# CS302
# Operating System
# Lab 3

# Process2

HUANG Bo, ZENG Xinxun, ZHANG Shiqi

# Outline

- Recap Exercise on Lecture
- Lab2 report review
- Linux process organization

# Exercise on Lecture : fork() + exec()

- Try the code **fork_exec.c** see what happens

# Exercise on Lecture : fork() + exec() + wait()

- Try the code **fork_exec2.c** see what happens

# Exercise on Lecture : fork() detail

- All process info can be seen in /proc:
  - Compile and run fork_detail.c
  - Try cpid and ppid on “less /proc/pid/status”

```
int getMemory(){ //Note: this value is in MB!
   FILE* file = fopen("/proc/self/status", "r");
   int result = -1;
   char line[128];
   while (fgets(line, 128, file) != NULL){
       if (strncmp(line, "VmSize:", 6) == 0){
           result = parseLine(line);
           break;
       }
  }
   fclose(file);
   return result/1024;
}
```

# Exercise on Lecture : Write to same file

- How can parent and child process write to same file?
  - Compile and run share_file_fwrite.c
  - Compile and run share_file_write.c

# Exercise on Lecture : Zombie

- Compile and run wait_exit.c
- Look the status of the child process

```c
int main(void)
{
    int pid;
    if( (pid = fork()) !=0 ) {
        printf("Look at the status of the child process %d\n", pid);
        while( getchar() != '\n' );
        wait(NULL);
        printf("Look again!\n");
        while( getchar() != '\n' );
    }
    return 0;
}
```

"enter" here

"enter" here

This program requires you to type "enter" twice before the process terminates.

You are expected to see **the status of the child process changes (ps aux [PID])** between the 1st and the 2nd "enter".

```
chuang@master:~$ ps 5814
  PID TTY      STAT   TIME COMMAND
 5814 pts/2    Z+     0:00 [wait_exit] <defunct>
chuang@master:~$ ps 5814
  PID TTY      STAT   TIME COMMAND
chuang@master:~$
```

# Lab2 report review

- How to realize inter-process communication
  - Pipe, named pipe(FIFO)
  - Message queue
  - Signal
  - Semaphore
  - Shared memory
  - Socket

What's different between PIPE, FIFO and MQ?

# Pipe vs FIFO

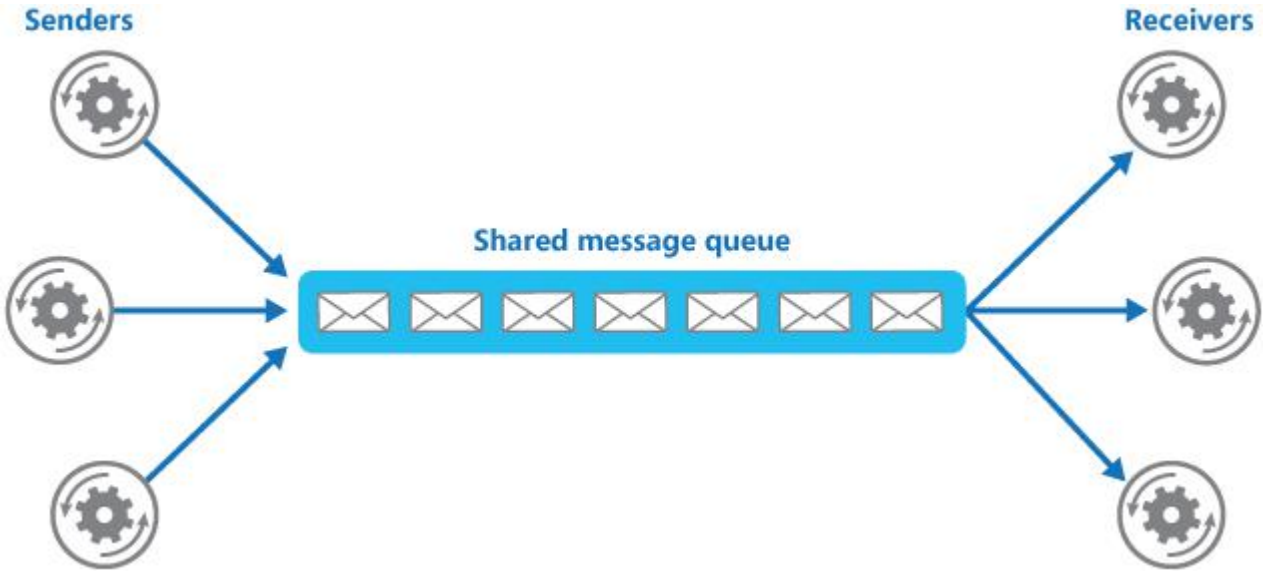
| PIPE | FIFO |
| --- | --- |
| PIPE is un-named IPC object | FIFO is named IPC object |
| PIPE doesn’t exist in the filesystem, vanish after program exit or one of end closed. | FIFO exists in the filesystem. Even the program exit, FIFO file remain till system reboot. |
| PIPE can only communicate among the related process (parent and child, sibling). | FIFO can be used for unrelated process. Even not in a local computer (in network). |
| PIPE is often used between two processes. | FIFO is often used in multiple processes communicating. |
| PIPE no control over ownership and permission. | FIFO, as a file, need to control ownership and permission. |

