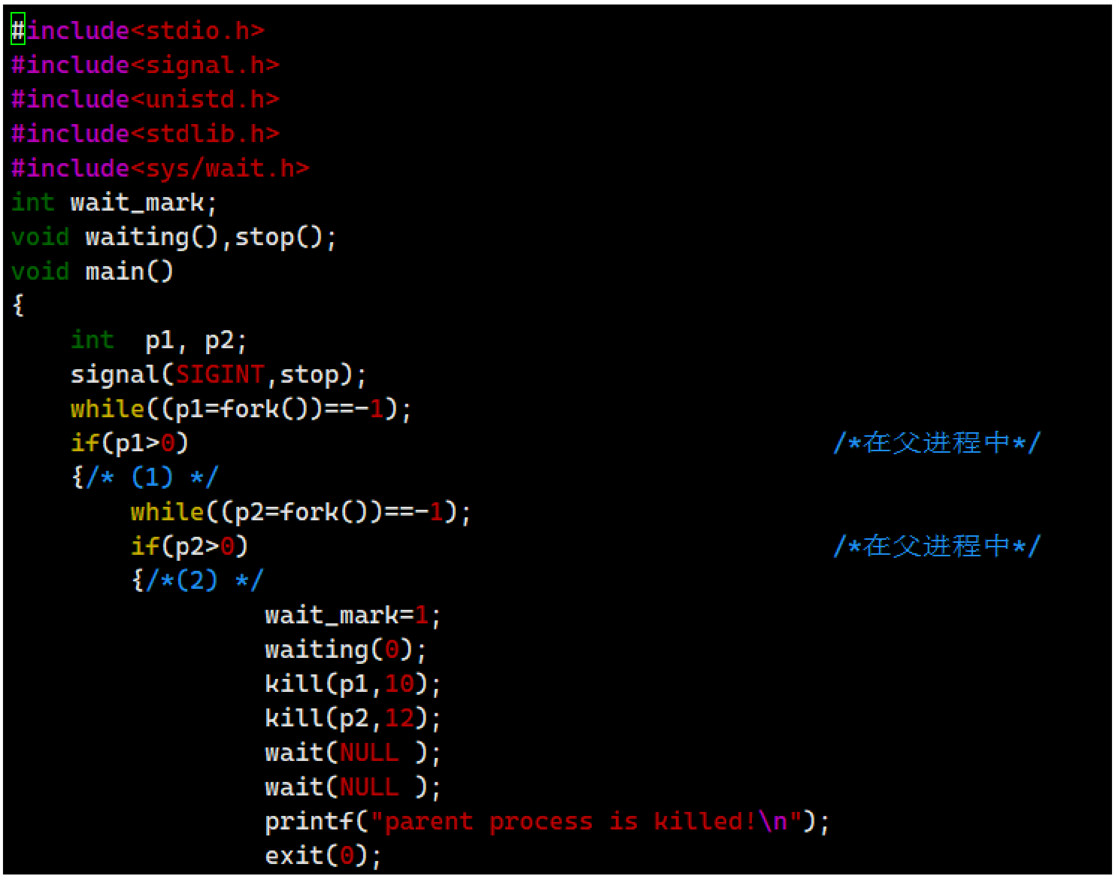
   熟悉 Linux 进程之间信号、管道、消息队列（信号量、共享内存）等通信的实现方法。

2. 内容（原理、方法）

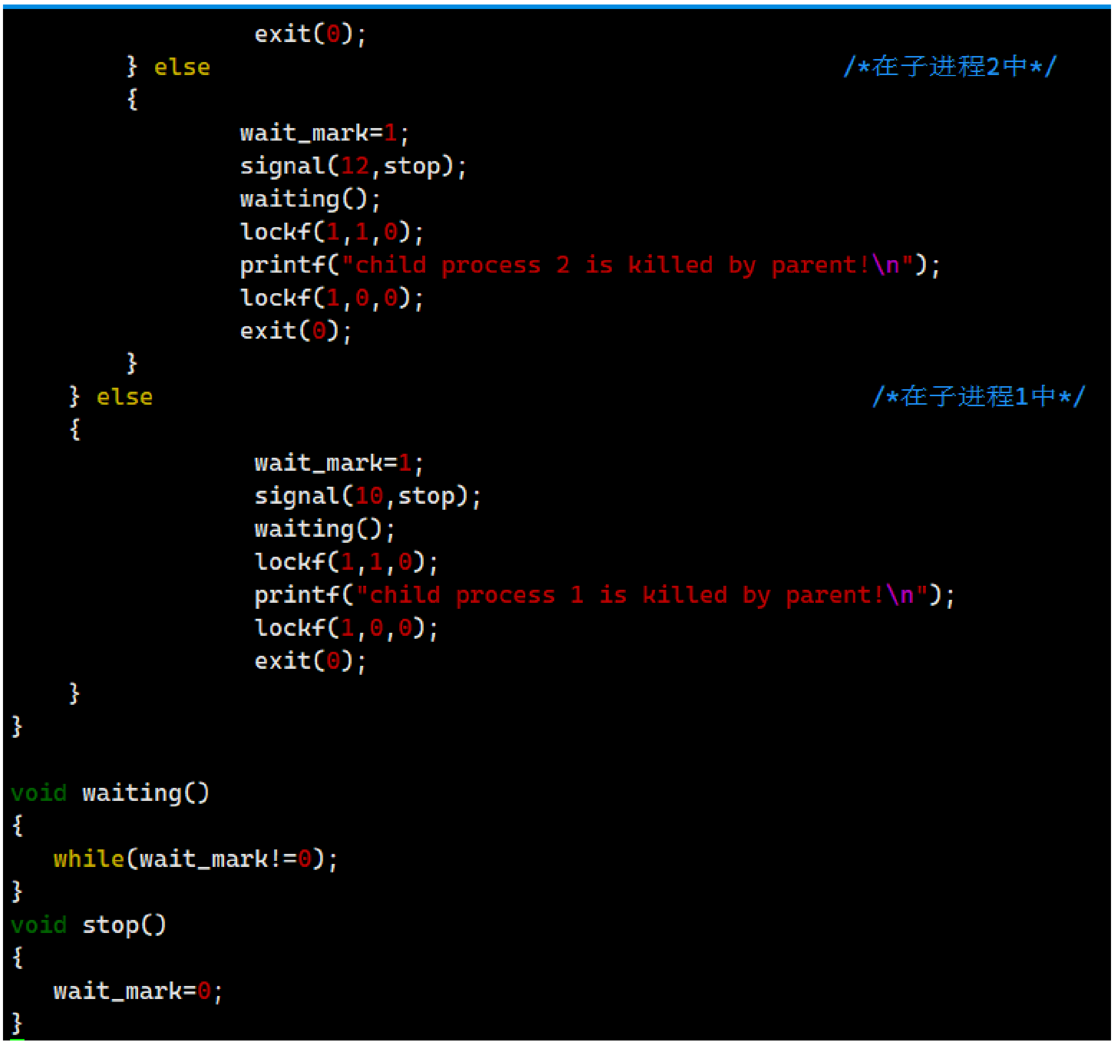
   分别使用信号、管道和消息队列实现进程之间的通信。

3. 结果、分析与建议

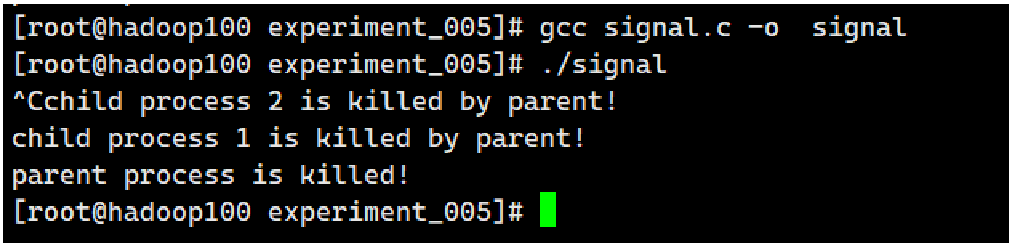
   - 基于信号的进程通信



```
#include<stdio.h>
#include<signal.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/wait.h>
int wait_mark;
void waiting(),stop();
void main()
{
    int  p1, p2;
    signal(SIGINT,stop);
    while((p1=fork())==-1);
    if(p1>0)                                                    /*在父进程中*/
    {/* (1) */
        while((p2=fork())==-1);
        if(p2>0)                                                /*在父进程中*/
        {/*(2) */
                 wait_mark=1;
                 waiting(0);
                 kill(p1,10);
                 kill(p2,12);
                 wait(NULL );
                 wait(NULL );
                 printf("parent process is killed!\n");
                 exit(0);
```

