select — Waiting for I/O completion

This module provides access to the select() and poll() functions available in most operating systems, devpoll() available on Solaris and derivatives, epoll() available on Linux 2.5+ and kqueue() available on most BSD. Note that on Windows, it only works for sockets; on other operating systems, it also works for other file types (in particular, on Unix, it works on pipes). It cannot be used on regular files to determine whether a file has grown since it was last read.

The module defines the following:

exception select.error

A deprecated alias of OSError.

select.devpoll()

(Only supported on Solaris and derivatives.) Returns a /dev/poll polling object; see section /dev/poll Polling Objects below for the methods supported by devpoll objects.

devpoll() objects are linked to the number of file descriptors allowed at the time of instantiation. If your program reduces this value, devpoll() will fail. If your program increases this value, devpoll() may return an incomplete list of active file descriptors.

select.epoll(sizehint=-1, flags=0)

(Only supported on Linux 2.5.44 and newer.) Return an edge polling object, which can be used as Edge or Level Triggered interface for I/O events.

sizehint informs epoll about the expected number of events to be registered. It must be positive, or -1 to use the default. It is only used on older systems where epoll_create1() is not available; otherwise it has no effect (though its value is still checked).

flags is deprecated and completely ignored. However, when supplied, its value must be 0 or select.EPOLL_CLOEXEC, otherwise OSError is raised.

See the Edge and Level Trigger Polling (epoll) Objects section below for the methods supported by epolling objects.

epoll objects support the context management protocol: when used in a with statement, the new file descriptor is automatically closed at the end of the block.

select.poll()

(Not supported by all operating systems.) Returns a polling object, which supports registering and unregistering file descriptors, and then polling them for I/O events; see section Polling Objects below for the methods supported by polling objects.

select.kqueue()

(Only supported on BSD.) Returns a kernel queue object; see section Kqueue Objects below for the methods supported by kqueue objects.

select.kevent(ident, filter=KQ_FILTER_READ, flags=KQ_EV_ADD, fflags=0, data=0, udata=0)

(Only supported on BSD.) Returns a kernel event object; see section Kevent Objects below for the methods supported by kevent objects.

select.select(rlist, wlist, xlist[, timeout])

This is a straightforward interface to the Unix select() system call. The first three arguments are iterables of 'waitable objects': either integers representing file descriptors or objects with a parameterless method named fileno() returning such an integer:

    - rlist: wait until ready for reading
    - wlist: wait until ready for writing
    - xlist: wait for an "exceptional condition" (see the manual page for what your system considers such a condition)
- Empty iterables are allowed, but acceptance of three empty iterables is platform-dependent. (It is known to work on Unix but not on Windows.) The optional timeout argument specifies a time-out as a floating point number in seconds. When the timeout argument is omitted the function blocks until at least one file descriptor is ready. A time-out value of zero specifies a poll and never blocks.
- The return value is a triple of lists of objects that are ready: subsets of the first three arguments. When the time-out is reached without a file descriptor becoming ready, three empty lists are returned.
- Among the acceptable object types in the iterables are Python file objects (e.g. sys.stdin, or objects returned by open() or os.popen()), socket objects returned by socket.socket(). You may also define a wrapper class yourself, as long as it has an appropriate fileno() method (that really returns a file descriptor, not just a random integer).

select.PIPE_BUF

The minimum number of bytes which can be written without blocking to a pipe when the pipe has been reported as ready for writing by select(), poll() or another interface in this module. This doesn't apply to other kind of file-like objects such as sockets.

/dev/poll Polling Objects

Solaris and derivatives have /dev/poll. While select() is O(highest file descriptor) and poll() is O(number of file descriptors), /dev/poll is O(active file descriptors).

/dev/poll behaviour is very close to the standard poll() object.

devpoll.close()

Close the file descriptor of the polling object.

devpoll.closed

devpoll.fileno()

Return the file descriptor number of the polling object.

rue if the polling object is closed.

devpoll.register(fd[, eventmask])

Register a file descriptor with the polling object. Future calls to the poll() method will then check whether the file descriptor has any pending I/O events. fd can be either an integer, or an object with a fileno() method that returns an integer. File objects implement fileno(), so they can also be used as the argument.

eventmask is an optional bitmask describing the type of events you want to check for. The constants are the same that with poll() object. The default value is a combination of the constants POLLIN, POLLPRI, and POLLOUT.

devpoll.modify(fd[, eventmask])

This method does an unregister() followed by a register(). It is (a bit) more efficient that doing the same explicitly.

devpoll.unregister(fd)

Remove a file descriptor being tracked by a polling object. Just like the register() method, fd can be an integer or an object with a fileno() method that returns an integer.

