NAME

userver − Micro-server
(A fairly simple web server designed for experimentation)

SYNOPSIS

userver

[**-A**|**--accepts-only**|**--close-after-accept**] [**-B**|**--block-for-send_events**] [**-C**|**--caching-on**] [**-D**|**--debug-mask** *[0x]N*] [**-E**|**--send-events**] [**-F**|**--reset-on-close**] [**-G**|**--use-getpid**] [**-H**|**-h**|**-?**|**--help**] [**-I**|**--rcv-sock-low-wat** *N*] [**-J**|**--snd-sock-low-wat** *N*] [**-K**|**--kdebug-mask** *[0x]N*] [**-L**|**--full-read** *N*] [**-M**|**--multi-accept**] [**-N**|**--send-loop**] [**-O**|**--select-timeout** *N*] [**-P**|**--count-sigpipes**] [**-Q**|**--ecb-hi-water** *N*] [**-R**|**--read-sockbuf-size** *N*] [**-S**|**--dont-intr-select**] [**-T**|**--port** *N*] [**-U**|**--auto-accept**] [**-V**|**--use-sendfile**] [**--auto-accept-aperture**] [**-W**|**--write-sockbuf-size** *N*] [**-Y**|**--send-io-events**] [**-Z**|**--content-type**] [**-a**|**--async-mode**] [**-b**|**--ecb-buf-entries** *N*] [**-c**|**--max-conns** *N*] [**-d**|**--delay** *N*] [**-e**|**--extra-accepts** *N*] [**-f**|**--max-fds** *N*] [**-g**|**--accept-on-close**] [**-i**|**--interactive**] [**-j**|**--kernel-info**] [**-k**|**--sigio-use-procmask**] [**-l**|**--listenq** *N*] [**-m**|**--accept-count** *N*] [**-n**|**--no-accepts**|**--listen-only**] [**-o**|**--process-fd-order** [**up**|**down**|**writes-down**|**writes-up**|**lru**|**lifo**|**fifo**]] [**-p**|**--procs** *N*] [**-q**|**--ecb-low-water** *N*] [**-r**|**--eager-reads**] [**-s**|**--conns-in-server-loop**] [**-t**|**--free-fd-thresh-old** *N*] [**-u**|**--track-max-fd** *N*] [**-v**|**--use-poll**] [**-w**|**--eager-writes**] [**-x**|**--sigio-accepts**] [**-y**|**--memcpy**] [**-z**|**--rejection-rate**] [**-1**|**--use-epoll**] [**-2**|**--use-epoll2**] [**-3**|**--use-epoll-ctlv**] [**--cfg-filename** *path*] [**--close-after-sock-init**] [**--close-after-read**] [**--close-after-parse**] [**--send-polls-for-accepts**] [**--cache-table-size** *N*] [**--cache-max-bytes** *N*] [**--cache-max-file-size** *N*] [**--cache-max-load-factor** *N*] [**--cache-lock-pages**] [**--cache-table-print**] [**--cache-for-spec**] [**--cache-warm=path**] [**--trace-summary**] [**--trace-summary-only**] [**--use-tcp-cork**] [**--use-madvise**] [**--use-accept-send**] [**--ip-addr** *N.N.N.N[:P]*] [**--use-aio-accept**] [**--use-aio-read**] [**--use-aio-write**] [**--use-aio-close**] [**--use-socket-aio**] [**--use-aio-wait**] [**--aio-read-before-accept**] [**--aio-read-before-write**] [**--aio-write-events-limit** *N*] [**--aio-read-events-limit** *N*] [**--aio-accept-events-limit** *N*] [**--aio-accept-thold** *N*] [**--aio-complq-count** *N*] [**--pid-filename** *path*] [**--trace-filename** *path*] [**--read-buffer-size** *N*] [**--reply-buffer-size** *N*] [**--dyn-buffer-size** *N*] [**--num-dyn-buffers** *N*] [**--num-dyn-buffers-per-app** *N*] [**--num-dyn-buffers-per-appserver** *N*] [**--dyn-lock-pages**] [**--dyn-touch-pages**] [**--stats-interval** *N*] [**--ignore-fd-setsize**] [**--use-cpu-mask** *[0x]N*] [**--idle-threshold** *N*] [**--doc-root** *path*] [**--hostname** *name*] [**--app** *uri,type[,path[,count]]*] [**--start-app-server** *path[,count][=exec_string]*] [**--app-req-queue-size** *N*] [**--skip-header**] [**--victim** *filepath*] [**--victim-skip** *N]* [**--cache-miss-skip** *N]* [**--call-stats** *N]*

DESCRIPTION

userver is a micro web server that is meant to be used to experiment with the design and implementation of web servers. In particular the original intention was to permit experimental comparisons of different event dispatch mechanisms within the same application framework. For this reason there are lots and lots and lots of parameters. They are mainly used to control the behaviour of the server.

