1. 什么是用户级线程和内核级线程？两者之间的区别是什么？在什么情况下一种类型比另一种更有优势？

用户级线程：在**用户空间中管理和调度**的线程。它们由应用程序自身的线程库（Thread Library）实现和控制，而**操作系统对其毫无感知**。用户级线程的创建、切换和调度等操作都由用户空间的线程库完成。

内核级线程：由**操作系统内核管理和调度**的线程。内核级线程由操作系统直接创建和管理，并且操作系统负责线程的调度和切换。

区别：

1. 用户级线程**由应用程序线程库创建和管理**，而内核级线程**由操作系统内核创建和管理。**
2. 用户级线程调度和切换是**在用户空间中完成**，而内核级线程的调度和切换**由操作系统内核负责。**
3. 用户级线程的并发性（concurrency）是由应用程序线程库控制的，而内核级线程的并发性由操作系统内核控制。内核级线程可以实现真正的并行执行，而用户级线程不行。

选择：

- 用户级线程适用于需要轻量级、快速切换的应用程序，例如服务器程序或多线程计算密集型任务。
- 内核级线程适用于需要更好的并行性和多核利用率的应用程序，例如需要高度并行计算或I/O密集型任务。

2. 以下哪个程序状态在多线程的进程中是跨线程共享的

b. 堆内存

c. 全局变量

可以通过编程验证

3. 考虑以下代码回答相关问题

```
pid_t pid;

pid = fork();
if (pid == 0) { /* child process */
    fork();
    thread_create(...);
}
fork();
```

- 创建了多少个进程
    - 6个
    - 进程数量只和`fork`的执行有关，一开始代码本身有1个进程，称为A1
    - 经过第一个`fork`，创建子进程A2
    - 第2个`fork`仅子进程A2可以执行，创建子进程A3
    - 第3个`fork`，A1、A2、A3都可以执行，所以最后有6个进程
    - 综上，此代码会创建6个进程
- 创建了多少个线程

- 2个(不包含进程的线程)或8个(包含进程的线程)
- 线程数量主要和 `thread_create` 相关
- 由进程处的解析可知，只有进程A2、A3会执行 `thread_create`，所以创建了2个线程
- 考虑到进程本身也会有1个线程，所以答案可以回答2个(不包含进程的线程)或8个(包含进程的线程)

4. 考虑以下程序，给出Line C和Line P的输出。(3+3分)

```
#include <pthread.h>
#include <stdio.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);
}
```


- Line C
  - `CHILD: value=5`
  - 因为线程可以共享全局变量
- Line P
  - `PARENT: value=0`
  - 因为进程间不共享全局变量

5. 考虑一个多核系统和一个使用多对多线程模型编写的多线程程序。让程序中用户级线程的数量大于系统中处理核心的数量。讨论以下场景对性能的影响(2+2+2分)

- 内核线程数小于处理器核心数
  - 系统的一些核心会闲置，算力无法完全发挥
- 内核线程数等于处理器核心数
  - 一个内核线程可以与处理核心一一对应，在没有IO等待、系统调用的情况下可以实现最佳的利用率
- 内核线程数大于处理器核心数，但小于用户级线程数
  - 内核线程之间存在竞争，需要进行调度，切换占用处理核心，可能造成一定的性能损失
  - 在某个内核线程进行IO或系统调用等停止使用CPU时，其余内核线程可以被调度执行，在实际情况中可能具有更好的利用率