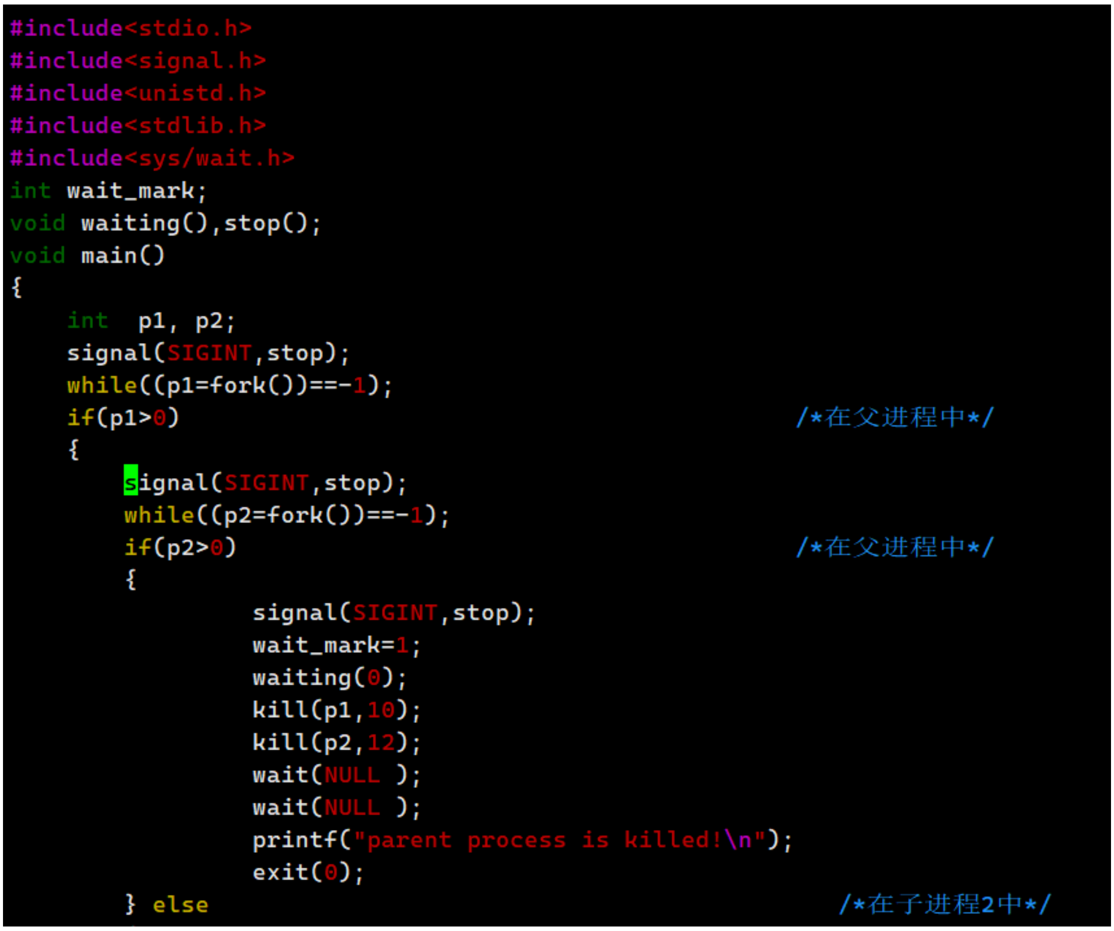


```
                 exit(0);
        } else                                                   /*在子进程2中*/
        {
                wait_mark=1;
                signal(12,stop);
                waiting();
                lockf(1,1,0);
                printf("child process 2 is killed by parent!\n");
                lockf(1,0,0);
                exit(0);
        }
    } else                                                         /*在子进程1中*/
    {
                 wait_mark=1;
                 signal(10,stop);
                 waiting();
                 lockf(1,1,0);
                 printf("child process 1 is killed by parent!\n");
                 lockf(1,0,0);
                 exit(0);
    }
}

void waiting()
{
   while(wait_mark!=0);
}
void stop()
{
   wait_mark=0;
}
```

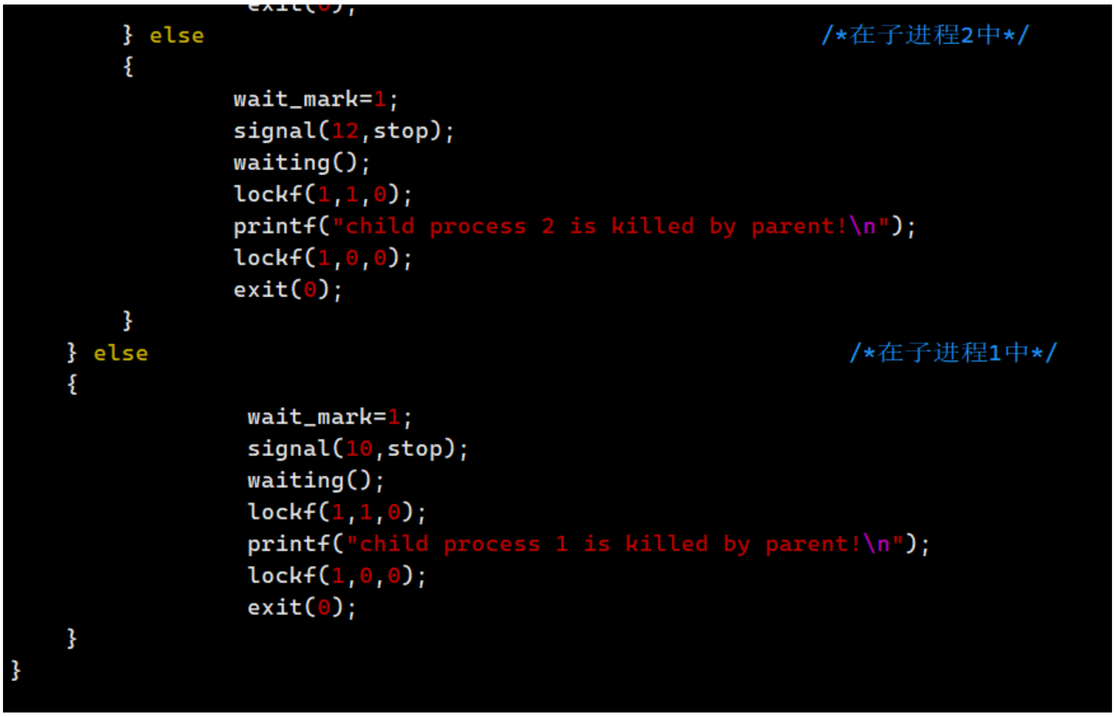


```
[root@hadoop100 experiment_005]# gcc signal.c -o  signal
[root@hadoop100 experiment_005]# ./signal
^Cchild process 2 is killed by parent!
child process 1 is killed by parent!
parent process is killed!
[root@hadoop100 experiment_005]# 
```

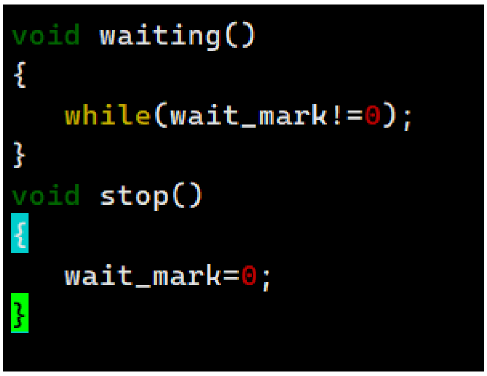
○ 把参考程序 signal(SIGINT,stop)放在 /*(1) */和 /*(2) */位置



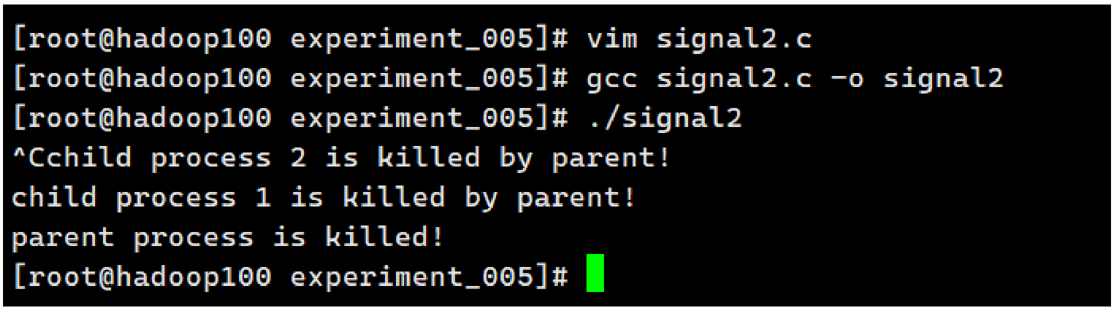
```
#include<stdio.h>
#include<signal.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/wait.h>
int wait_mark;
void waiting(),stop();
void main()
{
    int  p1, p2;
    signal(SIGINT,stop);
    while((p1=fork())==-1);
    if(p1>0)                                           /*在父进程中*/
    {
        signal(SIGINT,stop);
        while((p2=fork())==-1);
        if(p2>0)                                       /*在父进程中*/
        {
                 signal(SIGINT,stop);
                 wait_mark=1;
                 waiting(0);
                 kill(p1,10);
                 kill(p2,12);
                 wait(NULL );
                 wait(NULL );
                 printf("parent process is killed!\n");
                 exit(0);
        } else                                            /*在子进程2中*/
```



```
                 exit(0);
        } else                                           /*在子进程2中*/
        {
                wait_mark=1;
                signal(12,stop);
                waiting();
                lockf(1,1,0);
                printf("child process 2 is killed by parent!\n");
                lockf(1,0,0);
                exit(0);
        }
    } else                                                 /*在子进程1中*/
    {
                 wait_mark=1;
                 signal(10,stop);
                 waiting();
                 lockf(1,1,0);
                 printf("child process 1 is killed by parent!\n");
                 lockf(1,0,0);
                 exit(0);
    }
}
```



```
void waiting()
{
   while(wait_mark!=0);
}
void stop()
{
   wait_mark=0;
}
```

