

3.1 论述长期调度、中期调度和短期调度的差异

长期调度

将已进入系统并处于后备状态的作业按某种算法选择一个或一批，为其建立进程，并进入主机。当该作业执行完毕时，还负责回收系统资源。

中期调度

将进程从内存或从CPU竞争中移出，从而降低多道程序设计的程度，之后进程能被重新调入内存，并从中断处继续执行。

短期调度

又称为进程调度、低级调度或微观调度。主要任务是按照某种策略和算法将处理机分配给一个处于就绪状态的进程，分为抢占式和非抢占式。

区别：主要区别在于使用的地方和使用频率不一致

短期调度使用频率最高

短期调度在内存作业中选择就绪执行的作业，并为他们分配CPU，于是需要多次被调用

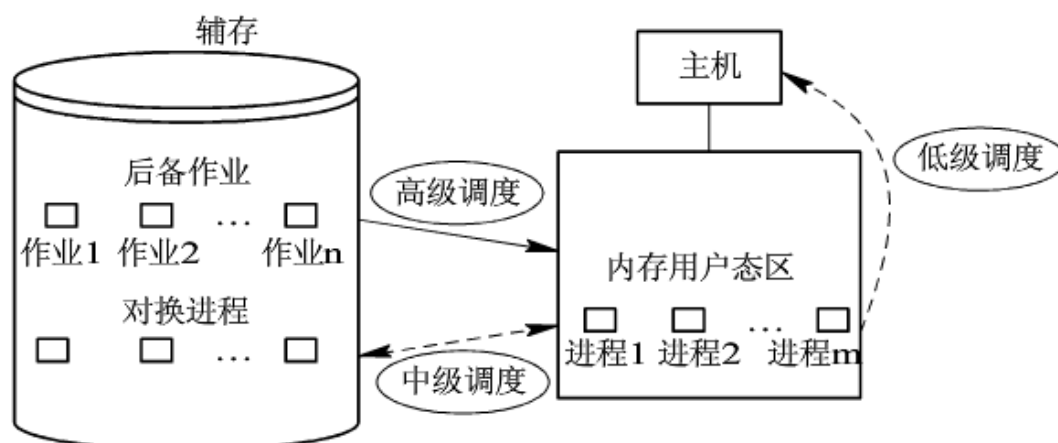
中期调度次之

中期调度作为一种中等程度的调度程序，主要被用于分时系统，将部分运行程序移出内存，之后，从中断处继续执行。

长期调度最少被调用

长期调度确定哪些作业调入内存以执行，并不频繁，可能在进程离开系统时才被唤起。

下图展示了三种调度算法的使用位置的不同



3.6 何时到达第J行

```

int pid=fork();
if(pid < 0){
//失败，该用户的进程数达到限制或者内存被用光了
}
else if(pid == 0){
//子进程执行的代码
}
else{
//父进程执行的代码
}

```

所以如果是在子进程中的话会返回pid=0，输出LINE J

3.6 对图 3-31 所示的标记为 printf("LINE J") 的行所能执行的环境，请解释一下。

```

#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
    pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
        printf("LINE J");
    }
    else { /* parent process */
        /* parent will wait for the child to complete */
        wait(NULL);
        printf("Child Complete");
    }

    return 0;
}

```

图 3-31 何时到达第 J 行

3.7 采用3-32所示的程序，确定行ABCD中的PID值（假定父进程和子进程的PID分别为2600和2603）

A中为0

B中为2603

C中为2603

D中为2600

```

#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
    pid_t pid, pid1;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (pid == 0) { /* child process */
        pid1 = getpid();
        printf("child: pid = %d",pid); /* A */
        printf("child: pid1 = %d",pid1); /* B */
    }
    else { /* parent process */
        pid1 = getpid();
        printf("parent: pid = %d",pid); /* C */
        printf("parent: pid1 = %d",pid1); /* D */
        wait(NULL);
    }

    return 0;
}

