## Zadanie 1 – serwer sieciowej usługi „czat”(10 pkt.)

Składnia programu:

```
czatserver [opcje]
```

Opcje:

`-p, --port=port_number` – opcja wymagana, argumentem jest port tcp na którym słucha serwer

`-i, --interface=ip_address` – opcja, w której można podać adres IP interfejsu, na którym ma nasłuchiwać serwer (domyślnie: wszystkie interfejsy)

Obsługa opcji powinna być zgodna ze standardem POSIX (proponowane użycie biblioteki getopt), w szczególności opcje można podawać w dowolnej kolejności.

Przykładowe wywołania:

```
czatserver –p 10001
czatserver –-port=10001
czatserver –i 127.0.0.1 –-port=10001
czatserver –-port 10001 –-interface=127.0.0.1
```

Po uruchomieniu program nasłuchuje na wybranym porcie (tcp) na zgłoszenia klientów. Program musi być w stanie obsługiwać jednocześnie dowolną liczbę klientów (ograniczeniem będą tylko zasoby systemu operacyjnego).
Przykładowe działanie programu:

1. Z klientem telnet
   - Po nawiązaniu komunikacji z klientem (za pomocą polecenia `telnet`) serwer rozpocznie jego obsługę wyświetlając prompt „>” i reagując na trzy typy żądań:
     - polecenie *get* - w wyniku, którego zostanie wyświetlonych ostatnie 10 komunikatów jakie zostały przesłane przez dowolnego klienta do serwera,
     - polecenie *put* „przykładowy tekst” – w wyniku, którego klient przesyła jako argument polecenia dowolny komunikat tekstowy do serwera, komunikat ten powinien być zapisany w pamięci współdzielonej, a następnie może być odczytany przez dowolnego klienta,
     - polecenie *quit* - zakończenie połączenia z danym klientem.

     w przypadku podania nieobsługiwanego polecenia program powinien wyświetlić stosowny komunikat o błędzie.
   - po wykonaniu polecenia program powinien ponownie wyświetlić prompt „>” i zaczekać na kolejne polecenie.
2. Z własnym klientem: odbieranie i wyświetlanie komunikatów od innych odbywa się na bieżąco

Proces obsługujący połączenie z klientem powinien przychwytywać sygnały TERM i INT: powinien najpierw wysłać komunikat do klienta informujący o przerwaniu pracy, po czym poprawnie zakończyć połączenie.

Należy zabezpieczyć się przed zakończeniem procesu w trakcie odbierania i wysyłania danych z pamięci współdzielonej, która powinna służyć, jako miejsce przechowywania komunikatów od klientów (jeżeli klient zażądał danych, wysłał *get*, proces serwera obsługujący połączenie może zakończyć się dopiero po wysłaniu odpowiedzi klientowi).

W przypadku odebrania sygnałów TERM i INT przez główny proces serwera, serwer powinien rozesłać ten sygnał do wszystkich swoich procesów potomnych i zaczekać na ich zakończenie.

W przypadku błędów w wywołaniu funkcji bibliotecznych należy zareagować w odpowiedni sposób.

Ocenie podlega:

- zaimplementowanie wymaganej funkcjonalności
- obsługa błędów
- przejrzystość i formatowanie kodu
- sprzątanie (na bieżąco) pozostałości po zakończonych procesach potomnych