# FIFO demonstration

- Try mkfifo myfifo
- Try file myfifo
- Try echo hello > myfifo
- Try cat myfifo (in another terminal)

# What is Message queue

Message queue is a component provides an asynchronous communications protocol, letting that multiple senders and receivers can interact asynchronously but in correct order .




POSIX MQ
RabbitMQ
RocketMQ
WEIXINMQ

# FIFO vs Message queue

| FIFO | MQ |
| --- | --- |
| Synchronous | Asynchronous |
| Only bytes steam | User define struct |
| Unconditionally read | Selectively read |
| Both local and remote | Both local and remote * |
| In memory, cannot recover | Can be in memory and hard disk * |
| Content can be read only once | Content can be read until be removed * |

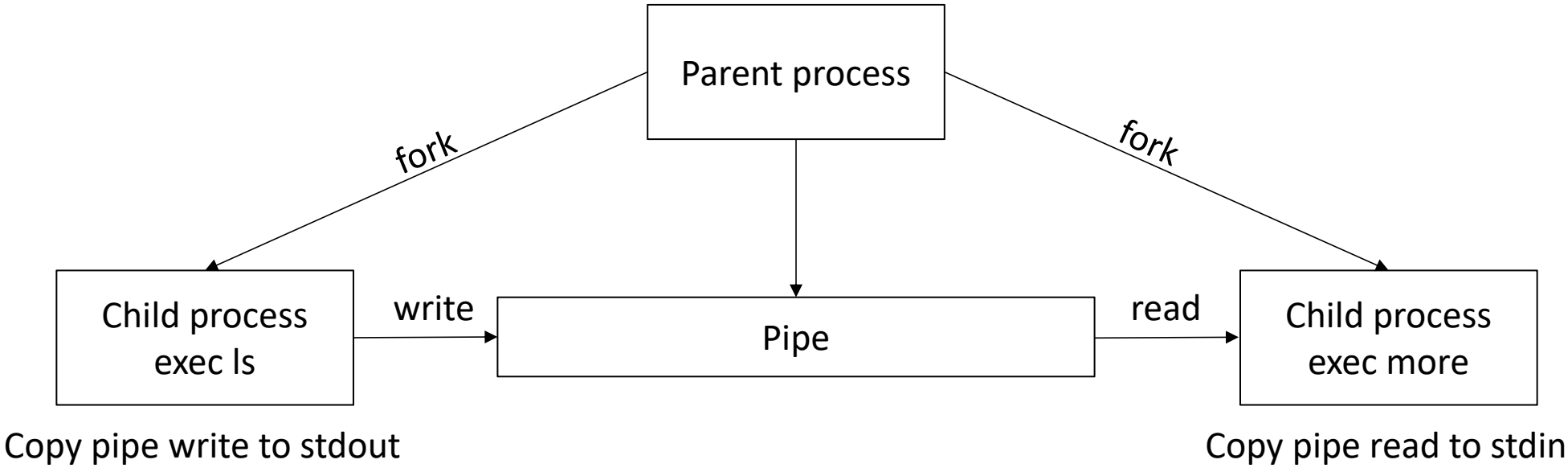
The * means POSIX MQ does not provide these characters, but other MQ can do that.

# MQ demonstration

- Try the code **mq_write.c and mq_read.c** see what happens
- Try multiple writer and reader see what happens.