```

图 3-32 pid 值是什么

3.8 普通管道有时比命名管道更合适，而命名管道有时比普通管道更合适。请举例说明

普通管道：用于有血缘关系之间的进程间的通信（PIPE）且只能是单向通信

eg: 隔空投送，广播

命名管道：用于任意进程间的通信（FIFO）且可以是双向通信

eg: 聊天

注：FIFO严格遵循先进先出，对管道及FIFO的读总是从开始处返回数据，写则把数据添加到末尾。不支持seek()等文件定位操作。

3.12

- 3.12 使用 UNIX 或 Linux 系统，编写一个 C 程序，以便创建一个子进程并最终成为一个僵尸进程。这个僵尸进程在系统中应保持至少 10 秒。进程状态可以从下面的命令中获得

```
ps -l
```

进程状态位于列 S；状态为 Z 的进程为僵尸。子进程的进程标识符 (pid) 位于列 PID；而父进程的则位于列 PPID。

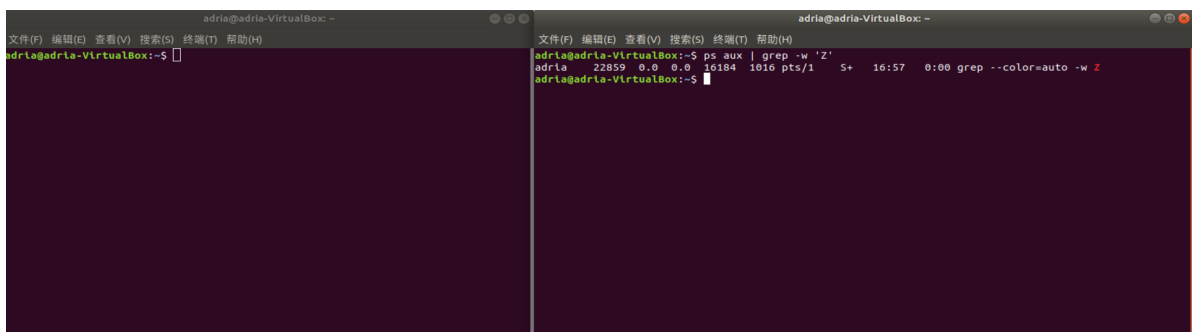
为了确定子进程确实是一个僵尸，或许最简单的方法是：运行所写的程序于后台（使用 &），然后运行命令 `ps -l` 以便确定子进程是否是一个僵尸进程。因为系统不想要过多的僵尸进程存在，所以需要删除所生成的。最简单的做法是通过命令 `kill` 来终止父进程。例如，如果父进程的 pid 是 4884，那么可输入

```
kill -9 4884
```

选择了sleep的方式生成僵尸进程：

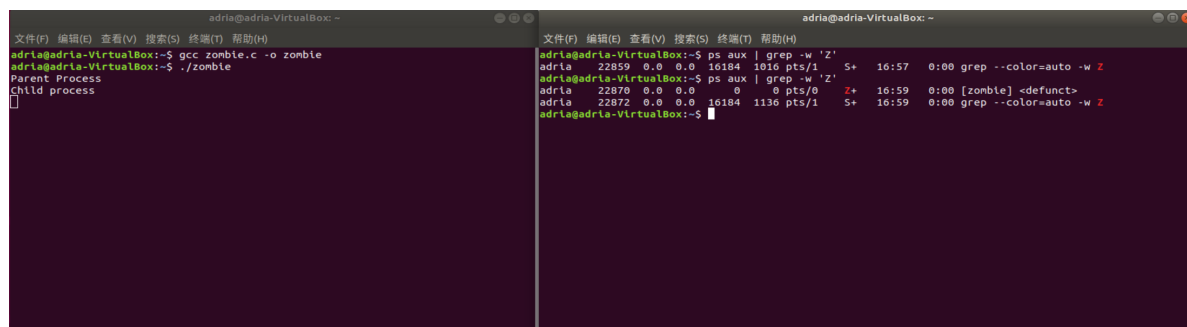
```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main(void)
{
    pid_t pid;
    pid = fork();
    if (pid < 0)
    {
        printf("Fork Failed");
    }
    else if (pid == 0)
    {
        printf("Child process\n");
    }
    else
    {
        printf("Parent Process\n");
        sleep(30); //父进程休眠30s保证子进程先退出并成为僵尸进程
        printf("weak up and quit\n");
    }
    return 0;
}
```

