COMP4621 Lab 7

Name: CHAN, Chun Hin
Student ID: 20853893
Email: chchanec@connect.ust.hk

1. Describe the components of the structure “pollfd”. What’s the difference between “events” and “revents”?

Inside the structure “pollfd”, there are three components.
The first component is **fd** of int data type, which is the socket descriptor being polled.
The second component is **events** of short data type, which is a bitmask which specifies what events to monitor given a file descriptor.
The third descriptor is **revents** of short data type, which shows a bitmap of what events are actually occurred when poll() return.

The difference between “events” and “revents” is that “events” specifies the bitmap of events that we are interested in, while “revents” shows the bitmap of events that are actually occurred, and it is set by the kernel.

2. In the select() and poll(), how can we know which socket is ready for reading?

For select(), we can know whether a socket is ready for reading by using FD_ZERO() and FD_SET() and FD_ISSET(). We firstly use FD_ZERO() to initialize the **readfds** bit set of descriptors to check for reading. Then, we can call FD_SET() for each socker in **readfds** to make the socket be tested. Note that only the sockets that are ready for reading would still remain inside **readfds** after **readfds** is modified in place. Thus, next, we can call FD_ISSET() to check whether a socket member in **readfds** is being set in the ready for reading state.

For poll(), we can know which socket is ready for reading by using the **revents** data members in **pollfd** structure. Firstly, we should call poll() function. Then, we can use for loops or while loop to loop through the **pollfd** structure array. Next, we can check the **revents** data member inside each **pollfd** structure. If the **revents** data member has POLLIN bit set, then this means the socket is ready for reading.

3. To enable the concurrency, what is the main difference between fork() and select()/poll()?

For fork(), it creates one new child process by simply duplicating the parent process. Note that the child process contains the same copy of information of the parent process, such as file descriptors. Note that the PID of the child process is different from that of the parent process. After the fork() function is successfully executed, the child process would start to execute as the same point the parent create itself. It is worth noticing that a child process is independent of the parent process. Hence, using fork() allows multiple processes to execute different tasks at the same time to enable concurrency.

For select()/poll(), they are used to monitor several sockets in just one single process. To enable concurrency, instead of creating new process, select()/poll() allow to monitor whether different sockets are ready to be read and to be concurrent using the built-in data members and structure as discussed in this lab. This hence allows concurrency in just one process, and avoid blocking in I/O. (Note: It is worth noticing that if we are using select()/poll(), as we are just using one single process, and the concurrency in this case is driven be events, this means that the server can only handle only one request at one time when an event is triggered.)

4. Practical Exercise: Implement RDT 3.0 on top of UDP client server. (Hint: this will be needed in the programming project later)

Please see the codes.