苏州大学实验报告

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院系 | 计算机学院 | | 年级专业 | | 21计科 | | 姓名 | 方浩楠 | 学号 | 2127405048 |
| 课程名称 | | 操作系统课程实践 | | | | | | | 成绩 |  |
| 指导教师 | | 王红玲 | | 同组实验者 | | 无 | | 实验日期 | 2024.3.6 | |

|  |  |
| --- | --- |
| 实 验 名 称 | 实验2 进程创建 |

1. 实验目的

1. 加深对进程概念的理解，进一步认识并发执行的实质

2. 掌握 Linux 操作系统的进程创建和终止操作

3. 掌握在 Linux 系统中创建子进程后并加载新映像的操作。

1. 实验内容

1. 编写一个 C 程序，使用系统调用 fork( )创建一个子进程。要求：①在子进程中分别输出当

前进程为子进程的提示、当前进程的 PID 和父进程的 PID、根据用户输入确定当前进程的返

回值、退出提示等信息。②在父进程中分别输出：当前进程为父进程的提示、当前进程的

PID 和子进程的 PID、等待子进程退出后获得的返回值、退出提示等信息。

2. 编写 C 程序，使用系统调用 fork()创建一个子进程，子进程调用 exec 族函数执行系统命

令 ls。

3. 调试并运行下列代码，回答下述问题。

①一共创建了多少个进程？

② 画出创建的进程树。

③ 给出程序执行的结果。

④ 若删除代码中的“wait(&rtn);”语句，程序的运行结果是否相同？为什么？

int main()

{

int rtn=0;

int i;

for(i=0;i<2;i++)

{

if(fork()==0)

printf("Child: the parent pid is: %d, the current pid is:%d\n", getppid(),getpid());

else

{

printf("Parent: the process pid is: %d\n",getpid());

wait(&rtn);

}

}

exit(0);