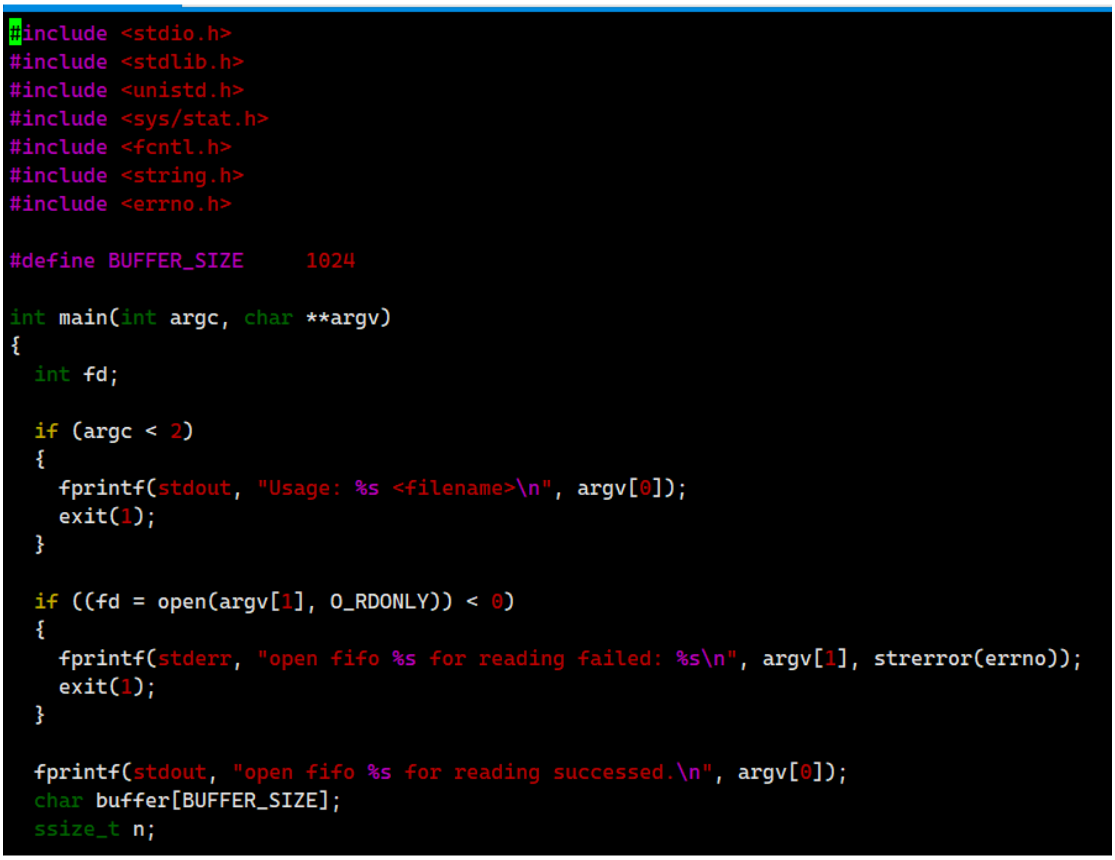


```
[root@hadoop100 experiment_005]# vim signal2.c
[root@hadoop100 experiment_005]# gcc signal2.c -o signal2
[root@hadoop100 experiment_005]# ./signal2
^Cchild process 2 is killed by parent!
child process 1 is killed by parent!
parent process is killed!
[root@hadoop100 experiment_005]# 
```

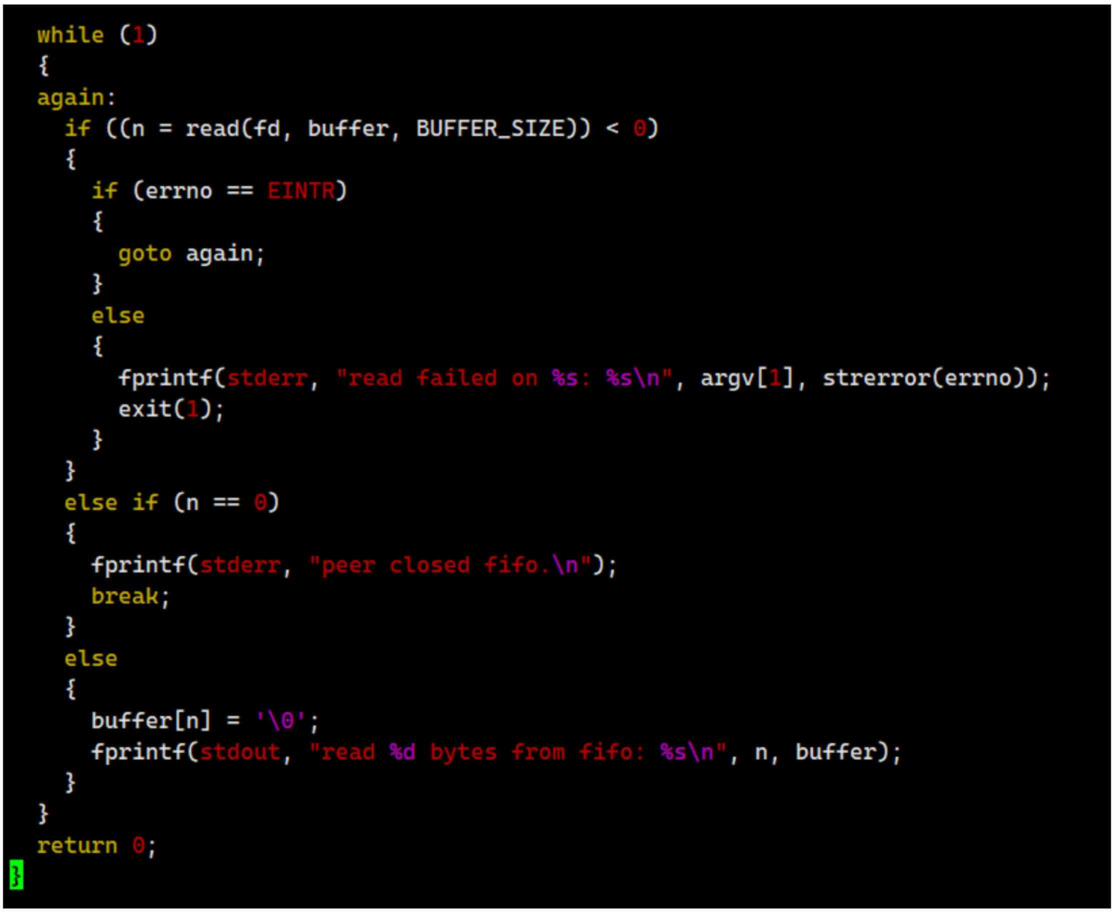
该程序段前面部分用了两个 wait(0),为什么？
wait(0) 暂时停止目前进程的执行，直到信号来到或子进程结束，如果在调用 wait(0) 时子进 程已经结束，则 wait(0) 会立即返回子进程结束状态值。

