# 实验三 进程间通信

# 实验目的

1.熟悉有关Linux 中进程通信的系统调用。

2.学习有关Linux 的进程创建，理解进程创建后两个并发进程的执行。

3.掌握管道、消息缓冲等进程通信方法并了解其特点和使用限制。

# 实验内容

1. Linux 消息缓冲通信

编写一个程序，实现以下功能。给出源程序代码和运行结果。父进程创建一个消息队列和一个子进程，由子进程发送消息，父进程等待子进程结束后接收子进

程发来的消息，并显示消息内容。以“end”作为结束消息。

2. 编写进程

分别编写发送进程和接收进程，由发送进程发送消息，接收进程接收消息。采用先执行发送进程后执行接收进程的方式同步。以“end”作为结束消息。

3.编写一段程序，使用系统调用fork( )创建两个子进程，再用系统调用signal( )

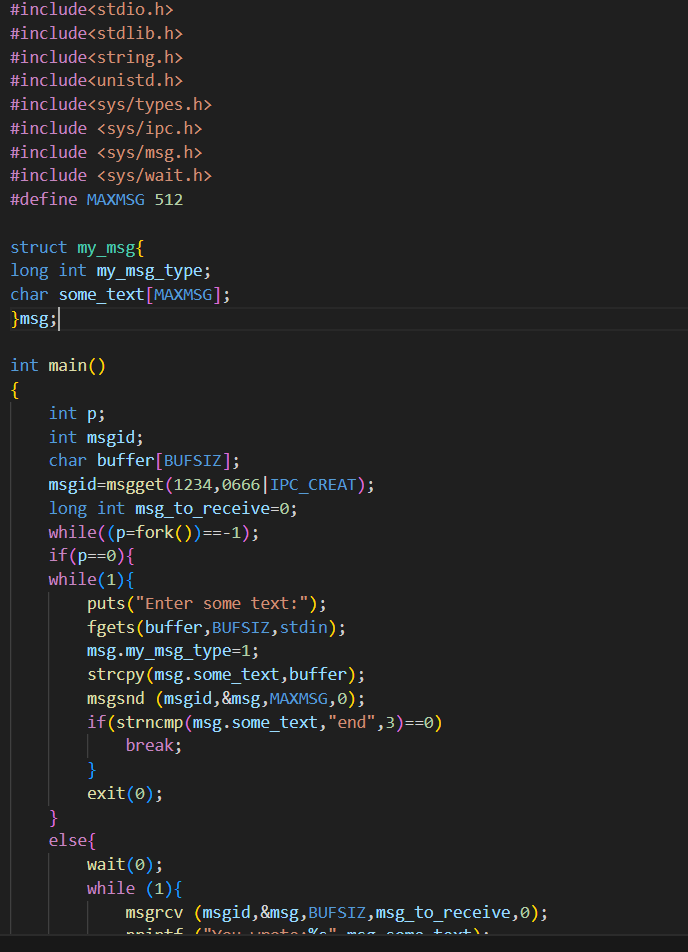
让父进程捕捉键盘上的中断信号（即按ctrl+c 键），当捕捉到中断信号后，父进程用系统调用kill()向两个子进程发出信号，子进程捕捉到信号后，分别输出下列信息

后终止：Child process 1 is killed by parent !Child process 2 is killed by parent!

父进程等待两个子进程终止后，输出以下信息后终止:Parent process is killed!

