APR: What Is It, and Why We Use It In Apache

Ryan Bloom

Why Programs Requires a Portable Run-Time 1
Why Write the Apache Portable Run-Time 1
What Does APR Provide 2
Status Values 3
File I/O 3
Network I/O 3
Critical Section Locking 4
Memory Mapped Files 4
Threads and Processes 4
Time 5
Memory Management 5
Functions Taken from Apache 1.3 5
Portability Routines 6
Shared Memory 6
APR Success Stories 6
Future Work in APR 6
ApacheBench with APR 7
ApacheBench without APR 20

Why Programs Requires a Portable Run-Time

Apache Portable Run-time (APR) is a library that provides a consistent API for standard system calls across all supported platforms. This allows programs to use native calls on all platforms while keeping the base code clean and easy to follow.

As programs are ported to platforms that do not support POSIX, they become harder to maintain. This is because most programs are ported using #ifdefs, so looking at the code, it is very difficult to determine which lines are used on which platforms. There is also the problem of native APIs. Most platforms that provide a set of native APIs do so because they perform better than POSIX calls on that platform. If this is the case, then it is always better for programs to use native calls when possible, APR makes it possible to use native calls without littering the code with more #ifdefs.

There are two basic options from which to choose when trying to fix these problems. The first is to continue using #ifdef's to separate code for different platforms. This has the advantage of keeping all of the code in one place, but it has the disadvantage of the code being much harder to read and to modify. It is also likely that a bug fix on one platform will not be carried over to all of the other platforms supported when using this method. The second option is to use a portable run-time to make all of the platforms look alike to the program. This option also had advantages and disadvantages. Using this option makes the code much easier to read and understand. It is also increases the probability that a bug fix will propagate through to all platforms. The disadvantage is some loss of control in the program. Because it is important to ensure that all platforms can work with code written for APR, APR uses incomplete types, meaning it is not possible to get access to the platforms' native types.

Why Write the Apache Portable Run-Time

It is always possible for programmers to create a new portable run-time for each project that requires one. However, there is no reason to duplicate effort and ignore the progress made by other groups who are focused on creating a portable run-time. When the developers started APR we wanted to focus on Apache, but we knew we needed a portable run-time, so we went looking for what was already available. The main options were to either use the Netscape Portability Run-Time (NSPR), the ADAPTIVE Communication Environment (ACE), or to try to convince a big company to Open-Source their existing run-time.

NSPR was an obvious choice because Dean Gaudet had already used it once to create a portable threaded version of Apache. NSPR was well designed and had already been proven to work cross-platform. There were a few problems that kept NSPR from being used with Apache. First, the initial license for NSPR was not compatible with the Apache license, which would have kept some of the Apache Group members from being able to contribute to Apache 2.0. After a discussion of the issue, Netscape promised to modify its license to accommodate Apache's needs. However, a few months passed and the release with the new license never came. A month after work was started on APR, NSPR was finally released with the compliant license. At this point, the ASF made a technical decision not to use NSPR because it had too many features that Apache didn't require. However, rather than ignore the work already done by NSPR, the APR developers are learning from it. Many times when design decisions are needed, we look to NSPR for guidance.

The next option was a package by the name of ACE. However, ACE was written in C++ not C, which basically made it useless to Apache. The Apache Group has had a long-standing position of not using C++ in Apache in order to make it easier for new people to join the Apache effort. The general feeling within the Apache Group is that more people know C than know C++ and that those who do know C++ can write C code.

The final option was to get a proprietary run-time library Open Sourced. Unfortunately, while it is generally easy to find closed source run-time libraries, getting them opened is usually quite difficult. This option was being explored in early 1999 – before most of the big companies started rushing to Open Source their code. Because of the huge amount of red tape involved with this alternative, this course was abandoned early on.

After exhausting these possibilities, the Apache Group began writing its own run-time library. At this point, the scope of APR needed to be established. APR could be limited to basic system functions or be broad enough to provide any function that is portable across platforms. Because Apache had already implemented some functions that would not be considered system functions in a portable manner, it was decided to go ahead and implement any function that was written to be portable across platforms.

What Does APR Provide

The first version of APR only provides functions that are useful to a server application because of the importance of its inclusion in Apache 2.0. Future versions of APR may provide additional functions that are not required by Apache. The following list of necessary functions was created by looking at the NSPR port that Dean created:

- Status Values
- File I/O
- Network I/O
- Critical Section Locking
- Memory Mapped Files
- Threads and Processes
- Time
- Memory Management
- Functions taken from Apache 1.3
- Portability Routines
- Shared Memory

Because most of these functions are implemented using incomplete types, application programmers can not just access internal fields for each type. Each incomplete type therefore provides a set of routines that allow access to the internal fields. For example, it is possible to retrieve a file's name by calling apr_get_filename and passing in an open file type.

Each APR type contains a pointer to a pool (these are described in the Memory Management section) except for the time types. The pool is used for memory management and to attach user data to any APR type.

Status Values

The status values in APR fall into four categories: *system error*, *error value*, *status value*, & *success*. *System errors* are values between zero and APR_OS_START_ERROR. These errors are returned when an APR function fails to complete because a native function has failed. If a platform does not provide a particular error value, such as EAGAIN, APR will provide a value APR_EAGAIN that can be used in its place.

APR *error values* are values between APR_OS_START_ERROR and APR_OS_START_STATUS. This category of status codes will be returned if an APR function fails to complete for any reason that is not related to the platform. For example, if an un-initialized file is passed to apr_get_filename, APR_ENOFILE will be returned.

APR *status values* are between APR_OS_START_STATUS and APR_OS_START_SYSERR. This category is used when an APR function has completed successfully but has more information that it must express. For example, when apr_wait needs to inform the caller that a child process is still running, it returns APR_CHILD_NOTDONE, but when it wishes to express that a child process has finished executing, it will return APR_CHILD_DONE.

The *success* value is returned whenever an APR function completes successfully and has no other information to express. It is represented by APR_SUCCESS.

File I/O

File I/O is used whenever we are reading from or writing to a file or pipe. Because of the differences between platforms, this is one place where the need for APR becomes very apparent. Files are handled very differently between Unix and Windows. Windows provides a HANDLE type that is used for files. Unix provides both buffered and unbuffered files, represented by an integer and a FILE type respectively. These differences make it very difficult to write code that works on both platforms. To solve this problem, APR has provided its own type apr_file_t that represents files on all supported platforms. All of the regular file operations are supported through this type. It is possible to open, read, write, close, delete, duplicate, and get a file's information. It is also possible to check for end of file and get the name of a file pointed to by a specific apr_file_t.

Getting a file's information is a bit different from other APR functions. APR's file information structure is one of the few non-incomplete types provided by APR. This is because the information it is possible to retrieve about files is pretty much the same across platforms. Information not provided by some platforms is represented by an error value. For example, on Windows the user field is always APR_ENOTIMPL (this value may be changed, or it may be implemented using ifdefs soon).

File I/O is also used for piped communication. As people who have followed Apache on Windows know, pipes on Windows do not always provide the same level of support that Unix pipes allow. For example, not all pipes on Windows can be timed out. To handle this, APR includes the ability to attach a timeout value to files. This value is meaningless on regular files, but on pipes it determines how long a program will attempt to read from or write to a file until it gives up.

Finally, File I/O encompasses directory access. Directories are represented using the apr_dir_t type. They can be opened, closed, created, read, and deleted if empty.

Network I/O

Network I/O is used for communicating over a network. Currently, the only transport layer supported is TCP. This will most likely change in time. Sockets are represented using the apr_socket_t type. It is possible to attach a timeout value to sockets to determine how long to wait while trying to read or write. If the timeout value expires, APR_ETIMEUP will be returned. Sockets can be bound to ports by first assigning a port to the socket using

apr_set_local_port and then calling apr_bind. Once bound, sockets are ready to listen for connections, which is done using apr_listen. APR also provides the apr_connect routine, which will make a connection between any two TCP sockets.

Reading and writing to sockets is accomplished using the apr_send and apr_recv routines. There has been some work done to implement apr_sendfile. If APR supports sendfile on a particular platform, the APR_HAS_SENDFILE feature macro will be defined to be TRUE.

Network I/O also incorporates polling sockets. This is achieved through the use of the apr_poll routines. These routines cannot be used to poll pipes or files, although the Apache Group hopes to eventually incorporate this ability. To set up a polling structure, a user must first call apr_setup_poll, then add the sockets to the poll structure by calling apr_add_poll_socket. To poll the current set of sockets, apr_poll should be called. It is then possible to check the results by calling apr_get_revents for each socket in the set. At some point in the future, APR will be modified to include a function that will return a list of sockets that were affected by the poll. Finally, t is possible to remove individual sockets from a poll set and to clear the results of the previous poll in anticipation of the next call to apr_poll.

Critical Section Locking

Critical sections are those sections of code that should only have either one thread or one process in them at any one time. This is accomplished using the apr_lock_t type. There are three ways to lock a section of code: APR_CROSS_PROCESS, APR_INTRAPROCESS, and APR_LOCKALL. An APR_CROSS_PROCESS lock is guaranteed to never allow more than one process into the affected section of code. This is useful if a platform suffers from thundering herd problems (waking up all processes waiting on a socket when a connection is established). It is important to note that this type of lock may or may not lock out threads as well as processes depending on the platform. An APR_INTRAPROCESS lock is basically a thread lock. This lock will guarantee that a section of code has only one thread in it at any given time. Finally, an APR_LOCKALL lock will protect a critical section from other threads and other processes. This is the safest kind of lock to use, because it guarantees that no other execution primitive will be allowed into the critical section. This is also usually a heavier weight lock than APR_CROSS_PROCESS, and should only be used if it is really what is needed. For example, if a program never spawns threads this lock should not be used because it is more expensive than a simple APR_CROSS_PROCESS.

Memory Mapped Files

Memory mapped files in APR are represented by apr_mmap_t types. Although memory mapped files are sometimes used as shared memory on certain platforms, apr_mmap_t types are not suitable for this use. APR has only implemented a very small subset of mmap'ed file uses because it needed to remain portable. APR's mmaped files can only be created, deleted, and read from. In order to create an mmap type, the file must be represented by an APR file type. If APR supports mmap'ed files on a platform the APR_HAS_MMAP feature macro will be defined as TRUE.

Threads and Processes

Threads are represented in APR by apr_thread_t types. It is possible to create and destroy threads. Threads can be created in either detached or non-detached states. If a thread is detached, then the resources it uses will be freed immediately when it dies. However, this will keep other threads in the same process from knowing when the thread has terminated. If the thread is not detached, then there must be another thread in the same process which is joined on that thread. This joined thread is then notified when the other thread has finished executing. It is possible to detach a currently running thread by calling apr_detach. It is also possible to kill a currently running thread. This is not recommended though because on most platforms, processes leak memory or become unstable when one of their threads are terminated.

Data can also be attached to currently running threads. This is commonly referred to as thread-local storage or thread-private storage. This data is represented as apr_threadkey_t types in APR. It is possible to attach data to a thread and retrieve that data. The data is associated with a key value that is used for retrieval.