Attempting to remove a file descriptor that was never registered is safely ignored.

devpoll.poll([timeout])

Polls the set of registered file descriptors, and returns a possibly-empty list containing (fd, event) 2-tuples for the descriptors that have events or errors to report. fd is the file descriptor, and event is a bitmask with bits set for the reported events for that descriptor — POLLIN for waiting input, POLLOUT to indicate that the descriptor can be written to, and so forth. An empty list indicates that the call timed out and no file descriptors had any events to report. If timeout is given, it specifies the length of time in milliseconds which the system will wait for events before returning. If timeout is omitted, -1, or None, the call will block until there is an event for this poll object.

Edge and Level Trigger Polling (epoll) Objects

eventmask

Constant	Meaning
EPOLLIN	Available for read
EPOLLOUT	Available for write
EPOLLPRI	Urgent data for read
EPOLLERR	Error condition happened on the assoc. fd
EPOLLHUP	Hang up happened on the assoc. fd
EPOLLET	Set Edge Trigger behavior, the default is Level Trigger behavior
EPOLLONESHOT	Set one-shot behavior. After one event is pulled out, the fd is internally disabled
EPOLLEXCLUSIVE	Wake only one epoll object when the associated fd has an event. The default (if this flag is not set) is to wake all epoll objects polling on a fd.
EPOLLRDHUP	Stream socket peer closed connection or shut down writing half of connection.
EPOLLRDNORM	Equivalent to EPOLLIN
EPOLLRDBAND	Priority data band can be read.
EPOLLWRNORM	Equivalent to EPOLLOUT
EPOLLWRBAND	Priority data may be written.
EPOLLMSG	Ignored.

epoll.close()

Close the control file descriptor of the epoll object.

epoll.closed

True if the epoll object is closed.

epoll.fileno()

Return the file descriptor number of the control fd.

epoll.fromfd(fd)

Create an epoll object from a given file descriptor.

epoll.register(fd[, eventmask])

Register a fd descriptor with the epoll object.

epoll.modify(fd, eventmask)

Modify a registered file descriptor.

epoll.unregister(fd)

Remove a registered file descriptor from the epoll object.

Changed in version 3.9: The method no longer ignores the EBADF error.

epoll.poll(timeout=None, maxevents=-1)

Wait for events. timeout in seconds (float)

Changed in version 3.5: The function is now retried with a recomputed timeout when interrupted by a signal, except if the signal handler raises an exception (see PEP 475 for the rationale), instead of raising InterruptedError.

Polling Objects

The poll() system call, supported on most Unix systems, provides better scalability for network servers that service many, many clients at the same time. poll() scales better because the system call only requires listing the file descriptors of interest, while select() builds a bitmap, turns on bits for the fds of interest, and then afterward the whole bitmap has to be linearly scanned again. select() is O(highest file descriptor), while poll() is O(number of file descriptors).

poll.register(fd[, eventmask])

Register a file descriptor with the polling object. Future calls to the poll() method will then check whether the file descriptor has any pending I/O events. fd can be either an integer, or an object with a fileno() method that returns an integer. File objects implement fileno(), so they can also be used as the argument.

eventmask is an optional bitmask describing the type of events you want to check for, and can be a combination of the constants POLLIN, POLLPRI, and POLLOUT, described in the table below. If not specified, the default value used will check for all 3 types of events.

Constant	Meaning
POLLIN	There is data to read
POLLPRI	There is urgent data to read
POLLOUT	Ready for output: writing will not block
POLLERR	Error condition of some sort
POLLHUP	Hung up
POLLRDHUP	Stream socket peer closed connection, or shut down writing half of connection
POLLNVAL	Invalid request: descriptor not open

Registering a file descriptor that's already registered is not an error, and has the same effect as registering the descriptor exactly once.

poll.modify(fd, eventmask)

Modifies an already registered fd. This has the same effect as register(fd, eventmask). Attempting to modify a file descriptor that was never registered causes an OSError exception with errno ENOENT to be raised.

poll.unregister(fd)

Remove a file descriptor being tracked by a polling object. Just like the register() method, fd can be an integer or an object with a fileno() method that returns an integer.

