文章编号: 1009-4881(2012)03-0054-06
DOI: 10.3969/j.issn.1009-4881.2012.03.014

Linux下基于epoll+线程池高并发服务器实现研究

梁明刚,陈西曲
(武汉工业学院 电气与电子工程学院,湖北 武汉 430023)

摘 要: 在介绍linux系统下基于TCP/IP协议的socket网络编程的客户端和服务器(client/server)基本模型、线程池和epoll机制的基础上,给出了基于epoll+线程池高并发服务器设计的详细算法。最后,给出一个在linux环境下用C语言编写的基于epoll和线程池技术的服务器和客户端实验程序,实现了服务器和客户端之间并发通信以及实验结果的截图。实验结果表明,基于epoll+线程池技术设计出来的高并发服务器能够保证服务器和客户端之间稳定高效的通信。

关键词: linux系统; 高并发服务器; 线程池; epoll机制
中图分类号: TP 312 **文献标识码:** A

The researchon realization of high concurrent server base on epoll plus threadpool in Linux

LIANG Ming-gang, CHEN Xi-qu
(School of Electrical and Electronic Engineering, WuHan Polytechnic University, Wuhan 430023, China)

Abstract: This paper describles the normal client and server (client/server) model of network communication program based on the technology of socket with TCP/IP protocol, the technology of threadpool and epoll mechanism. And then it gives the detail algorithm with the technology of threadpool and epoll to accomplish the high concurrent server. In the end, this paper exhibits an application example based on the algorithm, which is written by C language, in the Linux system and the result photos. The result shows that the concurrent server with the technology of epoll plus threadpool can make sure the communication between server and clients is stable and efficient.

Key words: Linux; high concurrent server; threadpool; epoll mechanism

在数字化、网络化和信息化的当今,计算机网络技术得到了飞速的发展和广泛的应用[1]。在TCP/IP网络应用中,通信的两个进程间相互作用的主要模式是客户机/服务器模式(client/server)即客户机向服务器提出请求,服务器接收到请求后提供相应的服务。而基于流式套接字(SOCK_STREAM)的并发服务器是目前linux网络服务器主流形式。随着网络技术快速进入各行各业,服务器的并发请求量也成几何级数增加,并且对服务器的实时性和可靠性的要求也越来越高。而传统基于多进程和select机制的并发服务器由于受到系统硬件资源和linux内核版本的限制,已经不能满足高实时性、高可靠性

收稿日期: 2012-05-18.
作者简介: 梁明刚(1986-),男,硕士研究生,E-mail: 501253524@qq.com.
通讯作者: 陈西曲(1971-),男,副教授,E-mail: cxqdhl@21cn.com.
基金项目: 湖北省自然科学基金项目(2011CHB030).

和高并发性的要求。

本文针对传统并发服务器存在的这些不足之处，在介绍多线程和线程池编程知识以及 linux2.6 内核所采用的多路复用 I/O 机制中的 epoll 技术的基础上，给出了高并发服务设计的详细算法以及具体实现的实例。

1　高并发服务器的原理及所用技术

1.1　socket 套接字技术

1.1.1　socket 的概念

TCP/IP 网络中各种服务广泛地采用客户机和服务器的模型。为了能够方便的开发网络的应用软件，由美国伯克利大学在 UNIX 上推出一套直接访问通信协议的一种接口技术: 套接字(socket) ，其屏蔽了底层网络的一些复杂的协议，使普通的程序员可以方便地访问 TCP/IP，从而开发出各种产品的网络应用程序。

1.1.2　套接字分类

Linux 系统支持三种套接字类型[2] 如下。

(1) 数据报式套接字(SOCK_DGRAM) 。数据报式套接字是基于 UDP 协议实现的。其主要提供对网络的传输的差错率要求不高，但对实时性要求极高的面向无连接网络服务。

(2) 原始套接字(SOCK_RAW) 。原始套接字是主要用于新网络协议的测试。其最大的特点就是允许直接访问底层的网络协议。

(3) 流式套接字(SOCK_STREAM) 。流式套接字是基于 TCP 协议实现的。其主要是提供三次握手的无差错、不丢失、不重复、并且按序到达的面向连接的网络服务。

