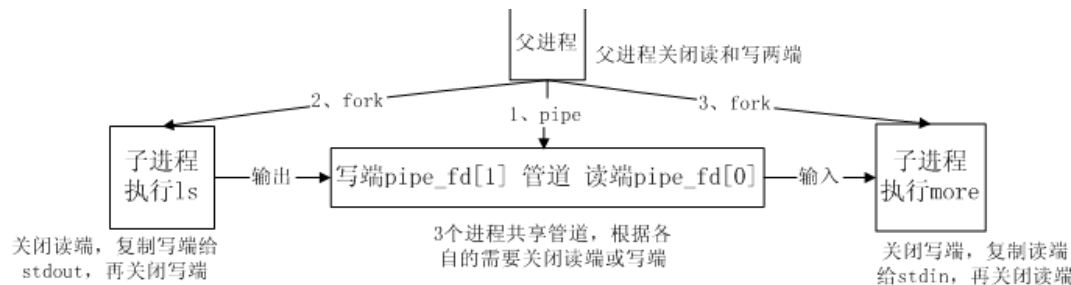


Lab 3

Dongping Zhang
2018.3.21

LAB 2 REPORT

- ✗ 进程间的通信：指进程之间相互交换信息的过程，以实现数据传输、共享数据、事件通知、资源共享和进程控制。通过一些相应的通信机制（包括共享存储区通信机制、消息传递通信机制和管道通信机制）来实现进程间的通信
- ✗ 进程间的连接：进程用pipe命令进行创建，用fork与子进程共享管道，用read和write传递数据



LAB 2 REPORT

- ✗ `int execvp(const char *file, char * const argv []);`
- ✗ `execvp()` 会从 `PATH` 环境变量所指的目录中查找符合参数 `file` 的文件名, 找到后便将第二个参数 `argv` 传给该文件并执行

EXEC()

Member name	Using pathname	Using filename	Argument List	Argument Array	Original ENV	Provided ENV
exec1()	YES		YES		YES	
exec1p()		YES	YES		YES	
execle()	YES		YES			YES
execv()	YES			YES	YES	
execvp()		YES		YES	YES	
execve()	YES			YES		YES

LAB 2 REPORT

- ✗ fork.c: 等效于Shell的命令 “/bin/ls -l”
- ✗ pipe.c: 通过赋值进程启动时的参数/bin/ls -l /etc/ 和 /bin/more来通过管道实现进程间通信。等同于: \$ls-l/etc>temp (输出重定向到temp) , \$more<temp (或者more temp) , \$ rm temp (删除临时文件)
 - + 由上结果可以看execvp(prog2_argv[0],prog2_argv)执行了

LAB 2 REPORT

✗ signal.c

- + 创建了一对父子进程，接收改变信号，并通过 `sigaction` 来改变进程接收到特定信号后的行为
- + `pid_t wait(int *status);`
- + `pid_t waitpid(pid_t pid, int *status, int options);`
- + `wait` 函数是 `waitpid` 函数的一个特例。 `waitpid(-1, &status, 0);`

LAB 2 REPORT

✖ process.c

```
19     if (cpid<0)
20         exit(-1);
21
22     if (!cpid)
23     {
24         fprintf(stdout, "ID(child)=%d\n", getpid());
25
26         //         setpgid(0,0);
27         //         tcsetpgrp(0,getpid());
28         execl("/bin/vi", "vi", NULL);
29         exit(-1);
30     }
31
32     fprintf(stdout, "ID(parent)=%d\n", ppid);
33     //     setpgid(cpid,cpid);
34     //     tcsetpgrp(0,cpid);
35     waitpid(cpid, NULL, 0);
36     //     tcsetpgrp(0,ppid);
37
```

EXEC()

Pathname VS Filename

/home/tywong/os/example.c

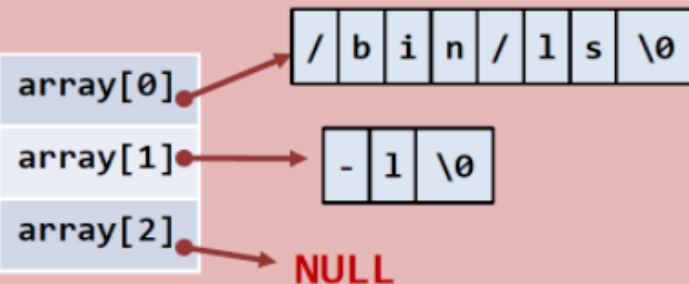
The entire string is a pathname.

