Sieci, sockety, protokoły, procesy

Bartosz Szreder

Po co?

- Komunikacja pomiędzy procesami na różnych maszynach.
- Komunikacja pomiędzy procesami na jednej maszynie.

Nic nie stoi na przeszkodzie, żeby pisać programy komunikujące się po socketach nielokalnych (o rozróżnieniu potem), działające na jednym komputerze. Można w ten sposób wytestować kod sieciowy w kontrolowanym środowisku.

Nie należy jednak robić w ten sposób testów wydajnościowych czy odpornościowych.

Jak?

- Synchronicznie czy asynchronicznie?
- Oszczędnie czy reduntantnie?
- **Jak zaprojektować protokół?**

Czym?

- Protokół transportowy: TCP? UDP? Może jeszcze jakiś inny? [trochę ich jest...]
- Kiedy jaki i dlaczego?

UDP

- Minimalna nakładka na pakiety IP.
- Bezpołączeniowy.
- Brak pewności dostarczenia, brak gwarancji kolejności.
- Komunikacja w oparciu o datagramy.

TCP

- Połączeniowy.
- Strumieniowy.
- Gwarancja dostarczenia i kolejności.
- Dość sporo logiki dołożonej ponad IP.

- TCP wygląda świetnie.
- . . . nie zawsze go chcemy.
- *Latency*. Wiele gier FPS używa UDP i predykcji ruchu.
- Gry, gdzie nie jest wymagany znaczący pośpiech, a kluczowa jest informacja (strategie, niektóre MMORPG) preferują TCP.
- Na UDP możemy sobie zaimplementować własny mechanizm „odporności”.

Local sockets

Do komunikacji lokalnej możemy używać tzw. lokalnych gniazdek (`man 7 unix`).

Na poziomie systemu gniazdko to zwykły *file descriptor*:

```
int socket_fd = socket(AF_INET, SOCK_STREAM, 0);
bind(socket_fd, ...);
listen(socket_fd, 10); /* backlog */
int connection_fd = accept(socket_fd, ...);
...
close(connection_fd);
```

`man 2 socket`, `man 2 listen`, `man 2 accept`, `man 2 connect`, `man 2 send`, `man 2 recv`

Od biedy zamiast `recv` i `send` można `read` i `write`.

(A)synchroniczność

Trzeba uważać, żeby nie zawiesić procesu/wątku w oczekiwaniu na komunikację sieciową.

- Flaga `O_NONBLOCK` na deskryptorze, flaga `MSG_DONTWAIT` w `recv`.
- (De?)multipleksing: `man 2 select`, ew. `man 2 poll`, `man 7 epoll`.

. . . albo wywalić komunikację do innego wątku. Wtedy oczywiście trzeba zadbać o przepchanie danych do wątku głównego.

Pewnie już myśleliście, że nie powiem nic o Qt. ☺

- `QUdpSocket`, `QTcpSocket`, `QSslSocket`, `QTcpServer`.
- Mogą działać zarówno synchronicznie (metody `waitFor*()`) jak i asynchronicznie (sygnały).
- Dokumentacja ma świetne przykłady (np. pełny klient BitTorrent, ale też dużo drobniejsze programy), więc podaruję sobie wklejanie.
- Łatwo połączyć przyjemne z pożytecznym: wywalamy komunikację do oddzielnego wątku (`QThread`), robimy ją synchronicznie i przepychamy dane do wątku głównego sygnałami.