1.1.3　基于 TCP 的 socket 编程模型

本文仅讨论基于流式套接字(SOCK_STREAM) 客户端/服务器模式的编程模型。其通信示意图如图 1 所示。

服务器端程序编写的步骤: 第一步，调用 socket( ) 函数创建通讯所需要的流式套接字。第二步，调用 bind( ) 函数将创建的套接字绑定到一个本地地址和端口上。第三步，调用 listen( ) 函数将套接字设置成监听的模式。第四步，调用 accpet( ) 函数，阻塞等待客户端请求到来，当有客户端请求到来后，它会创建一个对应于此连接新套接字，随后就用这个套接字与指定的客户端进行通信。第五步: 调用 recv( ) 和 send( ) 函数与客户端进行数据通信。第六步，通信完成后调用 close( ) 函数来关闭连接套接字，接着回到第四步。第七步: 在服务器端程序关闭之前调用 close( ) 函数关闭监听套接字。

客户端程序编写的步骤: 第一步，调用 socket( ) 函数创建用于通信的套接字。第二步，调用 connect( ) 函数向指定的服务器发出连接请求。第三步，调用 recv( ) 和 send( ) 函数与服务器进行数据的通信。第四步: 通信完成之后调用 close( ) 函数关闭套接字。

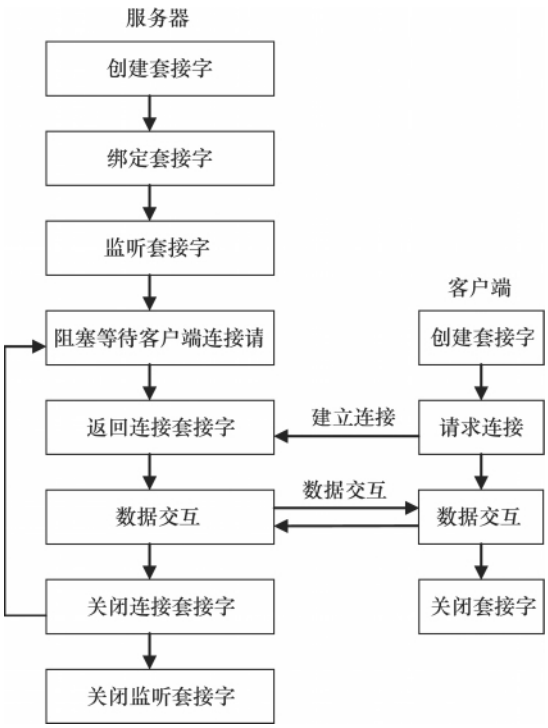

图 1　socket tcp 通信示意图

1.2　linux 多线程编程

Linux 下多线程遵循 POSIX(可移植的操作系统接口) 线程接口。和进程相比，线程是一种非常“节俭”的多任务操作方式，不但创建速度快，而且线程间通信也很方便，容易维护。因此，为了提高并发服务器的可靠性和高效性我们采用多线程机制。

1.2.1　linux 多线程编程

Linux 操作系统提供了 linux Threads 库，在这个库中提供了在 linux 系统下线程编程的相关函数。需要注意的一点: 进行多线程编程必须包含 pthread.h 头文件，在用 gcc 命令进行编译链接时，必需使用—lpthread 编译选项手动连接 libpthread.a 库。相关函数的原型如下所示[3] 。

(1) 线程创建

int pthread _ create (pthread _ t * ptid, const pthread_attr_t * attr,

void * (* start_rtn) (void *) ,void * arg) 功能: 创建子线程

(2) 线程退出

void pthread_exit(void * rval_ptr) 功能: 终止调用的线程

(3) 线程等待

int pthread_join(pthread_t ptid,void * * rval_ptr) 功能: 阻塞调用线程,直到指定的线程终止

(4) 线程异常处理函数

void pthread_cleanup_push(void (* rtn) (void *) ,void * arg) 功能: 将清除函数压入到清除栈

void pthread_cleanup_pop(int execute) 功能: 将清除函数弹出清除栈

1.2.2 线程池技术

在 linux 系统中,采用多线程机制可以实现服务器的并发请求,但对于高并发服务器而言,这里面存在一个致命的安全隐患[4]。因为系统每创建一个线程,都会为该线程分配一定的系统资源,所以一个进程创建的子线程数是有限制的。如果在同一时刻有大量的客户端并发请求服务器,这时服务器的主线程就不断地创建子线程处理连接到来的客户端,这样系统的资源就慢慢消耗殆尽,如果此时还有其他客户端请求服务器,这时服务器就没有额外资源来创建线程来处理这个客户端到来的连接,这时候就会引发系统异常,甚至造成整个服务器系统崩溃。