}

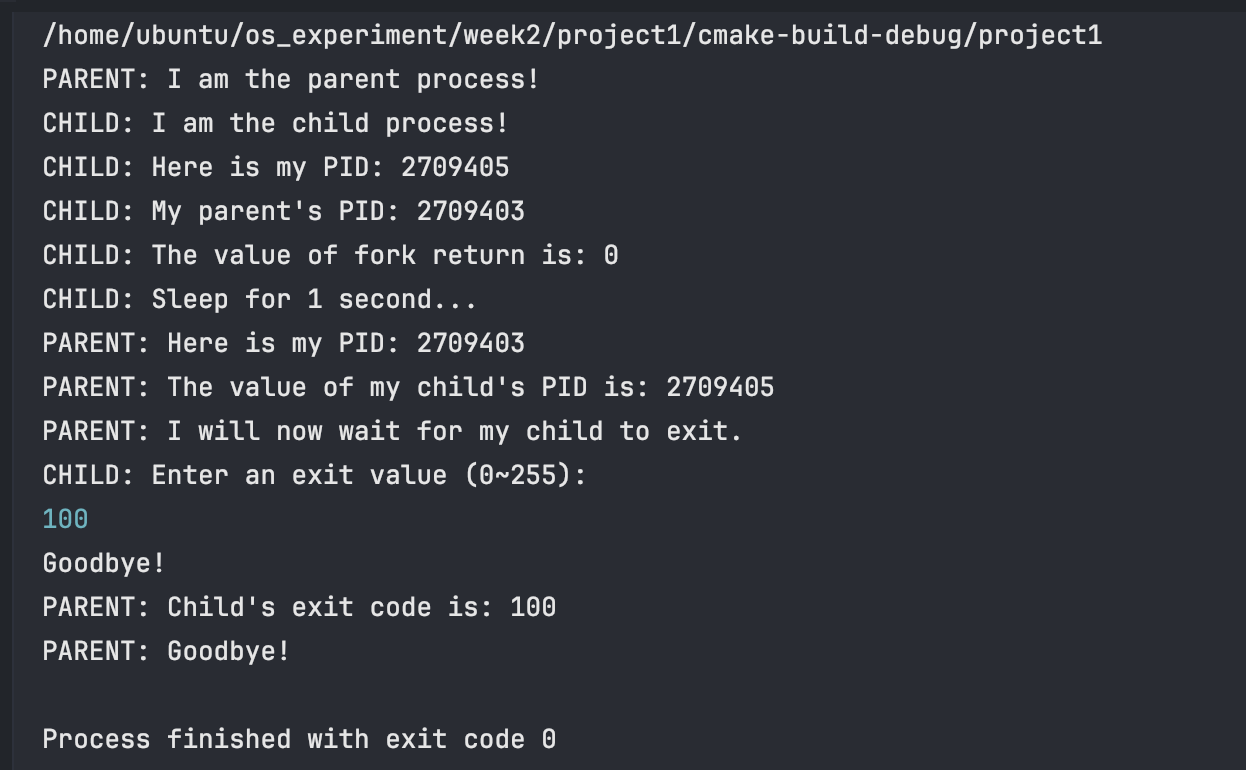
1. 实验步骤和结果

1.

c程序代码:



运行结果:

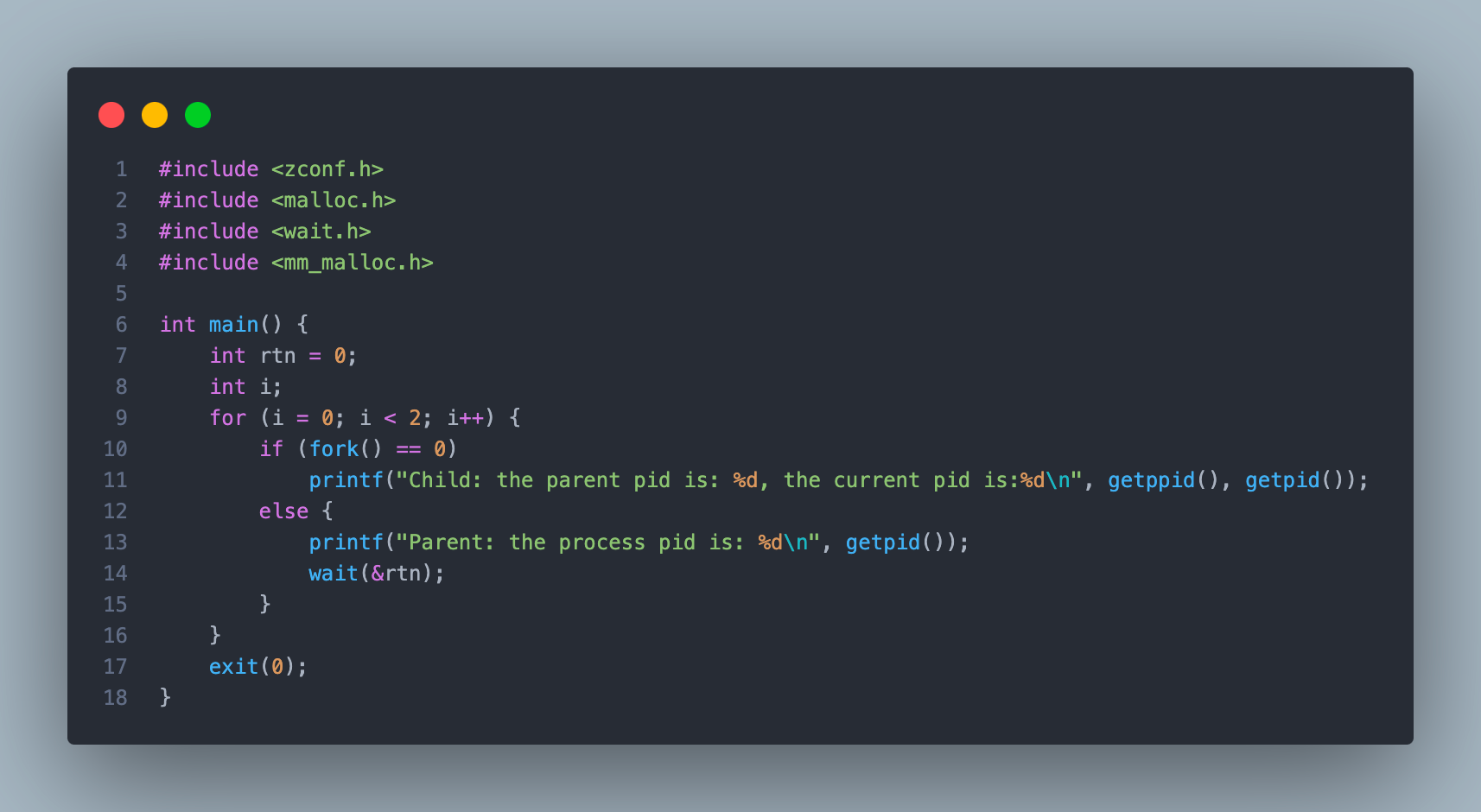


2.

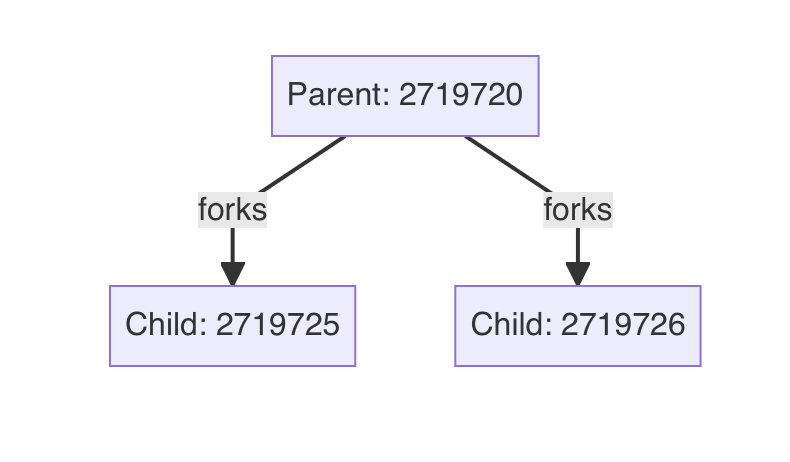
c程序代码:



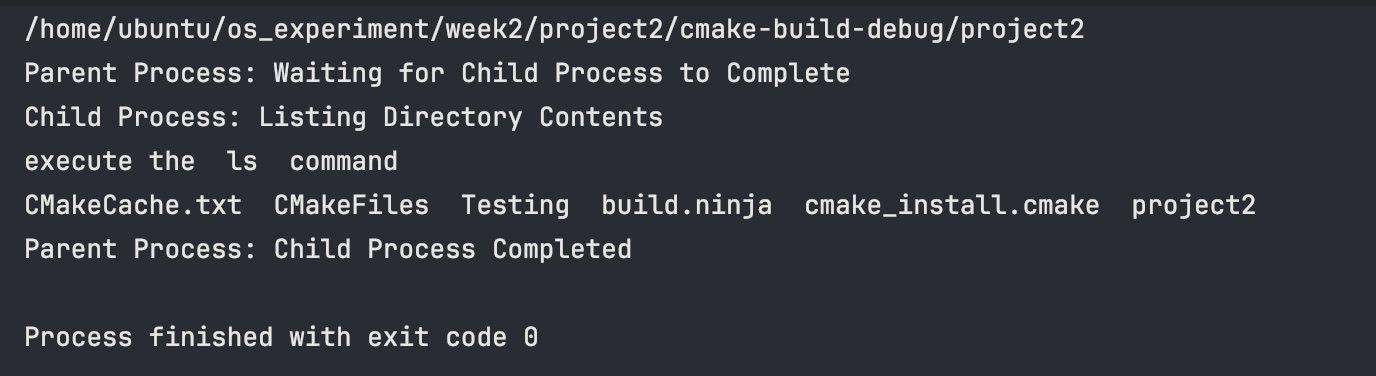
3.