Processes can also be created in APR. APR does provide an apr_fork function but it is not portable. If your platform provides an apr_fork function, then the APR feature macro APR_HAS_FORK is defined to be TRUE. The more portable method of creating a new process is apr_create_process. The difference between the two is that apr_fork creates a new process and starts that process at the same place in the code whereas apr_create_process creates a new process and executes a new program. It is possible when using apr_create_process to setup up to three pipes to be used as stdin, stdout, and stderr.

There are two more operations for processes. The first is determining if a child process has terminated. This function returns either APR_CHILD_DONE or APR_CHILD_NOTDONE. In the second operation the user can send a signal to another process. Currently, the only signal really supported is the signal to terminate the process. It is very possible that the apr_kill function will be redefined to only allow this signal in the near future.

Time

Time is the only APR category that does not use any incomplete types. There are two different time types in APR. The first is apr_time_t, which is the number of microseconds since the epoch, defined as 00:00:00 January 1, 1970 UTC. The time library also has one of the only functions that doesn't return a status value. This function is apr_now, which returns the current time in apr_time_t format. This function should be used like a thread-safe global variable. The second time type is apr_exploded_time_t. This is a broken down representation of the time value. There are functions to convert between the two time types, as well as a couple of functions to convert the time value to a character string. The first two functions to convert to a string use a common string format. The first, apr_rfc822_date, uses the string format specified by RFC 822. The second, apr_ctime, uses the same format as the POSIX function ctime. Finally, there is a function to write the time in a user-defined format, apr_strftime, which uses the same format specifiers as the POSIX strftime function.

Memory Management

Memory management is implemented using pools in APR. Pools contain a pointer to memory pools, a pointer for user data, and a function pointer. The memory pools do not require the programmer to explicitly free the memory. All memory is allocated using either apr_palloc or apr_pcalloc. When all of the memory in a specific pool is no longer needed, it is all freed using either apr_clear_pool or apr_destroy_pool. It is also possible to create a sub-pool by calling apr_create_pool and supplying a non-NULL pool. Anytime memory is allocated out of a pool, it is possible to register a function to be used to cleanup the memory when the pool is freed. The user data is associated with a key and can be set and retrieved. It is currently not possible to de-associate user data, but it will automatically be freed when the pool is destroyed. Lastly, the function pointer is used when an error occurs while allocating memory. If the function is NULL, then the memory allocation routine will return an error condition. If it is not NULL, the function is executed. This ability to register a function has been provided to allow backward compatibility with Apache 1.3. In Apache 1.3, if memory allocation failed Apache quit immediately and output an error string.

Functions Taken from Apache 1.3

There is a set of functions taken directly from the Apache 1.3 ap library, including tables, getopt, apr_snprintf, apr_cpystrn, and routines for MD5 encryption. Because they were included in Apache 1.3, these functions will not be discussed in detail here.

Portability Routines

One of the great truths that APR developers have had to live with is that not all programs currently use APR (hopefully that will change one day). This is driven home when dealing with Apache, mod_php, and mod_perl. Mod_php and mod_perl are two of the biggest modules used with Apache; however, neither of these modules is likely to move to APR anytime soon. Despite this, it important to be sure that they will work with Apache 2.0. If Apache requires APR, then it must be easy for programmers to go from APR to native types. To accommodate this, there are a set of functions named apr_get_os_* and apr_put_os_*. These two functions convert from APR to native types and back respectively. These are very stable functions, but they take time to perform, and for this reason they should only be used when transitioning from an APR program to one which doesn't use APR.

Shared Memory

Currently, APR's shared memory support is not complete. We are using Ralf Engelschall's MM library to provide shared memory on most platforms, but MM does not work on non-Unix platforms. Work is currently progressing to provide shared memory on Windows.

APR's shared memory is very basic, providing a way to allocate memory, apr_shm_malloc and apr_shm_calloc, and a way to free that memory, apr_shm_free. Shared memory is implemented in a pool fashion, so shared memory must be initialized and destroyed. This is done with apr_shm_init and apr_shm_destroy. When using MM, shared memory is always anonymous. This means that all of the shared memory is opened in the parent and inherited by the child processes. APR does not impose this limitation. This means that if the memory is not anonymous, then the program must be able to get and set the name of the shared memory. This is done through apr_get_shm_name and apr_set_shm_name. There are two options for shared memory names in APR. Either the shared memory is named as a file name, or a key. If the memory is implemented using a file, the macro APR_USES_FILEBASED_SHM is 1, if it is named with a key, then APR_USES_KEYBASED_SHM is 1. If the shared memory is anonymous, then APR_USES_ANONYMOUS_SHM is 1. There are two more operations for shared memory in APR, apr_open_shmem, which opens a shared memory block in a child process, and apr_shm_avail, which determines how much shared memory is available.

APR Success Stories

APR has already proven itself as a viable library. ApacheBench is a benchmarking tool that has always come with Apache. The problem is that ApacheBench historically only worked on Unix systems. There has been some work recently to port ApacheBench to Windows, but the Windows modifications were very intrusive when reading the code. While APR was in development, it was used to easily port ApacheBench to every APR supported program. ApacheBench now works on Unix, Windows, OS/2, and BeOS. The changes made to ApacheBench are minimal and have little to no effect on the readability of the code. Both version of ApacheBench have been included at the end of this document.

APR has also been used to port htpasswd, a utility used to create Apache password files. One final success story is htdigest, a utility for manipulating digest password files. The Apache Group is currently in the process of porting suexec, logresolve, and rotatelogs to APR. These are all programs from the Apache support directory. It is projected that some non-Apache-related programs will be ported to APR after Apache 2.0 is released.

Future Work in APR

APR version 1.0 is very close to being finished. However, there is still much that can be done to make APR more attractive to people who are writing programs that need to work on multiple platforms. APR also need to be tested on more platforms. APR has thus far been tested on BeOS, Windows, OS/2, Linux, AIX, FreeBSD, and Digital Unix. This is a very small number of platforms in comparison to the number of platforms that Apache currently runs on.

Every APR library has a corresponding test program. The more platforms which validate that the tests always work, the better APR will become.

Finally, APR needs to be publicized. APR is a very robust and flexible tool, but that doesn't do anybody any good unless large numbers of programmers know about it. Having Apache 2.0 rely on APR will go a long way towards getting its name out to other programmers, but that is not enough. There has been talk of getting some bigger companies to use APR internally in some of their projects. With the Apache Software Foundation expanding into ever more projects, it is hoped that more ASF projects will choose to use APR to make their code easier to read and easier to port to more platforms.