对此本文提出了线程池的解决方案,使高并发的服务器在有限资源的 linux 系统中得到实现。在服务器的主线程中预先创建含有限个任务线程的线程池,一旦有客户请求到来就启动这些任务线程来处理客户端到来的连接。超出部分的客户就先在任务队列中等待,等任务线程处理完任务后成为空闲的任务线程,空闲出来的任务线程就可以转而执行等待队列中的任务。线程池工作原理如图 2 所示。

主线程: 初始化线程池结构体变量。在初始化线程池变量时一次性创建并且启动 m 个任务线程的线程池。线程池变量中有一个记录任务链表节点数的变量 n,当主线程向线程的任务链表头(H) 中抛入一任务节点就让任务链表节点数加 1,同时调用 pthread_cond_signal 函数唤醒空闲挂起的任务线程。

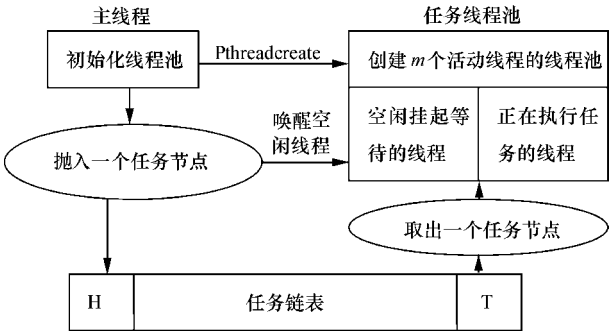

图 2 线程池工作流程图

任务线程池: 任务线程是在线程池结构体变量初始化时启动的线程。任务线程池中的任务线程如果检测到任务链表中的任务节点数不为零就从任务链表尾(T) 取出任务节点并执行相应的任务,同时让任务链表中的任务节点数减 1。如果任务链表中的任务数为零,就调用 pthread_cond_wait 使任务线程挂起,等待主线程向任务链表中抛入一个任务并且调用 pthread_cond_signal 再次唤醒空闲挂起的任务线程执行任务。

1.3 linux2.6 内核的 I/O 复用 epoll 机制

1.3.1 epoll 机制的工作原理

在 linux 系统下开发高并发 TCP 连接的服务器程序时,必须要选用合适的 I/O 技术和 I/O 分派机制。并发的服务器设计在很长一段时间内使用的是 I/O 技术中的 select 机制。

但是 select 机制存在着两点致命的缺陷: 第一,select 是采用的轮询的机制,其工作效率与监听的事件数成反比。第二,select 监听的事件的数量是有明显限制的。由于存在这两个“瓶颈”,select 监听机制并不能实现真正意义上的高并发服务器。

在 linux2.6 以上版本的新内核中提出来 epoll 监听机制。epoll 机制的工作原理: 它无须遍历整个被侦听的描述符集,如果注册在 epoll 等待队列中的 socket 文件描述符事件状态发生了变化,内核就直接将这些事件放到 events 数组中直接返回,这点明显不同与 select 机制的轮询策略。因此它的这种触发机制,就不会随着监听事件数的增加而效率显剧下降。epoll 中所监听的事件数没有限制,它只与系统的资源有关。

1.3.2 epoll 机制的编程函数

epoll 的相关编程函数的声明在头文件 sys/epoll.h 头文件中。其编程函数的原型如下。

(1) int epoll_create(int size)

功能: 创建一个 epoll 文件描述符。

(2) int epoll_ctl(int epfd, int op, int fd, struct epoll_ent * ev)

功能: 控制 epoll 文件描述符上的事件，包括事件添加、删除以及修改。

(3) int * epoll_wait(int epfd, struct epoll_event * events, int max, int timeout)

功能: 等待注册在 epoll 文件描述符上的事件的产生。一旦 epoll 文件描述符注册的事件的状态发生了变化，就将这些事件用 events 事件数组提取出来进行处理。

2　基于线程池 + epoll 技术高并发服务器实现

前面已经讲述了基于套接字的客户端/服务器的 TCP 网络应用程序编写的框架模型、linux 下多线程编程和 I/O 复用技术中的 epoll 机制。由分析可知，传统多线程并发服务器存的一些问题归根到底是由于系统资源有限造成的。本文给出在资源有限的 linux 系统下基于线程池和 epoll 机制的高并发服务器详细算法如下图 3 所示。