The last part is a filename.

Argument list VS Argument array

```
execv("/bin/ls", array);
```

```
execl("/bin/ls",  
"/bin/ls", "-l", NULL);
```



The Command: `"/bin/ls -l"`

理论课回顾

✗ time_1.c time_2.c

```
$ time ./time_example
```

```
real    0m0.001s
user    0m0.000s
sys     0m0.000s
$ _
```

```
$ time ./time_example
```

```
real 0m2.795s
user 0m0.084s
sys 0m0.124s
$ _
```

See? Accessing hardware costs the process more time.

```
int main(void) {
    int x = 0;
    for(i = 1; i <= 10000; i++) {
        x = x + i;
        // printf("x = %d\n", x);
    }
    return 0;
}
```

Commented on purpose.

```
int main(void) {
    int x = 0;
    for(i = 1; i <= 10000; i++) {
        x = x + i;
        printf("x = %d\n", x);
    }
    return 0;
}
```

Comment released.

理论课回顾

理论课回顾

✗ Time_example_slow.c time_example_fast.c

- ◆ When writing a program, you must consider both the user time and the sys time.
 - ◆ E.g., the output of the following two programs are exactly the same. But, their running time is not.

```
#define MAX 1000000

int main(void) {
    int i;
    for(i = 0; i < MAX; i++)
        printf("x\n");
    return 0;
}
```

```
#define MAX 1000000

int main(void) {
    int i;
    for(i = 0; i < MAX / 5 ; i++)
        printf("x\nx\nx\nx\nx\n");
    return 0;
}
```

理论课回顾

✗ Time_example_slow.c time_example_fast.c

```
#define MAX 1000000
```

```
int main(void) {  
    int i;  
    for(i = 0; i < MAX; i++)  
        printf("x\n");  
    return 0;  
}
```

```
$ time ./time_example_slow
```

```
real 0m1.562s  
user 0m0.024s  
sys 0m0.108s  
$ _
```

```
#define MAX 1000000
```

```
int main(void) {  
    int i;  
    for(i = 0; i < MAX / 5; i++)  
        printf("x\nx\nx\nx\nx\n");  
    return 0;  
}
```

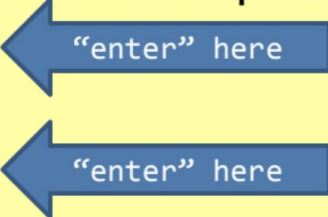
```
$ time ./time_example_fast
```

```
real 0m1.293s  
user 0m0.012s  
sys 0m0.084s  
$ _
```

理论课回顾

✗ wait_and_exit.c

```
1 int main(void)
2 {
3     int pid;
4     if( (pid = fork()) !=0 ) {
5         printf("Look at the status of the child process %d\n", pid);
6         while( getchar() != '\n' );
7         wait(NULL);
8         printf("Look again!\n");
9         while( getchar() != '\n' );
10    }
11    return 0;
12 }
```



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”.

理论课回顾

```
1 int main(void) {
2     int i;
3     if(fork() == 0) {
4         for(i = 0; i < 5; i++) {
5             printf("(%d) parent's PID = %d\n",
6                 getpid(), getppid() );
7             sleep(1);
8         }
9     }
10    else
11        sleep(1);
12    printf("(%d) bye.\n", getpid());
13 }
```

`getppid()` is the system call that returns the parent's PID of the calling process.

```
$ ./reparent
(1235) parent's PID = 1234
(1235) parent's PID = 1234
(1234) bye.
$ (1235) parent's PID = 1
(1235) parent's PID = 1
(1235) parent's PID = 1
(1235) bye.
$ _
```

作业调度

实验目的

- ✘ 理解操作系统中调度的概念和调度算法。
- ✘ 学习Linux下进程控制以及进程之间通信的知识。
- ✘ 理解在操作系统中作业是如何被调度的，如何协调和控制各个作业对CPU的使用。

- ✘ 程序：静态的指令集合，不占用系统的运行资源，可以长久保存在磁盘
- ✘ 进程：进程实体（程序、数据和进程控制块构成）的运行过程，是系统进行资源分配和调度的一个独立单位。

相关基础知识——程序与进程2

- ✘ 进程执行程序，但进程与程序之间不是一一对应的。通过多次运行，同一程序可以对应多个进程(fork 父子进程)；通过调用关系，一个进程可以包含多个程序（一个DLL 文件可一被多个程序运用）。