该程序段中每个进程退出时都用了语句 exit(0),为什么？
为了进程的正常终止，在正常终止时， exit() 函数返回进程结束状态。

- 基于管道的进程通信
  - ○ 读管道程序

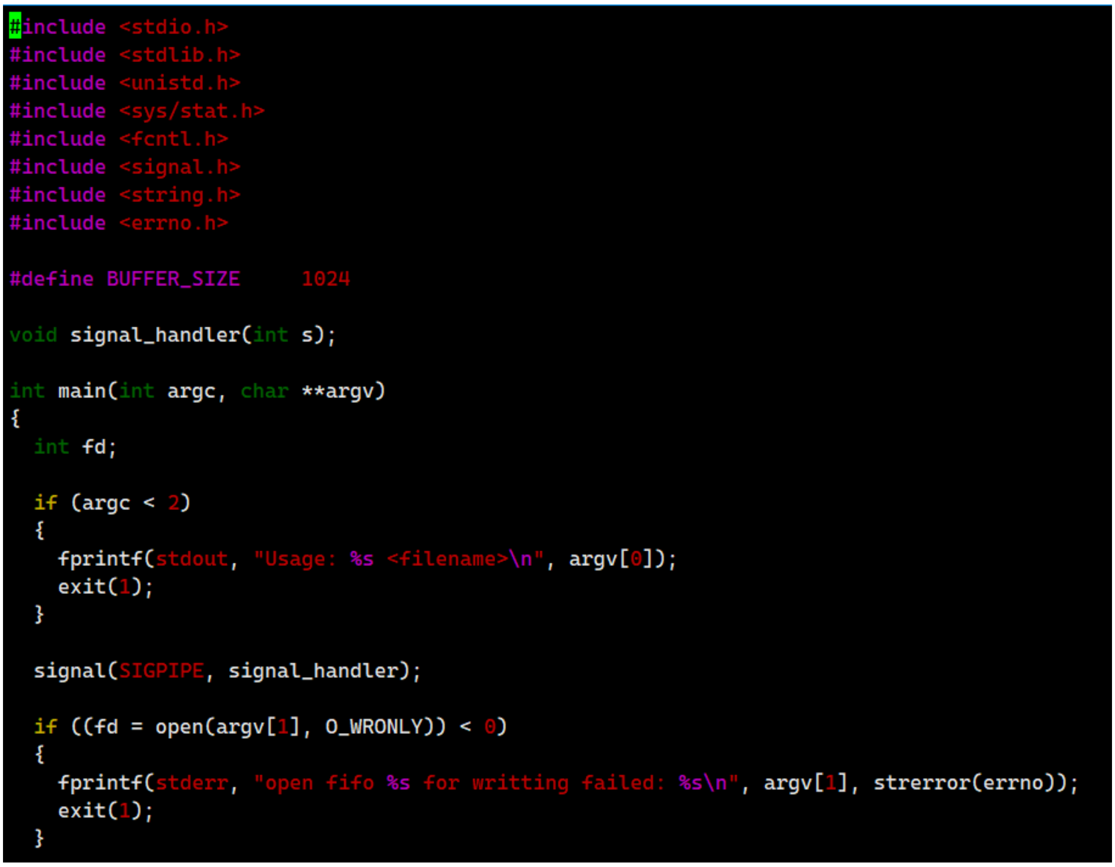


```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <errno.h>

#define BUFFER_SIZE     1024

int main(int argc, char **argv)
{
  int fd;

  if (argc < 2)
  {
    fprintf(stdout, "Usage: %s <filename>\n", argv[0]);
    exit(1);
  }

  if ((fd = open(argv[1], O_RDONLY)) < 0)
  {
    fprintf(stderr, "open fifo %s for reading failed: %s\n", argv[1], strerror(errno));
    exit(1);
  }

  fprintf(stdout, "open fifo %s for reading successed.\n", argv[0]);
  char buffer[BUFFER_SIZE];
  ssize_t n;
```

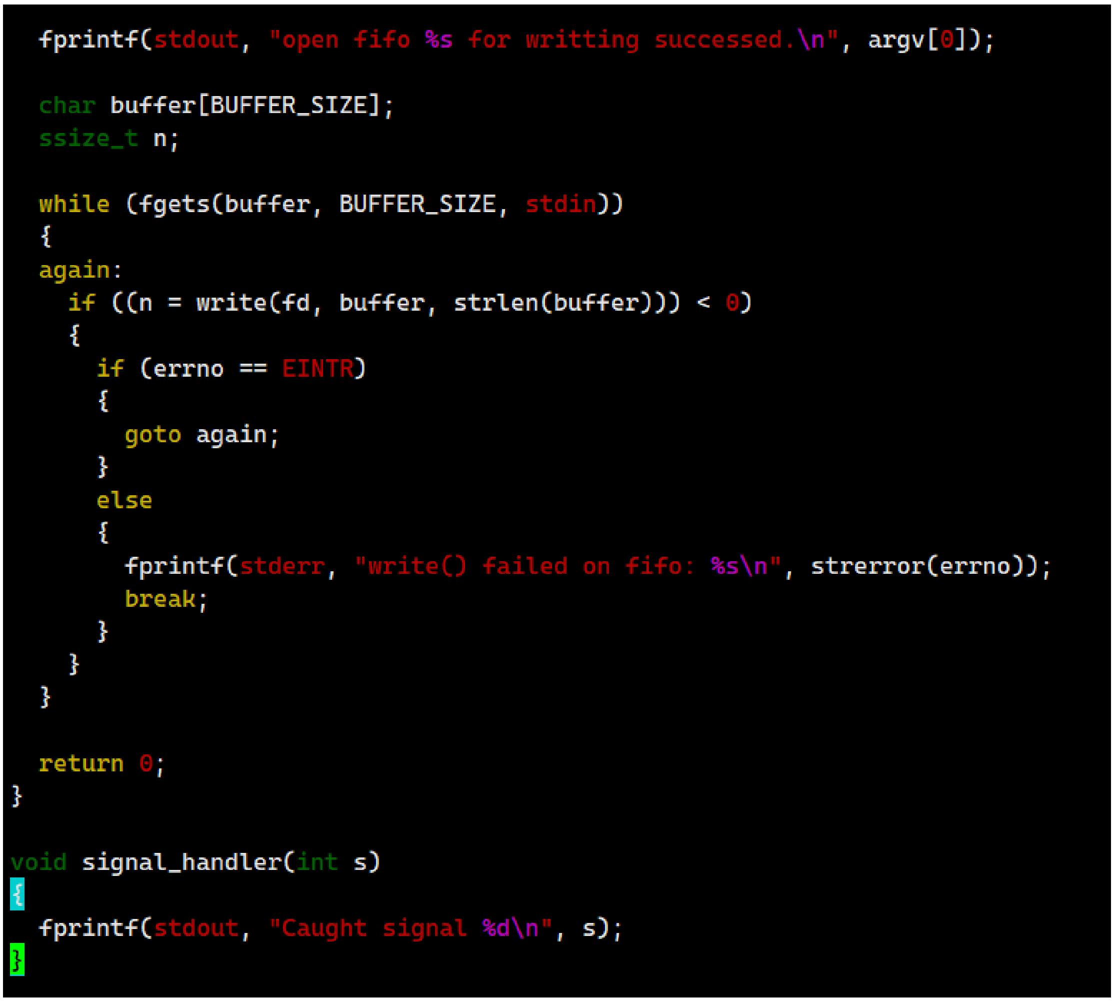


```
  while (1)
  {
  again:
    if ((n = read(fd, buffer, BUFFER_SIZE)) < 0)
    {
      if (errno == EINTR)
      {
        goto again;
      }
      else
      {
        fprintf(stderr, "read failed on %s: %s\n", argv[1], strerror(errno));
        exit(1);
      }
    }
    else if (n == 0)
    {
      fprintf(stderr, "peer closed fifo.\n");
      break;
    }
    else
    {
      buffer[n] = '\0';
      fprintf(stdout, "read %d bytes from fifo: %s\n", n, buffer);
    }
  }
  return 0;
}
```

