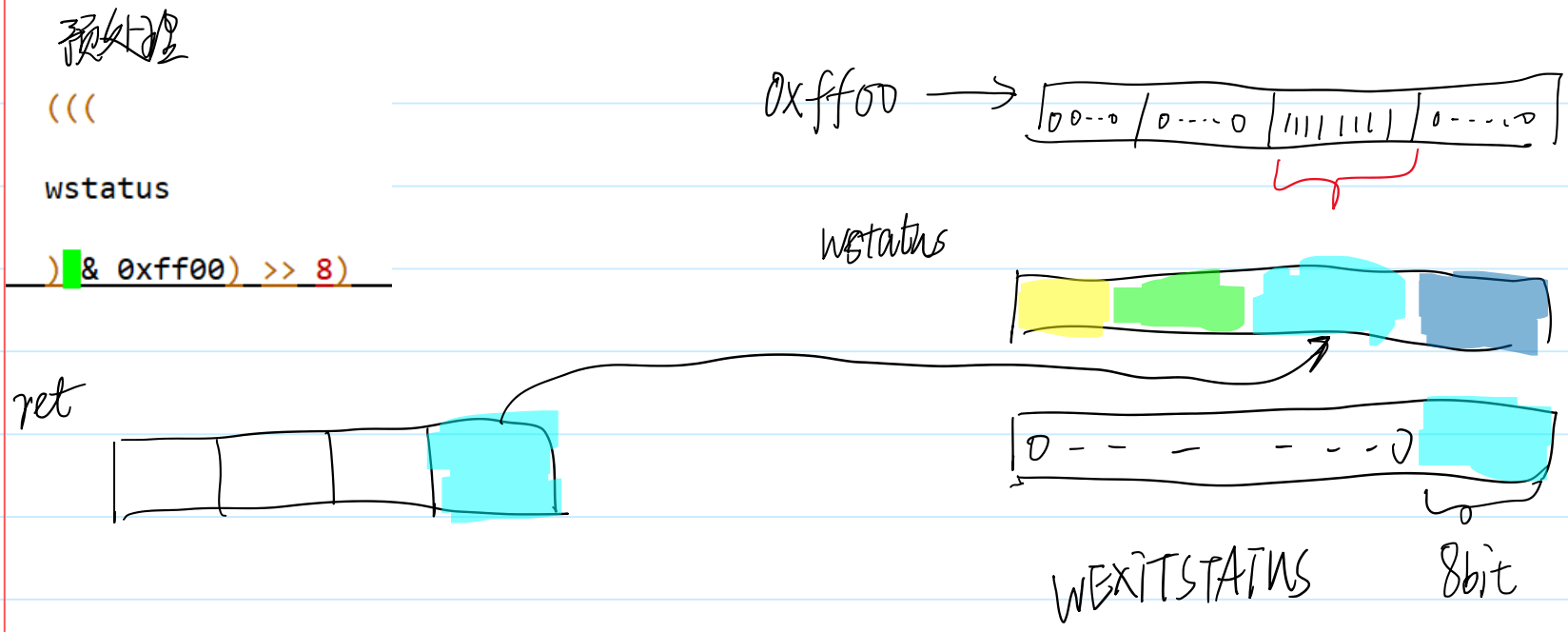
WEXITSTATUS的原理

2023年4月26日 9:55

预处理

```
(((
wstatus
)  & 0xff00) >> 8)
```



进程的终止

2023年4月26日 10:02

1. 正常 在main函数内return.

```c
int main()
{
    printf("You can see me!\n");
    return -1;
    printf("You can't see me!\n");
}
```

确定FILE.

2. 正常 在任何位置 exit

```c
#include <stdlib.h>

void exit(int status);

int test()
{
    //return -1;
    exit(-1);
}
int main(){
    printf("You can see me!\n");
    test();
    printf("You can't see me?\n");
}
```

```c
int main()
{
    printf("Hello");
    //return 0;
    exit(0);
}
```

_Exit _exit

2023年4月26日 10:15

3 正常 _exit /_Exit 不会清理FILE

```
#include <unistd.h>

void _exit(int status);

#include <stdlib.h>

void _Exit(int status);
```

```
#include <49func.h>
int main()
{
    printf("Hello");
    //return 0;
    //exit(0);
    _exit(0);
}
```

进程的异常终止

2023年4月26日 10:17

1. 进程被信号终止。

   Ctrl + C

   kill -9

   SIGPIPE

2. 进程自动放弃。

```
NAME
       abort - cause abnormal process termination

SYNOPSIS
       #include <stdlib.h>

       void abort(void);
```

```
[liao@ubuntu Linuxday_12]$ ./2_exit_abnormal
exit abnormally, term sig = 6
```

操作系统管理多进程

2023年4月26日 10:23

进程组：进程的集合

组ID：组长的PID 组长终止了，组ID不变

父 —fork→ 子. 子进程刚被创建时，组ID和父进程一致

一个组长进程 不能指离原进程组

```
int setpgid(pid_t pid, pid_t pgid);
pid_t getpgid(pid_t pid);
```

setpgid(0,0) → 以自己为组长建立新进程组.