ApacheBench with APR

```
/* ====================================================================
 * The Apache Software License, Version 1.1
 *
 * Copyright (c) 2000 The Apache Software Foundation.  All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The end-user documentation included with the redistribution,
 *    if any, must include the following acknowledgment:
 *       "This product includes software developed by the
 *        Apache Software Foundation (http://www.apache.org/)."
 *    Alternately, this acknowledgment may appear in the software itself,
 *    if and wherever such third-party acknowledgments normally appear.
 *
 * 4. The names "Apache" and "Apache Software Foundation" must
 *    not be used to endorse or promote products derived from this
 *    software without prior written permission. For written
 *    permission, please contact apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache",
 *    nor may "Apache" appear in their name, without prior written
 *    permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 * ====================================================================
 *
 * This software consists of voluntary contributions made by many
 * individuals on behalf of the Apache Software Foundation.  For more
 * information on the Apache Software Foundation, please see
 * <http://www.apache.org/>.
 *
 * Portions of this software are based upon public domain software
 * originally written at the National Center for Supercomputing Applications,
 * University of Illinois, Urbana-Champaign.
 */

/*
   ** This program is based on ZeusBench V1.0 written by Adam Twiss
   ** which is Copyright (c) 1996 by Zeus Technology Ltd. http://www.zeustech.net/
   **
   ** This software is provided "as is" and any express or implied waranties,
   ** including but not limited to, the implied warranties of merchantability and
   ** fitness for a particular purpose are disclaimed.  In no event shall
   ** Zeus Technology Ltd. be liable for any direct, indirect, incidental, special,
   ** exemplary, or consequential damaged (including, but not limited to,
   ** procurement of substitute good or services; loss of use, data, or profits;
   ** or business interruption) however caused and on theory of liability.  Whether
   ** in contract, strict liability or tort (including negligence or otherwise)
   ** arising in any way out of the use of this software, even if advised of the
```

```
   ** possibility of such damage.
   **
 */

/*
   ** HISTORY:
   **    - Originally written by Adam Twiss <adam@zeus.co.uk>, March 1996
   **      with input from Mike Belshe <mbelshe@netscape.com> and
   **      Michael Campanella <campanella@stevms.enet.dec.com>
   **    - Enhanced by Dean Gaudet <dgaudet@apache.org>, November 1997
   **    - Cleaned up by Ralf S. Engelschall <rse@apache.org>, March 1998
   **    - POST and verbosity by Kurt Sussman <kls@merlot.com>, August 1998
   **    - HTML table output added by David N. Welton <davidw@prosa.it>, January 1999
   **    - Added Cookie, Arbitrary header and auth support. <dirkx@webweaving.org>,
April 1999
   **
 */

/*
 * BUGS:
 *
 * - uses strcpy/etc.
 * - has various other poor buffer attacks related to the lazy parsing of
 *   response headers from the server
 * - doesn't implement much of HTTP/1.x, only accepts certain forms of
 *   responses
 * - (performance problem) heavy use of strstr shows up top in profile
 *   only an issue for loopback usage
 */

#define AB_VERSION "1.3c"

/*  ------------------------------------------------------------------ */

#if 'A' != 0x41
/* Hmmm... This source code isn't being compiled in ASCII.
 * In order for data that flows over the network to make
 * sense, we need to translate to/from ASCII.
 */
#define NOT_ASCII
#endif

/* affects include files on Solaris */
#define BSD_COMP

#include "apr_network_io.h"
#include "apr_file_io.h"
#include "apr_time.h"
#include "apr_getopt.h"
#include "ap_base64.h"
#ifdef NOT_ASCII
#include "apr_xlate.h"
#endif
#ifdef HAVE_STRING_H
#include <string.h>
#endif
#ifdef HAVE_STDIO_H
#include <stdio.h>
#endif
#ifdef HAVE_STDLIB_H
#include <stdlib.h>
#endif
#ifdef HAVE_CTYPE_H
#include <ctype.h>
#endif

/* ------------------- DEFINITIONS -------------------------- */

/* maximum number of requests on a time limited test */
#define MAX_REQUESTS 50000

/* good old state hostname */
#define STATE_UNCONNECTED 0
#define STATE_CONNECTING  1
#define STATE_READ        2

#define CBUFFSIZE       2048

struct connection {
    apr_socket_t *aprsock;
    int state;
    int read;                 /* amount of bytes read */
    int bread;                /* amount of body read */
    int length;                     /* Content-Length value used for keep-alive */
    char cbuff[CBUFFSIZE];      /* a buffer to store server response header */
    int cbx;                  /* offset in cbuffer */
    int keepalive;            /* non-zero if a keep-alive request */
    int gotheader;            /* non-zero if we have the entire header in
                               * cbuff */
    apr_time_t start, connect, done;
```

```
    int socknum;
};

struct data {
    int read;                   /* number of bytes read */
    int ctime;                  /* time in ms to connect */
    int time;                   /* time in ms for connection */
};

#define ap_min(a,b) ((a)<(b))?(a):(b)
#define ap_max(a,b) ((a)>(b))?(a):(b)

/* -------------------- GLOBALS ---------------------------- */

int verbosity = 0;              /* no verbosity by default */
int posting = 0;                /* GET by default */
int requests = 1;               /* Number of requests to make */
int concurrency = 1;            /* Number of multiple requests to make */
int tlimit = 0;                     /* time limit in cs */
int keepalive = 0;              /* try and do keepalive connections */
char servername[1024];              /* name that server reports */
char hostname[1024];            /* host name */
char path[1024];                /* path name */
char postfile[1024];            /* name of file containing post data */
char *postdata;                     /* *buffer containing data from postfile */
apr_ssize_t postlen = 0;            /* length of data to be POSTed */
char content_type[1024];          /* content type to put in POST header */
char cookie[1024],              /* optional cookie line */
     auth[1024],                /* optional (basic/uuencoded)
                                 * authentification */
     hdrs[4096];                /* optional arbitrary headers */
int port = 80;                  /* port number */
time_t aprtimeout = 30 * APR_USEC_PER_SEC; /* timeout value */

int use_html = 0;               /* use html in the report */
const char *tablestring;
const char *trstring;
const char *tdstring;

int doclen = 0;                     /* the length the document should be */
int totalread = 0;              /* total number of bytes read */
int totalbread = 0;             /* totoal amount of entity body read */
int totalposted = 0;            /* total number of bytes posted, inc. headers */
int done = 0;                   /* number of requests we have done */
int doneka = 0;                     /* number of keep alive connections done */
int started = 0;                /* number of requests started, so no excess */
int good = 0, bad = 0;              /* number of good and bad requests */

/* store error cases */
int err_length = 0, err_conn = 0, err_except = 0;
int err_response = 0;

apr_time_t start, endtime;

/* global request (and its length) */
char request[512];
apr_ssize_t reqlen;

/* one global throw-away buffer to read stuff into */
char buffer[8192];

struct connection *con;             /* connection array */
struct data *stats;             /* date for each request */
apr_pool_t *cntxt;

apr_pollfd_t *readbits;
#ifdef NOT_ASCII
apr_xlate_t *from_ascii, *to_ascii;
#endif

/* ------------------------------------------------------- */

/* simple little function to write an error string and exit */

static void err(char *s)
{
    fprintf(stderr, "%s", s);
    exit(1);
}

/* simple little function to write an APR error string and exit */

static void apr_err(char *s, apr_status_t rv)
{
    char buf[120];

    fprintf(stderr,
            "%s: %s (%d)\n",
            s, apr_strerror(rv, buf, sizeof buf), rv);
    exit(rv);
```

```
}

/* ------------------------------------------------------- */

/* write out request to a connection - assumes we can write
   (small) request out in one go into our new socket buffer  */

static void write_request(struct connection *c)
{
    apr_ssize_t len = reqlen;
    c->connect = apr_now();
    apr_setsocketopt(c->aprsock, APR_SO_TIMEOUT, 30 * APR_USEC_PER_SEC);
    if (apr_send(c->aprsock, request, &reqlen) != APR_SUCCESS ||
        reqlen != len) {
        printf("Send request failed!\n");
    }
    if (posting) {
        apr_send(c->aprsock, postdata, &postlen);
        totalposted += (reqlen + postlen);
    }

    c->state = STATE_READ;
    apr_add_poll_socket(readbits, c->aprsock, APR_POLLIN);
}

/* ------------------------------------------------------- */

/* calculate and output results */

static void output_results(void)
{
    int timetaken;

    endtime = apr_now();
    timetaken = (endtime - start) / 1000;

    printf("\r
\r");
    printf("Server Software:        %s\n", servername);
    printf("Server Hostname:        %s\n", hostname);
    printf("Server Port:            %d\n", port);
    printf("\n");
    printf("Document Path:          %s\n", path);
    printf("Document Length:        %d bytes\n", doclen);
    printf("\n");
    printf("Concurrency Level:      %d\n", concurrency);
    printf("Time taken for tests:   %d.%03d seconds\n",
           timetaken / 1000, timetaken % 1000);
    printf("Complete requests:      %d\n", done);
    printf("Failed requests:        %d\n", bad);
    if (bad)
        printf("   (Connect: %d, Length: %d, Exceptions: %d)\n",
               err_conn, err_length, err_except);
    if (err_response)
        printf("Non-2xx responses:      %d\n", err_response);
    if (keepalive)
        printf("Keep-Alive requests:    %d\n", doneka);
    printf("Total transferred:      %d bytes\n", totalread);
    if (posting)
        printf("Total POSTed:           %d\n", totalposted);
    printf("HTML transferred:       %d bytes\n", totalbread);

    /* avoid divide by zero */
    if (timetaken) {
        printf("Requests per second:    %.2f\n", 1000 * (float) (done) / timetaken);
        printf("Transfer rate:          %.2f kb/s received\n",
               (float) (totalread) / timetaken);
        if (posting>0) {
            printf("                        %.2f kb/s sent\n",
                   (float) (totalposted) / timetaken);
            printf("                        %.2f kb/s total\n",
                   (float) (totalread + totalposted) / timetaken);
        }
    }

    {
        /* work out connection times */
        int i;
        int totalcon = 0, total = 0;
        int mincon = 9999999, mintot = 999999;
        int maxcon = 0, maxtot = 0;

        for (i = 0; i < requests; i++) {
            struct data s = stats[i];
            mincon = ap_min(mincon, s.ctime);
            mintot = ap_min(mintot, s.time);
            maxcon = ap_max(maxcon, s.ctime);
            maxtot = ap_max(maxtot, s.time);
            totalcon += s.ctime;
            total += s.time;
```

```
        }
        if (requests > 0) { /* avoid division by zero (if 0 requests) */
            printf("\nConnnection Times (ms)\n");
            printf("              min   avg   max\n");
            printf("Connect:    %5d %5d %5d\n", mincon, totalcon / requests, maxcon);
            printf("Processing: %5d %5d %5d\n",
                   mintot - mincon, (total / requests) - (totalcon / requests),
                   maxtot - maxcon);
            printf("Total:      %5d %5d %5d\n", mintot, total / requests, maxtot);
        }
    }
}

/* ------------------------------------------------------- */

/* calculate and output results in HTML  */

static void output_html_results(void)
{
    int timetaken;

    endtime = apr_now();
    timetaken = (endtime - start) / 1000;

    printf("\n\n<table %s>\n", tablestring);
    printf("<tr %s><th colspan=2 %s>Server Software:</th>"
           "<td colspan=2 %s>%s</td></tr>\n",
           trstring, tdstring, tdstring, servername);
    printf("<tr %s><th colspan=2 %s>Server Hostname:</th>"
           "<td colspan=2 %s>%s</td></tr>\n",
           trstring, tdstring, tdstring, hostname);
    printf("<tr %s><th colspan=2 %s>Server Port:</th>"