2.1　算法的工作原理

该算法主要是实现主线程和线程池中的任务线程之间的协同工作。主线程和任务线程池中的线程的工作过程如下。

主线程: 第一步，调用一个线程的初始化函数来初始化线程池结构体变量中的互斥锁变量和条件变量，以及创建一个包含 *m* 个活动线程的任务线程池。第二步，调用 epoll_create 函数来创建一个 epollfd 的文件描述符。第三步，调用 socket 相关函数来创建监听套接字 listenfd。第四步，调用 epoll_ctl 函数将监听套接字 listenfd 注册到 epoll 事件的等待队列中。第五步，在一个“死循环”中，调用调用 epoll_wait 阻塞等待注册在 epoll 上的 sockfd 事件的

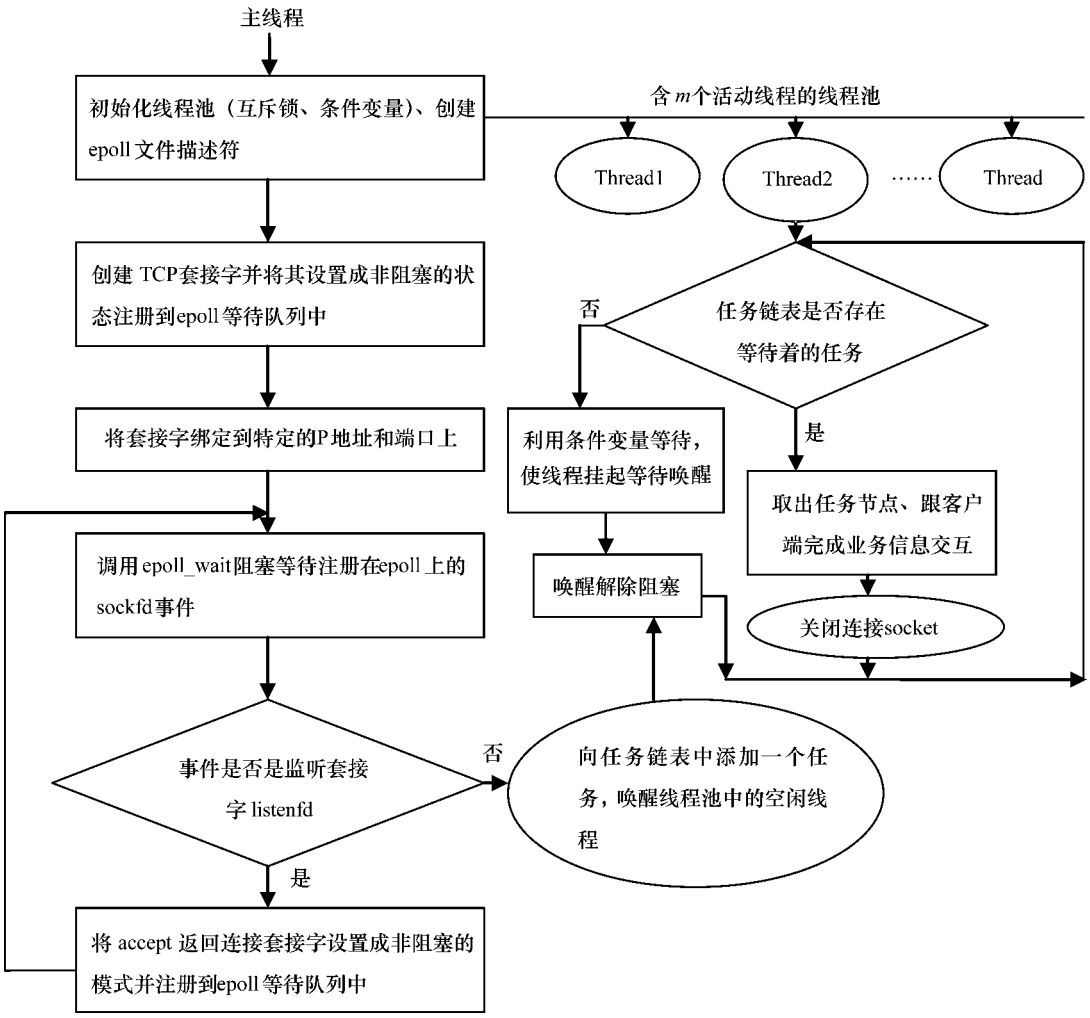

图 3　epoll + 线程池高并发服务算法

发生，依次处理到来的事件，如果到来的的事件是监听 sockfd 事件，就调用 accept 函数来产生连接套接字 connfd，并且将这个 connfd 也设置成非阻塞的状态注册到 epoll 的事件等待的队列中，如果到来的事件是 connfd 事件就表明有已经连接的客户端的通信数据到来，就向任务线程池中抛入一个任务。并且调用 pthread_cond_signal 函数来唤醒空闲的任务线程来处理相关的数据通信。

