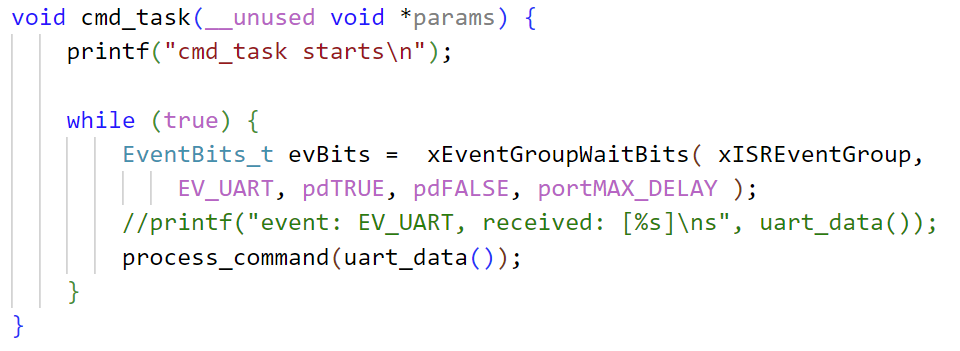
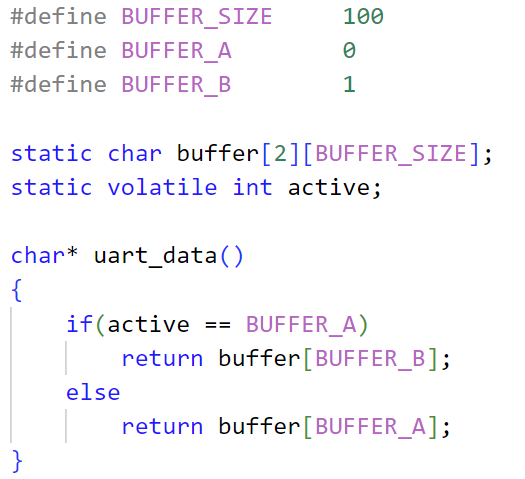
# SWA-Lab5. Zadania

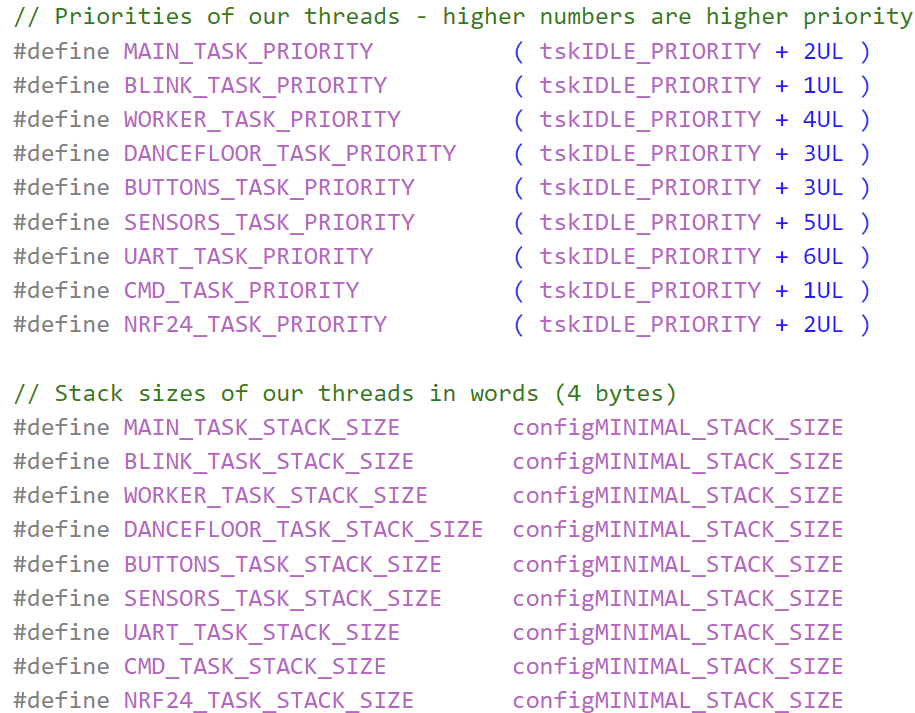
Przygotowałem demonstracyjny projekt freertos\_demo.zip - gotowy do rozpakowania i użycia. Zawiera już SDK i FreeRTOS, przez to trochę waży, ale może dzięki temu łatwiej będzie zmusić go do działania.

