Laboratorium 5B (7pkt)

Przygotowany jest szkielet programu wraz z funkcją main. W funkcji main zadeklarowane jest przykładowe użycie funkcji, które mają zostać zaimplementowane w kolejnych etapach. Program ma na celu stworzenie systemu obsługi wiadomości przesyłanych jako ciąg znaków. Wiadomości są oznaczane znacznikami MESSAGE_START i MESSAGE_END (np. MESSAGE_START = "|MESSAGE_START|")

System, inspirowany komunikatorami opartymi na protokole UDP, wyodrębnia wiadomości z napływającego strumienia danych. Ponieważ wiadomości mogą przychodzić w częściach lub zawierać zakłócenia, dane są najpierw gromadzone w buforze. Następnie system analizuje bufor, aby odnaleźć i przetworzyć wszystkie kompletne wiadomości. Elementy związane z przesyłaniem danych przez sieć zostały pominięte.

Słowniczek:
Skrótowy opis funkcji, które mogą się przydać na dzisiejszych laboratoriach.

1. Funkcja **memset** wypełnia blok pamięci określoną wartością. Wywołanie: memset(pointer, value, size). Pointer to początek obszaru pamięci, value to wartość, którą ustawiamy, a size to liczba bajtów.
2. Funkcja **memcpy** kopiuje dane z jednego miejsca w pamięci do drugiego. Wywołanie: memcpy(destination, source, size). Destination to miejsce docelowe, source to dane do skopiowania, a size to liczba bajtów. Przykład: memcpy(manager->buffer + manager->buffer_len, data, data_len).
3. Funkcja **strncpy_s** kopiuje określoną liczbę znaków z jednego ciągu do drugiego, zapewniając bezpieczeństwo przed przepełnieniem bufora. Wywołanie: strncpy_s(destination, destsz, source, count). destination to miejsce docelowe, destsz to rozmiar bufora docelowego, source to źródło, a count to liczba znaków do skopiowania. Gwarantuje, że ciąg docelowy będzie null-terminowany.
4. Funkcja **strlen** oblicza długość ciągu znaków (liczbę znaków przed pierwszym znakiem null-terminującym). Wywołanie: strlen(str). Zwraca liczbę znaków w ciągu str, nie wliczając znaku null-terminującego.
5. Funkcja **strstr** wyszukuje pierwsze wystąpienie podłańcucha w ciągu znaków. Wywołanie: strstr(text, subtext). text to ciąg, w którym szukamy, a subtext to podłańcuch, który chcemy znaleźć. Zwraca wskaźnik do pierwszego wystąpienia podłańcucha lub NULL, jeśli nie znaleziono.
6. Funkcja **_countof** oblicza liczbę elementów w statycznej tablicy. Wywołanie: _countof(array). array to statyczna tablica, której liczba elementów jest obliczana. Funkcja jest przydatna do określenia limitów operacji na tablicach, takich jak kopiowanie danych, aby uniknąć przepełnienia bufora.
7. Funkcja **snprintf** pozwala na zapisanie sformatowanego tekstu, zamiast wyświetlania go na konsoli, do zmiennej. Na przykład, zamiast używać `printf` do wypisania tekstu, możesz zapisać go do bufora, co pozwala na dalsze przetwarzanie lub przekazywanie tego tekstu w programie.

Etap 1 (1pkt)
Należy zaimplementować trzy następujące funkcję:

1. init_message_manager(MessageManager* manager, int initial_capacity)
   Funkcja inicjalizuje strukturę MessageManager, ustawiając początkową pojemność tablicy wiadomości, zerując bufor i ustawiając wartości początkowe dla liczby wiadomości (count) oraz długości bufora (buffer_len).
2. free_message_manager(MessageManager* manager)
   Funkcja zwalnia pamięć zaalokowaną dla tablicy wiadomości w strukturze MessageManager.
3. print_all_messages(MessageManager* manager)
   Funkcja wypisuje wszystkie przechowywane wiadomości, wyświetlając ich nagłówki oraz treści w czytelnej formie.

Etap 2 (1 pkt)
Należy zaimplementować funkcję, która dodaje nowe dane do bufora

Laboratorium 5B (7pkt)

“append_to_buffer(MessageManager* manager, const char* data, size_t data_len)”
Funkcja powinna najpierw sprawdzać przepełnienie bufora, jeśli dodanie nowych danych przekroczy jego rozmiar, należy wyświetlić komunikat o błędzie i wyzerować długość bufora (buffer_len).
Jeśli bufor nie jest przepełniony nalezy dodać otrzymaną wiadomość na koniec bufora. Pamiętaj aby dodać znak końca ciągu ('\0') na końcu bufora, aby zapewnić, że bufor zawsze będzie poprawnym ciągiem znaków.

Etap 3 (1 pkt)

Należy zaimplementować funkcję, która resetuje bufor w strukturze MessageManager. Funkcja powinna ustawić długość bufora (buffer_len) na 0, a następnie wyczyścić zawartość bufora, wypełniając go zerami przy użyciu funkcji memset „clear_buffer”.

Etap 4 (3 pkt)

Należy zaimplmenentować dwie funkcję, pierwsza ma za zadanie wyodrębnić wiadomość z buffora, a druga dodać ją do struktury MessageManager.

1. Funkcja: process_buffer przechodzi po bufferze i znajduje wszystkie wiadomości. W buforze może znajdować się więcej niż jedna wiadomość, a na końcu może wystąpić nie dokończona wiadomość, jeśli brakuje znaku końca wiadomości. Funkcja process_buffer przeszukuje bufor, aby znaleźć wszystkie pełne wiadomości, a wyodrębnione wiadomości przekazuje do funkcji add_message. Ważne jest, aby po wyodrębnieniu wiadomości bufor nie był modyfikowany – pozostałe części bufora nie są usuwane.
2. Funkcja add_message tworzy nową wiadomość, przypisuje jej całą wyodrębnioną treść jako "content" i dodaje ją do struktury MessageManager. W tej części nie aktualizujemy nagłówka wiadomości „header”. Po dodaniu wiadomości wypisz o tym informację do konsoli przykładowo: „printf("New message added %s \n", new_message.content);”.

Etap 5 (1 pkt)

Zaimplementuj funkcję process_all_messages, która przetwarza wszystkie wiadomości w strukturze MessageManager. Funkcja powinna ustawiać pole header każdej wiadomości na wartość `"Receiver: <IP>"`, gdzie `<IP>` to przekazany jako argument adres IP. Użyj `snprintf` do stworzenia nagłówka i `strncpy_s` do bezpiecznego kopiowania danych, zapewniając null-terminację i brak przepełnienia bufora. Iteruj po wszystkich wiadomościach, wykorzystując pole `count` w strukturze `MessageManager`.