○ 写管道程序

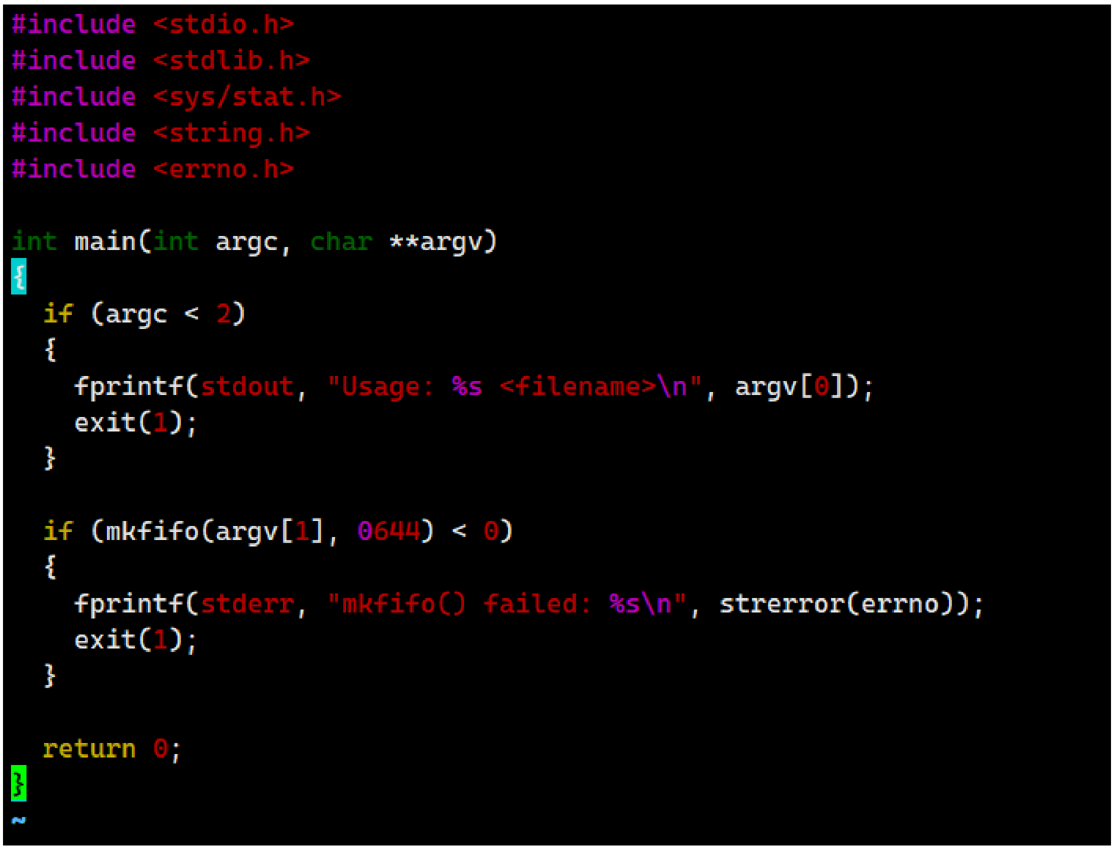


```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h>
#include <string.h>
#include <errno.h>

#define BUFFER_SIZE     1024

void signal_handler(int s);

int main(int argc, char **argv)
{
  int fd;

  if (argc < 2)
  {
    fprintf(stdout, "Usage: %s <filename>\n", argv[0]);
    exit(1);
  }

  signal(SIGPIPE, signal_handler);

  if ((fd = open(argv[1], O_WRONLY)) < 0)
  {
    fprintf(stderr, "open fifo %s for writting failed: %s\n", argv[1], strerror(errno));
    exit(1);
  }
```



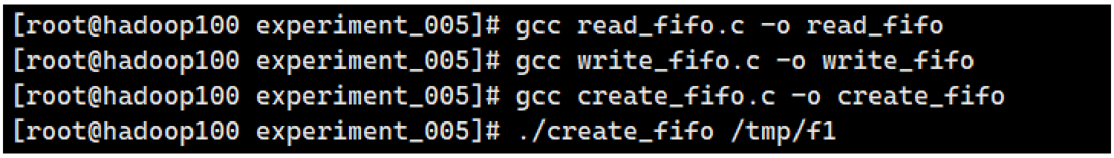
```
  fprintf(stdout, "open fifo %s for writting successed.\n", argv[0]);

  char buffer[BUFFER_SIZE];
  ssize_t n;

  while (fgets(buffer, BUFFER_SIZE, stdin))
  {
  again:
    if ((n = write(fd, buffer, strlen(buffer))) < 0)
    {
      if (errno == EINTR)
      {
        goto again;
      }
      else
      {
        fprintf(stderr, "write() failed on fifo: %s\n", strerror(errno));
        break;
      }
    }
  }

  return 0;
}

void signal_handler(int s)
{
  fprintf(stdout, "Caught signal %d\n", s);
}
```

○ 创建管道程序

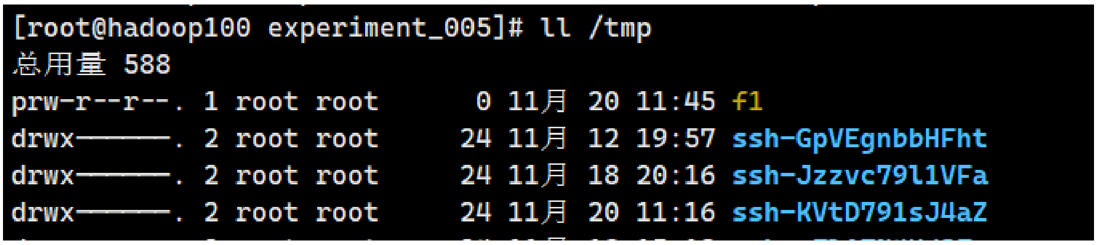


```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <string.h>
#include <errno.h>

int main(int argc, char **argv)
{
  if (argc < 2)
  {
    fprintf(stdout, "Usage: %s <filename>\n", argv[0]);
    exit(1);
  }

  if (mkfifo(argv[1], 0644) < 0)
  {
    fprintf(stderr, "mkfifo() failed: %s\n", strerror(errno));
    exit(1);
  }

  return 0;
}
~
```

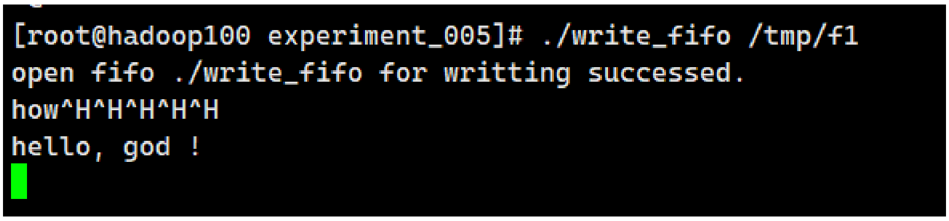
○ 编译运行



```
[root@hadoop100 experiment_005]# gcc read_fifo.c -o read_fifo
[root@hadoop100 experiment_005]# gcc write_fifo.c -o write_fifo
[root@hadoop100 experiment_005]# gcc create_fifo.c -o create_fifo
[root@hadoop100 experiment_005]# ./create_fifo /tmp/f1
```

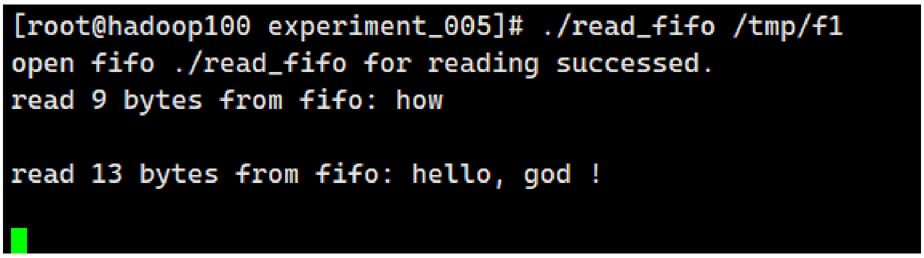


```
[root@hadoop100 experiment_005]# ll /tmp
总用量 588
prw-r--r--. 1 root root     0 11月 20 11:45 f1
drwx------. 2 root root    24 11月 12 19:57 ssh-GpVEgnbbHFht
drwx------. 2 root root    24 11月 18 20:16 ssh-Jzzvc79l1VFa
drwx------. 2 root root    24 11月 20 11:16 ssh-KVtD791sJ4aZ
```



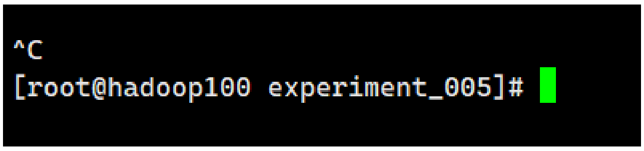
```
[root@hadoop100 experiment_005]# ./write_fifo /tmp/f1
open fifo ./write_fifo for writting successed.
how^H^H^H^H
hello, god !
```

另一终端



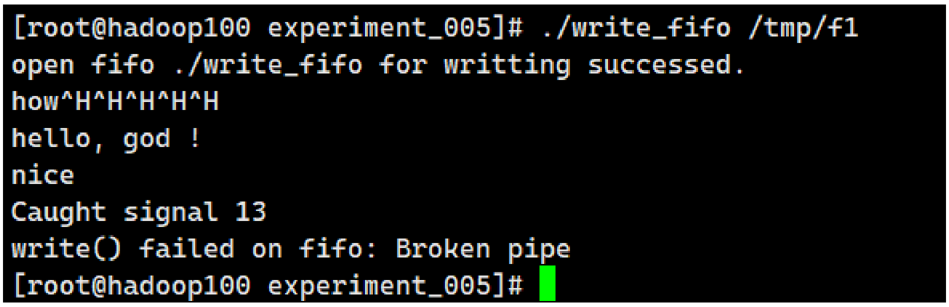
```
[root@hadoop100 experiment_005]# ./read_fifo /tmp/f1
open fifo ./read_fifo for reading successed.
read 9 bytes from fifo: how

read 13 bytes from fifo: hello, god !