Jest to realizacja zadania z wcześniejszych zajęć RTOS - miganie diodą 0/1/2 po wybraniu przycisku 0/1/2. Przykład demonstruje praktyczne użycie grup zdarzeń wraz z obsługą przerwania.

1. Pobierz archiwum, rozpakuj i spróbuj skompilować oraz uruchomić projekt.
2. Dodaj obsługę UART - użyj swojego wcześniejszego kodu, który zbiera odebrane znaki w buforze. Będzie to osobne zadanie z przerwaniem UART - analogicznie jak to, które obsługuje przyciski.   
   Hint-1: Znaki możesz gromadzić w buforze (tablicy), ale sprytniej będzie zdefiniować dwa bufory (A, B) - w danej chwili jeden gromadzi znaki odbierane przez UART, a drugi w tym czasie może być przetwarzany przez dekoder komend.   
   Hint-2: Jeszcze lepiej byłoby przekształcić to w kolejkę o ustalonej pojemności, do której zadanie task\_uart wrzuca odebrane linie tekstu, a task\_cmd pobiera, dekoduje i wykonuje (bufory A/B są uproszczoną wersją kolejki 2-elementowej)  
   Hint-3: Zastosuj sekcję krytyczną - mamy do czynienia z systemem wielozadaniowym, więc operacje na kolejce muszą być zabezpieczone. Podobnie gdy masz bufory A/B - zadanie task\_uart powinno być zabezpieczone przed użyciem bufora, którego nie zdążyło obsłużyć zadanie task\_cmd.
3. Do istniejących zdarzeń “guzikowych” dodaj zdarzenie związane z odebraniem komendy przez UART - zdarzenie ma być ustawiane, gdy pojawi się znak 13 (\r, CR) - czyli inaczej enter/return. Dawniej w tym miejscu była wywoływana funkcja dekodująca i wykonująca komendę, ale to nie było dobre rozwiązanie - nadal jesteśmy w kontekście przerwania UART!
4. Do projektu dodaj pliki .c, .h odpowiedzialne za dekoder komend (z wcześniejszych zajęć). W tym celu zdefiniuj osobne zadanie, którego funkcja będzie wyglądać tak:  
   Zwróć uwagę na brak sleep - nie musi go być, funkcja xEvent… grzecznie czeka na pojawienie się zdarzenie nie obciążając procesora.  
     
   Funkcja uart\_data() zwraca wskaźnik do danych z kolejki (lub z bufora A/B) gotowych do przetworzenia.  
     
   Przykładowa implementacja przełączanych buforów A/B (uwaga! bez sekcji krytycznej!)  
   Słowo kluczowe static w modułach C ogranicza widoczność zmiennych globalnych, jest to dobra praktyka znana z programowania obiektowego jako hermetyzacja (sekcje private, protected znane z C++). Jednocześnie funkcja uart\_data() stała się odpowiednikiem akcesora lub gettera - na zewnątrz modułu nie ma dostępu do zmiennej buffer czy active, jest tylko uart\_data() zwracająca gotowy do użycia zestaw znaków. Mogę kiedyś przerobić kod z przełączanych buforów A/B na kolejkę, a uart\_data pozostawić bez zmian.  
     
   

W wersji dojrzalszej projektu zadań będzie zapewne więcej:





Proponowana struktura plików projektu (chociaż można już pomyśleć o zgrupowaniu plików źródłowych zadań w podkatalogu tasks):