# 实验结果

1. Linux 消息缓冲通信

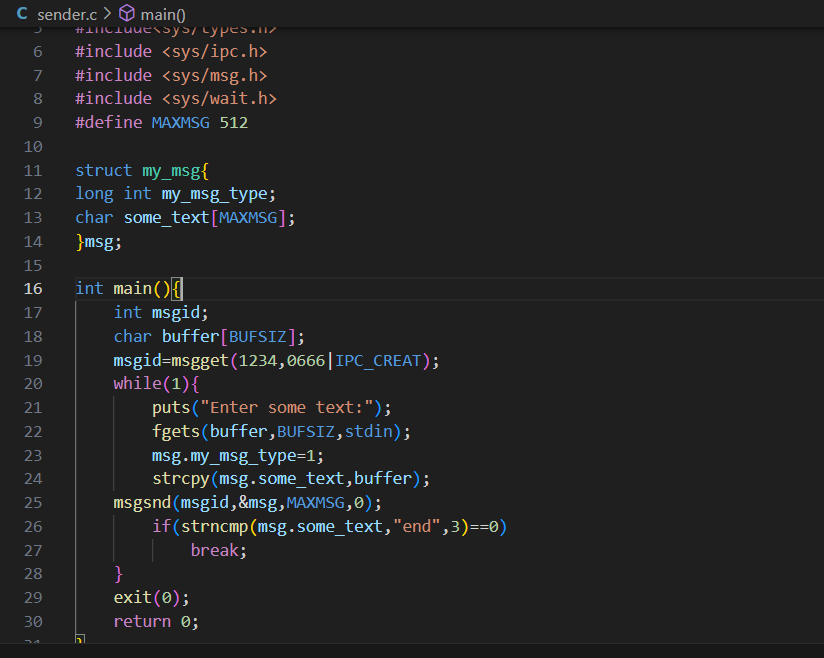


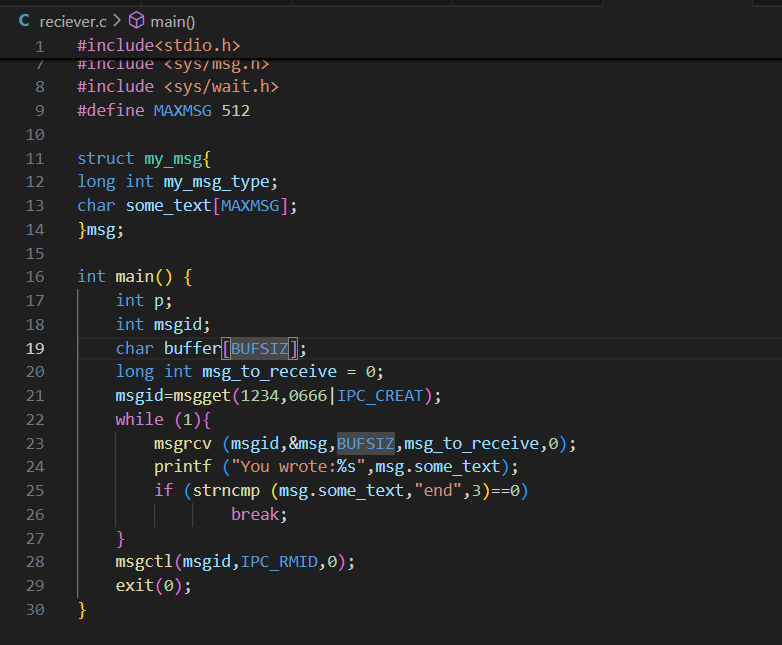
实验程序



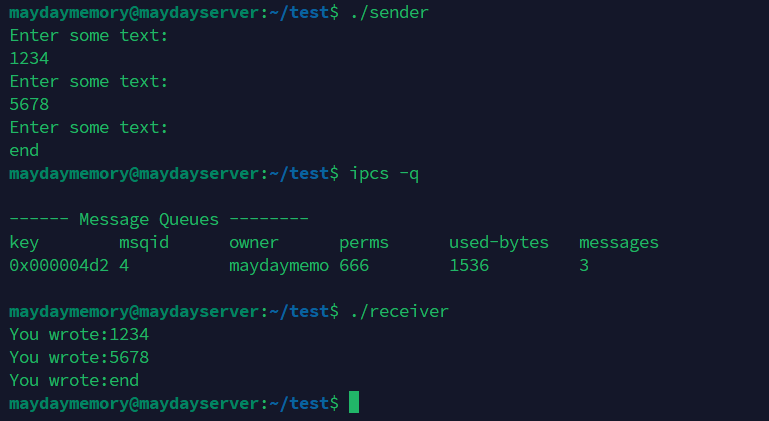
运行结果，父进程正确显示了子进程发来的消息，并显示消息内容

2. 编写进程

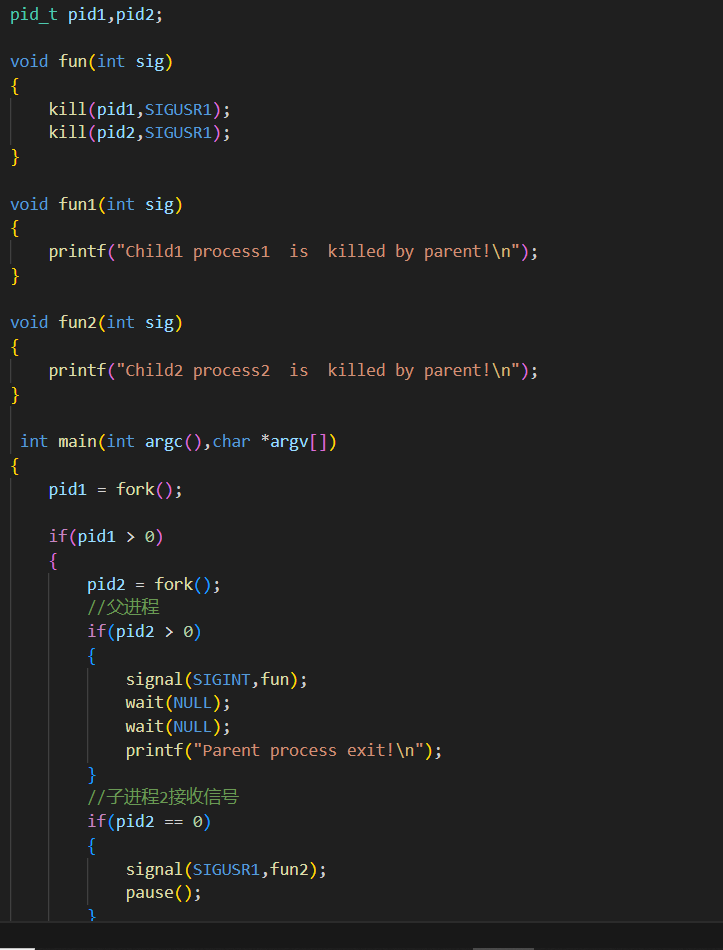




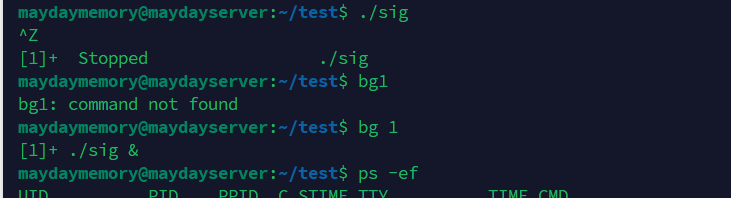
编写sender和receiver程序

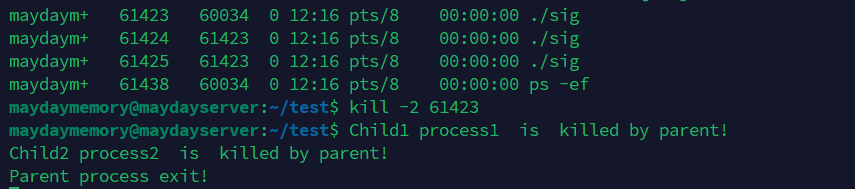


用ipcs查看了消息队列，并且进程间正确通讯



程序代码





运行程序并后台运行，然后使用ps-ef查找进程id，并用kill发送SIGINT信号（即按ctrl+c产生的信号），输出了正确的信息。

# 实验小结

本次实验主要围绕Linux系统中的进程间通信（IPC）展开，通过消息队列和信号机制实现了不同进程之间的信息传递与控制。在第一个实验中，父进程创建消息队列并派生子进程，子进程发送消息，父进程接收并显示，使用“end”作为结束信号，展示了消息队列在进程间同步中的有效性。第二个实验通过独立的发送和接收进程，增强了程序的模块性，同样使用消息队列并确保发送进程在接收进程之前启动。最后，在第三个实验中，通过fork()创建两个子进程，并在父进程中捕获SIGINT信号，当捕获到信号时，父进程向子进程发送终止信号，子进程输出相应信息后退出，这一过程展示了如何优雅地处理进程控制。通过这些实验，我们不仅掌握了基本的IPC方法，还深化了对Linux进程管理和信号处理的理解，为实现复杂的多进程应用打下了坚实基础。