```

断开写进程



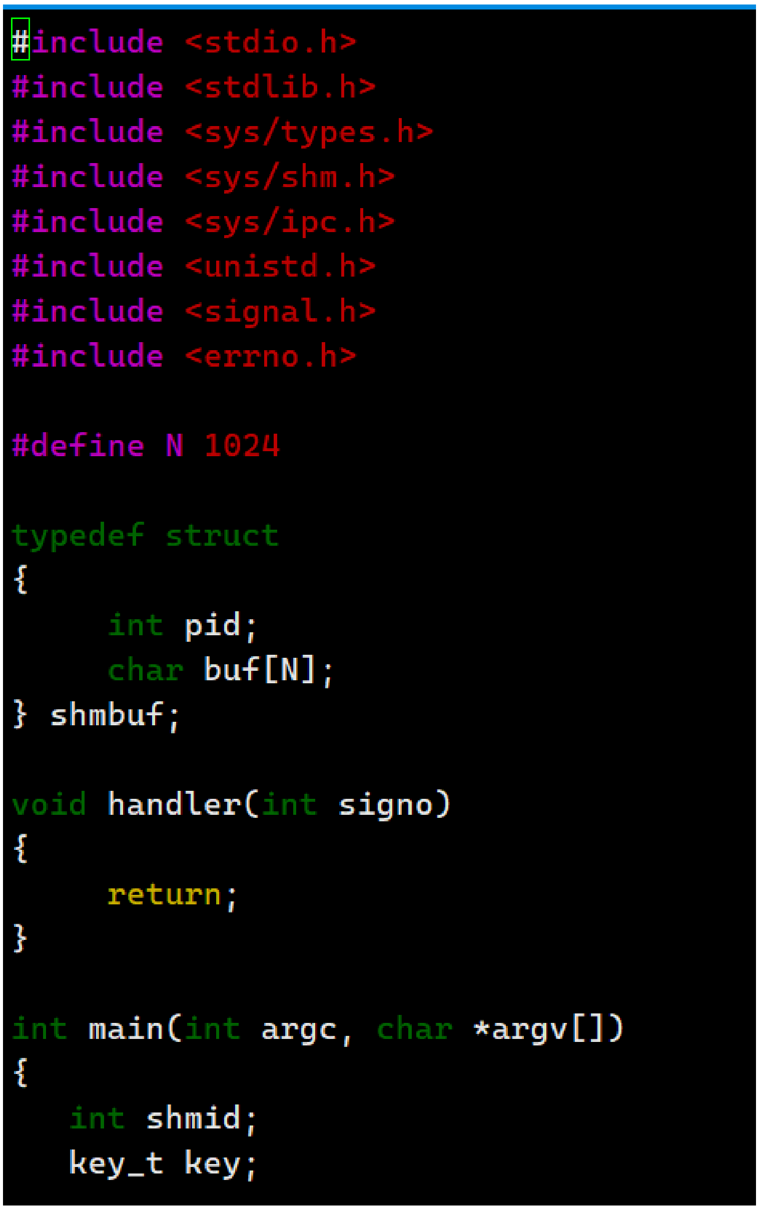
```

^C
[root@hadoop100 experiment_005]# 
```

读进程也自动断开

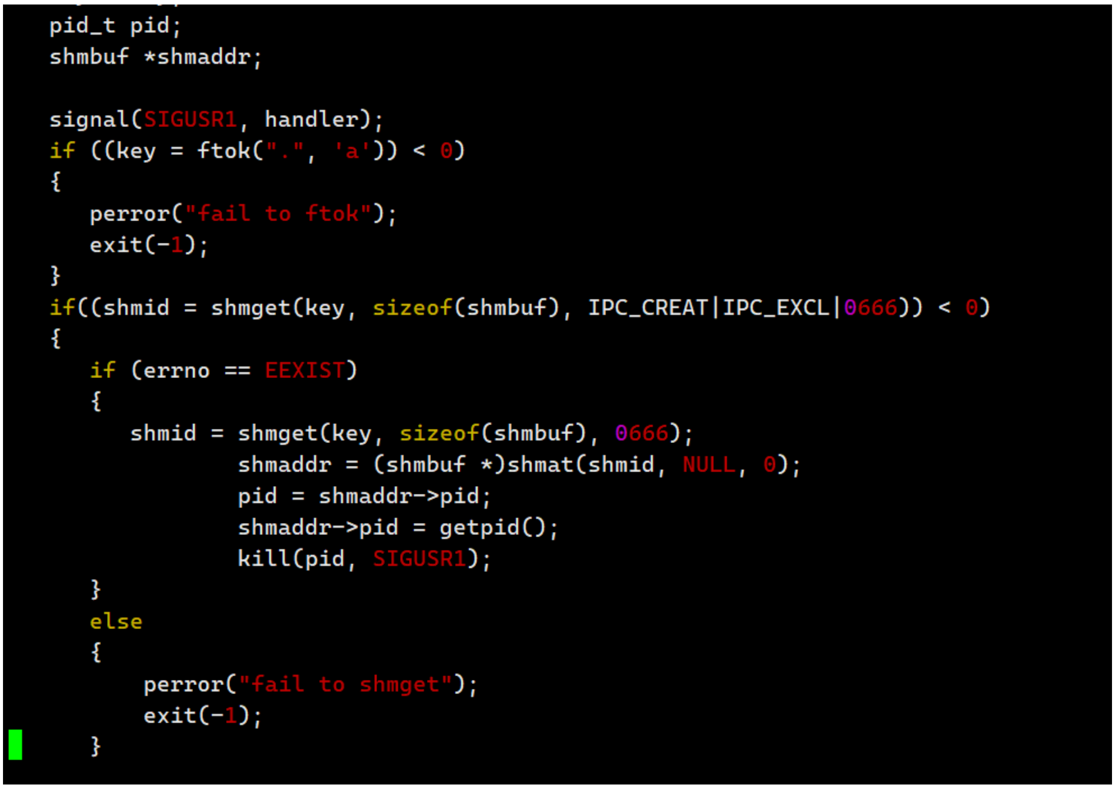


```
[root@hadoop100 experiment_005]# ./write_fifo /tmp/f1
open fifo ./write_fifo for writting successed.
how^H^H^H^H
hello, god !
nice
Caught signal 13
write() failed on fifo: Broken pipe
[root@hadoop100 experiment_005]# 
```

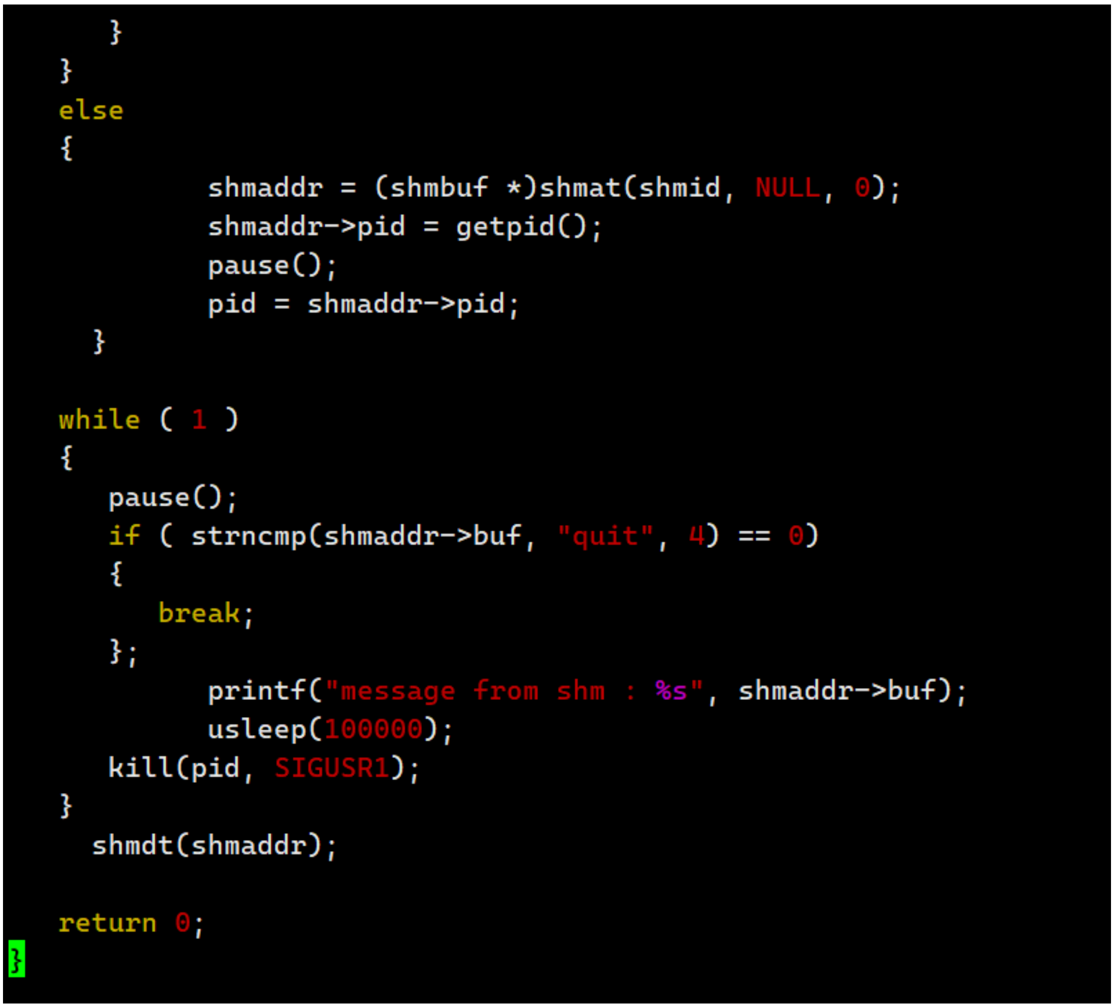
- 基于共享内存的进程通信
  - 共享内存读



```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <unistd.h>
#include <signal.h>
#include <errno.h>

#define N 1024

typedef struct
{
	int pid;
	char buf[N];
} shmbuf;

void handler(int signo)
{
	return;
}

int main(int argc, char *argv[])
{
    int shmid;
    key_t key;
```