刚开始抓取到的僵尸进程：

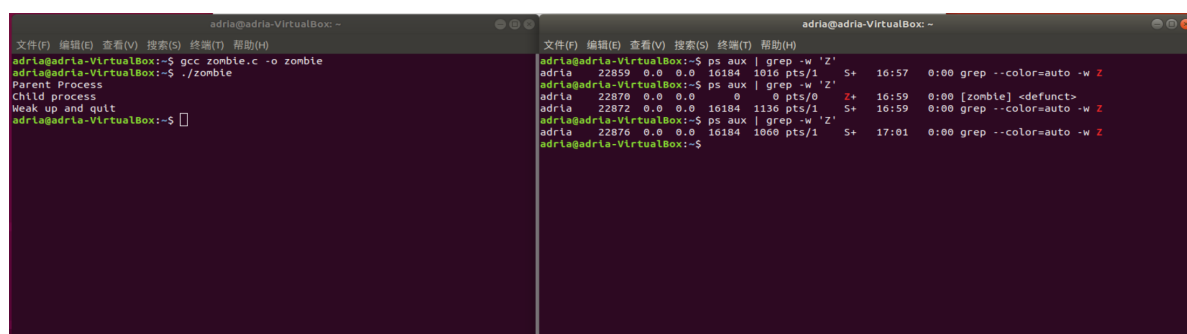


编译zombie.c运行获得的僵尸进程：S+为睡眠状态

可以清楚地看到 [zombie] 为 Z 状态，即僵尸状态



程序结束状态：



3.20

3.20 利用普通管道设计一个文件复制程序 filecopy。此程序有两个参数：原文件名称和新文件名称。该程序将创建一个普通管道，并将要复制的文件内容写入管道。子进程将从管道中读取该文件，并将它写入目标文件。例如，如果我们按如下调用该程序：

```
filecopy input.txt copy.txt
```

那么文件 input.txt 将被写入管道。子进程将读取这个文件的内容，然后写入目标文件 copy.txt。你可以利用 UNIX 或 Windows 管道来写这个程序。

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/ioctl.h>
#include <dirent.h>
#include <linux/input.h>
#include <signal.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <pthread.h>
int main(int argc, char **argv)
{
    char buf[1024] = {0};
    memset(buf, 0, sizeof(buf));    //定义一个字符数组并且清空

    int mk = mkfifo("/tmp/tran", 0777);    //创建一个管道
```

```

    if(mk == -1)
    {
        printf("establish mk fail.\n");    //判断是否创建有名管道成功
    }

    int fifo = open("/tmp/tran", O_RDWR);    //打开管道

    int fd_src = open(argv[1], O_RDWR);    //打开需要复制的文件
    if(fd_src < 0)
    {
        printf("open %s fail.\n", argv[1]);    //判断是否打开成功
    }

    int fd_desc = open(argv[2], O_CREAT|O_RDWR|O_TRUNC, 0777);    //创建新的文件
    if(fd_desc < 0)
    {
        printf("open %s fail.\n", argv[2]);    //判断是否打开成功
    }

    while(1)    //读写文件
    {
        int dre = read(fd_src, buf, sizeof(buf));

        write(fifo, buf, dre);    //从文件中读取文件并且写入管道

        memset(buf, 0, sizeof(buf));

        int re = read(fifo, buf, sizeof(buf));    //从管道中读取文件并且写入新的文件中

        int ret = write(fd_desc, buf, re);

        memset(buf, 0, sizeof(buf));

        if(ret < 1024)    //判断文件是否写入完成，完成则退出。
            break;
    }
    printf("copy succes.\n");
    close(fd_src);
    close(fd_desc);    //关闭文件
    return 0;
}

```

```

adria@adria-VirtualBox:~$ gcc pipe.c -o pipe
adria@adria-VirtualBox:~$ ./pipe 1.txt 2
copy succes.
adria@adria-VirtualBox:~$

```

结果：1.txt中写入"hello world"，运行结束后生成了2 内容也为 "hello world"