Some of the design philosophy and experimental results obtained using this server are described in [Brecht-2001]. One of note is that all data structures are allocated at initialization and are not resized (grown). This helps to ensure that experiments can be repeated because with dynamic sizing of data structures different loads could result in different server behaviour.

The userver uses an event driven architecture. The idea is to run one event driven server per processor. If using sendfile on Linux the file system buffer cache will be shared across all servers. For high-performance one would want to use sendfile since it doesn't involve extra copying from the file buffer cache to socket buffers (i.e., it's a zero copy implementation).

IMPORTANT: This micro web server is not meant to be a full-blown web server. It is specifically designed for conducting performance experiments. For that reason design decisions were made in favour of producing repeatable experiments.

EXAMPLES

userver
: This starts the **userver** with all of its default options.

userver --max-conns 300 --accept-count 25
: This command causes the **userver** server to limit the number of simultaneous connections to 300. When an event indicating that a new connection is requested accept is called continuously until either 25 new connections are accepted, or there are no more outstanding connections.

userver --max-conns 300 --accept-count 0
: This command causes the **userver** server to limit the number of simultaneous connections to 300. When an event indicating that a new connection is requested accept is called repeatedly until there are no more outstanding connections. (Note that --accept-count=0 means an infinite number of connections, i.e., no limits).

userver --ip-addr 192.168.10.105 --max-conns 300 --listenq 128

userver --ip-addr 192.168.20.105 --max-conns 300 --listenq 128
: This will start up two copies of the userver each accepting connections from a different interface.

userver --ip-addr 192.168.10.105:6801 --max-conns 300 --listenq 128

userver --ip-addr 192.168.20.105:6802 --max-conns 300 --listenq 128
: Same as above except one listens on port 6801 and the other on port 6802.

userver --port 6800 --max-conns 300 --listenq 128
: This command causes **userver** to listen for incoming connection requests on port 6800. It uses a listen queue of length 128 and it will permit up to 300 simultaneous connections. Once there are 300 open connections it will not accept new connections until an existing connection is closed.

userver --port 6800 --max-conns 300 --listenq 128 --use-poll
: This is the same as above except using *select()* calls to get events the **userver** will use *poll().*

userver --port 6800 --max-conns 300 --listenq 128 --use-epoll
: This is the same as above except using *poll()* calls to get events the **userver** will use *epoll().*

userver -C --cache-table-size=10000 --cache-max-bytes=262144000 --cache-max-file-size=1048576
: Caching is turned on. Caching parameters limit the number of files that are cached to 10000, the total number of bytes cached to 250 MB, and the size of the largest file that can be cached to 1 MB.

OPTIONS

The operation of **userver** can be controlled through a number of options. The tool supports both short (one-character) and long (arbitrary-length) option names. Short options are prefixed with a single leading dash (-), long options with a double-dash (--). Multiple short options can be grouped together (e.g., ‘‘**-rw**’’ is equivalent to ‘‘**-r -w**’’) and long options can be abbreviated so long as they remain unique. Parameters to options can be specified either by following the long option name with an equal sign and the parameter value (e.g., **--port=6800**) or by separating the option name and value with whitespace (e.g., **--port 6800**).

OPTIONS (GENERAL)

--idle-threshold=N

In a situation where the number of open connections has reached the limit (see --max-conns), the userver will close the least recently used connection if it has not seen activity in the last N seconds.

--use-cpu-mask= [0x]N

On systems that support the *sched_setaffinity()* system call, specifies a mask of schedulable CPUs for the userver process. If prefixed with ’0x’, the argument is interpreted as hexadecimal; otherwise it is interpreted as decimal. The least significant bit corresponds to the first logical processor number on the system.

-f N

--max-fds=*N*

The maximum number of open file descriptors permitted. I can’t recall if this is actually enforced but it is used to size internal data structures. Note that there is a relationship between **--max-conns** and **--max-fds.** If files are not being cached you could (in the extreme case) have one fd used for the socket connection and one fd used for the open file. So one needs to set **--max-fds** = more than 2 x **--max-conns.** Note: it should be greater because of some descriptors that are used up by stdin, out, err, and a few open files, etc.

-m N

--accept-count=*N*

Call accept repeatedly until either, **--max-conns** is reached, **accept()** returns EWOULDBLOCK, or *N* new connections have been accepted. Note that the value of *N* can make a substantial difference in the performance of event dispatch mechanisms under heavy loads. See [Brecht-2001].

-T N

--port=*N*

Specify the port number that the server should listen for connections on.