1. 一共创建了三个进程
2. 进程树:



(3)代码运行结果:



1. 删除后不相同.删除后父进程将不会等待子进程结束.程序的同步行为将被移除，父子进程的执行顺序可能变化，导致父子进程的执行更加独立和不可预测。
2. 实验总结

调用 fork() 函数的三种返回情况

当在Linux操作系统中调用fork()函数时，会有三种不同的返回情况，这些返回值对于控制程序流程非常关键：

返回值为-1：当fork()调用失败时，会返回-1。这通常是由于系统资源限制或达到了进程数量的上限。在这种情况下，新的子进程不会被创建，程序应该检查这种错误并适当地处理。

返回值为0：当fork()调用在子进程中成功时，会返回0。这意味着你现在在子进程的上下文中运行，可以在此基础上执行子进程特有的代码逻辑。通常，子进程会执行与父进程不同的任务或者调用exec系列函数加载新的程序映像。

返回值大于0：当fork()调用在父进程中成功时，会返回子进程的PID（进程ID）。这允许父进程获取子进程的标识符，以便于之后进行进程间通信、发送信号或等待子进程的结束。

fork() 和 wait() 配合使用的情况

fork()和wait()函数经常一起使用来控制父子进程之间的同步执行。fork()用于创建新的子进程，而wait()用于使父进程暂停执行，直到一个子进程结束。正常使用情况：在父进程中调用wait()函数可以等待子进程的完成。wait()会阻塞父进程的执行直到至少有一个子进程结束。在此期间，父进程不消耗CPU资源。wait()还能够获取子进程的退出状态，使父进程可以了解子进程的结束原因和状态。

取消wait()函数的影响：如果在父进程中取消wait()函数，父进程将不会等待子进程结束就继续执行。这会导致几个可能的问题：

僵尸进程：子进程结束后，如果父进程没有通过wait()或waitpid()调用来回收子进程的状态信息，子进程会成为僵尸进程。虽然僵尸进程释放了大部分资源，但仍然保留在进程表中，占用系统资源。

父进程提前结束：如果父进程在子进程之前结束，子进程将变成孤儿进程。孤儿进程将被init进程（PID为1的进程）接管，init进程会自动调用wait()回收孤儿进程的状态信息，防止孤儿进程成为僵尸进程。

通过本实验，目的在于：

加深对进程概念的理解，深入认识并发执行的实质。

掌握Linux操作系统中进程的创建和终止操作。

学习在Linux系统中如何创建子进程并加载新的执行映像。

本次实验的成果:

创建并管理子进程：通过编写C程序，使用fork()系统调用创建子进程，实践了在Linux下的进程创建和管理。子进程中输出了其PID、父进程的PID以及根据用户输入决定的退出值，从而深入理解了进程间的关系和通信机制。

使用exec族函数：编写程序实现fork()后子进程调用exec族函数执行系统命令ls，学习了如何在子进程中替换当前进程的映像为新的程序映像，这对理解进程状态和程序执行的过程至关重要。

进程创建分析：通过对给定代码的调试和分析，学习到了进程创建树的概念，理解了在Linux下使用fork()创建进程的机制和父子进程间的同步方法。实验表明，在给定的代码执行过程中，一共创建了四个进程（一个父进程和三个子进程），进程创建树的分析有助于理解并发执行的内在机制。

并发执行的同步：通过比较包含和删除wait(&rtn);语句的程序运行结果，深入理解了进程同步的重要性。wait()函数使得父进程等待子进程的结束，保证了进程间的有序执行。移除wait()后，父子进程的执行顺序变得不可预测，可能导致资源管理上的问题，如僵尸进程的产生。