           "<td colspan=2 %s>%d</td></tr>\n",
           trstring, tdstring, tdstring, port);
    printf("<tr %s><th colspan=2 %s>Document Path:</th>"
           "<td colspan=2 %s>%s</td></tr>\n",
           trstring, tdstring, tdstring, path);
    printf("<tr %s><th colspan=2 %s>Document Length:</th>"
           "<td colspan=2 %s>%d bytes</td></tr>\n",
           trstring, tdstring, tdstring, doclen);
    printf("<tr %s><th colspan=2 %s>Concurrency Level:</th>"
           "<td colspan=2 %s>%d</td></tr>\n",
           trstring, tdstring, tdstring, concurrency);
    printf("<tr %s><th colspan=2 %s>Time taken for tests:</th>"
           "<td colspan=2 %s>%d.%03d seconds</td></tr>\n",
           trstring, tdstring, tdstring, timetaken / 1000, timetaken % 1000);
    printf("<tr %s><th colspan=2 %s>Complete requests:</th>"
           "<td colspan=2 %s>%d</td></tr>\n",
           trstring, tdstring, tdstring, done);
    printf("<tr %s><th colspan=2 %s>Failed requests:</th>"
           "<td colspan=2 %s>%d</td></tr>\n",
           trstring, tdstring, tdstring, bad);
    if (bad)
        printf("<tr %s><td colspan=4 %s >   (Connect: %d, Length: %d, Exceptions:
%d)</td></tr>\n",
               trstring, tdstring, err_conn, err_length, err_except);
    if (err_response)
        printf("<tr %s><th colspan=2 %s>Non-2xx responses:</th>"
               "<td colspan=2 %s>%d</td></tr>\n",
               trstring, tdstring, tdstring, err_response);
    if (keepalive)
        printf("<tr %s><th colspan=2 %s>Keep-Alive requests:</th>"
               "<td colspan=2 %s>%d</td></tr>\n",
               trstring, tdstring, tdstring, doneka);
    printf("<tr %s><th colspan=2 %s>Total transferred:</th>"
           "<td colspan=2 %s>%d bytes</td></tr>\n",
           trstring, tdstring, tdstring, totalread);
    if (posting>0)
        printf("<tr %s><th colspan=2 %s>Total POSTed:</th>"
               "<td colspan=2 %s>%d</td></tr>\n",
               trstring, tdstring, tdstring, totalposted);
    printf("<tr %s><th colspan=2 %s>HTML transferred:</th>"
           "<td colspan=2 %s>%d bytes</td></tr>\n",
           trstring, tdstring, tdstring, totalbread);

    /* avoid divide by zero */
    if (timetaken) {
        printf("<tr %s><th colspan=2 %s>Requests per second:</th>"
               "<td colspan=2 %s>%.2f</td></tr>\n",
           trstring, tdstring, tdstring, 1000 * (float) (done) / timetaken);
        printf("<tr %s><th colspan=2 %s>Transfer rate:</th>"
               "<td colspan=2 %s>%.2f kb/s received</td></tr>\n",
             trstring, tdstring, tdstring, (float) (totalread) / timetaken);
        if (posting>0) {
            printf("<tr %s><td colspan=2 %s> </td>"
                   "<td colspan=2 %s>%.2f kb/s sent</td></tr>\n",
                   trstring, tdstring, tdstring,
                   (float) (totalposted) / timetaken);
            printf("<tr %s><td colspan=2 %s> </td>"
                   "<td colspan=2 %s>%.2f kb/s total</td></tr>\n",
```

```
                trstring, tdstring, tdstring,
                (float) (totalread + totalposted) / timetaken);
        }
    }

    {
        /* work out connection times */
        int i;
        int totalcon = 0, total = 0;
        int mincon = 9999999, mintot = 999999;
        int maxcon = 0, maxtot = 0;

        for (i = 0; i < requests; i++) {
            struct data s = stats[i];
            mincon = ap_min(mincon, s.ctime);
            mintot = ap_min(mintot, s.time);
            maxcon = ap_max(maxcon, s.ctime);
            maxtot = ap_max(maxtot, s.time);
            totalcon += s.ctime;
            total += s.time;
        }

        if (requests > 0) { /* avoid division by zero (if 0 requests) */
            printf("<tr %s><th %s colspan=4>Connnection Times (ms)</th></tr>\n",
                  trstring, tdstring);
            printf("<tr %s><th %s> </th> <th %s>min</th>   <th %s>avg</th>   <th
%s>max</th></tr>\n",
                  trstring, tdstring, tdstring, tdstring, tdstring);
            printf("<tr %s><th %s>Connect:</th>"
                  "<td %s>%5d</td>"
                  "<td %s>%5d</td>"
                  "<td %s>%5d</td></tr>\n",
                  trstring, tdstring, tdstring, mincon, tdstring, totalcon / requests,
tdstring, maxcon);
            printf("<tr %s><th %s>Processing:</th>"
                  "<td %s>%5d</td>"
                  "<td %s>%5d</td>"
                  "<td %s>%5d</td></tr>\n",
                  trstring, tdstring, tdstring, mintot - mincon, tdstring,
                  (total / requests) - (totalcon / requests), tdstring, maxtot -
maxcon);
            printf("<tr %s><th %s>Total:</th>"
                  "<td %s>%5d</td>"
                  "<td %s>%5d</td>"
                  "<td %s>%5d</td></tr>\n",
                  trstring, tdstring, tdstring, mintot, tdstring, total / requests,
tdstring, maxtot);
        }
        printf("</table>\n");
    }
}

/* ------------------------------------------------------- */

/* start asnchronous non-blocking connection */

static void start_connect(struct connection *c)
{
    apr_status_t rv;

    if(!(started < requests)) return;

    c->read = 0;
    c->bread = 0;
    c->keepalive = 0;
    c->cbx = 0;
    c->gotheader = 0;

    if ((rv = apr_create_tcp_socket(&c->aprsock, cntxt)) != APR_SUCCESS) {
        apr_err("Socket:", rv);
    }
    if ((rv = apr_set_remote_port(c->aprsock, port)) != APR_SUCCESS) {
        apr_err("Port:", rv);
    }
    c->start = apr_now();
    if ((rv = apr_connect(c->aprsock, hostname)) != APR_SUCCESS) {
        if (apr_canonical_error(rv) == APR_EINPROGRESS) {
            c->state = STATE_CONNECTING;
            apr_add_poll_socket(readbits, c->aprsock, APR_POLLOUT);
            return;
        }
        else {
            apr_remove_poll_socket(readbits, c->aprsock);
            apr_close_socket(c->aprsock);
            err_conn++;
            if (bad++ > 10) {
                fprintf(stderr,
                        "\nTest aborted after 10 failures\n\n");
                apr_err("apr_connect()", rv);
            }
```

```
            start_connect(c);
            return;
        }
    }

    /* connected first time */
    started++;
    write_request(c);
}

/* ------------------------------------------------------- */

/* close down connection and save stats */

static void close_connection(struct connection *c)
{
    if (c->read == 0 && c->keepalive) {
        /* server has legitimately shut down an idle keep alive request */
        if (good) good--;     /* connection never happened */
    }
    else {
        if (good == 1) {
            /* first time here */
            doclen = c->bread;
        }
        else if (c->bread != doclen) {
            bad ++;
            err_length++;
        }
        /* save out time */
        if (done < requests) {
            struct data s;
            c->done = apr_now();
            s.read  = c->read;
            s.ctime = (c->connect - c->start) / 1000;
            s.time  = (c->done - c->start) / 1000;
            stats[done++] = s;
        }
    }

    apr_remove_poll_socket(readbits, c->aprsock);
    apr_close_socket(c->aprsock);

    /* connect again */
    start_connect(c);
    return;
}

/* ------------------------------------------------------- */

/* read data from connection */

static void read_connection(struct connection *c)
{
    apr_ssize_t r;
    apr_status_t status;
    char *part;
    char respcode[4];         /* 3 digits and null */

    r = sizeof(buffer);
    apr_setsocketopt(c->aprsock, APR_SO_TIMEOUT, aprtimeout);
    status = apr_recv(c->aprsock, buffer, &r);
    if (r == 0 || (status != 0 && apr_canonical_error(status) != APR_EAGAIN)) {
        good++;
        close_connection(c);
        return;
    }

    if (apr_canonical_error(status) == APR_EAGAIN)
        return;

    c->read += r;
    totalread += r;

    if (!c->gotheader) {
        char *s;
        int l = 4;
        int space = CBUFFSIZE - c->cbx - 1;  /* -1 to allow for 0 terminator */
        int tocopy = (space < r) ? space : r;
#ifdef NOT_ASCII
        apr_size_t inbytes_left = space, outbytes_left = space;

        status = apr_xlate_conv_buffer(from_ascii, buffer, &inbytes_left,
                                       c->cbuff + c->cbx, &outbytes_left);
        if (status || inbytes_left || outbytes_left) {
            fprintf(stderr, "only simple translation is supported (%d/%u/%u)\n",
                    status, inbytes_left, outbytes_left);
            exit(1);
        }
#else
```

```
        memcpy(c->cbuff + c->cbx, buffer, space);
#endif /*NOT_ASCII */
        c->cbx += tocopy;
        space -= tocopy;
        c->cbuff[c->cbx] = 0;  /* terminate for benefit of strstr */
        if (verbosity >= 4) {
            printf("LOG: header received:\n%s\n", c->cbuff);
        }
        s = strstr(c->cbuff, "\r\n\r\n");
            /* this next line is so that we talk to NCSA 1.5 which blatantly
             * breaks the http specifaction
             */
        if (!s) {
            s = strstr(c->cbuff, "\n\n");
            l = 2;
        }

        if (!s) {
            /* read rest next time */
            if (space) {
               return;
            }
            else {
                /* header is in invalid or too big - close connection */
                apr_remove_poll_socket(readbits, c->aprsock);
                apr_close_socket(c->aprsock);
                err_response++;
                if (bad++ > 10) {
                    err("\nTest aborted after 10 failures\n\n");
                }
                start_connect(c);
            }
        }
        else {
            /* have full header */
            if (!good) {
               /* this is first time, extract some interesting info */
               char *p, *q;
               p = strstr(c->cbuff, "Server:");
               q = servername;
               if (p) {
                   p += 8;
                   while (*p > 32)
                       *q++ = *p++;
               }
               *q = 0;
            }

            /* XXX: this parsing isn't even remotely HTTP compliant...
             * but in the interest of speed it doesn't totally have to be,
             * it just needs to be extended to handle whatever servers
             * folks want to test against. -djg */

            /* check response code */
            part = strstr(c->cbuff, "HTTP");   /* really HTTP/1.x_ */
            strncpy(respcode, (part + strlen("HTTP/1.x_")), 3);
            respcode[3] = '\0';
            if (respcode[0] != '2') {
               err_response++;
               if (verbosity >= 2)
                   printf("WARNING: Response code not 2xx (%s)\n", respcode);
            }
            else if (verbosity >= 3) {
               printf("LOG: Response code = %s\n", respcode);
            }

            c->gotheader = 1;
            *s = 0;             /* terminate at end of header */
            if (keepalive &&
               (strstr(c->cbuff, "Keep-Alive")
               || strstr(c->cbuff, "keep-alive"))) {  /* for benefit of MSIIS */
               char *cl;
               cl = strstr(c->cbuff, "Content-Length:");
               /* handle NCSA, which sends Content-length: */
               if (!cl)
                   cl = strstr(c->cbuff, "Content-length:");
               if (cl) {
                   c->keepalive = 1;
                   c->length = atoi(cl + 16);
               }
            }
            c->bread += c->cbx - (s + l - c->cbuff) + r - tocopy;
            totalbread += c->bread;
        }
    }
    else {
        /* outside header, everything we have read is entity body */
        c->bread += r;
        totalbread += r;
    }
```

```
    if (c->keepalive && (c->bread >= c->length)) {
        /* finished a keep-alive connection */
        good++;
        doneka++;
        /* save out time */
        if (good == 1) {
            /* first time here */
            doclen = c->bread;
        }
        else if (c->bread != doclen) {
            bad++;
            err_length++;
        }
        if (done < requests) {
            struct data s;
           c->done = apr_now();
            s.read = c->read;
           s.ctime = (c->connect - c->start) / 1000;
           s.time = (c->done - c->start) / 1000;
            stats[done++] = s;
        }
        c->keepalive = 0;
        c->length = 0;
        c->gotheader = 0;
        c->cbx = 0;
        c->read = c->bread = 0;
        write_request(c);
        c->start = c->connect; /* zero connect time with keep-alive */
    }
}

/* ------------------------------------------------------- */