-l N

--listenq=*N*

Specify the length of the application’s listen queue. This specifies how many connection requests are allowed to be queued in the application before the kernel rejects incoming connection requests. WARNING: on all version of Linux I’ve ever seen (up to and including several 2.6.5) the kernel silently converts any values of N that are greater than 128 to 128. For details see sys_listen and the value of SOMAXCONNS.

-L[1 | 2]

--full-read=[1 | 2]

Affects how calling read works. If this is set to 1 the server will loop on reading the socket until the read fails. This mainly happens either because there is nothing left to read because the other end has been closed or until it would have blocked (EWOULDBLOCK).

If a value of 1 is used the loop on reads will only occur on calls to read that occur as a result of an event indicating that the first read should be done. If a value of 2 is used the loop on reads occur as above but if **--eager-reads** is also set the server will loop on eager reads (i.e., those that will be done immediately following a the acceptance and setup of a connection).

-H

-h

-?

--help Print out the usage message.

-M N

--multi-accept=*N*

Use the *multiaccept()* system call to perform accepts. NOTE: this was a system call that we added to and experimented with in Linux. The idea here is that since we are repeatedly calling *accept()* to accept a bunch of connections, why not provide a system call that allows up to a maximum of *N* connections to be accepted in one system call.

-S

--dont-intr-select

Disable asynchronous event notification when calling *select().* I believe that this was initially meant to work for SIGIO and for SEND.

-d N

--delay=*N*

The idea was to introduce a delay that might simulate work being done by the server (like some computation and/or a dynamic request). I couldn't find a good method for delaying for a specified amount of time (note that it should be small) with a enough accuracy.

-e N

--extra-accepts=*N*

While in the middle of processing events that have been obtained (e.g., from select, or some other mechanism) periodically poll for new connection requests. In this case every time *N* descriptors are processed we check for new connection requests. Note that this open was never used effectively. I found it easier to use and tune the **--accept-count** option.

-g

--accept-on-close

When a connection is closed check to see if there are any outstanding connection requests. Note: that this only occurs if the maximum number of connections was reached and now a closed connection will reduce the number of connections below **--max-conns.**

-o [up|down|writes-down|writes-up|lru|lifo|fifo]

--process-fd-order

[up|down|writes-down|writes-up|lru|lifo|fifo]

Change the order in which file descriptors are handled. They are numbered from 0 to *N* so the options are:

up: from 0 to N

down: from N to 0

writes-down:
to writes from N to 0, then reads from N to 0

writes-up:
: to writes from 0 to N, then reads from 0 to N

lru: least recently descriptor touched first

lifo: first descriptor (connection) added to the set first

filo: first descriptor (connection) added to the set last

-p N

--procs=*N*
: Start *N* copies of the server. Note that this option hasn't been tested in a long time and is almost certain to cause problems.

-s

--conns-in-server-loop
: Add an additional poll for new connections each time through the server loop.

-t N

--free-fd-threshold=*N*
: When the maximum number of connections **--max-conns** is reached the server stops checking for incoming connections. This options controls when the server starts checking again. It starts checking again after *N* connections can be accommodated before reaching **--max-conns.**

-u

--track-max-fd
: The server currently keeps track of the maximum fd ever used and uses that as a parameter to calls like select. This option implements code to dynamically keep track of the maximum file descriptor currently used. Note that this requires a bit of extra processing but it's probably not noticeable, especially if it can save extra overhead on copying fdsets and processing time in event mechanisms (e.g., select).

-r

--eager-reads
: try to eagerly read from new connections. Call *read()* to try to read the request as soon as the new connection is accepted. In this mode the server optimistically assumes that data will be available for reading when a connection is made on a new socket (or very shortly after). If the assumption is incorrect, the read call simply returns with the error EWOULDBLOCK. Later the event notification mechanism will indicate when the socket is readable.

-w

--eager-writes
: try to eagerly perform writes to new connections when the response is available. Both **--eager-read** and **--eager-write** also try to take advantage of any potential locality effects by working on the most recently used file descriptor and socket.

-y

--memcpy
: Use the **memcpy** library call to copy interest sets before calling select. This is done rather than doing a straight assignment. This exists because the size of the interest set could be large and I don't know how clever a compiler would be about assigning large data structures. NOTE: it

currently copies the side of an fd_set. This should be changed to only copy the amount of data required for the maximum fd of interest.

-z

--rejection-rate
Use the **rejection-rate** is not yet implemented. The ideas would be to specify the frequency with which new connection requests should be rejected.

-V

--use-sendfile
Use the **sendfile()** call on systems that have it available. This is only used for uncached files and dynamic content.

--use-madvise
Use *madvise()* calls to provide the kernel with hints that files are being read sequentially and when they should no longer be cached.

-v

--use-poll
Use *poll()* instead of *select()* to get events.

-1

--use-epoll
Use *epoll()* instead of *select()* to get events.