# Lab2 report review

- How to realize inter-process connection



# Lab2 report review

- process.c, debug and analysis

1. Why cannot open vi?

```
if (!cpid)
{
    fprintf(stdout,"ID(child)=%d\n",getpid());
    /* 使子进程所在的进程组成为前台进程组，然后执行vi */
    setpgid(0,0);
    tcsetpgrp(0,getpid());
    execvp("/bin/vi","vi",NULL);
    exit(-1);
}
```

```
if (!cpid)
{
    fprintf(stdout,"ID(child)=%d\n",getpid());
    /* 使子进程所在的进程组成为前台进程组，然后执行vi */
    setpgid(0,0);
    tcsetpgrp(0,getpid());
    execlp("vi","./process",NULL);
    exit(-1);
}
```

2. Why parent process will be suspended?

**DESCRIPTION**

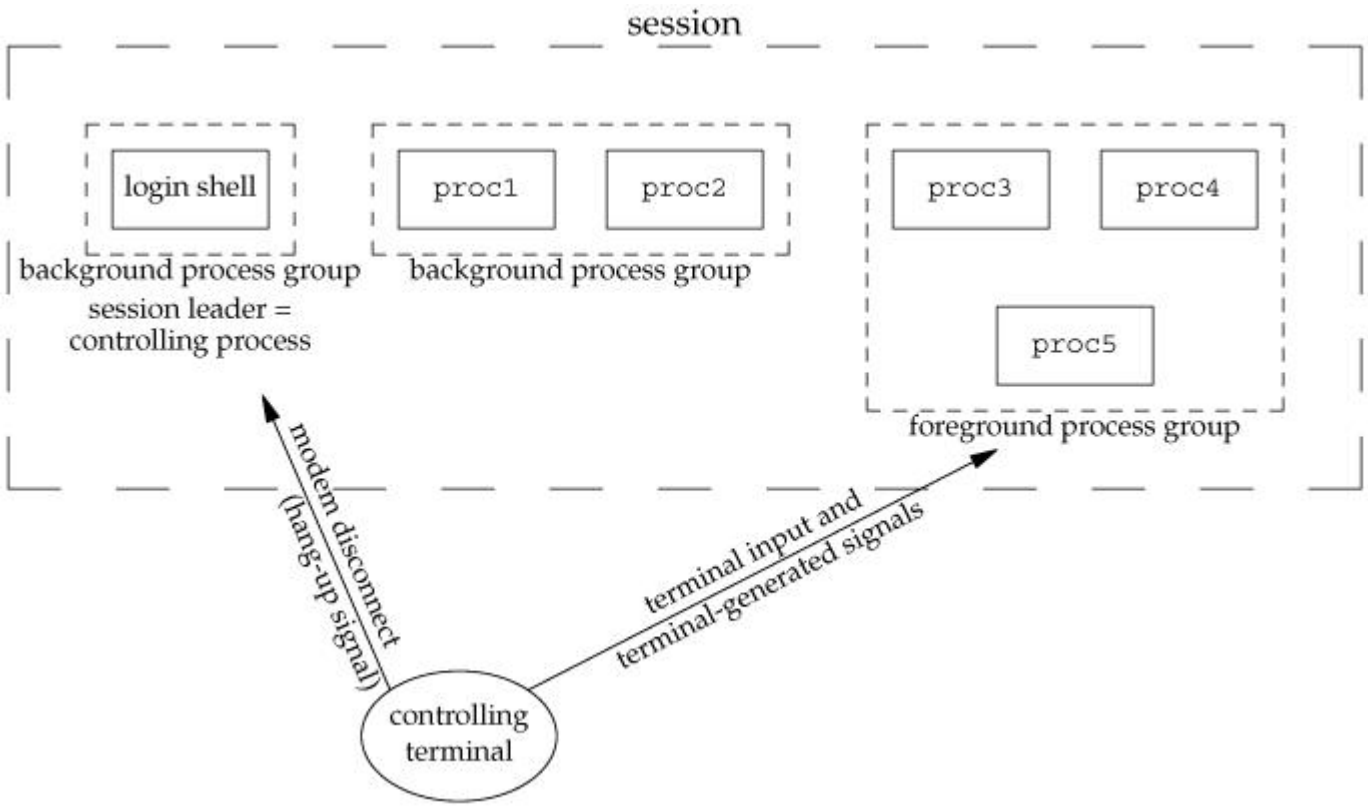
The function **tcgetpgrp**() returns the process group ID of the foreground process group on the terminal associated to fd, which must be the controlling terminal of the calling process.

The function **tcsetpgrp**() makes the process group with process group ID pgrp the foreground process group on the terminal associated to fd, which must be the controlling terminal of the calling process, and still be associated with its session. Moreover, pgrp must be a (nonempty) process group belonging to the same session as the calling process.

If **tcsetpgrp**() is called by a member of a background process group in its session, and the calling process is not blocking or ignoring **SIGTTOU**, a **SIGTTOU** signal is sent to all members of this background process group.