/* run the tests */

static void test(void)
{
    apr_time_t now;
    apr_interval_time_t timeout;
    apr_int16_t rv;
    int i;
    apr_status_t status;
#ifdef NOT_ASCII
    apr_size_t inbytes_left, outbytes_left;
#endif

    if (!use_html) {
        printf("Benchmarking %s (be patient)...", hostname);
        fflush(stdout);
    }

    now = apr_now();

    con = malloc(concurrency * sizeof(struct connection));
    memset(con, 0, concurrency * sizeof(struct connection));

    stats = malloc(requests * sizeof(struct data));
    apr_setup_poll(&readbits, concurrency, cntxt);

    /* setup request */
    if (!posting) {
        sprintf(request, "%s %s HTTP/1.0\r\n"
                "User-Agent: ApacheBench/%s\r\n"
                "%s" "%s" "%s"
                "Host: %s\r\n"
                "Accept: */*\r\n"
                "%s" "\r\n",
                (posting == 0) ? "GET" : "HEAD",
                path,
                AB_VERSION,
                keepalive ? "Connection: Keep-Alive\r\n" : "",
                cookie, auth, hostname, hdrs);
    }
    else {
        sprintf(request, "POST %s HTTP/1.0\r\n"
                "User-Agent: ApacheBench/%s\r\n"
                "%s" "%s" "%s"
                "Host: %s\r\n"
                "Accept: */*\r\n"
                "Content-length: %d\r\n"
                "Content-type: %s\r\n"
                "%s"
                "\r\n",
                path,
                AB_VERSION,
                keepalive ? "Connection: Keep-Alive\r\n" : "",
                cookie, auth,
                hostname, postlen,
                (content_type[0]) ? content_type : "text/plain", hdrs);
```

```
    }

    if (verbosity >= 2)
        printf("INFO: POST header == \n---\n%s\n---\n", request);

    reqlen = strlen(request);

#ifdef NOT_ASCII
    inbytes_left = outbytes_left = reqlen;
    status = apr_xlate_conv_buffer(to_ascii, request, &inbytes_left,
                                   request, &outbytes_left);
    if (status || inbytes_left || outbytes_left) {
        fprintf(stderr, "only simple translation is supported (%d/%u/%u)\n",
                status, inbytes_left, outbytes_left);
        exit(1);
    }
#endif /*NOT_ASCII*/

    /* ok - lets start */
    start = apr_now();

    /* initialise lots of requests */
    for (i = 0; i < concurrency; i++) {
        con[i].socknum = i;
        start_connect(&con[i]);
    }

    while (done < requests) {
        apr_int32_t n;
        apr_int32_t timed;

        /* check for time limit expiry */
        now = apr_now();
        timed = (now - start) / APR_USEC_PER_SEC;
        if (tlimit && timed > (tlimit * 1000)) {
            requests = done;   /* so stats are correct */
        }
        /* Timeout of 30 seconds. */
        timeout = 30 * APR_USEC_PER_SEC;

        n = concurrency;
        status = apr_poll(readbits, &n, timeout);
        if (status != APR_SUCCESS)
            apr_err("apr_poll", status);

        if (!n) {
            err("\nServer timed out\n\n");
        }

        for (i = 0; i < concurrency; i++) {
            apr_get_revents(&rv, con[i].aprsock, readbits);

            /* Note: APR_POLLHUP is set after FIN is received on some
             * systems, so treat that like APR_POLLIN so that we try
             * to read again.
             */
            if ((rv & APR_POLLERR) || (rv & APR_POLLNVAL)) {
               bad++;
               err_except++;
               start_connect(&con[i]);
               continue;
            }
            if ((rv & APR_POLLIN) || (rv & APR_POLLPRI) || (rv & APR_POLLHUP))
               read_connection(&con[i]);
            if (rv & APR_POLLOUT)
               write_request(&con[i]);
        }
    }
    if (use_html)
        output_html_results();
    else
        output_results();
}

/* ------------------------------------------------------- */

/* display copyright information */
static void copyright(void)
{
    if (!use_html) {
        printf("This is ApacheBench, Version %s\n", AB_VERSION " <$Revision: 1.26 $>
apache-2.0");
        printf("Copyright (c) 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/\n");
        printf("Copyright (c) 1998-2000 The Apache Software Foundation,
http://www.apache.org/\n");
        printf("\n");
    }
    else {
        printf("<p>\n");
```

```
        printf(" This is ApacheBench, Version %s <i>&lt;%s&gt;</i> apache-2.0<br>\n",
AB_VERSION, "$Revision: 1.26 $");
        printf(" Copyright (c) 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/<br>\n");
        printf(" Copyright (c) 1998-2000 The Apache Software Foundation,
http://www.apache.org/<br>\n");
        printf("</p>\n<p>\n");
    }
}

/* display usage information */
static void usage(char *progname)
{
    fprintf(stderr, "Usage: %s [options] [http://]hostname[:port]/path\n", progname);
    fprintf(stderr, "Options are:\n");
    fprintf(stderr, "    -n requests     Number of requests to perform\n");
    fprintf(stderr, "    -c concurrency  Number of multiple requests to make\n");
    fprintf(stderr, "    -t timelimit    Seconds to max. wait for responses\n");
    fprintf(stderr, "    -p postfile     File containg data to POST\n");
    fprintf(stderr, "    -T content-type Content-type header for POSTing\n");
    fprintf(stderr, "    -v verbosity    How much troubleshooting info to print\n");
    fprintf(stderr, "    -w              Print out results in HTML tables\n");
    fprintf(stderr, "    -i              Use HEAD instead of GET\n");
    fprintf(stderr, "    -x attributes   String to insert as table attributes\n");
    fprintf(stderr, "    -y attributes   String to insert as tr attributes\n");
    fprintf(stderr, "    -z attributes   String to insert as td or th attributes\n");
    fprintf(stderr, "    -C attribute    Add cookie, eg. 'Apache=1234.
(repeatable)\n");
    fprintf(stderr, "    -H attribute    Add Arbitrary header line, eg. 'Accept-
Encoding: zop'\n");
    fprintf(stderr, "                    Inserted after all normal header lines.
(repeatable)\n");
    fprintf(stderr, "    -A attribute    Add Basic WWW Authentication, the
attributes\n");
    fprintf(stderr, "                    are a colon separated username and
password.\n");
    fprintf(stderr, "    -p attribute    Add Basic Proxy Authentication, the
attributes\n");
    fprintf(stderr, "                    are a colon separated username and
password.\n");
    fprintf(stderr, "    -V              Print version number and exit\n");
    fprintf(stderr, "    -k              Use HTTP KeepAlive feature\n");
    fprintf(stderr, "    -h              Display usage information (this message)\n");
    exit(EINVAL);
}

/* ------------------------------------------------------- */

/* split URL into parts */

static int parse_url(char *url)
{
    char *cp;
    char *h;
    char *p = NULL; /* points to port if url has it */

    if (strlen(url) > 7 && strncmp(url, "http://", 7) == 0)
        url += 7;
    h = url;
    if ((cp = strchr(url, ':')) != NULL) {
        *cp++ = '\0';
        p = cp;
        url = cp;
    }
    if ((cp = strchr(url, '/')) == NULL)
        return 1;
    strcpy(path, cp);
    *cp = '\0';
    strcpy(hostname, h);
    if (p != NULL)
        port = atoi(p);
    return 0;
}

/* ------------------------------------------------------- */

/* read data to POST from file, save contents and length */

static int open_postfile(const char *pfile)
{
    apr_file_t *postfd = NULL;
    apr_finfo_t finfo;
    apr_fileperms_t mode = APR_OS_DEFAULT;
    apr_ssize_t length;

    if (apr_open(&postfd, pfile, APR_READ, mode, cntxt) != APR_SUCCESS) {
        printf("Invalid postfile name (%s)\n", pfile);
        return errno;
    }
```

```
    apr_getfileinfo(&finfo, postfd);
    postlen = finfo.size;
    postdata = (char *)malloc(postlen);
    if (!postdata) {
        printf("Can\'t alloc postfile buffer\n");
        return ENOMEM;
    }
    length = postlen;
    if (apr_read(postfd, postdata, &length) != APR_SUCCESS &&
        length != postlen) {
        printf("error reading postfilen");
        return EIO;
    }
    return 0;
}

/* ------------------------------------------------------- */

/* sort out command-line args and call test */
int main(int argc, char **argv)
{
    int r, l;
    char tmp[1024];
    apr_status_t status;
    apr_getopt_t *opt;
    const char *optarg;
    char c;

    /* table defaults  */
    tablestring = "";
    trstring = "";
    tdstring = "bgcolor=white";
    cookie[0] = '\0';
    auth[0] = '\0';
    hdrs[0] = '\0';

    apr_initialize();
    atexit(apr_terminate);
    apr_create_pool(&cntxt, NULL);

#ifdef NOT_ASCII
    status = apr_xlate_open(&to_ascii, "ISO8859-1", APR_DEFAULT_CHARSET, cntxt);
    if (status) {
        fprintf(stderr, "apr_xlate_open(to ASCII)->%d\n", status);
        exit(1);
    }
    status = apr_xlate_open(&from_ascii, APR_DEFAULT_CHARSET, "ISO8859-1", cntxt);
    if (status) {
        fprintf(stderr, "apr_xlate_open(from ASCII)->%d\n", status);
        exit(1);
    }
    status = ap_base64init_ebcdic(to_ascii, from_ascii);
    if (status) {
        fprintf(stderr, "ap_base64init_ebcdic()->%d\n", status);
        exit(1);
    }
#endif

    apr_initopt(&opt, cntxt, argc, argv);
    while ((status = apr_getopt(opt, "n:c:t:T:p:v:kVhwix:y:z:C:H:P:A:", &c, &optarg))
== APR_SUCCESS) {
        switch (c) {
        case 'n':
            requests = atoi(optarg);
            if (!requests) {
               err("Invalid number of requests\n");
            }
            break;
        case 'k':
            keepalive = 1;
            break;
        case 'c':
            concurrency = atoi(optarg);
            break;
        case 'i':
            if (posting == 1)
                err("Cannot mix POST and HEAD\n");
            posting = -1;
            break;
        case 'p':
            if (posting != 0)
                err("Cannot mix POST and HEAD\n");

            if (0 == (r = open_postfile(optarg))) {
               posting = 1;
            }
            else if (postdata) {
               exit(r);
            }
            break;
```

```
        case 'v':
            verbosity = atoi(optarg);
            break;
        case 't':
            tlimit = atoi(optarg);
            requests = MAX_REQUESTS;  /* need to size data array on something */
            break;
        case 'T':
            strcpy(content_type, optarg);
            break;
        case 'C':
            strncat(cookie, "Cookie: ", sizeof(cookie));
            strncat(cookie, optarg, sizeof(cookie));
            strncat(cookie, "\r\n", sizeof(cookie));
            break;
        case 'A':
            /* assume username passwd already to be in colon separated form.
             * Ready to be uu-encoded.
             */
            while(isspace(*optarg))
                optarg++;
            l=ap_base64encode(tmp, optarg, strlen(optarg));
            tmp[l]='\0';

            strncat(auth, "Authorization: basic ", sizeof(auth));
            strncat(auth, tmp, sizeof(auth));
            strncat(auth, "\r\n", sizeof(auth));
            break;
        case 'P':
            /*
             * assume username passwd already to be in colon separated form.
             */
            while(isspace(*optarg))
                optarg++;
            l=ap_base64encode(tmp, optarg, strlen(optarg));
            tmp[l]='\0';

            strncat(auth, "Proxy-Authorization: basic ", sizeof(auth));
            strncat(auth, tmp, sizeof(auth));
            strncat(auth, "\r\n", sizeof(auth));
            break;
        case 'H':
            strncat(hdrs, optarg, sizeof(hdrs));
            strncat(hdrs, "\r\n", sizeof(hdrs));
            break;
        case 'w':
            use_html = 1;
            break;
            /*
             * if any of the following three are used, turn on html output
             * automatically
             */
        case 'x':
            use_html = 1;
            tablestring = optarg;
            break;
        case 'y':
            use_html = 1;
            trstring = optarg;
            break;
        case 'z':
            use_html = 1;
            tdstring = optarg;
            break;
        case 'h':
            usage(argv[0]);
            break;
        case 'V':
            copyright();
            return 0;
        }
    }

    if (opt->ind != argc - 1) {
        fprintf(stderr, "%s: wrong number of arguments\n", argv[0]);
        usage(argv[0]);
    }

    if (parse_url((char*)opt->argv[opt->ind++])) {
        fprintf(stderr, "%s: invalid URL\n", argv[0]);
        usage(argv[0]);
    }

    copyright();
    test();

    return 0;
}
```