相关基础知识——进程与作业

- ✕ 作业由一组统一管理和操作的进程集合构成，是用户要求计算机系统完成的一项相对独立的工作。
- ✕ 作业可以是完成了编译、链接之后的一个用户程序
- ✕ 作业也可以是各种命令构成的一个脚本

处理器调度:

- ✗ 用户作业从提交给系统开始，直到运行结束退出系统为止，将经历高级调度、中级调度和低级调度

✕ **高级调度**，又称**作业调度**，不涉及处理机的分配，主要任务是按一定的原则从外存上处于后备状态的作业中挑选一个（或多个）**作业调入主存**，为其分配内存、I/O设备等必要的资源，并建立相应的进程，安排在就绪队列上，以使进程获得竞争处理机的权利。

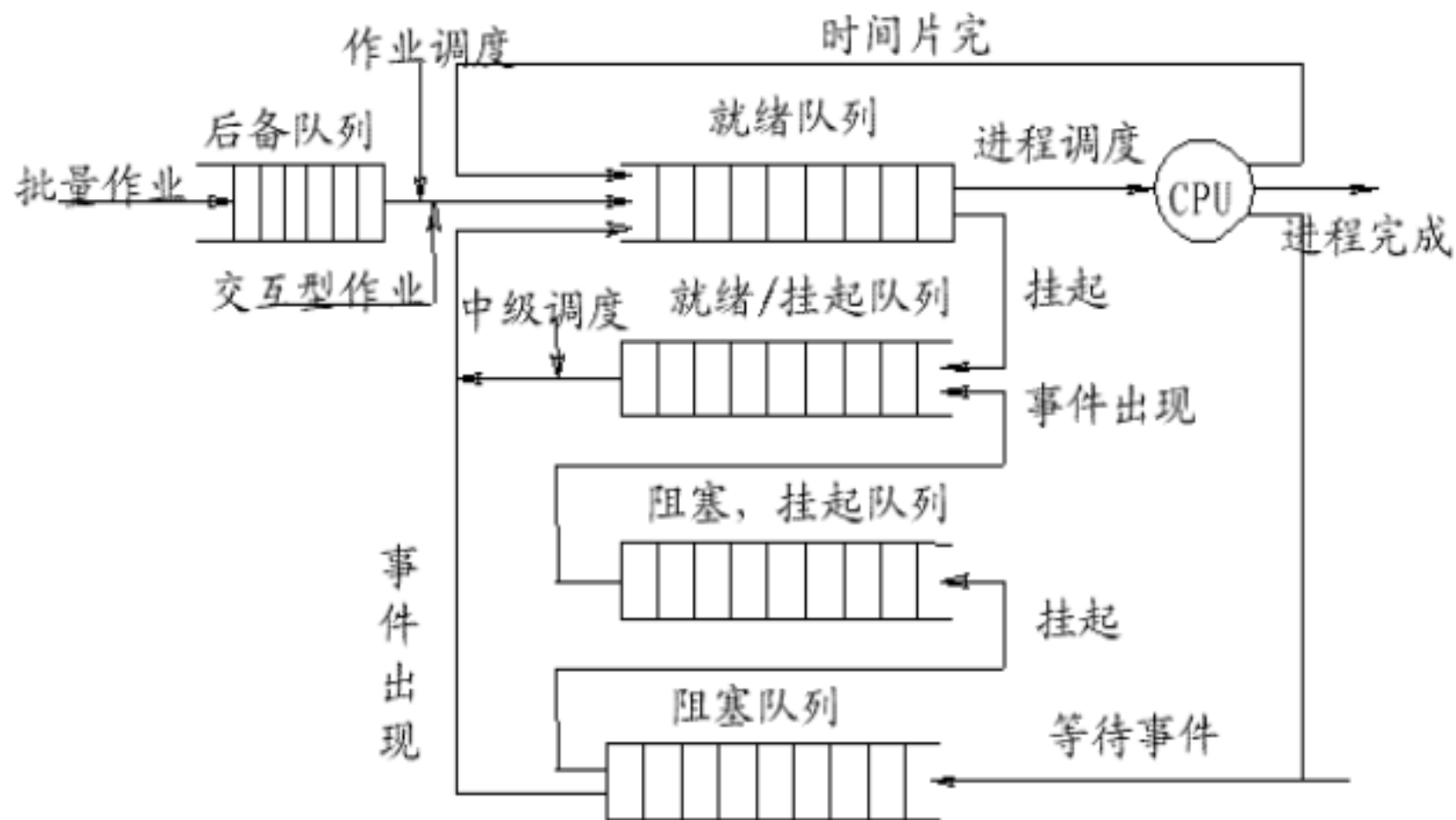
相关基础知识——处理器调度3

- ✘ **中级调度**，又称**交换调度**，是为了提高内存利用率和平衡系统负载而采取的一种利用外存补充内存的措施。中级调度实际上是**存储器管理的对换功能**。中级调度根据内存中能够接纳的进程数来平衡系统负载，起到在一定时间内平滑和调整系统负载的作用。
- ✘ 简而言之：中级调度是在换出内存的进程中确定需要进入内存的进程。

相关基础知识——处理器调度4

- ✘ **低级调度**，又称**进程调度**，是按照一定的调度算法从内存的就绪进程队列中选择进程，为进程分配处理器，避免进程对处理器竞争的方法。
- ✘ 在处理器的三级调度中，进程调度发生的频率最高。作业调度频率最低。中级调度主要用于内存管理。

处理器的三级调度模型



相关基础知识——处理器调度5

✧ 进程（作业）调度方式：

非抢占方式：当某一进程正在处理机上执行时，即使有某个更为重要或紧迫的进程进入就绪队列，该进程**不会让出处理机资源**，而是继续执行，直到进程完成或发生某事件而阻塞时，才把处理机分配给其他进程。

✕ **抢占方式**：当某一进程正在处理机上执行时，若有某个更为重要或紧迫的进程需要使用处理机，则**立即暂停正在执行的进程**，将处理机分配给这个更重要或紧迫的进程。

相关基础知识——系统调度7

- × 抢占方式遵循的原则：
- × **优先权原则**：允许**优先权高**的新到进程抢占当前进程的处理机
- × **短作业（进程）优先原则**：允许**估计运行时间短**的新到作业（进程）抢占当前较长作业（进程）的处理机

相关基础知识——系统调度8

