**进程 线程**

**进程 v.s. 线程**

基

略

**进程和线程的通信方式**

基

略

略

**进程和线程的同步和互斥**

**线程同步**是指线程之间所具有的一种制约关系，一个线程的执行依赖另一个线程的消息，当它没有得到另一个线程的消息时应等待，直到消息到达时才被唤醒。一般是当多个线程相互协作，存在相互依赖的关系。

**线程互斥**是指对于共享的进程系统资源，在各单个线程访问时的排它性。当有若干个线程都要使用某一共享资源时，任何时刻最多只允许一个线程去使用，其它要使用该资源的线程必须等待，直到占用资源者释放该资源。线程互斥可以看成是一种特殊的线程同步。

线程同步。包括临界资源等的访问，相互线程之间是互斥访问。

举个例子，设有一个全局变量global，为了保证线程安全，我们规定只有当主线程修改了global之后下一个子线程才能访问global，这就需要同步主线程与子线程，可用关键段实现。当一个子线程访问global的时候另一个线程不能访问global，那么就需要互斥。

**进程线程的状态转换**

基

略

**PCB在进程状态转换过程中的行为**

基

略

**线程、内核线程、用户线程与轻量级进程，优缺点**

基

略

**进程调度方法**

基

略

**五大缓存问题**

**内存管理**

**分页系统相对于交换系统的优点**

基

略

**缺页中断处理**

基

略

**页面置换算法**

基

略

**磁盘调度算法**

基

略

调度

**零碎的问题**

**磁盘臂调度算法**

**中断**

硬中断：由磁盘、网卡、键盘、时钟等硬件设备发出，处理中断的程序在cpu上运行，cpu因处理中断而被中断，中断通常只会中断一个cpu.

软中断：软件中断不会直接中断CPU。只有当前正在运行的代码才能生成软件中断。中断是请求内核为正在运行的进程做一些事情（通常是I / O）。一个特殊的软件中断是yield调用，它要求内核调度程序检查是否可以运行其他进程。

**大端和小端**

**如果有个bug，开发不认怎么办**

**漏测**

漏测是不能杜绝的，确定漏测发生后，我们需要学会思考，吸取经验教训，尽可能的降低缺陷的漏测量。

事前：

事后：分析漏测的原因，解决问题并总结经验。

1. 需求规格不明确，导致测试用例编写过于粗犷。
   1. 先进行需求分析，找出需求规格说明书中不明确、或有疑虑的地方，与需求人员（产品）确认商讨，给出明确定义。
   2. 在测试过程中发现没有明确和有疑惑点的，也要与需求人员确认商讨，要求给出明确写定义，之后完成测试用例。
   3. 无法及时确定的，可先编写大概框架，之后再将测试用例细化，补充完善。
2. 需求规格变更，测试用例未及时更新
   1. 根据新规格补充完善测试用例，对存在有疑问的地方需要和产品或设计进行沟通和确认，可以要求需求规格进行明确定义，事后将新增的、修改的测试用例整理成文，发给组内同事组织评审，并将评审之后的用例更新到用例库中去。
3. 测试用例覆盖不全面，场景出现遗漏
   1. 测试用例场景设计导致缺陷遗漏是在所难免的，编写测试用例的同事不可能把所有的场景都能想周全，把所有的场景下的情况都写成测试用例这也是不大现实的。对于外部反馈的缺陷，是因为场景设计不全引起的，我们先分**析出现问题的场景是客户必须的场景还是偶然的场景**，如果该场景是客户操作习惯，我们可以通过和技术接口人沟通，确认该场景的一些具体细节，在完善测试用例的过程中我们也要考虑一些和该场景相关联的场景，将多种场景下测试用例及时完善、评审，增加到用例库中.
4. 测试过程中未严格按照测试用例执行
5. 时间不充足，导致一些功能点在测试过程中被忽略。
   1. 根据功能模块划分测试优先级，主要的功能模块优先级最高，安排有经验的人测试，安排新手测试一些不重要的功能模块或者很少使用的功能模块，在后续测试过程中，由有经验的同学将新手测试过的模块进行冒烟测试，确认是否有明显BUG；
   2. 尽量避免在一些和开发扯不清的情况下浪费自己的时间，如果因为开发人员排查问题占用的时间较长，可以告诉测试负责人，由测试负责人采取相应措施，通过协商来避免类似问题蔓延；
   3. 增加测试人手
   4. 加班
6. 测试环境受限，导致缺陷漏测
   1. 环境的组合是无穷的，没有足够的时间、人力和其他资源成本在足够在足够多的环境中测试。措施：保证主要的操作系统环境，网络环境。操作系统：针对当前使用比例来排序。网络环境：正常网速、低网速
7. 开发人员引入的新BUG
   1. 根据开发人员的水平，选择合适的回归测试策略。

**一些词**

cpu利用率

系统吞吐量

周转时间

等待时间

响应时间