-2

--use-epoll2
Use *epoll()* instead of *select()* to get events, but attempt some optimizations to reduce the number of *epoll_ctl()* calls. We found the number of epoll_ctl calls to be excessive so this options calls epoll_ctl when a new connection is added and sets the interest to be READ and WRITE. This means that epoll_wait may return events that we aren't interested in (e.g., that a socket is writable even though we are busy reading) but we are exploring the tradeoffs between the two approaches (--use-epoll and --use-epoll2).

-3

--use-epoll-ctlv
Use *epoll_ctlv()* to update events. This requires the addition of a new system call, epoll_ctlv. This call permits one to collect a bunch of epoll_ctl calls and change interest in a bunch of fds at once instead of having to do one system call (epoll_ctl) per change.

--cfg-filename path
Read additional options from a file. The format of the file is the same as the command line, i.e. long and/or short options separated by whitespace (spaces, tabs, linefeeds). Lines with a '#' character in the first column are treated as comments and are ignored. Config files can use **--cfg-filename** to nest other config files.

--ip-addr N.N.N.N

--ip-addr N.N.N.N:P

Specify an IP address to listen for new connections on. Optionally takes a port number P as well. The special option --ip-addr 0.0.0.0 specifies that the userver should listen on any address (this is the default).

--pid-filename path

Sets the name of the file where the userver stores its process ID number while running. Specify an empty string ("") to suppress pid file generation. If this option is not specified, a default value of "userver.pid" is used.

--read-buffer-size N

The number of bytes to allocate at initialization time for *read()* operations on each connection.

--reply-buffer-size N

The number of bytes to allocate at initialization time for *write()* operations on each connection (for static requests only).

--stats-interval N

Print out some simple stats about related to server performance at a regularly specified interval. The interval is specified in seconds. NOTE: this options overrides the --select-timeout option in order to be able to print out stats according to the selected interval.

--ignore-fd-setsize

If permitted this causes the userver to ignore limits placed on the number of open connections and the maximum values of an open file descriptors that are required when using select (i.e., the FD_SETSIZE). This option does not work if --track-max-fd is defined or if the event mechanism being used is select or SEND.

-Z

--content-type

Use heuristics to guess what Content-Type header to return, based on the uri from the request. For example, if the uri ended in ".gif", the userver would assume a content type of "image/gif".

--doc-root path

Set the directory of where to find the documents being server to the specified directory. When the document root is specified all requests are assumed to be relative to the specified directory.

--hostname name

Set the name of the machine that the userver is running on. By default, userver uses *gethostname()* to determine the hostname. The hostname is only used to set the SERVER_NAME CGI parameter when fulfilling FastCGI requests.

--skip-header

Assume that all files being served contain a header in the actual file so there is no need for the server to generate and send a header. Useful for looking at the impact of things like cork/uncork when used with sendfile (i.e., if the header is in the file it's one sendfile system call instead of cork, write, sendfile, uncork).

--call-stats N
: Track statistics for up to N calls/requests. This is designed to read the Client-Id: header that we have httperf send in order to correlate what is happening with requests from different clients on different servers.

OPTIONS (FOR CONTROLLING SOCKETS)

-F

--reset-on-close
: Calls *shutdown()* instead of *close()*.
WARNING do not use this for a real system!!!

-I N

--rcv-sock-low-wat=*N*
: Sets the sockets receive buffer's low water mark to *N*. Not available on all systems.

-JN

--snd-sock-low-wat=N
: Sets the sockets send buffer's low water mark to *N*. Not available on all systems.

-O N

--select-timeout=*N*
: Set the timeout option to the *select()* system call.

-R N

--read-sockbuf-size=*N*

-W N

--write-sockbuf-size=*N*
: Set the size of the read or write socket buffers. Note that increasing these may help significantly for large transfers.

--use-tcp-cork
: Cork the TCP queue when writing the header for the reply. Currently only implemented for the sendfile portion of the code.

-c N

--max-conns=*N*
: The maximum number of simultaneous connections the server will handle. Note that this is typically limited by the number of open file descriptors permitted. Once the maximum number of open connections is reached incoming connection requests are refused until an existing connection is close. Note that idle connections can be timed out eventually (but this hasn't been tested lately). This is also used to size a number of internal data structures that are allocated at initialization time.

OPTIONS (FOR SERVING DYNAMIC CONTENT)

--app uri,type[,path[,count]]

Define an application. Applications are used to generate dynamic content. An application is uniquely identified by the uri string, which is case sensitive. The type string is not case sensitive, and can be one of

FastCGI:

General-purpose FastCGI application.

The path argument specifies the address of a listening FastCGI application, in the form of either an INET domain socket address (addr:port) or a UNIX domain socket address (file path). The optional count argument can be specified with INET domain socket addresses to specify that count consecutive ports, starting with port, are all listening FastCGI applications. It is a shortcut to specifying them individually using multiple --app arguments.