✦ **时间片原则**：各进程按时间片运行，
当一个时间片用完后，便暂停该进
程的执行而重新进行调度。

相关基础知识——Linux进程间通信

本实验使用命名管道（FIFO）：

普通管道与命名管道

特点	普通管道	命名管道
使用限制	相关进程使用，如父子进程，单向通信，双向要创建两个管道	不相关的进程可以通信
文件名称	没有文件名	有文件名
数据读写	先进先出，read、write（或重定向读写）	先进先出，open、read、write
创建方式	调用 pipe 创建	调用 mkfifo、mknod 创建
删除方式	无需删除，用完会自动删除	调用 rm 或 unlink 命令

作业调度模型要求及特点1

基本特性：

- ✗ **时间片**为100毫秒，即作业每次执行的基本单位为100毫秒，在这100毫秒内，作业一直在执行，直到时间片到期或执行结束

作业调度模型要求及特点2

✕ 三种作业状态：

READY：作业准备就绪可以运行

RUNNING：作业正在运行

DONE：作业已经运行结束，可以退出。

✕ 设4个等级的优先级：0、1、2和3，3最高。

作业调度模型要求及特点3

每个作业有两种优先级：

- ✖ **初始优先级** (initial priority) ，在作业提交时指定，作业执行过程中保持不变；
- ✖ **当前优先级** (current priority) ，Scheduler总是选择当前优先级最高的作业来执行，作业执行过程中可变，更新的情况有两种：

作业调度模型要求及特点4

- 1、若作业在就绪队列中等待了100毫秒，则将它的前优先级加1（最高为3）
- 2、若当前运行的作业时间片到，使其暂停执行，将其放入就绪队列中，当前优先级恢复为初始优先级。

作业调度模型要求及特点5

- ✘ 建立一个就绪队列，每个提交的作业都放在就绪队列中，scheduler遍历该队列，找出优先级最高的作业让它执行。

作业调度模型要求及特点6

✕作业调度进程scheduler:

负责整个系统的运行，处理作业的入队、出队及状态查看请求，在合适的时间调度各作业运行。

具体是：

作业调度模型要求及特点7

- 1、如果有新的作业到来，为其创建一个进程，其状态为就绪，然后将其放入**就绪队列**中。
- 2、如果有出队请求则使该作业出队，然后清除相关的数据结构，若该作业当前正在运行，则发信号给它，使它停止运行，然后出队。