Attempting to remove a file descriptor that was never registered causes a KeyError exception to be raised.

poll.poll([timeout])

Polls the set of registered file descriptors, and returns a possibly-empty list containing (fd, event) 2-tuples for the descriptors that have events or errors to report. fd is the file descriptor, and event is a bitmask with bits set for the reported events for that descriptor — POLLIN for waiting input, POLLOUT to indicate that the descriptor can be written to, and so forth. An empty list indicates that the call timed out and no file descriptors had any events to report. If timeout is given, it specifies the length of time in milliseconds which the system will wait for events before returning. If timeout is omitted, negative, or None, the call will block until there is an event for this poll object.

Kqueue Objects

kqueue.close()

Close the control file descriptor of the kqueue object.

kqueue.closed

True if the kqueue object is closed.

kqueue.fileno()

Return the file descriptor number of the control fd.

kqueue.fromfd(fd)

Create a kqueue object from a given file descriptor.

kqueue.control(changelist, max_events[, timeout]) → eventlist

Low level interface to kevent

- changelist must be an iterable of kevent objects or None
- max_events must be 0 or a positive integer
- timeout in seconds (floats possible); the default is None, to wait forever

- Changed in version 3.5: The function is now retried with a recomputed timeout when interrupted by a signal, except if the signal handler raises an exception (see PEP 475 for the rationale), instead of raising InterruptedError.

Kevent Objects

kevent.ident

Value used to identify the event. The interpretation depends on the filter but it's usually the file descriptor. In the constructor ident can either be an int or an object with a fileno() method. kevent stores the integer internally.

kevent.filter

Name of the kernel filter.

Constant	Meaning
KQ_FILTER_READ	Takes a descriptor and returns whenever there is data available to read

KQ_FILTER_WRITE	Takes a descriptor and returns whenever there is data available to write
KQ_FILTER_AIO	AIO requests
KQ_FILTER_VNODE	Returns when one or more of the requested events watched in fflag occurs
KQ_FILTER_PROC	Watch for events on a process id
KQ_FILTER_NETDEV	Watch for events on a network device [not available on Mac OS X]
KQ_FILTER_SIGNAL	Returns whenever the watched signal is delivered to the process
KQ_FILTER_TIMER	Establishes an arbitrary timer

kevent.flags

Filter action.

Constant	Meaning
KQ_EV_ADD	Adds or modifies an event
KQ_EV_DELETE	Removes an event from the queue
KQ_EV_ENABLE	Permitscontrol() to returns the event
KQ_EV_DISABLE	Disablesevent
KQ_EV_ONESHOT	Removes event after first occurrence
KQ_EV_CLEAR	Reset the state after an event is retrieved
KQ_EV_SYSFLAGS	internal event
KQ_EV_FLAG1	internal event
KQ_EV_EOF	Filter specific EOF condition
KQ_EV_ERROR	See return values

kevent.fflags

Filter specific flags.

KQ_FILTER_READ and KQ_FILTER_WRITE filter flags:

Constant	Meaning
KQ_NOTE_LOWAT	low water mark of a socket buffer

KQ_FILTER_VNODE filter flags:

Constant	Meaning
KQ_NOTE_DELETE	unlink() was called
KQ_NOTE_WRITE	a write occurred
KQ_NOTE_EXTEND	the file was extended
KQ_NOTE_ATTRIB	an attribute was changed
KQ_NOTE_LINK	the link count has changed
KQ_NOTE_RENAME	the file was renamed
KQ_NOTE_REVOKE	access to the file was revoked

KQ_FILTER_PROC filter flags:

Constant	Meaning
KQ_NOTE_EXIT	the process has exited
KQ_NOTE_FORK	the process has called fork()
KQ_NOTE_EXEC	the process has executed a new process
KQ_NOTE_PCTRLMASK	internal filter flag
KQ_NOTE_PDATAMASK	internal filter flag
KQ_NOTE_TRACK	follow a process across fork()
KQ_NOTE_CHILD	returned on the child process for NOTE_TRACK
KQ_NOTE_TRACKERR	unable to attach to a child

KQ_FILTER_NETDEV filter flags (not available on Mac OS X):

Constant	Meaning
KQ_NOTE_LINKUP	link is up
KQ_NOTE_LINKDOWN	link is down
KQ_NOTE_LINKINV	link state is invalid

kevent.data

Filter specific data.

kevent.udata

User defined value.