线程池：每一个任务线程都在一个 while 的“死循环”中不断的检测任务链表中是否有任务节点，如果没有任务节点，就调用 pthread_cond_wait 函数将该线程设置成为空闲阻塞等待的状态，等待任务链表中有任务节点后唤醒该空闲的任务线程来处理任务。如果任务链表中存在着任务节点，就取出任务作相应的任务处理。通信完毕后就调用 close 函数关闭相应的连接套接字。

2.2 高并发通信系统编程实例

下面我们结合上面讲述的线程池 + epoll 机制的高并发服务器的算法，给出一个具体的 linux 下用 C 语言编写的通信实例。服务器和客户端实现的功能如下。

服务器：先初始化一个含有 2 个活动的任务线程的线程池，创建 epoll 文件描述符，然后，将监听套接字加入到 epoll 事件等待队列中，当有客户端连接到来时，就显示连接客户端的点分十进制的 IP 地址，然后将 accept 返回的连接套接字注册到 epoll 等待队列中。当已经连接的客户有数据要发送的时候，就向线程池任务链表中添加一个任务节点，并唤醒空闲的任务线程进行信息的接收，并将接收到信息打印显示，随后将接受到的信息原样返回给客户端。客户端：建立 TCP 连接套接字，与服务器进行连接。连接成功则显示 connect sucessfully，然后提示用户输入等待发送的信息，随后接收服务器返回的信息并打印出来。

3 结果及分析

为了体现服务器并发请求和线程池处理多任务的功能，将线程池中的线程的活动情况实时打印出来，用 2 个活动的线程处理 3 个客户并发请求。在同一台虚拟机下，先启动 1 个终端运行服务器程序，随后启动 3 个终端运行客户端程序，第一个客户端向服务器发送球类的信息，第二个客户端向服务器发送动物类的信息，第三个客户端向服务器发送家具信息。运行程序使用本机的回路地址 127.0.0.1。实验的结果截图如图 4—图 7 所示。

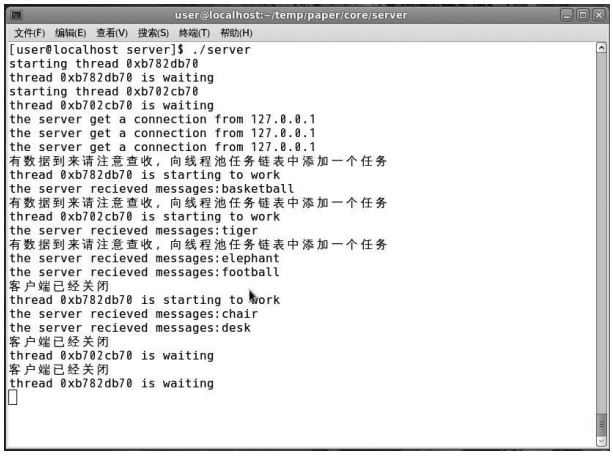

图 4 服务器截图

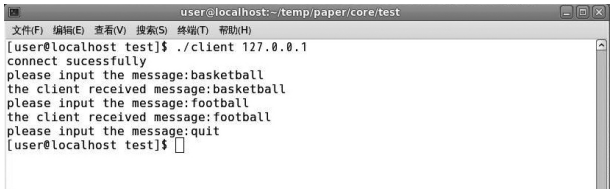

图 5 客户端 1 截图

图 6 客户端 2 截图

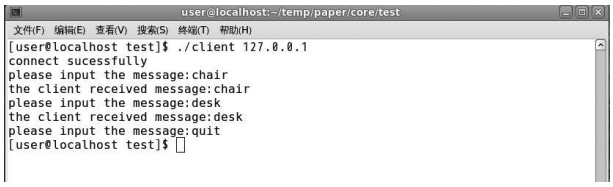

图 7 客户端 3 截图

从上面的客户端和服务器的截图可知，服务器：创建一个含有 2 个活动线程的线程池，打印出这 2 个任务线程的线程 ID，并且使这 2 个活动线程处于等待状态。当 3 个客户端请求到来时，就分别打印出这三个客户端的的 IP 地址。随后实时的打印客户端发送的数据，以及线程池中任务线程的工作状态。客户端：当第一个客户端向服务器发送球类数据的同时第二个客户端也可以向服务器发送动物类的数据，但是这时第三个客户端向服务器端发送数