# Linux process organization

- Process Group
- Session
- Controlling Terminal
- Job Control



# Process Groups

- **A process group** is a collection of (related) processes.
- It usually associated with the same job that can receive signals from the same terminal.
- Each group has a process group ID.
- Each group has a group leader who pid = pgid

www.cs.fsu.edu/~xyuan/cop5570/lect7_session.ppt

# Process Groups

- To get the group ID of a process:
  - pid_t getpgrp(void): return the pgpid of calling process
  - pid_t getpgid(pid_t pid): return the pgpid of given process
  - ***getpgid(0)*** is equivalent to ***getpgrp()***

- A signal can be sent to the whole group of processes.
  - int killpg(int pgrp, int sig);
  - If pgrp is 0, killpg() sends sig to process group of calling process

www.cs.fsu.edu/~xyuan/cop5570/lect7_session.ppt

# Process Groups

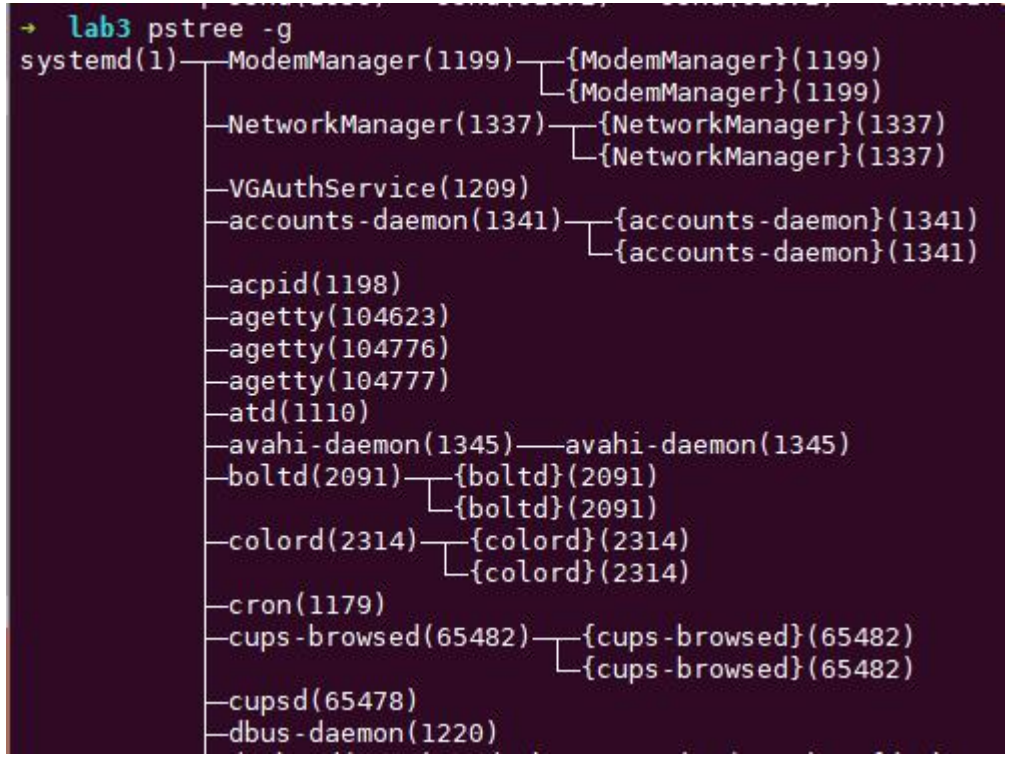
- To set the group ID:
  - pid_t setpgid(pid_t pid, pid_t pgid)
    - Set the process group ID of pid to pgid
    - If the two arguments are equal, the process with *pid* becomes a process **group leader**.
  - pid_t setpgrp(void)
    - Set the group ID of calling process to its process ID
    - Equavalent to
  - A process can set group ID of itself or its children

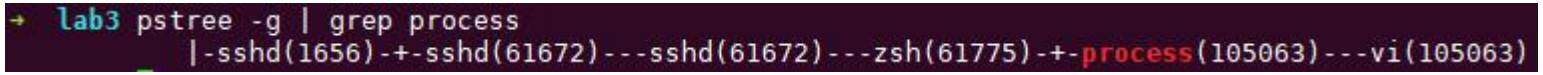
www.cs.fsu.edu/~xyuan/cop5570/lect7_session.ppt

# pstree command

- the pstree command displays a tree of processes.
- Try “man pstree” to see the options

