Instructions to run mTCP based applications

Following are the generic instructions for running example programs provided in this repository. The **'src'** directory contains parent directories of the code of mTCP integrated with our kernel bypass API. Within these parent directories, is the **testing** directory which contains our simple example applications. Following are the steps to run any one of this application.

1. Look into **Makefile** to ensure all the include paths are consistent with your system.
2. Run **'make clean'** and **'make'** in the parent directory to generate 'mtcp_lib.o'. We suggest running 'make clean' every time before you run 'make' , as dependencies are not properly verified within makefile.
3. Look into **compile.sh** to ensure all the include paths are consistent with your system.
4. Run **cd** to go into **testing/<application_directoy>.**
5. Run compile.sh to compile the application and link it together with mtcp library generated in parent directory.
6. Then run applications with sudo permissions.

About Applications

There are two notable things related to these application: they are mTCP based and they follow eventpoll paradigm. In this document, we explain the basic mTCP API and various configuration parameters involved used for these applications. While most of this information is also available at [2], the intention of this document is to accumulate pieces of information important to users of this repository. **Note that this manual is heavily based on [1] and [2] , and our experimentation with mTCP.**

mTCP API

While mTCP provides an exhaustive API [2], we explain the bare minimum of that in this section.

- **mtcp_init(char * conf_file) :** This function initializes configuration parameters of mTCP such as send/receive buffer size, interfaces to use and so on based on information provided in conf_file.
- **mtcp_core_affinitize(int core):** This function will affinitize this particular application thread to CPU core specified by core.

- **mtcp_create_context(int core):** This function will spawn mTCP thread for this application thread, which in turn initialize mTCP internal data structures based on parameters set in mtcp_init(). It returns mTCP context object, which is used in all the operations henceforth, to identify core at which core these operations are being performed. This is the consequence of the per-core design paradigm of mTCP.
- **mtcp_epoll_create(mctx_t mctx, MAX_EVENTS):** This returns identifier which can be used afterwards to invoke the eventpoll system of the mTCP.
- **mtcp_socket, mtcp_bind, mtcp_listen, mtcp_connect, mtcp_accept, mtcp_read, mtcp_write:** All these function take all parameters as their counterparts in the linux based API, along with one more parameter, i.e., mctx (mTCP context which was created by mtcp_create_context).
- **mtcp_epoll_ctl(), mtcp_epoll_wait() :** Similar to kernel eventpoll system, along with one extra parameter, i.e., mctx (mTCP context which was created by mtcp_create_context).

Order of invocation

Order of invocation at the mTCP initialization is important. And after mtcp is set up, then any of the API functions can be invoked. Order of initialization is as follows:

1. mtcp_init
2. mtcp_core_affinitize
3. mtcp_create_context

Configuration parameters and their significance

Following are configuration parameters:

- **io:** Values can be netmap/dpdk. This specifies which kernel bypass API to use.
- **num_cores:** Number of cores mTCP application can use.
- **port:** Interface which would be used by kernel bypass mechanisms. Values differ for both Netmap and DPDK. Look into conf files in respective folders for details.
- **max_concurrency:** Number of maximum concurrent sockets that could be opened on a core. Various data-structures are pre-allocated based on this value. Hence exercise caution here.
- **max_num_buffers:** Number of socket buffers preallocated per-core.
- **rcvbuf:** Size of TCP socket receive buffer.
- **sndbuf:** Size of TCP socket send buffer.

There are some other option also which were not used by us during experimentation. Look into mTCP's original documentation for description about them.

References

1. Eun Young Jeong, Shinae Woo, Muhammad Jamshed, Haewon Jeong, Sunghwan Ihm, Dongsu Han, and KyoungSoo Park. **mtcp: A highly scalable user-level tcp stack for multicore systems.** In Proc. of NSDI'14, 2014.
2. Link to mTCP repository: https://github.com/eunyoung14/mtcp