作业调度模型要求及特点8

3、如果是状态查看请求，则输出当前运行的作业及就绪队列中所有作业的信息。

作业调度模型要求及特点9

✕ 三个作业控制命令:

1、作业入队命令enq

给scheduler调度程序发出**入队请求**，将作业提交给系统运行。

Scheduler调度程序为每个作业分配一个唯一的jid（作业号）；为每个作业创建一个进程，并将其状态置为READY，然后放入就绪队列中。

作业调度模型要求及特点10

格式: `enq [-p num] e_file args`

`-p num`: 可选, 该选项指定作业的初始优先级

`e_file args`: `e_file`是可执行文件的
名字, `args`是可执行文件的参数。

作业调度模型要求及特点11

2、作业出队命令deq:

给scheduler调度程序发出一个出队请求

格式: deq jid

作业调度模型要求及特点12

3、作业状态查看命令stat:

在标准输出上打印出当前运行作业及就绪队列中各作业的信息，包括：

- ✗ 进程的pid;
- ✗ 作业提交者的user name;
- ✗ 作业执行的时间;
- ✗ 在就绪队列中总的等待时间;
- ✗ 作业创建的时刻;
- ✗ 此时作业的状态。

作业调度模型要求及特点13

采用**多级反馈的循环**（Round Robin）**调度策略**（时间片与优先级结合）：

- 1、调度程序以**时间片为单位**进行作业调度（在时间片内，即使有高优先级的作业到来，调度程序不会马上调度）

作业调度模型要求及特点14

- 2、每个作业有其**动态的优先级**，在用完分配的时间片后，可以被优先级更高的作业抢占运行。
- 3、等待队列中的作业**等待时间越长，其优先级越高**。

作业调度模型要求及特点15

- ✘作业调度命令及三个作业控制命令作为Linux系统的命令，也受**操作系统本身的进程调度**，本实验忽略考虑。

程序的实现——数据结构 1

✖ 作业信息结构:

```
struct jobinfo {  
    int    jid;           /* job id */  
    int    pid;           /* process id */  
    char** cmdarg;        /* the command &  
                           args to execute */  
    int    defpri;        /* default priority */  
    int    curpri;        /* current priority */  
};
```

程序的实现——数据结构2

```
int  ownerid;      /* the job owner id */
int  wait_time;    /* the time job in
                    waitqueue */

time_t create_time; /* the time job create */
int  run_time;      /* the time job running */
enum  jobstate state; /* job state */

};
```

程序的实现——数据结构3

✕ 作业调度命令结构:

```
struct jobcmd {  
    enum    cmdtype type;  
    int      argnum;  
    int      owner;  
    int      defpri;  
    char     data[BUFLen];  
};
```

