1. 下图程序在LINE A处的输出是什么？为什么会有这样的输出?

```
#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;
   }
}
```

输出为5，因为父子**进程**不共享全局变量，由判断条件`pid > 0`可知，此处是父进程执行，所以value保持初始化的值5

2. 下图程序运行过程中一共会出现多少个进程（包括主进程）？

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
    /* fork a child process */
    fork();

    /* fork another child process */
    fork();

    /* and fork another */
    fork();

    return 0;
}
```

会出现8个进程，每次`fork`都会创建一个子进程，且`fork`后父子进程执行开始的位置相同，因此本代码中，每个`fork`会使进程数量翻倍，即$1 \rightarrow 2 \rightarrow 4 \rightarrow 8$，共8个进程

3. 当一个进程使用fork（）操作创建一个新进程时 下列状态之一在父进程和子进程之间共享

选C. 共享内存段，AB不共享，可以通过编程验证

4. 描述内核对进程之间的上下文切换所采取的操作（）

大体步骤如下：

1. 保存当前进程上下文到PCB：内核会保存当前正在运行进程的寄存器状态、程序计数器值以及其他必要的上下文信息。这样可以在切换回该进程时，能够继续执行之前的状态。
2. 根据调度算法选择下一个执行的进程：内核会根据调度算法选择下一个要运行的进程。
3. 加载下一个进程的上下文：内核会从已选择的进程的PCB中恢复寄存器状态、程序计数器值以及其他必要的上下文信息。
4. 开始执行新进程：一旦下一个进程的上下文已被恢复，内核会将处理器控制权转移到该进程，使其开始执行。

5. 举一个普通管道比命名管道更适合的情况的例子，以及一个命名管道比普通管道更适合情况的例子。

普通管道只能用于父子进程或兄弟进程间的通信，因为普通管道通过fork调用来拷贝文件描述符的，在文件系统中，普通管道并不对应物理文件。

命名管道以文件的形式存在于文件系统中，及时无亲缘关系的进程在系统中打开该文件即可通信

普通管道的例子：有亲缘关系的进程间通信，如`ls | grep 'xxx'`、父进程获取子进程的执行结果等

命名管道的例子：无亲缘关系的进程间通信，如日志分析系统监控多个应用进程的报错日志等

6. 假设父和子进程的实际pid分别为2600和2603，请给出A、B、C、D 处的输出

```
#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;
}
```

- A：`child: pid = 0`，由判断条件（`pid == 0`）可知
- B：`child: pid1 = 2603`，由判断条件（`pid == 0`）可知，此处代码由子进程执行，因此`pid1=2603`
- C：`parent: pid = 2603`，由判断条件（`pid == 0`）可知，此处代码由父进程执行，则`pid`为子进程进程号，即`pid=2603`
- D：`parent: pid1 = 2600`，由判断条件（`pid == 0`）可知，此处代码由父进程执行，则`pid1`为父进程进程号，即`pid=2600`