SPECweb99:

SPECweb99 server application (only available if support is compiled in). The userver will treat requests of the specified uri as SPECweb99 requests, and will generate responses on the fly.

The path and count arguments are not applicable and must be omitted.

--start-app-server path[,count][=exec_string]

Tell the userver to start up the specified application server(s). (By default, userver assumes that they are already started through some other means, e.g. manually or through a script). The path and optional count arguments are as described for the **--app** option, above. It is an error to try to start up any application servers that aren't defined using the **--app** option. The optional exec_string argument is a string suitable for passing to execl() to start the application server(s). Note that if exec_string contains spaces or shell metacharacters, you will have to enclose it in quotes and/or escape those metacharacters in order for the string to be seen correctly by userver. If exec_string is omitted, then it is inferred by interpreting the uri in the corresponding application as an actual directory path rooted at the document root (see the **--doc-root** option).

--app-req-queue-size N

The number of queue spaces, per application (see the **--app** option above), to reserve for dynamic requests. Requests need to be queued whenever an application server or some other resource (such as dynamic buffer space, see above) is temporarily unavailable. Should the request queue ever become full, userver will respond to arriving requests with an HTTP 503 error. If not specified, this option takes on the same value as **--max-conns** so that it is effectively impossible to overflow the request queue.

--app=APP,PROTO,HOSTNAME:PORT,NUM

Registers one or more application servers with the userver. This means that dynamic requests for APP will be forwarded to the application server(s) running on HOSTNAME and listening on PORT. The optional NUM argument specifies the number of application servers running on HOSTNAME. When NUM is greater than one, the application servers listen on consecutive ports, starting with PORT. The PROTO flag specifies the communication protocol. Currently, only the FASTCGI protocol is supported. Here is an example:

--app=specweb99-fcgi.pl,FASTCGI,localhost:9000,24

In this example, dynamic requests for the specweb99-fcgi.pl are handled by 24 application servers running on the same machine as the userver. These application servers are listening on ports 9000 - 9023, and communicate with the userver using the FASTCGI protocol.

--start-app-server=localhost:PORT,NUM=PATH,MASK

Causes the userver to start one or more application servers on the userver's localhost. The NUM parameter specifies the number of copies of the application server that will be started. If NUM is greater than 1, then the application server copies will listen on consecutive ports starting with PORT. The PATH parameter gives an absolute path to the application server executable. The MASK parameter is optional, and specifies the CPU affinity mask that will apply to the newly started application servers. Here is an example:

--start-app-server=localhost:9000, 18=/usr/spec/specweb99-fcgi.pl, 0x000d

In this example, 18 copies of the /usr/spec/specweb99-fcgi.pl script will be started. These application servers will listen on ports 9000 - 9017 on the userver's localhost. The MASK of 0x000d is the CPU affinity mask, as interpretd by the sched_setaffinity system call.

--dyn-buffer-size N

The number of bytes to allocate at initialization time for buffering replies to dynamic requests. The default is one megabyte (1048576 bytes). The number of such buffers can be specified using **--num-dyn-buffers** or **--num-dyn-buffers-per-app** or **--num-dyn-buffers-per-appserver** (see below). If none of these options is specified, then by default the userver allocates 2 buffers per application server.

--num-dyn-buffers N

The number of buffers to allocate at initialization time for buffering replies to dynamic requests. If this option is specified, then the **--num-dyn-buffers-per-app** and **--num-dyn-buffers-per-appserver** options are ignored.

--num-dyn-buffers-per-app N

The number of buffers, per application (see the **--app** option below), to allocate at initialization time for buffering replies to dynamic requests. If this option is specified, then the **--num-dyn-buffers-per-appserver** option is ignored. This option is ignored if the **--num-dyn-buffers** option is specified.

--num-dyn-buffers-per-appserver N

The number of buffers, per application server (see the **--app** option below), to allocate at initialization time for buffering replies to dynamic requests. The default is 2. This option is ignored if either the **--num-dyn-buffers** or the **--num-dyn-buffers-per-app** option is specified.

--dyn-lock-pages

Lock all dynamic buffer pages into memory (must run as root for this to work).

--dyn-touch-pages

Touch each dynamic buffer page at initialization time. This should have the effect of faulting them into memory. However, in the absense of the **--dyn-lock-pages** option, these pages are still subject to being paged out.

OPTIONS (FOR USE WITH SIGIO)

-a

--sigio-accepts
Use asynchronous notification via SIGIO to accept new connections. Asynchronous notification may sound costly at first but consider that some form of asynchronous notification may be the only way for an application that is in the middle of something (e.g., computation) to find out about a connection requests.