据就不能被打印出来，因为这个时候两个活动的线程都处于繁忙的状态，所以就无法处理第三个客户端到来的任务请求，故这个任务请求只能是保存在任务链表中。当第一个客户端将数据处理完后，任务线程池就有空闲的线程来处理第三个客户到来的任务请求，可进行相应的数据通信，并且将其通信的信息打印输出。因为这里客户端数多于线程池中活动线程数，但是服务器和客户端之间仍然可以高效稳定地进行通信。由此可知，线程池和 epoll 机制巧妙的结合能开发出可靠稳定高并发服务器程序。

4　结束语

本文通过介绍线程池技术和 epoll 多路复用的 I/O 机制，揭示了传统并发服务器存在的一些问题。针对这些问题，提出了基于 epoll + 线程池技术高并发服务的详细算法。这种算法可以大大提高服务器并发量，很好克服由于系统的资源限制而导致系统异常的问题。本文最后给出一个 linux 下基于这种算法的通信实例，以及相应的实验结果。从结果分析可知，基于这种算法的高并发服务器是完全有可以实现的。

参考文献:

[1] 谢希仁. 计算机网络 [M]. 北京: 电子工业出版社，2009.

[2] 孙鑫，余安萍. VC + + 深入详解 [M]. 北京: 电子工业出版社，2006.

[3] Richard Stevens W，Stephen A Rago. UNIX 环境高级编程 [M]. 尤晋元，张亚英，戚正伟. 北京: 人民邮电出版社，2006.

[4] 刘新强，曾兵义. 用线程池解决服务器的并发请求的方案设计 [J]. 现代电子技术，2011(8).

(上接第 53 页)

向敏感性，提高了配准精度，使检测可靠性较高; ③制作高质量的样本不占用检测时间。

算法有待改进之处: 当待测图像的对比度不高时，应先增强图像再进行检测。同时，阈值 M 的选取会影响实验结果。

4.2　实验结果

实验采集 PCB 图像的分辨率为 800 dpi，两像素之间的距离约为 31 μm，而 PCB 板中导电区的导线宽约为 100 μm，要将影响 PCB 线路导通的缺陷处检出，故将其连通区规模控制为至少 3 点，只有高于三点的点集中程度才能构成缺陷。按照本文所提出的检测方法，能检测出短路、断路、毛刺、缺口等缺陷。检测一块 80 × 100 mm 的 PCB 板。所花费的时间是 30 ms。图 5 为检测到待测板的毛刺、短路缺陷。

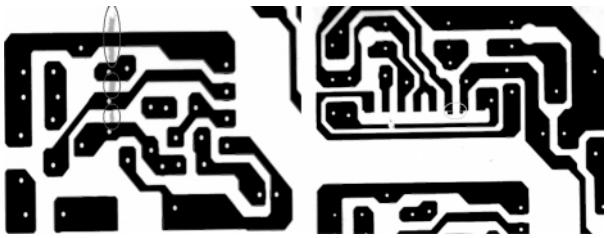

图 5　待测板缺陷

5　结束语

缺陷检测算法研究作为印刷电路板自动光学检测系统开发的一个主要内容，研究目标可归纳为提高系统的实时性和精度，这就要求利用缺陷算法快速准确的搜索到缺陷处，本文结合边缘梯度矢量提出了一种仅用轮廓点检测缺陷实现方法，实验表明: 该算法是一个计算量小、实现简单、检测精度和准确性较高的算法，能够满足工业应用的 AOI 系统的实时性要求。

参考文献:

[1] 刘国忠，李运生，周玺，等. 基于分层参考比对法印刷电路板自动检测技术 [J]. 制造业自动化，2010，32(10): 6-10.

[2] 范小涛，夏雨人. 一种基于图像处理 PCB 检测算法研究 [J]. 计算机工程与应用，2004，40(13): 91-93.

[3] 贾永红. 数字图像处理 [M]. 武汉: 武汉大学出版社，2003.

[4] 何伟，李薇，张玲. 基于计算机图像处理的电路印刷板缺陷检测 [J]. 计算机测量与控制，2007，15(10): 1295-1297.

[5] 胡涛. 基于轮廓对比的 PCB 裸板缺陷检测算法研究 [D]. 武汉: 华中科技大学，2009.