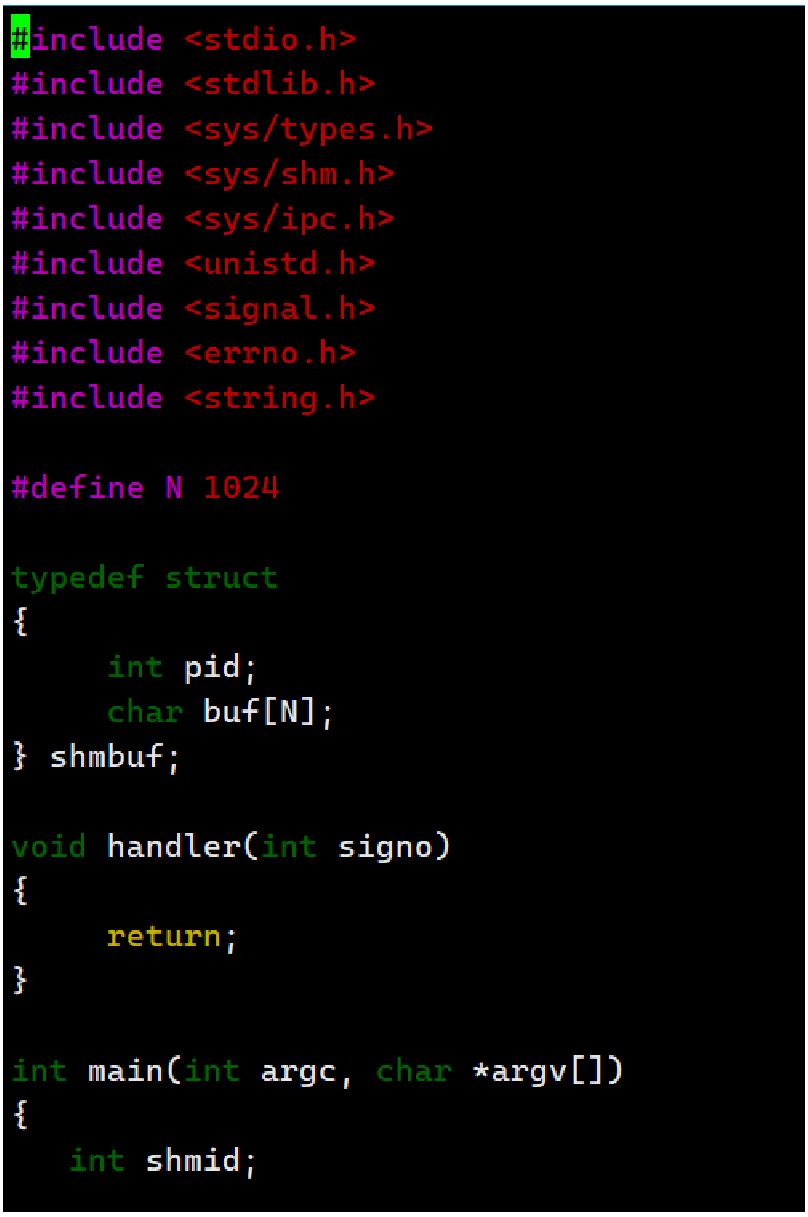


```
    pid_t pid;
    shmbuf *shmaddr;

    signal(SIGUSR1, handler);
    if ((key = ftok(".", 'a')) < 0)
    {
       perror("fail to ftok");
       exit(-1);
    }
    if((shmid = shmget(key, sizeof(shmbuf), IPC_CREAT|IPC_EXCL|0666)) < 0)
    {
       if (errno == EEXIST)
       {
          shmid = shmget(key, sizeof(shmbuf), 0666);
                  shmaddr = (shmbuf *)shmat(shmid, NULL, 0);
                  pid = shmaddr->pid;
                  shmaddr->pid = getpid();
                  kill(pid, SIGUSR1);
       }
       else
       {
           perror("fail to shmget");
           exit(-1);
       }
```

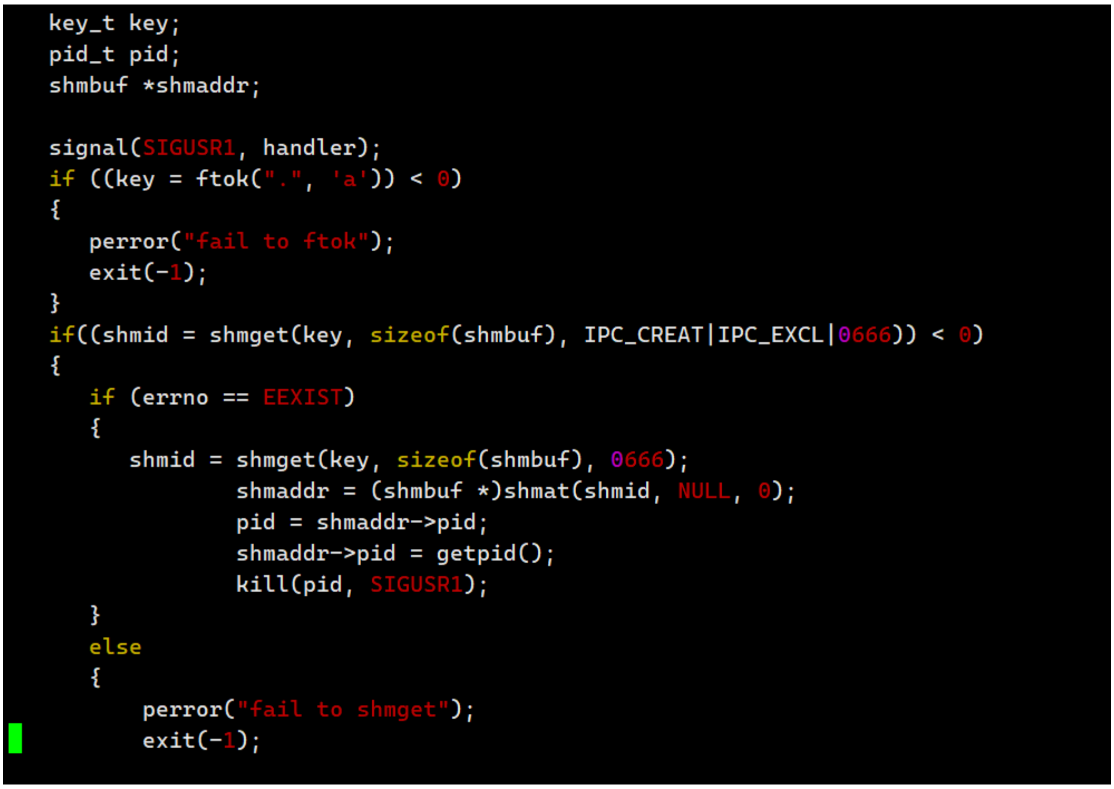


```
       }
    }
    else
    {
             shmaddr = (shmbuf *)shmat(shmid, NULL, 0);
             shmaddr->pid = getpid();
             pause();
             pid = shmaddr->pid;
      }

    while ( 1 )
    {
       pause();
       if ( strncmp(shmaddr->buf, "quit", 4) == 0)
       {
          break;
       };
             printf("message from shm : %s", shmaddr->buf);
             usleep(100000);
       kill(pid, SIGUSR1);
    }
      shmdt(shmaddr);

    return 0;
}
```