ApacheBench without APR

```
/* ====================================================================
 * Copyright (c) 1998-1999 The Apache Group.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the Apache Group
 *    for use in the Apache HTTP server project (http://www.apache.org/)."
 *
 * 4. The names "Apache Server" and "Apache Group" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache"
 *    nor may "Apache" appear in their names without prior written
 *    permission of the Apache Group.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the Apache Group
 *    for use in the Apache HTTP server project (http://www.apache.org/)."
 *
 * THIS SOFTWARE IS PROVIDED BY THE APACHE GROUP ``AS IS'' AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE APACHE GROUP OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
 *
 * This software consists of voluntary contributions made by many
 * individuals on behalf of the Apache Group and was originally based
 * on public domain software written at the National Center for
 * Supercomputing Applications, University of Illinois, Urbana-Champaign.
 * For more information on the Apache Group and the Apache HTTP server
 * project, please see <http://www.apache.org/>.
 *
 */

/*
   ** This program is based on ZeusBench V1.0 written by Adam Twiss
   ** which is Copyright (c) 1996 by Zeus Technology Ltd. http://www.zeustech.net/
   **
   ** This software is provided "as is" and any express or implied waranties,
   ** including but not limited to, the implied warranties of merchantability and
   ** fitness for a particular purpose are disclaimed.  In no event shall
   ** Zeus Technology Ltd. be liable for any direct, indirect, incidental, special,
   ** exemplary, or consequential damaged (including, but not limited to,
   ** procurement of substitute good or services; loss of use, data, or profits;
   ** or business interruption) however caused and on theory of liability.  Whether
   ** in contract, strict liability or tort (including negligence or otherwise)
   ** arising in any way out of the use of this software, even if advised of the
   ** possibility of such damage.
   **
 */

/*
   ** HISTORY:
   **    - Originally written by Adam Twiss <adam@zeus.co.uk>, March 1996
   **      with input from Mike Belshe <mbelshe@netscape.com> and
   **      Michael Campanella <campanella@stevms.enet.dec.com>
   **    - Enhanced by Dean Gaudet <dgaudet@apache.org>, November 1997
   **    - Cleaned up by Ralf S. Engelschall <rse@apache.org>, March 1998
   **    - POST and verbosity by Kurt Sussman <kls@merlot.com>, August 1998
   **    - HTML table output added by David N. Welton <davidw@prosa.it>, January 1999
   **    - Added Cookie, Arbitrary header and auth support. <dirkx@webweaving.org>,
April 1999
   **
 */
```

```
/*
 * BUGS:
 *
 * - uses strcpy/etc.
 * - has various other poor buffer attacks related to the lazy parsing of
 *   response headers from the server
 * - doesn't implement much of HTTP/1.x, only accepts certain forms of
 *   responses
 * - (performance problem) heavy use of strstr shows up top in profile
 *   only an issue for loopback usage
 */

#define VERSION "1.3c"

/*  ------------------------------------------------------------------ */

/* affects include files on Solaris */
#define BSD_COMP

/* allow compilation outside an Apache build tree */
#ifdef NO_APACHE_INCLUDES
#include <sys/time.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <errno.h>
#include <sys/ioctl.h>
#include <string.h>
#include <sys/types.h>
#include <sys/uio.h>

#define ap_select       select
#else                           /* (!)NO_APACHE_INCLUDES */
#include "ap_config.h"
#include "ap.h"
#ifdef CHARSET_EBCDIC
#include "ebcdic.h"
#endif
#include <fcntl.h>
#ifndef MPE
#include <sys/time.h>
#endif

#ifndef NO_WRITEV
#include <sys/types.h>
#include <sys/uio.h>
#endif

#endif                          /* NO_APACHE_INCLUDES */
/* ------------------- DEFINITIONS -------------------------- */

/* maximum number of requests on a time limited test */
#define MAX_REQUESTS 50000

/* good old state hostname */
#define STATE_UNCONNECTED 0
#define STATE_CONNECTING  1
#define STATE_READ        2

#define CBUFFSIZE       512

struct connection {
    int fd;
    int state;
    int read;                   /* amount of bytes read */
    int bread;                  /* amount of body read */
    int length;                       /* Content-Length value used for keep-alive */
    char cbuff[CBUFFSIZE];      /* a buffer to store server response header */
    int cbx;                    /* offset in cbuffer */
    int keepalive;              /* non-zero if a keep-alive request */
    int gotheader;              /* non-zero if we have the entire header in
                                 * cbuff */
    struct timeval start, connect, done;
};

struct data {
    int read;                   /* number of bytes read */
    int ctime;                  /* time in ms to connect */
    int time;                   /* time in ms for connection */
};

#define ap_min(a,b) ((a)<(b))?(a):(b)
#define ap_max(a,b) ((a)>(b))?(a):(b)
```

```
/* --------------------- GLOBALS --------------------------- */

int verbosity = 0;            /* no verbosity by default */
int posting = 0;              /* GET by default */
int requests = 1;             /* Number of requests to make */
int concurrency = 1;          /* Number of multiple requests to make */
int tlimit = 0;                    /* time limit in cs */
int keepalive = 0;            /* try and do keepalive connections */
char servername[1024];        /* name that server reports */
char hostname[1024];          /* host name */
char path[1024];              /* path name */
char postfile[1024];          /* name of file containing post data */
char *postdata;                    /* *buffer containing data from postfile */
int postlen = 0;              /* length of data to be POSTed */
char content_type[1024];      /* content type to put in POST header */
char cookie[1024],            /* optional cookie line */
     auth[1024],              /* optional (basic/uuencoded)
                               * authentification */
     hdrs[4096];              /* optional arbitrary headers */
int port = 80;                /* port number */

int use_html = 0;             /* use html in the report */
char *tablestring;
char *trstring;
char *tdstring;

int doclen = 0;                    /* the length the document should be */
int totalread = 0;            /* total number of bytes read */
int totalbread = 0;           /* totoal amount of entity body read */
int totalposted = 0;          /* total number of bytes posted, inc. headers */
int done = 0;                 /* number of requests we have done */
int doneka = 0;                    /* number of keep alive connections done */
int good = 0, bad = 0;        /* number of good and bad requests */

/* store error cases */
int err_length = 0, err_conn = 0, err_except = 0;
int err_response = 0;

struct timeval start, endtime;

/* global request (and its length) */
char request[512];
int reqlen;

/* one global throw-away buffer to read stuff into */
char buffer[8192];

struct connection *con;            /* connection array */
struct data *stats;           /* date for each request */

fd_set readbits, writebits;   /* bits for select */
struct sockaddr_in server;    /* server addr structure */

#ifndef BEOS
#define ab_close(s) close(s)
#define ab_read(a,b,c) read(a,b,c)
#define ab_write(a,b,c) write(a,b,c)
#else
#define ab_close(s) closesocket(s)
#define ab_read(a,b,c) recv(a,b,c,0)
#define ab_write(a,b,c) send(a,b,c,0)
#endif

/* ------------------------------------------------------- */

/* simple little function to perror and exit */

static void err(char *s)
{
    if (errno) {
        perror(s);
    }
    else {
        printf("%s", s);
    }
    exit(errno);
}

/* ------------------------------------------------------- */

/* write out request to a connection - assumes we can write
   (small) request out in one go into our new socket buffer  */

static void write_request(struct connection * c)
{
#ifndef NO_WRITEV
    struct iovec out[2]; int outcnt = 1;
#endif
    gettimeofday(&c->connect, 0);
```

```
#ifndef NO_WRITEV
    out[0].iov_base = request;
    out[0].iov_len = reqlen;

    if (posting>0) {
        out[1].iov_base = postdata;
        out[1].iov_len = postlen;
        outcnt = 2;
        totalposted += (reqlen + postlen);
    }
    writev(c->fd,out, outcnt);
#else
    ab_write(c->fd,request,reqlen);
    if (posting>0) {
        ab_write(c->fd,postdata,postlen);
        totalposted += (reqlen + postlen);
    }
#endif

    c->state = STATE_READ;
    FD_SET(c->fd, &readbits);
    FD_CLR(c->fd, &writebits);
}

/* ------------------------------------------------------- */

/* make an fd non blocking */

static void nonblock(int fd)
{
    int i = 1;
#ifdef BEOS
    setsockopt(fd, SOL_SOCKET, SO_NONBLOCK, &i, sizeof(i));
#else
    ioctl(fd, FIONBIO, &i);
#endif
}

/* ------------------------------------------------------- */

/* returns the time in ms between two timevals */

static int timedif(struct timeval a, struct timeval b)
{
    register int us, s;

    us = a.tv_usec - b.tv_usec;
    us /= 1000;
    s = a.tv_sec - b.tv_sec;
    s *= 1000;
    return s + us;
}

/* ------------------------------------------------------- */

/* calculate and output results */

static void output_results(void)
{
    int timetaken;

    gettimeofday(&endtime, 0);
    timetaken = timedif(endtime, start);

    printf("\r
\r");
    printf("Server Software:        %s\n", servername);
    printf("Server Hostname:        %s\n", hostname);
    printf("Server Port:            %d\n", port);
    printf("\n");
    printf("Document Path:          %s\n", path);
    printf("Document Length:        %d bytes\n", doclen);
    printf("\n");
    printf("Concurrency Level:      %d\n", concurrency);
    printf("Time taken for tests:   %d.%03d seconds\n",
           timetaken / 1000, timetaken % 1000);
    printf("Complete requests:      %d\n", done);
    printf("Failed requests:        %d\n", bad);
    if (bad)
        printf("   (Connect: %d, Length: %d, Exceptions: %d)\n",
               err_conn, err_length, err_except);
    if (err_response)
        printf("Non-2xx responses:      %d\n", err_response);
    if (keepalive)
        printf("Keep-Alive requests:    %d\n", doneka);
    printf("Total transferred:      %d bytes\n", totalread);
    if (posting>0)
        printf("Total POSTed:           %d\n", totalposted);
    printf("HTML transferred:       %d bytes\n", totalbread);
```

```
    /* avoid divide by zero */
    if (timetaken) {
        printf("Requests per second:    %.2f\n", 1000 * (float) (done) / timetaken);
        printf("Transfer rate:          %.2f kb/s received\n",
               (float) (totalread) / timetaken);
        if (posting>0) {
            printf("                        %.2f kb/s sent\n",
                   (float) (totalposted) / timetaken);
            printf("                        %.2f kb/s total\n",
                   (float) (totalread + totalposted) / timetaken);
        }
    }

    {
        /* work out connection times */
        int i;
        int totalcon = 0, total = 0;
        int mincon = 9999999, mintot = 999999;
        int maxcon = 0, maxtot = 0;

        for (i = 0; i < requests; i++) {
            struct data s = stats[i];
            mincon = ap_min(mincon, s.ctime);
            mintot = ap_min(mintot, s.time);
            maxcon = ap_max(maxcon, s.ctime);
            maxtot = ap_max(maxtot, s.time);
            totalcon += s.ctime;
            total += s.time;
        }
        if (requests > 0) { /* avoid division by zero (if 0 requests) */
            printf("\nConnnection Times (ms)\n");
            printf("              min   avg   max\n");
            printf("Connect:    %5d %5d %5d\n", mincon, totalcon / requests, maxcon);
            printf("Processing: %5d %5d %5d\n",
                   mintot - mincon, (total / requests) - (totalcon / requests),
                   maxtot - maxcon);
            printf("Total:      %5d %5d %5d\n", mintot, total / requests, maxtot);
