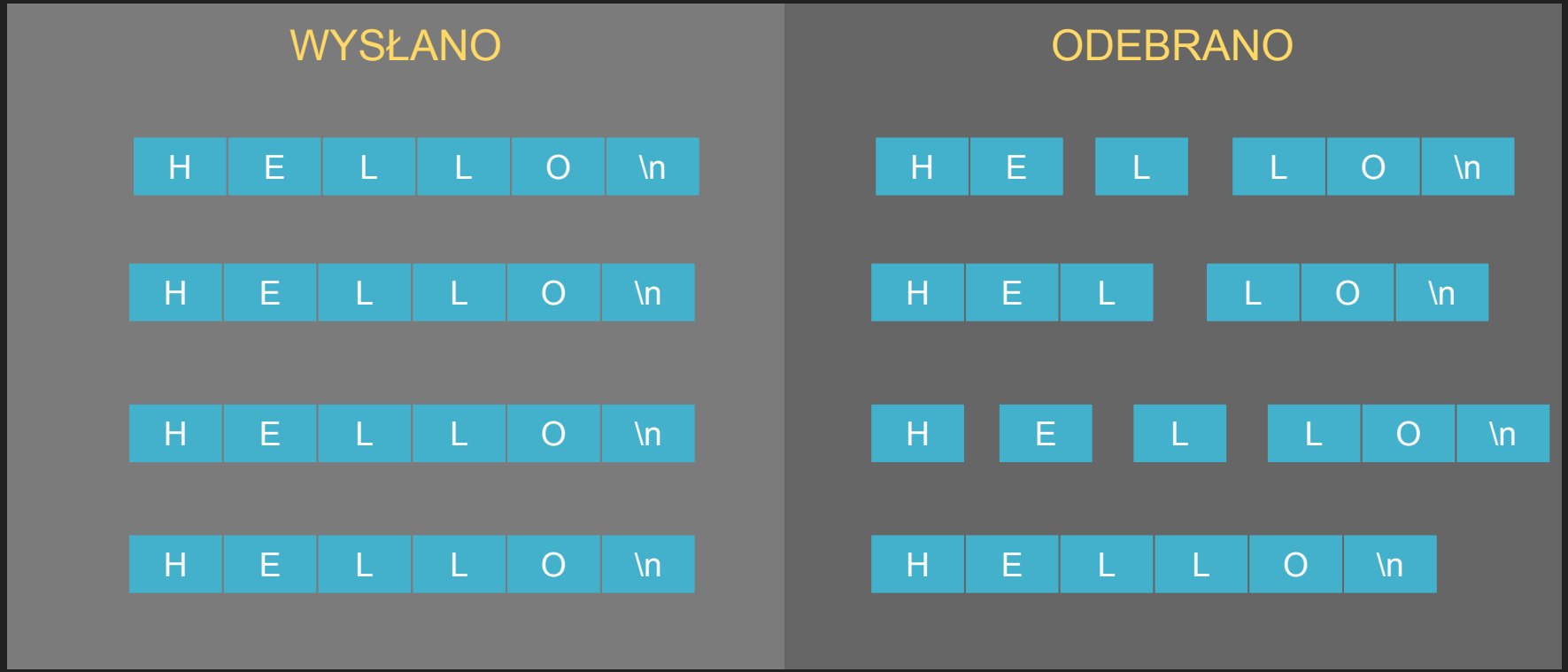
BSD sockets c.d.

