1使用下图所示的程序，说明LINE A的输出内容。

#include < sys/types.h >

#include < stdio.h > #include < unistd.h > int value **=** 5**;** int main**()**

**{** pid t pid**;** pid **=** fork**(); if (**pid **==** 0**)**

**{** /\* child process \*/ value **+=** 15**; return** 0**;**

**}**

# else if (pid > 0)

**{** /\* parent process \*/ wait**(NULL);**

printf**(**"PARENT: value = %d"**,** value**);** /\* LINE A \*/ **return** 0**;**

**}**

**}**

2包括初始父进程，下面显示的程序创建了多少进程？

#include < stdio.h > #include < unistd.h > int main**() {**

/\* fork a child process \*/ fork**();**

/\* fork another child process \*/

fork**();**

/\* and fork another \*/ fork**(); return** 0**;**

**}**

3当进程使用fork（）操作创建新进程时，父进程和子进程之间共享以下哪些状态？

a.堆

b.堆

c.共享内存段

4描述短期，中期和长期调度之间的差异。

5描述内核对进程之间上下文切换所采取的操作。

6包括初始父进程，下面显示的程序创建了多少进程？

#include < stdio.h > #include < unistd.h > int main**() {**

int i**; for (**i **=** 0**;** i **<** 4**;** i**++)** fork**();**

# return 0;

**}**

7说明将在下面的代码中标记printf（“LINE J”）的代码行的情况。

#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;

**}**

8使用以下程序，识别A，B，C和D行的pid值。假设父和子的实际pid分别为2600和2603。）

#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;

**}**

9举例说明普通管道比命名管道更合适的情况，以及命名管道比普通管道更合适的情况示例。

10使用下面显示的程序，解释X和Y行的输出结果。

#include < sys/types.h >

#include < stdio.h >

#include < unistd.h > #define SIZE 5 int nums**[**SIZE**] = {** 0**,**1**,**2**,**3**,**4 **} ;** int main**() {**

int i**;** pid t pid**;** pid **=** fork**(); if (**pid **==** 0**) { for (**i **=** 0**;** i **<** SIZE**;** i**++) {**

nums**[**i**] \*= -**i**;**

printf**(**"CHILD: %d "**,**nums**[**i**]);** /\* LINE X \*/ **}**

# } else if (pid > 0) { wait(NULL);

**for (**i **=** 0**;** i **<** SIZE**;** i**++)**

printf**(**"PARENT: %d "**,**nums**[**i**]);** /\* LINE Y \*/

# } return 0;

**}**

11创建线程时使用了哪些资源？它们与创建流程时使用的有何不同？

12提供两个编程示例，其中多线程不提供比单线程解决方案更好的性能。

13多线程进程中的线程之间共享以下哪个程序状态组件？

a.注册值

b.堆内存

c.全局变量

d.堆栈内存

14考虑以下代码段：

pid t pid; pid = fork();

if (pid == 0) { /\* child process \*/

fork();

thread create( . . .);

}

fork();

a.创建了多少个唯一进程？

b.创建了多少个唯一线程？

15下面显示的程序使用Pthreads API。 LINE C和LINE P的程序输出是什么？

#include <pthread.h>

#include <stdio.h>

#include <types.h> int value **=** 0**;**

void **\***runner**(**void **\***param**);** /\* the thread \*/ int main**(**int argc**,** char **\***argv**[]) {**

pid\_t pid**;** pthread\_t tid**;** pthread\_attr\_t attr**;** pid **=** fork**();**

**}**

**if (**pid **==** 0**) {** /\* child process \*/ pthread\_attr\_init**(&**attr**);** pthread\_create**(&**tid**,&**attr**,**runner**,NULL);** pthread\_join**(**tid**,NULL);**

printf**(**"CHILD: value = %d"**,**value**);** /\* LINE C \*/

**} else if (**pid **>** 0**) {** /\* parent process \*/ wait**(NULL);**

printf**(**"PARENT: value = %d"**,**value**);** /\* LINE P \*/

**}**

void **\***runner**(**void **\***param**) {**

value **=** 5**;**

pthread\_exit**(**0**);**

**}**

**6我们提到经常禁用中断会影响系统的时钟。解释为什么会发生这种情况以及如何最大限度地减少这种影响。**

**17繁忙等待一词的含义是什么？操作系统还有哪些其他类型的等待？可以一共忙碌等待吗？解释你的答案。**

**18解释为什么自旋锁不适用于单处理器系统，但通常用于多处理器系统。**

**19显示，如果wait（）和signal（）信号量操作未以原子方式执行，则可能违反了互斥。**

**20许多计算机系统都可以使用竞争条件。考虑一个银行系统，它通过两种功能维持账户余额：存款（金额）和提款（金额）。这两项功能通过了从银行账户余额中存入或取出的金额。假设丈夫和妻子共用一个银行账户。同时，丈夫调用withdraw（）函数，妻子调用deposit（）。描述竞争条件如何可能以及可以采取哪些措施来防止竞争条件的发生。**

**21解释为什么中断不适合在多处理器系统中实现同步原语。**