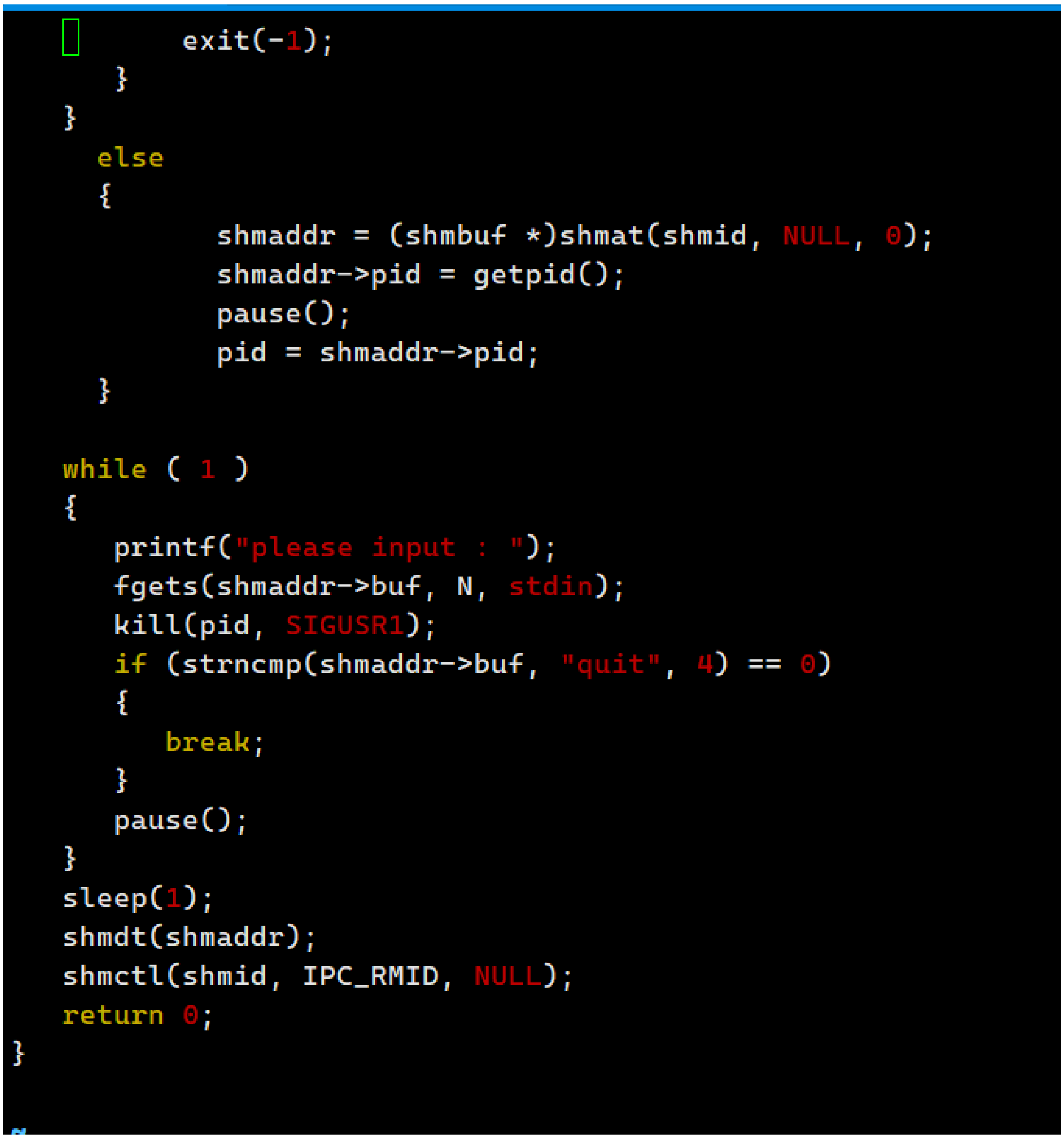
○ 共享内存写



```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <unistd.h>
#include <signal.h>
#include <errno.h>
#include <string.h>

#define N 1024

typedef struct
{
	int pid;
	char buf[N];
} shmbuf;

void handler(int signo)
{
	return;
}

int main(int argc, char *argv[])
{
   int shmid;
```

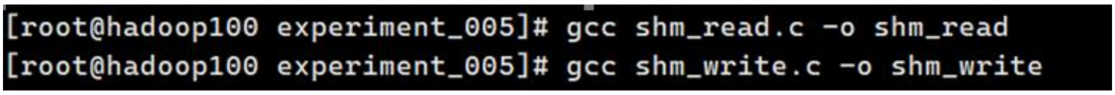


```
   key_t key;
   pid_t pid;
   shmbuf *shmaddr;

   signal(SIGUSR1, handler);
   if ((key = ftok(".", 'a')) < 0)
   {
      perror("fail to ftok");
      exit(-1);
   }
   if((shmid = shmget(key, sizeof(shmbuf), IPC_CREAT|IPC_EXCL|0666)) < 0)
   {
      if (errno == EEXIST)
      {
         shmid = shmget(key, sizeof(shmbuf), 0666);
                 shmaddr = (shmbuf *)shmat(shmid, NULL, 0);
                 pid = shmaddr->pid;
                 shmaddr->pid = getpid();
                 kill(pid, SIGUSR1);
      }
      else
      {
          perror("fail to shmget");
          exit(-1);
```

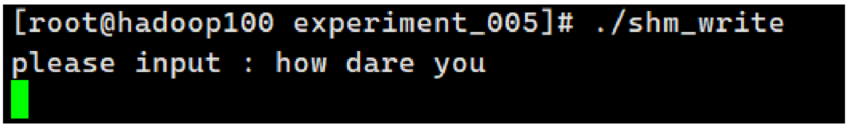


```
                exit(-1);
        }
    }
      else
      {
              shmaddr = (shmbuf *)shmat(shmid, NULL, 0);
              shmaddr->pid = getpid();
              pause();
              pid = shmaddr->pid;
      }

    while ( 1 )
    {
        printf("please input : ");
        fgets(shmaddr->buf, N, stdin);
        kill(pid, SIGUSR1);
        if (strncmp(shmaddr->buf, "quit", 4) == 0)
        {
            break;
        }
        pause();
    }
    sleep(1);
    shmdt(shmaddr);
    shmctl(shmid, IPC_RMID, NULL);
    return 0;
}
```

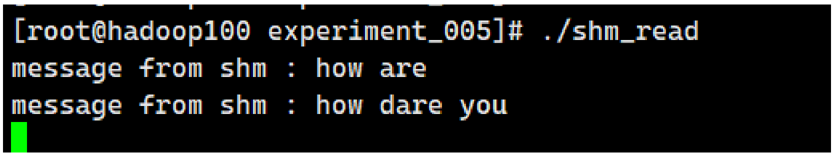
○ 运行代码



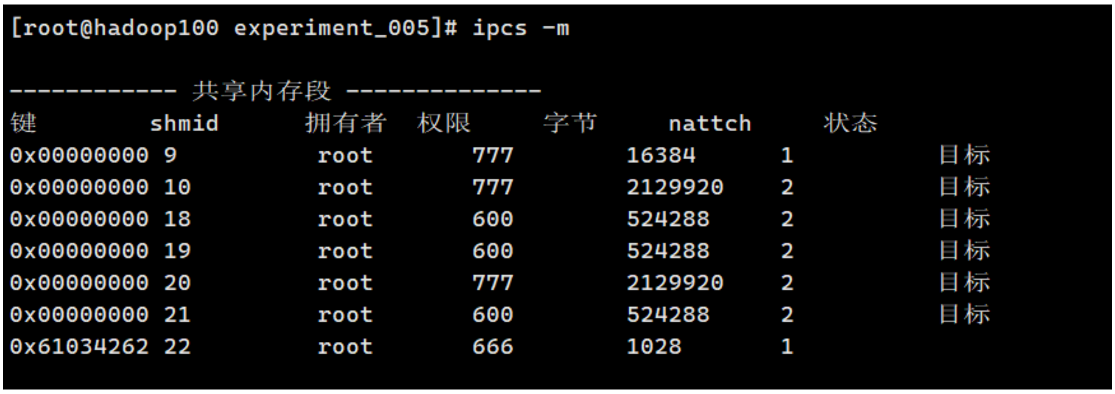
```
[root@hadoop100 experiment_005]# gcc shm_read.c -o shm_read
[root@hadoop100 experiment_005]# gcc shm_write.c -o shm_write
```



```
[root@hadoop100 experiment_005]# ./shm_write
please input : how dare you
```



```
[root@hadoop100 experiment_005]# ./shm_read
message from shm : how are
message from shm : how dare you
```



```
[root@hadoop100 experiment_005]# ipcs -m

------------ 共享内存段 --------------
键          shmid      拥有者  权限     字节      nattch     状态
0x00000000 9          root       777        16384      1               目标
0x00000000 10         root       777        2129920    2               目标
0x00000000 18         root       600        524288     2               目标
0x00000000 19         root       600        524288     2               目标
0x00000000 20         root       777        2129920    2               目标
0x00000000 21         root       600        524288     2               目标
0x61034262 22         root       666        1028       1
```

4. 附录(如源程序)

代码见上方截图