-k

--sigio-use-procmask
To disable and later enable asynchronous notification of incoming requests when using SIGIO one can use **fcntl()** to turn off and on asynchronous notification or use **sigprocmask().** Be careful of race conditions. By default the server uses **fcntl()** but this option says to use **sigprocmask()** instead.

OPTIONS (FOR ASYNCHRONOUS I/O)

This is experimental and currently the userver only makes use of asynchronous socket I/O. This is still under development and is being tested. See aio_layer.h for a definition of the interface that the userver uses to interact with the underlying asynchronous layer. Note that not all of the calls in aio_layer.h are meant to be asynchronous. Some are simply required in order to complete the layer. E.g., aio_sock_create is synchronous but it is used to create a socket that can be used asynchronously.

--use-aio-accept

--use-aio-read

--use-aio-write

--use-aio-close
Use the asynchronous version of the corresponding system calls for socket I/O. These definitions exist but are all currently only used together. It's pretty unlikely that any of these could be used independently.

--use-socket-aio
Use ansynchronous I/O for sockets. This turns on all of the above options.

--use-aio-wait
There are two ways to get events from the AIO layer. The aio_wait call returns available events of all types in a single array. It's then up to the userver to handle them in the order they appear or to sort through them. This is not on by default and by default the userver uses multiple calls to aio_sock_getevents which only gets events events of a specified type. Using aio_sock_getevents permits us to get all events of a specified type and to process them. This enables us to order events to be processed by event type.

--aio-accept-thold=**N**
This is meant to control the maximum number of outstanding accept calls that can be initiated (preposted). Using this in conjunction with --accept-count (-m) we should be able to impact the accept rate. I think that larger values of --aio-accept-thold used in conjunction with --accept-count 0 (-m 0) should be pretty aggressive about accepting new connections. This hasn't been tested very extensively. Especially in conjunction with different values of --accept-count.

--aio-read-before-accept

With this option the userver calls aio_sock_read_accept whenever a new connection should be accepted. This initiates a read call before initiating an accept call. This informs the underlying system of where the read buffer is located so that when data is available it can be immediately placed into the specified buffer.

--aio-read-before-write

With this option the userver initiates a read call before it tries to write a request. Again this is meant to permit arriving data to be copied directly into the specified read buffer. It can only be initiated after the request has been parsed and is deemed to be complete. Otherwise a prepost might possibly allow the read buffer to be overwritten.

--aio-accept-events-limit=**N**

Limits how many accept completion events to get out of the accept completion queue at one time. The idea is that it might be possible to use this and the read and write limits below in order to control how the work being processed is balanced. Note that one might actually prefer to adjust these values dynamically.

--aio-write-events-limit=**N**

Limits how many write completion events to get out of the write completion queue at one time.

--aio-read-events-limit=**N**

Limits how many read completion events to get out of the read completion queue at one time.

--aio-completion-order=**string**

Set the order in which completion events will get processed. The string must contain ’r’, ’w’, and ’a’ characters. Read completions are specified with ’r’, write completions with ’w’, and accept completions with ’a’. So "rwa" means process read completions, then, write completions followed by accept completions. Note that it is possible to specify a string like: "rwrwa".

--aio-complq-count=**[1|3]**

Specifies how many completion queus to use. Currently only supports 1 or 3. Note that for a single completion queues limits specified on --aio-write-events-limit, --aio-read-events-limit, or --aio-accept-events-limit don’t really apply. Instead the sum of these is used to limit the number of events handled at one time (in the 1 queue case).

OPTIONS (FOR OPEN FILE AND HEADER CACHING)

-C

--caching-on

Turns caching on.

--cache-table-size=*N*

Specifies the size of the hash table used to cached files. One entry is used per cached file. NOTE: if the **--cache-max-load-factor** is exceeded the server will print a message and exit. We do not automatically increase the size of the hash table because we want to ensure identical and repeatable behaviour from one experiment to the next.

--cache-max-load-factor=*N*

Specifies the maximum proportion of entries that can be filled in the hash table, expressed as a fraction of the table size. For example **--cache-max-load-factor = 0.70** will not permit the hash table to become greater than 70% full. If it does, a message is printed and the server exits.

--cache-max-bytes=*N*

Specifies the maximum number of bytes cached.

--cache-max-file-size=*N*

Don't cache files larger than *N* bytes.

--cache-table-print

When the program ends, print out the contents of the hash table used for caching.

--cache-for-spec

This enables a special hash function that does perfect hashing on URLs requested by SPECweb99. This is intended for experimentation only.

--cache-lock-pages

Lock all cached pages into memory (must run as root for this to work).

--cache-warm*<filename>*

If specified and caching is enabled then read a list of files and/or directories in <filename>. For each file add it to the userver cache and touch each page. If it's a directory, cache and touch each file in the directory. If --cache-table-print is declared then print the cache after it's been primed.