0 获取自己的pgid.

进程组id

2023年4月26日 11:07

```c
int main(int argc, char *argv[])
{
    if(fork() == 0){
        printf("Child, pid = %d, pgid = %d\n", getpid(), getpgid(0));
        setpgid(0,0);
        printf("Child, pid = %d, pgid = %d\n", getpid(), getpgid(0));
    }
    else{
        printf("Parent, pid = %d, pgid = %d\n", getpid(), getpgid(0));
        wait(NULL);
    }
    return 0;
}
```

会话 session

2023年4月26日 11:07

会话 是进程组的集合　　会话ID → 会话首进程

一个会话和一个终端关联　　会话的终端关闭了，会话内所有进程收到断开连接信号

在一个会话内 有最多一个前台进程组，任意个后台进程组

```c
int main(int argc, char *argv[])
{
    if(fork() == 0){
        printf("Child, pid = %d, pgid = %d\n", getpid(), getpgid(0));
        setpgid(0,0);
        printf("Child, pid = %d, pgid = %d\n", getpid(), getpgid(0));
        while(1);
    }
    else{
        printf("Parent, pid = %d, pgid = %d\n", getpid(), getpgid(0));
        wait(NULL);
    }
    return 0;
}
```

```
[liao@ubuntu Linuxday_12]$ ./5_foreground
Parent, pid = 33893, pgid = 33893
Child, pid = 33894, pgid = 33893
Child, pid = 33894, pgid = 33894
^C
```

守护进程 daemon

2023年4月26日 11:25 古

脱离了启动环境的进程 一般以d做为后缀

- 脱离了会话
  - setsid. `pid_t setsid(void);`
- 修改启动环境属性
  - cwd → "/"
  - umask → 0
  - 输入输出关闭

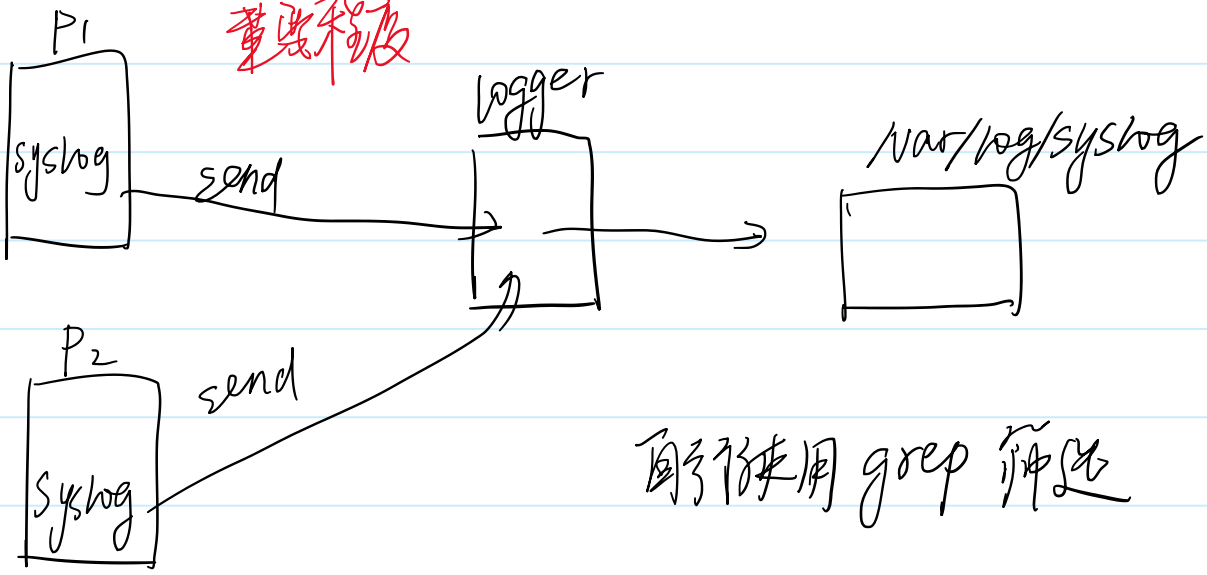
日志的故事

2023年4月26日 11:37

```
void syslog(int priority, const char *format, ...);
```

和printf一样.

重点和级



可以使用 grep 筛选

LOG_EMERG	system is unusable
LOG_ALERT	action must be taken immediately
LOG_CRIT	critical conditions
LOG_ERR	error conditions
LOG_WARNING	warning conditions
LOG_NOTICE	normal, but significant, condition
LOG_INFO	informational message
LOG_DEBUG	debug-level message

2023年4月26日 11:54

```c
#include <49func.h>
void Daemon(){
    if(fork() != 0){
        //父进程
        exit(0);
    }
    //子进程
    setsid();
    //关闭输入输出
    for(int i = 0; i < 3; ++i){
        close(i);
    }
    //修改进程的启动环境属性
    chdir("/");
    umask(0);

}
int main(int argc, char *argv[])
{
    Daemon();
    for(int i = 0; i < 100; ++i){
        syslog(LOG_INFO,"hello world! i = %d\n", i);
        sleep(2);
    }
    return 0;
}
```