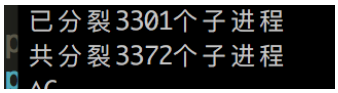
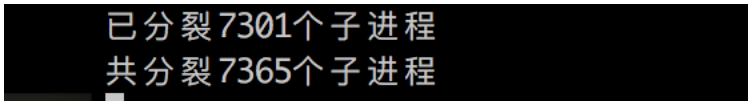
守护进程再次分裂子进程(极限测试)

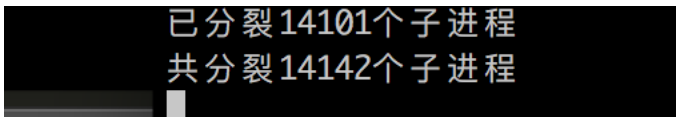
512MB 内存



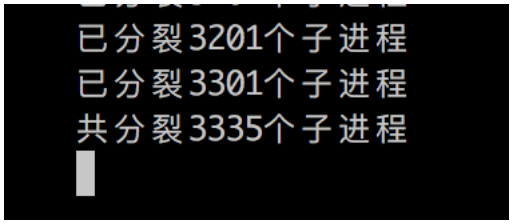
1024MB 内存



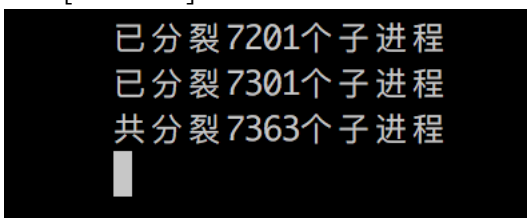
2048MB 内存



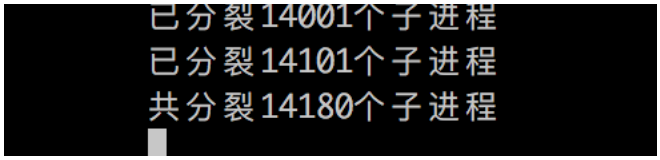
char[1024*10] 512MB



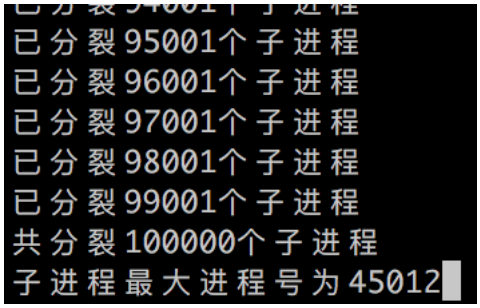
char[1024*10] 1024MB



char[1024*10] 2048MB



可以观察到进程中定义的数组大小即使增加十倍,能够同时存在的进程数还是几乎没有任何变化.
512->1024 内存扩大一倍 总进程数扩大一倍多点
可能系统的一部分内存用来运行其他必要进程了
1024->2048 内存扩大一倍总进程数扩大一倍左右
由于每次运行子进程数都不同 大致即为内存翻倍 相同的进程数翻倍
test4-2



可输出如图效果

上代码

root@RHEL-zby:/home/homework/1551265-000105-1/05 (ssh)

```
#include <stdio.h>
#include <sys/types.h>
#include <signal.h>
#include <string.h>
#include <sys/wait.h>
#include <unistd.h>
int numin=0,numout=0,max=0;
void func_waitpid(int signo) {
    pid_t pid;
    int stat;
    while( (pid = waitpid(-1, &stat, WNOHANG)) > 0 )
        numout++;
    return;
}
int sub()
{
char str[1024*10]={20};
    sleep(1);
}
int main(int argc,char *argv[])
{
signal(SIGCHLD, &func_waitpid);
    int i,j,num=0;
for(i=0;argv[1][i];i++)
num=num*10+argv[1][i]-'0';
    pid_t pid1;
    pid1=fork();
    if(pid1==-1)
    return 0;
    if(pid1==0)
    {
    for(i=0;i<num;)
    {
if(numin-numout<1500)
{
    pid_t pid = fork();
    if (pid == 0)
    {
if(i==num-1)
printf("子进程最大进程号为%d",(int)getpid());
        sub();
        return 0;
    }
   else if(pid==-1)
    {
        printf("共分裂%d个子进程\n",i+1 );
        break;
    }
else
{
numin++;
    if(!(i%1000))
       {
        printf("已分裂%d个子进程\n",i+1);
        sleep(1);
        }
        i++;
}
}
else
sleep(1);
}
if(i==num)
printf("共分裂%d个子进程\n",i);
    while(1)
    {
      sleep(5);
    }
return 0;}
if(pid1>0)
return 0;
}

~
-- INSERT --
```

几点说明:
1.程序用两个变量 numin 和 numout 来计数生成和释放的子进程个数
生成一个子进程时 numin+1

```
else
{
numin++;
    if(!(i%1000))
       {
        printf("已分裂%d个子进程\n",i+1);
```

