Struktura jąder systemów operacyjnych

Lista zadań nr 2

Na zajęcia 18 i 25 marca 2020

UWAGA! W trakcie prezentacji rozwiązań należy zdefiniować i wyjaśnić pojęcia, które zostały oznaczone **wytłuszczoną** czcionką.

W zadaniach 3–8 będziecie prezentować kod FreeRTOS z użyciem serwisu OpenGrok[1]. Przed zajęciami należy starannie przygotować notatki, które pozwolą sprawnie przejść przez zagadnienia z treści zadania. Za słabe przygotowanie prowadzący nie przydzieli pełnej liczby punktów.

Zadanie 1. Na podstawie strony „Memory Management[2]" wyjaśnij ogranicznia poszczególnych manadżerów pamięci sterty systemu FreeRTOS. Czemu autorzy dokumentacji odnoszą się z rezerwą do dynamicznego zarządzania pamięci w systemach czasu rzeczywistego? Podaj scenariusze użycia dynamicznego przydziału pamięci uzasadniające wybór dostępny wybór algorytmów zaimplementowanych w plikach «heap_?.c».

Zadanie 2. Dlaczego należy unikać funkcji rekurencyjnych programując z użyciem systemu FreeRTOS? Na podstawie strony „How big should the stack be?[3]" wyjaśnij jak wyznaczyć górne ograniczenie na rozmiar stosu. Czemu należy brać pod uwagę **procedury obsługi przerwań**? Na podstawie „Stack Usage and Stack Overflow Checking[4]" wyjaśnij strategie wykrywania **przepełnienia stosu** w systemach bez ochrony pamięci.

Zadanie 3 (P). Zadaniem struktury pcpu[5] (ang. *per-cpu*) w jądrze FreeBSD jest utrzymywanie globalnego stanu jądra dla każdego z procesorów. Do analogicznej struktury w jądrze NetBSD można uzyskać dostęp wywołując procedurę curcpu(9). W pliku tasks.c[6] zdefiniowano globalny stan jądra FreeRTOS. Opisz znaczenie pól «pxCurrentTCB», «pxReadyTasksLists», «xTickCount», a następnie postaraj się odnaleźć ich odpowiedniki w strukturze «pcpu».

Czemu wątki, które zakończyły swoje działanie wywołując procedurę «vTaskDelete(NULL)», są umieszczane na liście «xTasksWaitingTermination»? Kto zajmuje się zwolnieniem pamięci takich wątków?

Zadanie 4 (P). Struktura lwp[7] (ang. *lightweight process*) utrzymuje wszelkie informacje niezbędne jądru NetBSD do zarządzania danym wątkiem. Jej odpowiednikiem we FreeRTOS jest «tskTCB» w pliku «tasks.c». Opisz znaczenie pól «pxTopOfStack», «uxPriority», «uxBasePriority», «ulRunTimeCounter».

W trakcie życia wątek systemu FreeRTOS może być umieszczony jednocześnie na dwóch listach. Na które listy wpinamy wątek z użyciem węzła «xStateListItem», a na które z użyciem «xEventListItem»?

Zadanie 5 (P). Procedury «taskENTER_CRITICAL» i «taskEXIT_CRITICAL» służą do **blokowania przerwań**. Posługując się kodem jądra FreeRTOS podaj przykłady w których należy korzystać z blokowania przerwań, a kiedy z **blokowania wywłaszczania**.

Czemu liczba wywołań «taskENTER_CRITICAL» jest utrzymywana dla każdego wątku osobno (w polu «uxCriticalNesting» bloku kontrolnym wątku) zamiast globalnie?

Wskazkówka: Zauważ, że można zmienić kontekst, kiedy uxCriticalNesting > 0!

[1] https://mimiker.ii.uni.wroc.pl/source/xref/FreeRTOS-Amiga/
[2] https://freertos.org/a00111.html
[3] https://freertos.org/FAQMem.html#StackSize
[4] https://freertos.org/Stacks-and-stack-overflow-checking.html
[5] http://bxr.su/FreeBSD/sys/sys/pcpu.h#177
[6] https://mimiker.ii.uni.wroc.pl/source/xref/FreeRTOS-Amiga/FreeRTOS/tasks.c
[7] http://bxr.su/NetBSD/sys/sys/lwp.h#84

Zadanie 6 (P). W implementacji procedur, które należy wołać w ISR, można spotkać wywołania procedur «portSET_INTERRUPT_MASK_FROM_ISR» i «portCLEAR_INTERRUPT_MASK_FROM_ISR». Przy ich pomocy realizuje się **sekcje krytyczne** na architekturach dopuszczających **zagnieżdżanie przerwań** (np. m68k). Podobny problem w jądrze NetBSD rozwiązują procedury wymienione w spl(9).

Na podstawie rozdziału 6.8 książki „Mastering the FreeRTOS Real Time Kernel[8]" wyjaśnij dlaczego jest to koniecznie i o jakie sekcje krytyczne chodzi. Przeprowadź uczestników zajęć przez procedurę «ulPortSetIPL» (plik portasm.S) i wyjaśnij co ona robi.

Wskazówka: Rozważ pamięć współdzieloną między wszystkie procedury obsługi przerwań.

Zadanie 7 (P). Przeprowadź uczestników zajęć przez proces usypiania zadania na pewną liczbę taktów **zegara systemowego**. W trakcie prezentacji skup się na omówieniu **usypiania** zadania procedurą «vTaskDelay» i **wybudzania** po upłynięciu terminu w procedurze «xTaskIncrementTick».

Zadanie 8 (P). Kod procedury «vTaskSuspendAll» wyłączającej wywłaszczanie jest trywialny. Wydawałoby się, że jedyne co powinna zrobić procedura «xTaskResumeAll» to zmniejszyć wartość zmiennej «uxSchedulerSuspended» o 1, ale tak nie jest. Wyjaśnij zatem co robi ta procedura i dlaczego?

Wskazówka: Rozważ zadania, które przebywały długo w obrębie sekcji krytycznej z wyłączonym wywłaszczaniem.

[8] https://freertos.org/Documentation/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf