Sockets Backlog Parameter Explanation

Here are some web sites explaining backlog for sockets:
http://www.linuxjournal.com/files/linuxjournal.com/linuxjournal/articles/023/2333/2333s2.html
http://blog.stephencleary.com/2009/05/using-socket-as-server-listening-socket.html

A very good explanation is shown below and the links are for the website that contains the information
The following information comes from section 4.14 of the document:
http://tangentsoft.net/wskfaq/advanced.html (Winsock programmers FAQ)
http://tangentsoft.net/wskfaq/advanced.html#backlog (link to backlog part of the document)

4.14 - What is the connection backlog?

When a connection request comes into a network stack, it first checks to see if any program is listening on the requested port. If so, the stack replies to the remote peer, completing the connection. The stack stores the connection information in a queue called the connection backlog. (When there are connections in the backlog, the `accept()` call simply causes the stack to remove the oldest connection from the connection backlog and return a socket for it.)

One of the parameters to the `listen()` call sets the size of the connection backlog for a particular socket. When the backlog fills up, the stack begins rejecting connection attempts.

Rejecting connections is a good thing if your program is written to accept new connections as fast as it reasonably can. If the backlog fills up despite your program's best efforts, it means your server has hit its load limit. If the stack were to accept more connections, your program wouldn't be able to handle them as well as it should, so the client will think your server is hanging. At least if the connection is rejected, the client will know the server is too busy and will try again later.

The proper value for `listen()`'s `backlog` parameter depends on how many connections you expect to see in the time between `accept()` calls. Let's say you expect an average of 1000 connections per second, with a burst value of 3000 connections per second. (I picked these values because they're easy to manipulate, not because they're representative of the real world!) To handle the burst load with a short connection backlog, your server's time between `accept()` calls must be under 0.3 milliseconds. Let's say you've measured your time-to-accept under load, and it's 0.8 milliseconds: fast enough to handle the normal load, but too slow to handle your burst value. In this case, you could make `backlog` relatively large to let the stack queue up connections under burst conditions. Assuming that these bursts are short, your program will quickly catch up and clear out the connection backlog.

The traditional value for `listen()`'s `backlog` parameter is 5. This is actually the limit on the home and workstation class versions of Windows. On Windows Server, the maximum connection backlog size is 200, unless the dynamic backlog feature is enabled. (More info on dynamic backlogs below.) Because the stack will use its maximum backlog value if you pass in a larger value, you can pass a special constant, `SOMAXCONN`, to `listen()` which tells it to use whatever the platform maximum is, since the constant's value is 0x7FFFFFFF. There is no standard way to find out what backlog value the stack chose to use if it overrides your requested value.

If your program is quick about calling `accept()`, low backlog limits are not normally a problem. However, it does mean that concerted attempts to make lots of connections in a short period of time can fill the backlog queue. This makes non-Server flavors of Windows a bad choice for a high-load server: either a legitimate load or a `SYN` flood attack can overload a server on such a platform. (See below for more on `SYN` attacks.)

Beware that large backlogs make `SYN` flood attacks much more, shall we say, effective. When Winsock creates the backlog queue, it starts small and grows as required. Since the backlog queue is in non-pageable system memory, a `SYN` flood can cause the queue to eat a lot of this precious memory resource.

After the first `SYN` flood attacks in 1996, Microsoft added a feature to Windows NT 4.0 SP3 called the “dynamic backlog.” This feature is normally off for backwards compatibility, but when you turn it on, the stack can increase or decrease the size of the connection backlog in response to network conditions. (It can even increase the backlog beyond the normal maximum of 200, in order to soak up malicious `SYN`s.) The Microsoft Knowledge Base article that describes the feature also has some good practical discussion about connection backlogs.

You will note that `SYN` attacks are dangerous for systems with both very short and very long backlog queues. The point is that a middle ground is the best course if you expect your server to withstand `SYN` attacks. Either use Microsoft’s dynamic backlog feature, or pick a value somewhere in the 20-200 range and tune it as required.

A program can rely too much on the backlog feature. Consider a single-threaded blocking server: the design means it can only handle one connection at a time. However, it can set up a large backlog, making the stack accept and hold connections until the program gets around to handling the next one. (See this example to see the technique at work.) You should not take advantage of the feature this way unless your connection rate is very low and the connection times are very short. (Pedagogues excepted. Everyone else designing such a program should probably use 1, 0, or even close the listening socket while there is a client connected.)