QUESTIONS AND ANSWERS (from linux-kernel man page)

Q1 What happens if you add the same fd to an epoll_set twice?

A1 You will probably get EEXIST. However, it is possible that two hreads may add the same fd twice. This is a harmless condition.

Q2 Can two epoll sets wait for the same fd? If so, are events reported to both epoll sets fds?

A2 Yes. However, it is not recommended. Yes it would be reported to both.

Q3 Is the epoll fd itself poll/epoll/selectable?

A3 Yes.

Q4 What happens if the epoll fd is put into its own fd set?

A4 It will fail. However, you can add an epoll fd inside another epoll fd set.

Q5 Can I send the epoll fd over a unix-socket to another process?

A5 No.

Q6 Will the close of an fd cause it to be removed from all epoll sets automatically?

A6 Yes.

Q7 If more than one event comes in between epoll_wait(2) calls, are they combined or reported separately?

A7 They will be combined.

Q8 Does an operation on an fd affect the already collected but not yet reported events?

A8 You can do two operations on an existing fd. Remove would be meaningless for this case. Modify will re-read available I/O.

Q9 Do I need to continuously read/write an fd until EAGAIN when using the EPOLLET flag (Edge Triggered behaviour) ?

A9 No you don't. Receiving an event from epoll_wait(2) should suggest to you that such file descriptor is ready for the requested I/O operation. You have simply to consider it ready until you will receive the next EAGAIN. When and how you will use such file descriptor is entirely up to you. Also, the condition that the read/write I/O space is exhausted can be detected by checking the amount of data read/write from/to the target file descriptor. For example, if you call read(2) by asking to read a certain amount of data and read(2) returns a lower number of bytes, you can be sure to have exhausted the read I/O space for such file descriptor. Same is valid when writing using the write(2) function.

POSSIBLE PITFALLS AND WAYS TO AVOID THEM

- **Starvation (Edge Triggered)**

    - If there is a large amount of I/O space, it is possible that by trying to drain it the other files will not get processed causing starvation. This is not specific to epoll.

    - The solution is to maintain a ready list and mark the file descriptor as ready in its associated data structure, thereby allowing the application to remember which files need to be processed but still round robin amongst all the ready files. This also supports ignoring subsequent events you receive for fd's that are already ready.

- **If using an event cache**

- If you use an event cache or store all the fd's returned from epoll_wait(2), then make sure to provide a way to mark its closure dynamically (ie- caused by a previous event's processing). Suppose you receive 100 events from epoll_wait(2), and in eventi #47 a condition causes event #13 to be closed. If you remove the structure and close() the fd for event #13, then your event cache might still say there are events waiting for that fd causing confusion.

- One solution for this is to call, during the processing of event 47, epoll_ctl(EPOLL_CTL_DEL) to delete fd 13 and close(), then mark its associated data structure as removed and link it to a cleanup list.

- If you find another event for fd 13 in your batch processing, you will discover the fd had been previously removed and there will be no confusion.