        }
    }
}

/* ------------------------------------------------------- */

/* calculate and output results in HTML  */

static void output_html_results(void)
{
    int timetaken;

    gettimeofday(&endtime, 0);
    timetaken = timedif(endtime, start);

    printf("\n\n<table %s>\n", tablestring);
    printf("<tr %s><th colspan=2 %s>Server Software:</th>"
           "<td colspan=2 %s>%s</td></tr>\n",
           trstring, tdstring, tdstring, servername);
    printf("<tr %s><th colspan=2 %s>Server Hostname:</th>"
           "<td colspan=2 %s>%s</td></tr>\n",
           trstring, tdstring, tdstring, hostname);
    printf("<tr %s><th colspan=2 %s>Server Port:</th>"
           "<td colspan=2 %s>%d</td></tr>\n",
           trstring, tdstring, tdstring, port);
    printf("<tr %s><th colspan=2 %s>Document Path:</th>"
           "<td colspan=2 %s>%s</td></tr>\n",
           trstring, tdstring, tdstring, path);
    printf("<tr %s><th colspan=2 %s>Document Length:</th>"
           "<td colspan=2 %s>%d bytes</td></tr>\n",
           trstring, tdstring, tdstring, doclen);
    printf("<tr %s><th colspan=2 %s>Concurrency Level:</th>"
           "<td colspan=2 %s>%d</td></tr>\n",
           trstring, tdstring, tdstring, concurrency);
    printf("<tr %s><th colspan=2 %s>Time taken for tests:</th>"
           "<td colspan=2 %s>%d.%03d seconds</td></tr>\n",
           trstring, tdstring, tdstring, timetaken / 1000, timetaken % 1000);
    printf("<tr %s><th colspan=2 %s>Complete requests:</th>"
           "<td colspan=2 %s>%d</td></tr>\n",
           trstring, tdstring, tdstring, done);
    printf("<tr %s><th colspan=2 %s>Failed requests:</th>"
           "<td colspan=2 %s>%d</td></tr>\n",
           trstring, tdstring, tdstring, bad);
    if (bad)
        printf("<tr %s><td colspan=4 %s >   (Connect: %d, Length: %d, Exceptions:
%d)</td></tr>\n",
               trstring, tdstring, err_conn, err_length, err_except);
    if (err_response)
        printf("<tr %s><th colspan=2 %s>Non-2xx responses:</th>"
               "<td colspan=2 %s>%d</td></tr>\n",
               trstring, tdstring, tdstring, err_response);
    if (keepalive)
        printf("<tr %s><th colspan=2 %s>Keep-Alive requests:</th>"
```

```
                "<td colspan=2 %s>%d</td></tr>\n",
                trstring, tdstring, tdstring, doneka);
    printf("<tr %s><th colspan=2 %s>Total transferred:</th>"
           "<td colspan=2 %s>%d bytes</td></tr>\n",
           trstring, tdstring, tdstring, totalread);
    if (posting>0)
        printf("<tr %s><th colspan=2 %s>Total POSTed:</th>"
                "<td colspan=2 %s>%d</td></tr>\n",
                trstring, tdstring, tdstring, totalposted);
    printf("<tr %s><th colspan=2 %s>HTML transferred:</th>"
           "<td colspan=2 %s>%d bytes</td></tr>\n",
           trstring, tdstring, tdstring, totalbread);

    /* avoid divide by zero */
    if (timetaken) {
        printf("<tr %s><th colspan=2 %s>Requests per second:</th>"
                "<td colspan=2 %s>%.2f</td></tr>\n",
               trstring, tdstring, tdstring, 1000 * (float) (done) / timetaken);
        printf("<tr %s><th colspan=2 %s>Transfer rate:</th>"
                "<td colspan=2 %s>%.2f kb/s received</td></tr>\n",
             trstring, tdstring, tdstring, (float) (totalread) / timetaken);
        if (posting>0) {
            printf("<tr %s><td colspan=2 %s> </td>"
                   "<td colspan=2 %s>%.2f kb/s sent</td></tr>\n",
                   trstring, tdstring, tdstring,
                   (float) (totalposted) / timetaken);
            printf("<tr %s><td colspan=2 %s> </td>"
                   "<td colspan=2 %s>%.2f kb/s total</td></tr>\n",
                   trstring, tdstring, tdstring,
                   (float) (totalread + totalposted) / timetaken);
        }
    }

    {
        /* work out connection times */
        int i;
        int totalcon = 0, total = 0;
        int mincon = 9999999, mintot = 999999;
        int maxcon = 0, maxtot = 0;

        for (i = 0; i < requests; i++) {
            struct data s = stats[i];
            mincon = ap_min(mincon, s.ctime);
            mintot = ap_min(mintot, s.time);
            maxcon = ap_max(maxcon, s.ctime);
            maxtot = ap_max(maxtot, s.time);
            totalcon += s.ctime;
            total += s.time;
        }

        if (requests > 0) { /* avoid division by zero (if 0 requests) */
            printf("<tr %s><th %s colspan=4>Connnection Times (ms)</th></tr>\n",
                   trstring, tdstring);
            printf("<tr %s><th %s> </th> <th %s>min</th>   <th %s>avg</th>   <th
%s>max</th></tr>\n",
                   trstring, tdstring, tdstring, tdstring, tdstring);
            printf("<tr %s><th %s>Connect:</th>"
                   "<td %s>%5d</td>"
                   "<td %s>%5d</td>"
                   "<td %s>%5d</td></tr>\n",
                   trstring, tdstring, tdstring, mincon, tdstring, totalcon / requests,
tdstring, maxcon);
            printf("<tr %s><th %s>Processing:</th>"
                   "<td %s>%5d</td>"
                   "<td %s>%5d</td>"
                   "<td %s>%5d</td></tr>\n",
                   trstring, tdstring, tdstring, mintot - mincon, tdstring,
                   (total / requests) - (totalcon / requests), tdstring, maxtot -
maxcon);
            printf("<tr %s><th %s>Total:</th>"
                   "<td %s>%5d</td>"
                   "<td %s>%5d</td>"
                   "<td %s>%5d</td></tr>\n",
                   trstring, tdstring, tdstring, mintot, tdstring, total / requests,
tdstring, maxtot);
        }
        printf("</table>\n");
    }
}

/* ------------------------------------------------------- */

/* start asnchronous non-blocking connection */

static void start_connect(struct connection * c)
{
    c->read = 0;
    c->bread = 0;
    c->keepalive = 0;
    c->cbx = 0;
```

```
    c->gotheader = 0;

    c->fd = socket(AF_INET, SOCK_STREAM, 0);
    if (c->fd < 0)
        err("socket");

    nonblock(c->fd);
    gettimeofday(&c->start, 0);

    if (connect(c->fd, (struct sockaddr *) & server, sizeof(server)) < 0) {
        if (errno == EINPROGRESS) {
            c->state = STATE_CONNECTING;
            FD_SET(c->fd, &writebits);
            return;
        }
        else {
            ab_close(c->fd);
            err_conn++;
            if (bad++ > 10) {
                err("\nTest aborted after 10 failures\n\n");
            }
            start_connect(c);
        }
    }

    /* connected first time */
    c->state = STATE_CONNECTING;
    FD_SET(c->fd, &writebits);
}

/* ------------------------------------------------------- */

/* close down connection and save stats */

static void close_connection(struct connection * c)
{
    if (c->read == 0 && c->keepalive) {
        /* server has legitimately shut down an idle keep alive request */
        good--;               /* connection never happend */
    }
    else {
        if (good == 1) {
            /* first time here */
            doclen = c->bread;
        }
        else if (c->bread != doclen) {
            bad++;
            err_length++;
        }

        /* save out time */
        if (done < requests) {
            struct data s;
            gettimeofday(&c->done, 0);
            s.read = c->read;
            s.ctime = timedif(c->connect, c->start);
            s.time = timedif(c->done, c->start);
            stats[done++] = s;
        }
    }

    ab_close(c->fd);
    FD_CLR(c->fd, &readbits);
    FD_CLR(c->fd, &writebits);

    /* connect again */
    start_connect(c);
    return;
}

/* ------------------------------------------------------- */

/* read data from connection */

static void read_connection(struct connection * c)
{
    int r;
    char *part;
    char respcode[4];         /* 3 digits and null */

    r = ab_read(c->fd, buffer, sizeof(buffer));

    if (r == 0 || (r < 0 && errno != EAGAIN)) {
        good++;
        close_connection(c);
        return;
    }

    if (r < 0 && errno == EAGAIN)
        return;
```

```
    c->read += r;
    totalread += r;

    if (!c->gotheader) {
        char *s;
        int l = 4;
        int space = CBUFFSIZE - c->cbx - 1;   /* -1 to allow for 0
                                               * terminator */
        int tocopy = (space < r) ? space : r;
#ifndef CHARSET_EBCDIC
        memcpy(c->cbuff + c->cbx, buffer, tocopy);
#else                           /* CHARSET_EBCDIC */
        ascii2ebcdic(c->cbuff + c->cbx, buffer, tocopy);
#endif                          /* CHARSET_EBCDIC */
        c->cbx += tocopy;
        space -= tocopy;
        c->cbuff[c->cbx] = 0;  /* terminate for benefit of strstr */
        if (verbosity >= 4) {
            printf("LOG: header received:\n%s\n", c->cbuff);
        }
        s = strstr(c->cbuff, "\r\n\r\n");
        /*
         * this next line is so that we talk to NCSA 1.5 which blatantly
         * breaks the http specification
         */
        if (!s) {
            s = strstr(c->cbuff, "\n\n");
            l = 2;
        }

        if (!s) {
            /* read rest next time */
            if (space)
                return;
            else {
                /* header is in invalid or too big - close connection */
                ab_close(c->fd);
                if (bad++ > 10) {
                    err("\nTest aborted after 10 failures\n\n");
                }
                FD_CLR(c->fd, &writebits);
                start_connect(c);
            }
        }
        else {
            /* have full header */
            if (!good) {
                /* this is first time, extract some interesting info */
                char *p, *q;
                p = strstr(c->cbuff, "Server:");
                q = servername;
                if (p) {
                    p += 8;
                    while (*p > 32)
                        *q++ = *p++;
                }
                *q = 0;
            }

            /*
             * XXX: this parsing isn't even remotely HTTP compliant... but in
             * the interest of speed it doesn't totally have to be, it just
             * needs to be extended to handle whatever servers folks want to
             * test against. -djg
             */

            /* check response code */
            part = strstr(c->cbuff, "HTTP");  /* really HTTP/1.x_ */
            strncpy(respcode, (part + strlen("HTTP/1.x_")), 3);
            respcode[3] = '\0';
            if (respcode[0] != '2') {
                err_response++;
                if (verbosity >= 2)
                    printf("WARNING: Response code not 2xx (%s)\n", respcode);
            }
            else if (verbosity >= 3) {
                printf("LOG: Response code = %s\n", respcode);
            }

            c->gotheader = 1;
            *s = 0;             /* terminate at end of header */
            if (keepalive &&
                (strstr(c->cbuff, "Keep-Alive")
                 || strstr(c->cbuff, "keep-alive"))) {       /* for benefit of MSIIS */
                char *cl;
                cl = strstr(c->cbuff, "Content-Length:");
                /* handle NCSA, which sends Content-length: */
                if (!cl)
                    cl = strstr(c->cbuff, "Content-length:");
```

```
                if (cl) {
                    c->keepalive = 1;
                    c->length = atoi(cl + 16);
                }
            }
            c->bread += c->cbx - (s + l - c->cbuff) + r - tocopy;
            totalbread += c->bread;
        }
    }
    else {
        /* outside header, everything we have read is entity body */
        c->bread += r;
        totalbread += r;
    }

    /* cater for the case where we're using keepalives and doing HEAD requests */
    if (c->keepalive && ((c->bread >= c->length) || (posting < 0))) {
        /* finished a keep-alive connection */
        good++;
        doneka++;
        /* save out time */
        if (good == 1) {
            /* first time here */
            doclen = c->bread;
        }
        else if (c->bread != doclen) {
            bad++;
            err_length++;
        }
        if (done < requests) {
            struct data s;
            gettimeofday(&c->done, 0);