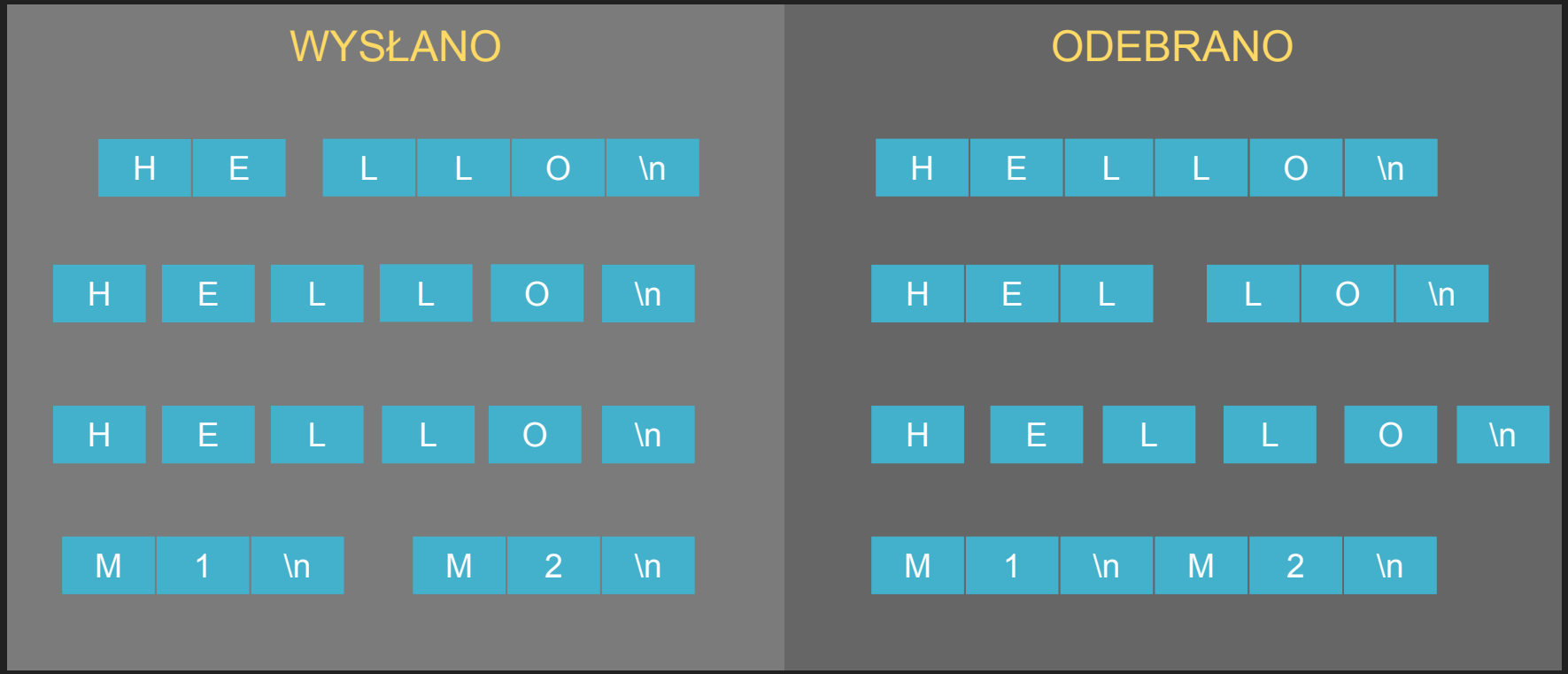
TCP - receive buffer (queue), send buffer (queue)

- Z każdym gniazdem sieciowym są skojarzone:
  - Bufor do odbierania danych (ang. receive buffer)
    - Przychodzące dane są umieszczane w buforze
    - Dane są przekazywane z buforu do aplikacji (zwrócenie kontroli z read/recv)
  - Bufor do zapisu (ang. send buffer)
    - Wysyłane dane są zapisywane do buforu
    - Dane z buforu są wysyłane przez sieć
- Rozmiary buforów zależą od ustawień systemu operacyjnego
  - Systemy wbudowane
- Możliwe jest zapełnienie buforów
- Częściowy odczyt - gdy w buforze do odczytu jest mniej danych niż oczekiwano
- Częściowy zapis - gdy w buforze do zapisu brakuje miejsca

TCP - fragmentacja



TCP - sklejanie



I/O multiplexing: select/poll/epoll

- Oczekiwanie na zdarzenia powiązane z deskryptorami plików
- select
  - przyjmuje zbiór deskryptorów,
  - powoduje modyfikację przekazanego zbioru (wynik),
  - raczej przestarzałe rozwiązanie, ale wspierane prawie wszędzie
  - maks. około 1000 deskryptorów
- poll
  - przyjmuje zbiór deskryptorów i interesujących zdarzeń,
  - nie modyfikuje zbioru
  - bez twardego limitu na liczbę deskryptorów