每次释放子进程时 numout+1

```
void func_waitpid(int signo) {
    pid_t pid;
    int stat;
    while( (pid = waitpid(-1, &stat, WNOHANG)) > 0 )
        numout++;
    return;
}
```

先说作用:
最主要的作用:比较生成和释放子程序数量,理论来说,应该每次都是生成比释放多 1000 个,因为我在程序中设置了一个单位时间(20s)生成的进程数量和释放数量相同,生成一个单位(1000 个)子进程 sleep 一个单位时间,也就是每个单位时间内都最多存在一个单位的子进程.
这个一个单位的数量选取是要不小于上一问测得的同时存在的进程数 3000 多点(极限值),又不能太少,否则程序运行的时间就会太长,等的很烦.于是我在此选了 1000
但是不知道为什么,我在修改各种参数进行测试后,每次几乎都会发生一个很严重的问题.每次打印的速度越来越快(本来应该是一个单位时间打印一次),比如我最开始设置一秒打印一次,在运行几千个进程后可以明显看到每次打印的时间根本不到 1 秒而且存在连续打印的情况,这样释放的速度就跟不上生成的速度了,在一秒内生成的进程超过这个极限值后(需要等待一秒才会得到释放),内存被耗尽,程序运行结束.
我试了很多种方法调整这个 sleep 的时间,但是都没有用.可能这对系统要求太高
只好换用另一种方法 强制控制同时存在的进程数 就是这种方法
用 numin-numout 来控制同时存在的子进程数,使之不能过大(大于每秒生成的子进程数又要小于极限值),我在这设置了 1.5 倍的每秒生成数(1500),如果检测到大于这个阈值,就让父进程停止继续生成子进程,并 sleep 一个单位时间,等待子进程释放,如果释放后小于阈值就继续生成,如果大于继续 sleep 等待
这样强制控制存在的进程数就避免了系统时钟的各种异常变化
另一个作用:为下一问做铺垫,直接可以用这两个变量计算最后的生成和释放数量是否相等.

用程序表达出来很简洁的话,写出来就很难解释清楚

关于两个变量的操作方式 我琢磨了很久
试了很多种方法 因为子进程执行的程序虽然和父进程是一样的代码 但是其中的计数器又被重置了
所以计数肯定就不能在子进程的相关函数中进行了
于是就只能看父进程如何获取生成一个子进程和释放一个子进程的信号
最后将两个监测点设置在这两个位置

这个程序当然还需要让子进程结束时完全退出 避免产生僵尸进程
这个可以继续用前面用过的 waitpid 函数

获取最大进程号这里我默认生成的子进程号是越来越大(因为我观察的每次都是这样的)为什么不将每次生成的子进程号与最大进程号进行比较并更新最大进程号呢?
我之前一直准备使用这种办法 但是每个子进程是独立的 每次的比较一定是在父进程中才能够进行 然而将子进程的 pid 传回父进程又显得比较麻烦 于是我直接输出了最后一次产生的子进程号作为最大进程号 使用 getpid()函数即可

第三问

```
[root@RHEL-zby 05]# ./test5-3 10000
[root@RHEL-zby 05]# 已分裂1个子进程
分裂1000  回收1
已分裂1001个子进程
分裂2000  回收695
已分裂2001个子进程
分裂3000  回收1518
已分裂3001个子进程
分裂4000  回收2501
已分裂4001个子进程
分裂5000  回收3506
已分裂5001个子进程
分裂6000  回收4577
已分裂6001个子进程
分裂7000  回收5501
已分裂7001个子进程
分裂8000  回收6501
已分裂8001个子进程
分裂9000  回收7677
已分裂9001个子进程
分裂10000  回收8501
共分裂10000个子进程
分裂个数10000
回收个数8752
分裂个数10000
回收个数8822
分裂个数10000
回收个数8835
分裂个数10000
回收个数8857
分裂个数10000
回收个数9001
分裂个数10000
回收个数9240
分裂个数10000
回收个数9426
分裂个数10000
回收个数9501
分裂个数10000
回收个数9817
子进程最大进程号为55151
分裂个数10000
回收个数10000
分裂个数10000
回收个数10000
```

主要代码和思路和上一问相同,上一问添加的两个全局变量 numin 和 numout 在这用到
我顺便输出了每产生 1000 个子进程时回收掉的进程数作对比

关于最后一个问题:当极限测试导致子进程分裂失败后，Linux 系统还能正常操作吗?
我只能说:不一定
第一次 512 极限测试时候,确实 bash 完全死了,不管什么指令都没有任何相应
强行关机重启后,之后的每次测试都再没死过,可以用 killall 把进程杀掉,bash 都可以正常相应,可能是被我坑一次以后长记性了?