程序的实现——数据结构4

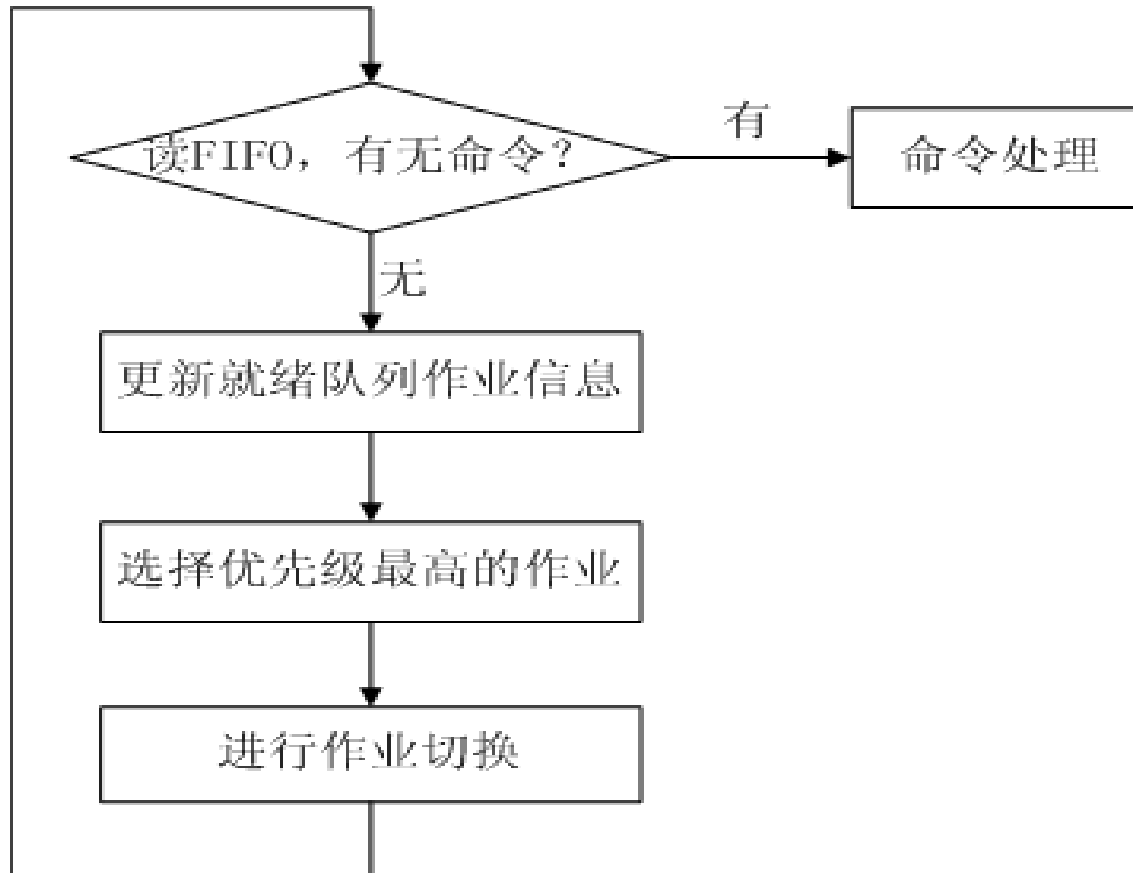
```
enum cmdtype {  
    ENQ = -1,  
    DEQ = -2,  
    STAT = -3  
};
```

程序的实现——数据结构5

✕就绪队列结构

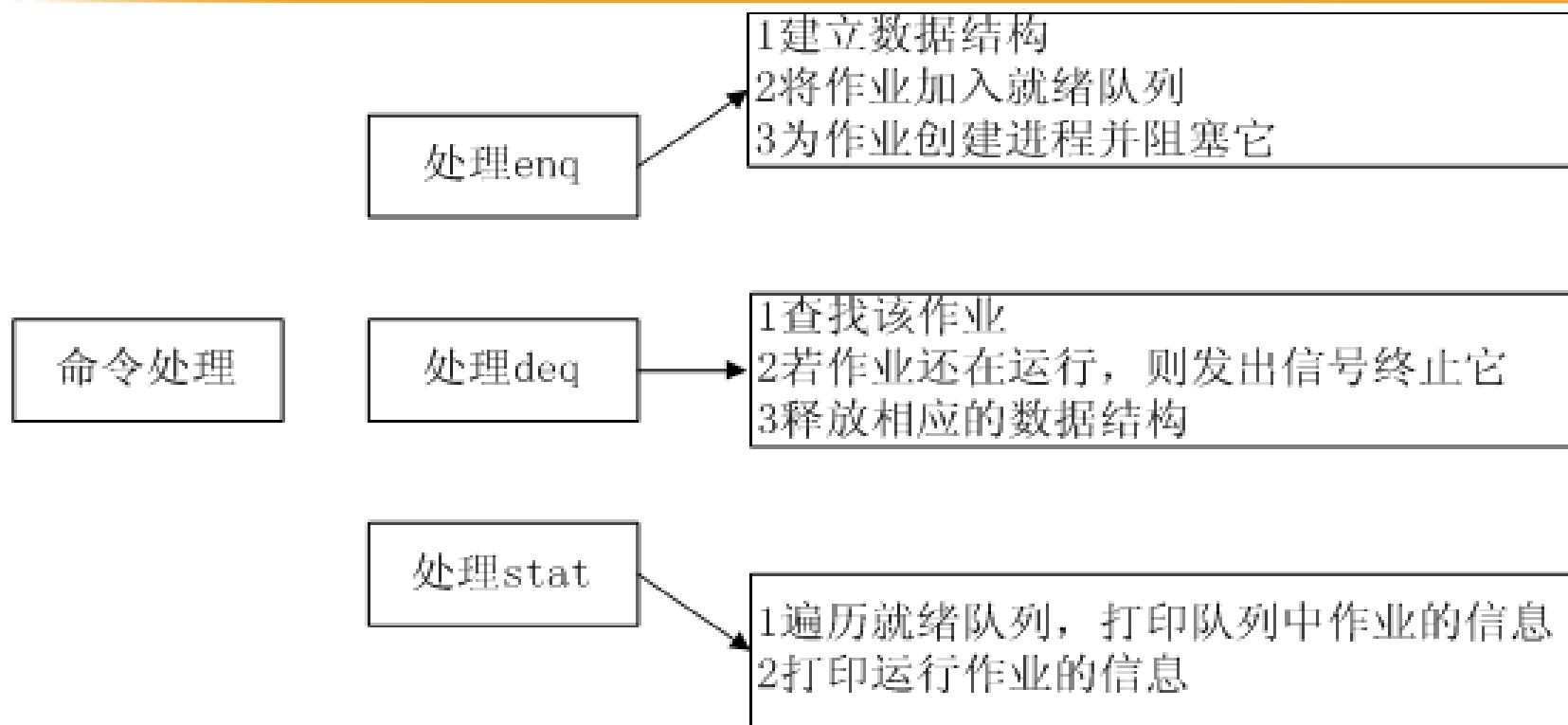
```
struct waitqueue {  
    /* double link list */  
    struct waitqueue *next;  
    struct jobinfo *job;  
};
```