            s.read = c->read;
            s.ctime = timedif(c->connect, c->start);
            s.time = timedif(c->done, c->start);
            stats[done++] = s;
        }
        c->keepalive = 0;
        c->length = 0;
        c->gotheader = 0;
        c->cbx = 0;
        c->read = c->bread = 0;
        write_request(c);
        c->start = c->connect; /* zero connect time with keep-alive */
    }
}

/* ------------------------------------------------------- */

/* run the tests */

static void test(void)
{
    struct timeval timeout, now;
    fd_set sel_read, sel_except, sel_write;
    int i;

    if (!use_html) {
        printf("Benchmarking %s (be patient)...", hostname);
        fflush(stdout);
    }

    {
        /* get server information */
        struct hostent *he;
        he = gethostbyname(hostname);
        if (!he)
            err("bad hostname");
        server.sin_family = he->h_addrtype;
        server.sin_port = htons(port);
        server.sin_addr.s_addr = ((unsigned long *) (he->h_addr_list[0]))[0];
    }

    con = malloc(concurrency * sizeof(struct connection));
    memset(con, 0, concurrency * sizeof(struct connection));

    stats = malloc(requests * sizeof(struct data));

    FD_ZERO(&readbits);
    FD_ZERO(&writebits);

    /* setup request */
    if (posting <= 0) {
        sprintf(request, "%s %s HTTP/1.0\r\n"
                "User-Agent: ApacheBench/%s\r\n"
                "%s" "%s" "%s"
                "Host: %s\r\n"
                "Accept: */*\r\n"
                "%s" "\r\n",
                (posting == 0) ? "GET" : "HEAD",
```

```
                path,
                VERSION,
                keepalive ? "Connection: Keep-Alive\r\n" : "",
                cookie, auth, hostname, hdrs);
    }
    else {
        sprintf(request, "POST %s HTTP/1.0\r\n"
                "User-Agent: ApacheBench/%s\r\n"
                "%s" "%s" "%s"
                "Host: %s\r\n"
                "Accept: */*\r\n"
                "Content-length: %d\r\n"
                "Content-type: %s\r\n"
                "%s"
                "\r\n",
                path,
                VERSION,
                keepalive ? "Connection: Keep-Alive\r\n" : "",
                cookie, auth,
                hostname, postlen,
                (content_type[0]) ? content_type : "text/plain", hdrs);
    }

    if (verbosity >= 2)
        printf("INFO: POST header == \n---\n%s\n---\n", request);

    reqlen = strlen(request);

#ifdef CHARSET_EBCDIC
    ebcdic2ascii(request, request, reqlen);
#endif                          /* CHARSET_EBCDIC */

    /* ok - lets start */
    gettimeofday(&start, 0);

    /* initialise lots of requests */
    for (i = 0; i < concurrency; i++)
        start_connect(&con[i]);

    while (done < requests) {
        int n;
        /* setup bit arrays */
        memcpy(&sel_except, &readbits, sizeof(readbits));
        memcpy(&sel_read, &readbits, sizeof(readbits));
        memcpy(&sel_write, &writebits, sizeof(readbits));

        /* check for time limit expiry */
        gettimeofday(&now, 0);
        if (tlimit && timedif(now, start) > (tlimit * 1000)) {
            requests = done;   /* so stats are correct */
        }

        /* Timeout of 30 seconds. */
        timeout.tv_sec = 30;
        timeout.tv_usec = 0;
        n = ap_select(FD_SETSIZE, &sel_read, &sel_write, &sel_except, &timeout);
        if (!n) {
            err("\nServer timed out\n\n");
        }
        if (n < 1)
            err("select");

        for (i = 0; i < concurrency; i++) {
            int s = con[i].fd;
            if (FD_ISSET(s, &sel_except)) {
                bad++;
                err_except++;
                start_connect(&con[i]);
                continue;
            }
            if (FD_ISSET(s, &sel_read))
                read_connection(&con[i]);
            if (FD_ISSET(s, &sel_write))
                write_request(&con[i]);
        }
    }
    if (use_html)
        output_html_results();
    else
        output_results();
}

/* ------------------------------------------------------- */

/* display copyright information */
static void copyright(void)
{
    if (!use_html) {
        printf("This is ApacheBench, Version %s\n", VERSION " <$Revision: 1.40 $>
apache-1.3");
```

```
        printf("Copyright (c) 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/\n");
        printf("Copyright (c) 1998-1999 The Apache Group, http://www.apache.org/\n");
        printf("\n");
    }
    else {
        printf("<p>\n");
        printf(" This is ApacheBench, Version %s <i>&lt;%s&gt;</i> apache-1.3<br>\n",
VERSION, "$Revision: 1.40 $");
        printf(" Copyright (c) 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/<br>\n");
        printf(" Copyright (c) 1998-1999 The Apache Group,
http://www.apache.org/<br>\n");
        printf("</p>\n<p>\n");
    }
}

/* display usage information */
static void usage(char *progname)
{
    fprintf(stderr, "Usage: %s [options] [http://]hostname[:port]/path\n", progname);
    fprintf(stderr, "Options are:\n");
    fprintf(stderr, "    -n requests     Number of requests to perform\n");
    fprintf(stderr, "    -c concurrency  Number of multiple requests to make\n");
    fprintf(stderr, "    -t timelimit    Seconds to max. wait for responses\n");
    fprintf(stderr, "    -p postfile     File containg data to POST\n");
    fprintf(stderr, "    -T content-type Content-type header for POSTing\n");
    fprintf(stderr, "    -v verbosity    How much troubleshooting info to print\n");
    fprintf(stderr, "    -w              Print out results in HTML tables\n");
    fprintf(stderr, "    -i              Use HEAD instead of GET\n");
    fprintf(stderr, "    -x attributes   String to insert as table attributes\n");
    fprintf(stderr, "    -y attributes   String to insert as tr attributes\n");
    fprintf(stderr, "    -z attributes   String to insert as td or th attributes\n");
    fprintf(stderr, "    -C attribute    Add cookie, eg. 'Apache=1234'
(repeatable)\n");
    fprintf(stderr, "    -H attribute    Add Arbitrary header line, eg. 'Accept-
Encoding: zop'\n");
    fprintf(stderr, "                    Inserted after all normal header lines.
(repeatable)\n");
    fprintf(stderr, "    -A attribute    Add Basic WWW Authentication, the
attributes\n");
    fprintf(stderr, "                    are a colon separated username and
password.\n");
    fprintf(stderr, "    -p attribute    Add Basic Proxy Authentication, the
attributes\n");
    fprintf(stderr, "                    are a colon separated username and
password.\n");
    fprintf(stderr, "    -V              Print version number and exit\n");
    fprintf(stderr, "    -k              Use HTTP KeepAlive feature\n");
    fprintf(stderr, "    -h              Display usage information (this message)\n");
    exit(EINVAL);
}

/* ------------------------------------------------------- */

/* split URL into parts */

static int parse_url(char *url)
{
    char *cp;
    char *h;
    char *p = NULL;

    if (strlen(url) > 7 && strncmp(url, "http://", 7) == 0)
        url += 7;
    h = url;
    if ((cp = strchr(url, ':')) != NULL) {
        *cp++ = '\0';
        p = cp;
        url = cp;
    }
    if ((cp = strchr(url, '/')) == NULL)
        return 1;
    strcpy(path, cp);
    *cp = '\0';
    strcpy(hostname, h);
    if (p != NULL)
        port = atoi(p);
    return 0;
}

/* ------------------------------------------------------- */

/* read data to POST from file, save contents and length */

static int open_postfile(char *pfile)
{
    int postfd, status;
    struct stat postfilestat;
```

```
    if ((postfd = open(pfile, O_RDONLY)) == -1) {
        printf("Invalid postfile name (%s)\n", pfile);
        return errno;
    }
    if ((status = fstat(postfd, &postfilestat)) == -1) {
        perror("Can\'t stat postfile\n");
        return status;
    }
    postdata = malloc(postfilestat.st_size);
    if (!postdata) {
        printf("Can\'t alloc postfile buffer\n");
        return ENOMEM;
    }
    if (read(postfd, postdata, postfilestat.st_size) != postfilestat.st_size) {
        printf("error reading postfilen");
        return EIO;
    }
    postlen = postfilestat.st_size;
    return 0;
}

/* ------------------------------------------------------- */

extern char *optarg;
extern int optind, opterr, optopt;

/* sort out command-line args and call test */
int main(int argc, char **argv)
{
    int c, r,l;
    char tmp[1024];

    /* table defaults  */
    tablestring = "";
    trstring = "";
    tdstring = "bgcolor=white";
    cookie[0] = '\0';
    auth[0] = '\0';
    hdrs[0] = '\0';
    optind = 1;
    while ((c = getopt(argc, argv, "n:c:t:T:p:v:kVhwix:y:z:C:H:P:A:")) > 0) {
        switch (c) {
        case 'n':
            requests = atoi(optarg);
            if (!requests) {
                err("Invalid number of requests\n");
            }
            break;
        case 'k':
            keepalive = 1;
            break;
        case 'c':
            concurrency = atoi(optarg);
            break;
        case 'i':
            if (posting==1)
                err("Cannot mix POST and HEAD");

            posting = -1;
            break;
        case 'p':
            if (posting!=0)
                err("Cannot mix POST and HEAD");

            if (0 == (r = open_postfile(optarg))) {
                posting = 1;
            }
            else if (postdata) {
                exit(r);
            }
            break;
        case 'v':
            verbosity = atoi(optarg);
            break;
        case 't':
            tlimit = atoi(optarg);
            requests = MAX_REQUESTS;  /* need to size data array on
                                       * something */
            break;
        case 'T':
            strcpy(content_type, optarg);
            break;
        case 'C':
            strncat(cookie, "Cookie: ", sizeof(cookie));
            strncat(cookie, optarg, sizeof(cookie));
            strncat(cookie, "\r\n", sizeof(cookie));
            break;
        case 'A':
            /* assume username passwd already to be in colon separated form. Ready
             * to be uu-encoded.
```

```
             */
            while(isspace(*optarg))
               optarg++;
            l=ap_base64encode(tmp,optarg,strlen(optarg));
            tmp[l]='\0';

            strncat(auth, "Authorization: Basic ", sizeof(auth));
            strncat(auth, tmp, sizeof(auth));
            strncat(auth, "\r\n", sizeof(auth));
            break;
        case 'P':
            /*
             * assume username passwd already to be in colon separated form.
             */
            while(isspace(*optarg))
               optarg++;
            l=ap_base64encode(tmp,optarg,strlen(optarg));
            tmp[l]='\0';

            strncat(auth, "Proxy-Authorization: Basic ", sizeof(auth));
            strncat(auth, tmp, sizeof(auth));
            strncat(auth, "\r\n", sizeof(auth));
            break;
        case 'H':
            strncat(hdrs, optarg, sizeof(hdrs));
            strncat(hdrs, "\r\n", sizeof(hdrs));
            break;
        case 'V':
            copyright();
            exit(0);
            break;
        case 'w':
            use_html = 1;
            break;
            /*
             * if any of the following three are used, turn on html output
             * automatically
             */
        case 'x':
            use_html = 1;
            tablestring = optarg;
            break;
        case 'y':
            use_html = 1;
            trstring = optarg;
            break;
        case 'z':
            use_html = 1;
            tdstring = optarg;
            break;
        case 'h':
            usage(argv[0]);
            break;
        default:
            fprintf(stderr, "%s: invalid option `%c'\n", argv[0], c);
            usage(argv[0]);
            break;
        }
    }
    if (optind != argc - 1) {
        fprintf(stderr, "%s: wrong number of arguments\n", argv[0]);
        usage(argv[0]);
    }

    if (parse_url(argv[optind++])) {
        fprintf(stderr, "%s: invalid URL\n", argv[0]);
        usage(argv[0]);
    }

    copyright();
    test();

    exit(0);
}
```