I/O multiplexing: select/poll/epoll

- epoll
  - wyłącznie dla systemu Linux,
  - inny mechanizm: komunikacja z jądrem systemu,
  - zwraca wyłącznie “aktywne” deskryptory (przy select/poll trzeba iterować po wszystkich deskryptorach i sprawdzać czy są aktywne),
  - bardziej skomplikowany w użyciu

select

```
fd_set fd_in, fd_out;
struct timeval tv;
FD_ZERO( &fd_in ); FD_ZERO( &fd_out );
FD_SET( sock1, &fd_in ); FD_SET( sock2, &fd_out );

int largest_sock = sock1 > sock2 ? sock1 : sock2;
tv.tv_sec = 10; tv.tv_usec = 0;

int ret = select( largest_sock + 1, &fd_in, &fd_out, NULL, &tv );
if ( ret == -1 )
else if ( ret == 0 )
else {
    if ( FD_ISSET( sock1, &fd_in ) )
        // input event on sock1

    if ( FD_ISSET( sock2, &fd_out ) )
        // output event on sock2
}
```

Przykład:
http://www.cs.put.poznan.pl/mkalewski/documents/select.php

poll

```
struct pollfd fds[2];
fds[0].fd = sock1;
fds[0].events = POLLIN;
fds[1].fd = sock2;
fds[1].events = POLLOUT;

int ret = poll( fds, 2, 10000 );
if ( ret == -1 )
else if ( ret == 0 )
else {
    if ( fds[0].revents & POLLIN )
        fds[0].revents = 0;
        // input event on sock1
    if ( fds[1].revents & POLLOUT )
        fds[1].revents = 0;
        // output event on sock2
}
```

Przykład:
https://beej.us/guide/bgnet/html/#poll

getaddrinfo() - użycie nazwy domenowej

int getaddrinfo(const char *node, const char *service, const struct addrinfo *hints, struct addrinfo **res);

node - nazwa domenowa; np. “wp.pl”
service - numer portu lub nazwa usługi; np. “http”, “80”
hints - wskazówki, filtry wyników; np. AF_INET (tylko IPv4), SOCK_STREAM (TCP)
res - wyniki w postaci linked list

Zwraca: 0 (ok) lub kod błędu

Przykład: http://www.cs.put.poznan.pl/mboron/templates/nameToIP.c

netstat

- Pokazuje m.in. aktywne połączenia (czy działa instancja serwera)
- netstat --tcp --listening -c

telnet

- zdalny dostęp do systemu
- telnet [nazwa serwera] [numer portu]
- może być użyty do debugowania aplikacji sieciowych przesyłających dane w formie tekstowej
  - np. połącz się z serwerem chatu z ostatnich zajęć przez telnet

Ciekawostka: wykonaj telnet towel.blinkenlights.nl 23

nc (netcat)

- arbitralne połączenia i nasłuchiwania TCP i UDP
- nc [nazwa serwera] [numer portu]
  - połączenie
- nc -l [numer portu]
  - nasłuchiwanie
- może być użyty do debugowania aplikacji sieciowych

Zadanie 1

Stwórz serwer TCP, który w pętli akceptuje nowe połączenia i je ignoruje (tzn. nie odbiera ani nie wysyła danych, nie zamyka połączeń). Stwórz klienta TCP, który w pętli: wysyła dane, wypisuje kolejny numer, wypisuje ilość wysłanych danych.

Obserwuj komunikaty wypisywane przez klienta. Obserwuj stan zapełnienia buforów przez netstat -tn (Recv-Q, Send-Q).

Zadanie 2

Napisz serwer TCP który po odebraniu połączenia, do jego zamknięcia, wykonuje w pętli: odbiera do 10 bajtów, wypisuje ile bajtów odebrał i co odebrał.

Napisz klienta TCP który wysyła cały alfabet po jednej literze. Podłącz się (wielokrotnie) do serwera.

Zadanie 3

Napisz serwer TCP który po odebraniu połączenia, do jego zamknięcia, wykonuje w pętli: odbiera do 10 bajtów, wypisuje ile bajtów odebrał i co odebrał.

Napisz klienta TCP który wysyła wielokrotnie po kilkanaście liter. Podłącz się (wielokrotnie) do serwera.

Literatura

- https://www.ulduzsoft.com/2014/01/select-poll-epoll-practical-difference-for-system-architects/
- Kerrisk rozdział 63
  - select/poll/epoll
- Kerrisk rozdział 61
  - partial r/w, netstat, tcpdump, UDP vs. TCP,
- https://beej.us/guide/bgnet/html/