作业调度程序的实现1



作业调度程序流程图

作业调度程序的实现2



命令处理示意图

调度进程与命令程序间的通信:FIFO

1、调度程序负责创建一个FIFO文件，

```
int mkfifo(const char* pathname,  
            mode_t mode)
```

2、命令程序负责把命令**按照struct
jobcmd格式**写进FIFO中，

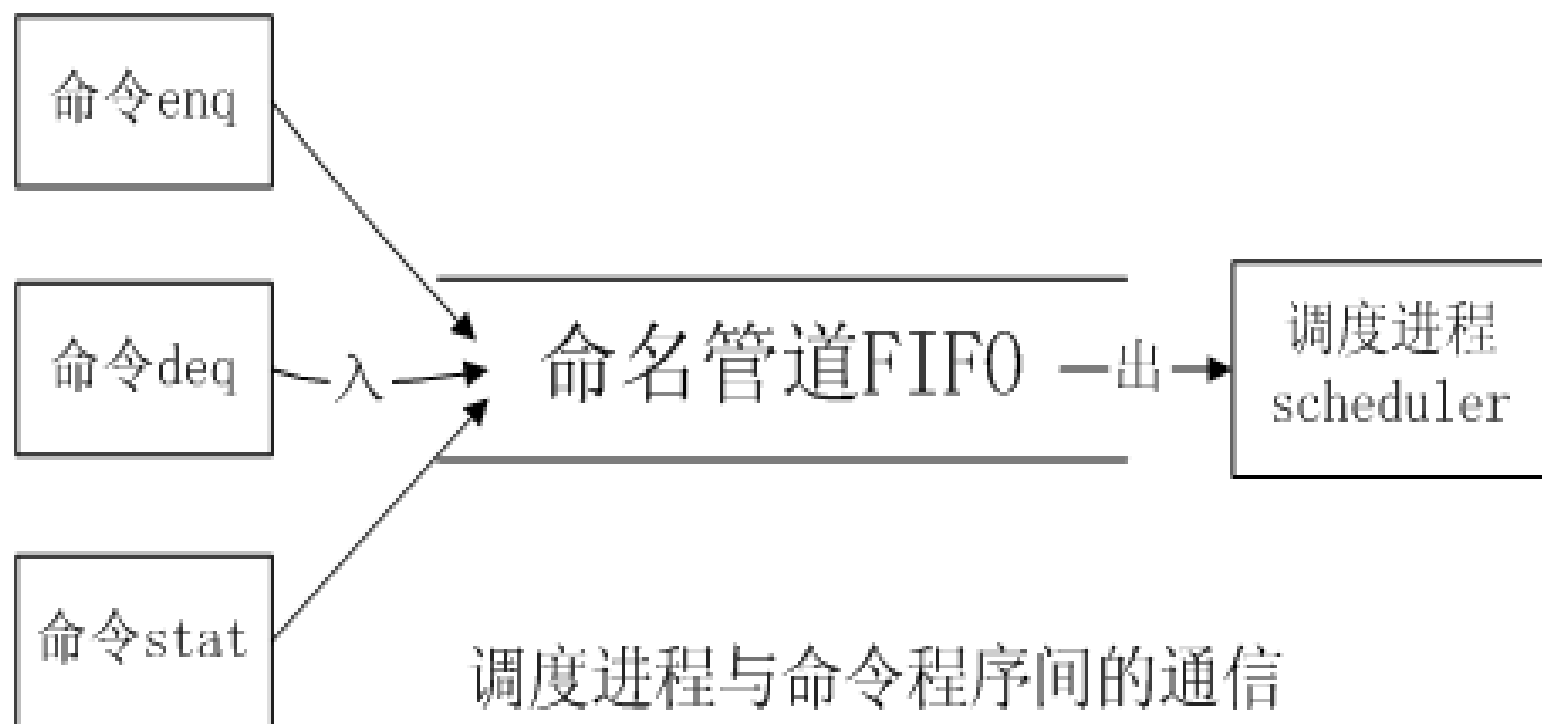
```
ssize_t write(int filedес, const void  
               *buff, size_t nbytes)
```