OPTIONS (FOR DEBUGGING)

-D [0x]N

--debug-mask=*[0x]N***]**

Specify a debug mask. If prefixed with '0x', the argument is interpreted as hexadecimal; otherwise it is interpreted as decimal. For maximum debugging detail use **--debug-mask =** *0xffffffff*. To turn debugging messages off use **--debug-mask =** *0x0*. See debug.h for details. Note: that to compile without debugging also see debug.h.

--trace-summary

Keep track of information regarding system calls and print a summary. If tracing is enabled a list of each event being traced will be dumped to a trace output file.

--trace-summary-only

Keep track of information regarding system calls and print a summary. If this option is enabled individual events are not actually recorded but instead information to produce a summary of each event type is tracked. For example, minimum, maximum and average call times for each system call being traced.

--trace-filename path

Specify the name of the file to write tracing output to, when tracing is enabled.

-i

--interactive

Turns on interactive mode. The server stops between calls to process events and prompts the user for input. This permits the user to query some of the state of the server or to step through server execution. At the prompt type 'h' or '?' for a list of commands.

-P

--count-sigpipes

Install a signal handler that simply counts the number of SIGPIPE signals received.

OPTIONS (FOR EXPERIMENTS RELATED TO BOUNDING PERFORMANCE)

These options are designed to have the server stop processing a request at various stages of a requests life. It stops handling the request by closing it's end of the socket and not working on that request anymore. I've attempted to list these in the order of earliest time to drop the request to latest. The idea is that by running a series of experiments by using each of these options one might be able to get a sense of how much additional stage costs.

-n

--no-accepts

--listen-only

This puts the server into a mode where it only ever listens to requests. It never makes any attempt to process requests. So it never even accepts any requests. The believe is that this might be used to provide insights into the amount of overhead incurred when kernel TCP SYN queues and application listen queues become full.

-A

--accepts-only

--close-after-accept

In this mode the server accepts and closes incoming connection requests as quickly as possible. At the time this was done *close()* would block until the call completed (despite the fd being in non blocking mode). So immediately after accepting, call *close().* to terminate the connection.

--close-after-sock-init

The server accepts a connection, initializes the socket (setting socket options like socket buffer sizes, non blocking mode, etc) and then it closes the connection.

--close-after-read

The server accepts a connection, initializes the socket reads the request and then closes the connection.

--close-after-parse

The server accepts a connection, initializes the socket reads the request, parses the requests and then closes the connection.

--fake-writing

The server accepts a connection, initializes the socket reads the request, parses the requests and then reads the entire file before sending a fake response (HTTP OK with size 0). Note that this only works when read/write are being used (currently this happens only when the file descriptor and response header are not cached and when sendfile is not used).

OPTIONS (FOR USE WITH THE SEND ENHANCED KERNEL)

WARNING: these options are highly experimental and subject to continuous change.

These options specify how the **userver** interacts with new Linux Scalable Event Notification and Delivery (SEND) kernel mechanisms. See [Ostrowski-2000] for more details on SEND.

-K [0x]N

--kdebug-mask=*[0x]N***]**
Specify a debugging mask which is passed to the *evtctl* system call to set a kernel debugging mask for the SEND related parts of the kernel. If prefixed with ’0x’, the argument is interpreted as hexadecimal; otherwise it is interpreted as decimal.

--send-polls-for-accepts
The send loop polls for new connections.

-B

--block-for-send_events
When there are not more events to process, block and wait for one or more new events by calling *evtctl().*

-E

--send-events
Use send events. This version does not do notification.

-G

--use-getpid
If we aren’t blocking to wait for events (by calling evtctl) we may wish to make a system call to give the kernel a chance to deliver events to the application (since they are delivered when returning from a syscall. In this case we use *getpid()* because it is likely fast.

-Y

--send-io-events
Use send events. This version does do notification.

-N

--send-loop
Use the send loop. Can potentially get and process send events even while using select. This bypasses the use of the select call.

-b N

--ecb-buf-entries=*N*
How many events can be stored in the application’s event control buffer (ecb).

-Q N

--ecb-hi-water=*N*
Set the high water mark used for the event control buffer (ecb). One of the things we've tried is to turn event delivery off when the *--ecb-hi-water* mark is reached and then to turn delivery back on when the *--ecb-low-water* mark is reached.

-q N

--ecb-low-water=*N*
Set the low water mark used for the event control buffer (ecb). One of the things we've tried is to turn event delivery off when the *--ecb-hi-water* mark is reached and then to turn delivery back on when the *--ecb-low-water* mark is reached.

-U

--auto-accept
Turn on the SEND kernel's auto accept mechanisms. When new connection requests arrive the kernel automatically accepts them (auto accepts 'em) and notifies the application with an event containing the new connection's fd.

