## Лекция 3. Технология клиент-сервер. Практические примеры

**Примитивный пример.**

Одна "станция" отправляет сообщение. Другая его принимает и отправляет обратно добавив в конце слово Echo. Первая станция распечатывает этот ответ. По ходу дела станции также печатают на экран всякую служебную информацию.

Пример для протокола UDP — cli0.cpp ser0.cpp.

Пример для протокола TCP — cli.cpp ser.cpp.

**Асинхронность.**

В данном примере есть сушественный недостаток — он абсолютно не учитывает асинхронность взаимодействия, т.е. тот опирается на то, что оба абонента будут отправлять свои сообщения в те моменты времени, когда другой абонент будет готов их принять и будет фактически ждать прихода сообщения. В реальной практике так модет не быть, поскольку соообщения могут прийти с задержкой или вообще не прийти, и тогда принимающая сторона будет бесконечно долго и безуспешно их ждать.

Для решения этого вопроса принимающая сторона должна иметь возможность ожидать ответа некоторое время, после чего принимать решение нужно ли продолжать это ожидание, или стоит закончить работу (или произвести новый запрос). Подобный механизм обычно называется таймером, устанавливающим определенный интервал времени на выполнение некоторых действий (или бездействий как при ожидании). Таймер модно реализовать на базе раличных системных вызовов. Здесь мы рассмотрим простейший частный случай таймера на ожидание ввода данных. Для этого используются два системных механизма

— отмена блокировки на вводе,

— блокирование (засыпание) процесса на заданный период времени.

Для отмены блокировки на вводе можно использовать системный вызов fcntl (установка режимов для канала обмена).

```
fcntl(fd, F_SETFL, O_NONBLOCK);
```

Если по умолчанию операция чтения данных из канала fd блокировалась до появляния возможности эти данные прочитать, то теперь каналу установлен флаг отмены блокировки, и вызов функции чтения завершится сразу, либо прочитав имеющиеся к моменту вызова данные, либо с указанием, что никаких данных не было прочитано. Теперь можно в цикле повторять попытки чтения до тех пор пока либо не придут данные для чтения, либо не истечет необходимое время. А вот необходимый интервал время можно отмерять, периодически "засыпая" на некоторое время.

Для засыпания в Linux можно использовать две функции

```
sleep (int num_seconds);
usleep (int num_micro_seconds);
```

При вызове этих функции процесс выключается на указанное время и не занимает системные ресурсы (процессор и т.п.), а потом возобнавляется и продолжает работу с кода, следующего за данным вызовом.

**Менее примитивный пример.**

Станция занимается рассылкой сообщений всем, кто у нее зарегистрируется. Сообщеним является информация о текущем времени. Эти сообщения возникают в случайные моменты времени. Регистриция получателей выполняется отсылкой запроса

на эту станцию со стороны клиента, который хочет эти сообщения получать. Будем для простоты считать, что таких клиентов может быть не более 8 и они уже не выходят из обслуживания до конца работы (станция будет просто сохранять их адреса в массиве по простейшей схеме).

В этой схеме у нас есть два источника асинхронных событий — запрос на обслуживание от клиента и сообщение о времени от “сервера”. Сообщения и запросы возникают в случайные моменты времени, и обе стороны взаимодействия не могут предугадать когда это произойдет. Основой реализации сервера является бесконечный цикл, в котором сервер проверяет наличие запросов от клиентов на регистрацию (и соответственно регистрирует их), а в некоторые случайные моменты этого цикла рассылает всем зарегистрированным клиентам сообщение — текущую дату и время.

Мы модифицируем предыдущий вариант UDP взаимодействия.

Пример для протокола UDP — cli2.cpp ser2.cpp.