调度进程与命令程序间的通信:FIFO

3、调度程序从FIFO中读取用户提交的命令，

```
ssize_t read(int filedes, void *buf,  
             size_t nbytes)
```

调度进程与命令程序间的通信:FIFO



调度进程与提交的作业间的通信:信号1

1、通过重定向（使用dup2系统调用）把所提交作业的输出送往**重定向标准输出**后的文件（本实验是/dev/null），而非屏幕。

调度进程与提交的作业间的通信:信号2

2、使用函数**sigaction**作业调度主进程处理接收到的特定信号:

SIGVTALRM: 表示定时到期, 此时主进程调用函数**schedule**开始调度

调度进程与提交的作业间的通信:信号3

SIGCHLD: 表示子进程状态已改变
子进程由运行状态变为停止状态,
或由停止状态变为运行状态, 父进
程都会接收到信号SIGCHLD, 所以
父进程还需判别子进程是否退出,
用**waitpid**(-1, &status, **WNOHANG**)

调度进程与提交的作业间的通信:信号4

3、调度程序的作业切换：通过系统调用kill实现。

原型：int **kill**(pid_t pid, int signo)

signo：信号类型，可以为：

SIGSTOP——暂停进程的允许。

SIGCONT——使停止的进程继续运行

SIGKILL——终止一个进程

实验步骤1

1、 拷贝程序到**自己的学号目录**下
(一个头文件job.h, 四个源文件
scheduler.c、 enq.c、 deq.c、 stat.c) ,

实验步骤2

2、调用vi编辑器修改job.h文件，为命名管道FIFO设置正确的路径，即在语句：

```
#define FIFO "/home/SVRFIFO"
```

中加入学号路径为：

```
#define FIFO "/home/学号  
/SVRFIFO"
```

实验步骤3

3、修改scheduler.c文件，添加作业的打印信息，即修改do_stat函数，要求再输出作业名称、当前优先级、默认优先级；

实验步骤4

在

```
printf( "JID\tPID\tOWNER\tRUNTIME\t  
WAITTIME\tCREATETIME\t\tSTATE\n");
```

语句中添加JOBNAME、CURPRI、
DEFPRI

实验步骤5

接下来的两个输出语句根据表头修改，注意**printf**语句的输出格式，输出的信息内容参照**jobinfo**结构体。

实验步骤6

- 4、用gcc分别编译连接作业调度程序、三个命令程序。
- 5、在一个远程登录窗口中**运行作业调度程序(scheduler)作为服务端。**
- 6、提交一个运行时间超过100毫秒的作业（**要求提供源程序**），并编译连接。

头文件：#include <stdio.h>, #include <sys/wait.h>

函数：usleep(n) //n微秒

函数：Sleep (n) //n毫秒

函数：sleep (n) //n秒

实验步骤7

7、再打开一个窗口登录服务器作为**客户端**，在其中**运行作业控制命令**（提交作业、删除作业、查看信息），在服务端观察调度情况，分析所提交作业的执行情况。

-
- ✗ Please download the report template and report code and complete the report
 - ✗ Name your report as OS_Lab3_XXX_YYYYYYYYY.doc
 - ✗ Replace XXX with your name and replace YYYYYYYY with your student id
 - ✗ Such as OS_Lab3_ 张三 _12345678.doc
 - ✗ Check the blackboard for deadline