--auto-accept-aperture
Control the difference between the number of connections the kernel has autoaccepted and the number of connections the application has closed. Without this type of control the autoaccepting can get out of control and the kernel can spend too much time auto accepting new connections and the application is not able to make much forward progress on existing connections.

--use-accept-send
Use a special version of accept that returns event information related to the accepted connection/socket.

-j

--kernel-info
Experimental: was to tell the application that the kernel has an interface available for obtaining more information about the size and number of entries in some of the kernel queues (e.g., the TCP SYNQs and the application's listenq).

OPTIONS (FOR CHOOSING AND CONTROLLING A VICTIM)

WARNING: These are just some hacks to try out some ideas.

-F

--victim string
Set the specified string as a victim string. Any uri that contains this string is declared a victim. How it is victimized depends on the value used in the --victim-skip option. If the --victim-skip option is not set there is no affect on the specified victim file.

--victim-skip N
If N is > 0, then writes to the client socket will be skipped N-1 times. The idea is to slow down the response to any client that requests the specified victim file. Specifically, when the event mechanism indicates that the socket is ready for writing the write will just be skipped. Note that this probably won't work with epoll if you are using edge triggered events. Also note that the amount of slow down depends on the number of other sockets that are ready for processing when the event mechanisim returns and the amount of processing required for each of those events.

If N = -1 then no writes are skipped but instead the server uses madvise after it has written the bytes to advise the operating system that it doesn't need the bytes for this file anymore.

--cache-miss-skip N

This is specific to BSD (possibly FreeBSD). If N is > 0, then sendfile calls that would block because of disk I/O (i.e., a file cache miss) will be skipped N times. The idea is to slow down the response to any client that requests a file that might require disk I/O. In theory this may permit us to favour requests for files that are in the file cache. Specifically, when the event mechanism indicates that the socket is ready for writing the sendfile call will be made with the flag SF_NODISKIO.

Currently this isn't intended to be used in conjunction with --victim and --victim-skip (i.e., I haven't thought through the behaviour).

OUTPUT

This section describes portions of the output produced during the execution of the **userver.**

The first portion of output lists the program name and the command line parameters. The second portion of output lists the setting of all options. This is the most reliable way to find out the default settings for some options.

The next two sections identify operating system and per user or process limits.

The remainder of the output provides more information about default settings of various parameters and a large number of statistics about the execution of the server.

AUTHORS

userver was developed by Tim Brecht by starting with some micro web servers originally developed by Abhishek Chandra and David Mosberger [Chandra-2001]. David Pariag contributed the current caching engine. One of the hash functions is from Bob Jenkin's web site.

BUGS

Probably many.

NOTE: very few if any checks are performed to see if the combination of options being used makes sense. For example, the behaviour from using both the **--select-loop** and **--send-loop** options is undefined.

Always be sure to double-check results and don't fall prey to measuring client-performance instead of server performance! The tool **httperf** is very good at generating loads.

Many of the single letter options are difficult to relate to anything meaningful. As the number of options grew I eventually started to run out of alphabet and added long options.

The user-interface definitely could be improved. A simple configuration file might be more suitable than the many command-line options.

REFERENCES

[Shukla-2006] Amol Shukla and Tim Brecht, TCP Connection Management Mechanisms for Improving Internet Server Performance, First IEEE Workshop on Web Systems and Technologies (HotWeb 2006), Boston, MA, November, 2006.

[Brecht-2006] Tim Brecht, G. (John) Janakiraman, Brian Lynn, Vikram Saletore, Yoshio Turner, Evaluating Network Processing Efficiency with Processor Partitioning and

Asynchronous I/O, Proceedings of EuroSys 2006, Leuven, Belgium, April 2006.

[Gammo-2004] Louay Gammo, Tim Brecht, Amol Shukla, and David Pariag, Comparing and Evaluating epoll, select, and poll Event Mechanisms, Ottawa Linux Symposium, July, 2004.

[Brecht-2004] Tim Brecht, Louay Gammo, and David Pariag, accept()able Strategies for Improving Web Server Performance, 2004 USENIX Annual Technical Conference: General Track, Boston, June, 2004.

[Brecht-2001] Tim Brecht and Michal Ostrowski, Exploring the Performance of Select-based Internet Servers, HP Labs Technical Report HPL-2001-314, November, 2001.

[Chandra-2001] A. Chandra and D. Mosberger, Scalability of Linux Event-Dispatch Mechanisms, Proceedings of the 2001 USENIX Annual Technical Conference, June 2001.

[Ostrowski-2000] Michal Ostrowski, A Mechanism for Scalable Event Notification and Delivery in Linux, M.Math Thesis, Department of Computer Science, University of Waterloo, November, 2000.