- pstree –g: show process group id
- pstree –g | grep process



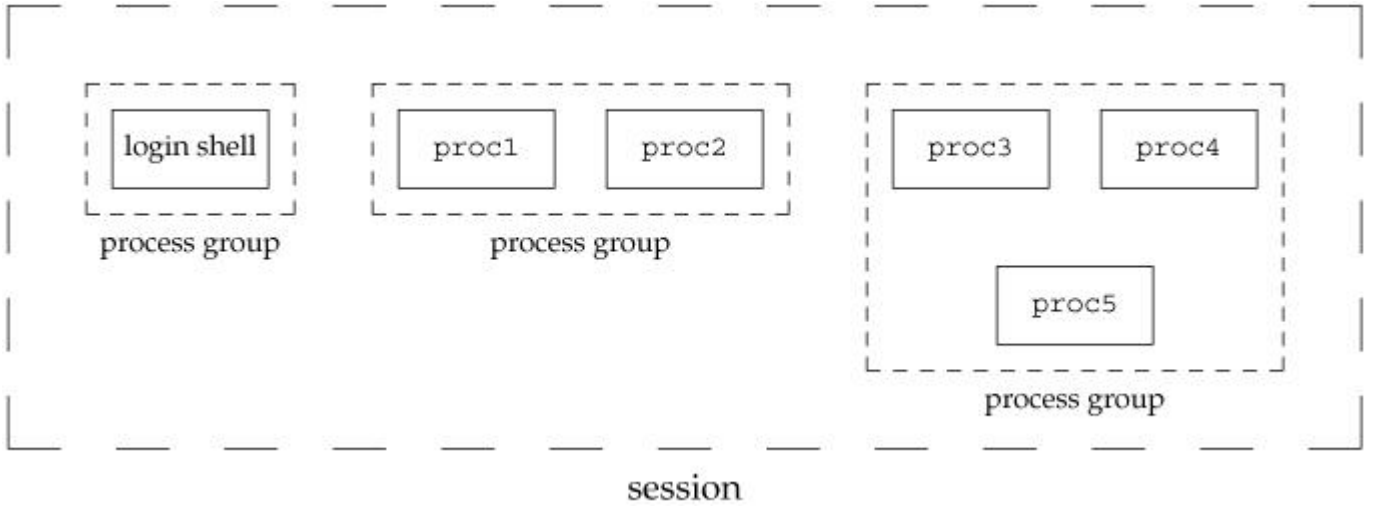
# Controlling Terminal

- The controlling terminal is a **terminal** that
  - controls processes by sending signals to them
  - provide I/O operation
- Two types of controlling terminals:
  - regular terminal devices
    - Ctrl-Alt-F1 to F7(TTY1-TTY6 and exit)
  - pseudo-terminal devices (e.g. terminal, Xshell)
    - Use ‘ps’ command, there are pts/* in TTY column

www.cs.fsu.edu/~xyuan/cop5570/lect7_session.ppt

# Sessions

- Sessions
  - A session is one or more process groups




  - login shell is the first process that executes under our user ID
    - A new session begins after login
    - A session terminate after logout(close terminal or type exit)

http://poincare.matf.bg.ac.rs/~ivana/courses/ps/sistemi_knjige/pomocno/apue/APUE/0201433079/ch09lev1sec5.html

# Sessions

- To establish a new session:

  pid setsid(void);

  - Calling process become the session leader
  - Calling process become a new group leader of a new group
  - Calling process has no controlling terminal (break up the old one)
    - Each login shell is a session. When a shell is created, a terminal must be setup.

www.cs.fsu.edu/~xyuan/cop5570/lect7_session.ppt

# Sessions and Controlling Terminal

- A session can have zero or a single controlling terminal
- The session leader that establishes the connection to the control terminal is called the *controlling process*.

www.cs.fsu.edu/~xyuan/cop5570/lect7_session.ppt

# Sessions and Process Group

- A session contains several process groups
  - One foreground process group
  - Many background process groups
  - Input
    - Only foreground group
    - Terminals' interrupt signals are only sent to the processes in the foreground group.
  - Output
    - Typically shared

www.cs.fsu.edu/~xyuan/cop5570/lect7_session.ppt

# Process and Job

- **Process**: an execution of certain program with its own address space

- **Jobs**: processes which are started by a **shell**
  - You can Ctrl+C, Ctrl+Z, bg, fg…

# Job Control

- Allows to start multiple jobs from a single terminal and control which job can access the terminal.
- Foreground jobs can access terminal.
- Background jobs may not:
  - When a background job try to read, SIGTTIN signal is sent.
  - When a background job try to change its mode, SIGTTOU signal is sent.