Struktura jąder systemów operacyjnych

Lista zadań nr 1

Na zajęcia 4 i 11 marca 2020

Należy przygotować się do zajęć czytając następujące rozdziały książek:

- *McKusick et al.* (wydanie drugie): 3.1 – 3.2
- *Tanenbaum* (wydanie czwarte): 11.3
- *Silberschatz* (wydanie dziesiąte): 5.6
- *M68000 Microprocessor's User Manual* (wydanie dziewiąte): 6

Zadania wymagające użycia rzutnika, oznaczenie **(P)**, należy starannie przygotować w domu – w tym celu należy sporządzić notatki. Każde zadanie należy mieć właściwie przygotowane do prezentacji przed zajęciami.

Referowanie kodu jądra systemu NetBSD należy przeprowadzić przy pomocy przeglądarki kodu dostępnej pod adresem bxr.su[1]. Będą nas interesować pliki w katalogach:

- /NetBSD/sys/arch/aarch64: zależne od architektury – w tym wypadku AArch64 (64-bitowy ARM),
- /NetBSD/sys/sys: pliki nagłówkowe jądra,
- /NetBSD/sys/kern: implementacja głównych funkcji jądra.

Referowanie kodu systemu operacyjnego FreeRTOS będziemy przeprowadzać przy pomocy lokalnej instancji oprogramowania OpenGrok zainstalowanej na serwerze mimiker.ii.uni.wroc.pl[2].

UWAGA! W trakcie prezentacji należy być gotowym do zdefiniowania pojęć oznaczonych **wytłuszczoną** czcionką.

Zadanie 1. W jakich przypadkach wywołanie stat(2) mogłoby zakończyć się awarią systemu, gdyby jądro nie używało funkcji kopiujących copy(9)? Czemu w trakcie przetwarzania wywołania systemowego należy skopiować dane pomiędzy **przestrzenią użytkownika** a **przestrzenią jądra**?

Zadanie 2. Które funkcje jądra BSD wykonują się w **górnej** (ang. *top half*), a które w **dolnej połówce** (ang. *bottom half*)? Jakie są konsekwencje zbyt długiego przebywania procesora w dolnej połówce? Kiedy kod w górnej połówce zostanie **wywłaszczony** (ang. *preempted*), a kiedy **wstrzymany** (ang. *suspended*)?

UWAGA! Pojęcia górnej i dolnej połówki w systemie Linux mają odwrotne znaczenie niż w systemach BSD.

Zadanie 3 (2,P). Kiedy i w jakim celu jądro woła funkcję userret(9)? Posługując się przeglądarką kodu zreferuj działanie procedur: «mi_userret», «cpu_need_resched», «lwp_userret», «preempt». Przeczytaj również podręczniki cpu_need_resched(9) i preempt(9). Ogranicz się do omawiania ciała wymienionych procedur. Należy skrótowo opisać znaczenie wykorzystywanych podprocedur.

Konwencja: Pierwszy człon nazwy procedury oznacza podsystem jądra. W tym przypadku mi oznacza machine independent (niskopoziomowy kod niezależny od architektury), a lwp oznacza light-weight processes (wątki jądra).

Wskazówka: IPI to skrót od inter-processor interrupt.

Zadanie 4. Jądro systemu WinNT jest łatwo **przenośne** (ang. *portable*) dzięki implementacji **warstwy abstrakcji sprzętu** (ang. *hardware abstraction layer*). Jakie zadania pełni HAL (§11.3)? Jakie korzyści daje HAL programistom implementującym **sterowniki urządzeń**?

W podręczniku intro(9) opisano podsystemy jądra NetBSD. Wśród wymienionych tam podsystemów zidentyfikuj te, które należą do warstwy abstrakcji sprzętu. Uzasadnij swój wybór przytaczając ustępy danych stron podręcznika. Jakie jest zastosowanie wymienionych podsystemów jądra?

[1] http://bxr.su

[2] https://mimiker.ii.uni.wroc.pl/source/xref/FreeRTOS-Amiga/

Zadanie 5. Najważniejszym kryterium w systemach czasu rzeczywistego jest **dotrzymywanie terminów**. Zatem zależy nam na minimalizacji **opóźnienia przetwarzania zdarzeń** (ang. *event latency*). Na podstawie §5.6.1 wymień fazy przetwarzania zdarzenia. Wyjaśnij z czego wynika **opóźnienie przetwarzania przerwania** (ang. *interrupt latency*) i **opóźnienie ekspedycji** (ang. *dispatch latency*).

Zadanie 6. Opisz zasadę działania algorytmów szeregowania zadań okresowych ze statycznym i dynamicznych przydziałem priorytetów: odpowiednio **RMS** (ang. *Rate Monotonic Scheduling*) i **EDF** (ang. *Earliest Deadline First*). Podaj kryterium **szeregowalności** dla powyższych algorytmów.

Dla zadań T_1 ... T_3 kolejno podano (w milisekundach) okres i czas wykonania, tj. odpowiednio (p_i, t_i):

- $(100, 20)$, $(150, 50)$, $(250, 100)$
- $(100, 20)$, $(150, 50)$, $(250, 120)$

Narysuj diagram (patrz obraz 5.22) przydziału czasu procesora dla pierwszych 750ms działania algorytmów.

Zadanie 7 (P). Instrukcja «`trap #n`» (gdzie $\mathtt{n} = 0...15$) procesora `M68000` może posłużyć do implementacji **wywołań systemowych** (§6.3.5). Spróbujmy zatem użyć «`trap #1`» do realizacji prostego wywołania systemowego, które drukuje ile razy zostało wywołane.

Umieść instrukcję wywołania systemowego `portTRAP(1)` w pętli procedury «`vRedTask`» (plik `main.c`). Następnie obsłuż instrukcję w procedurze «`vPortTrapHandler`» (plik `trap.c`).

Posługując się debuggerem zreferuj proces obsługi procedury wyjątku procesora od momentu wykonania instrukcji «`trap #1`», do momentu powrotu z procedury obsługi wyjątku przy pomocy instrukcji «`rte`». Pierwsza instrukcja procedury obsługi wyjątku znajduje się pod adresem «`TrapInstTrap`» (plik `trap.S`). Należy być przygotowanym do wydrukowania słów maszynowych, które odłożył / zdejmie procesor ze stosu w wyniku odpowiednio wejścia i powrotu z procedury obsługi wyjątku. **Wektor wyjątków procesora** (tabla 6-2) jest inicjowany w procedurze «`vPortSetupExceptionVector`».

Zadanie 8 (2,P). Peryferia komputera `Amiga 500` generują 14 różnych przerwań, które są odwzorowane w 6 przerwań zewnętrznych procesora `M68000` o rosnącym **priorytecie** (Level X Interrupt Autovector, tabela 6-2). Zapoznaj się pobieżnie z dokumentacją opisującą przerwania Interrupts[3]. Nas będzie interesować przerwanie **wygaszania pionowego** (ang. *vertical blank*), które zgłaszane jest 50 razy na sekundę.

Zauważ, że istnieją dwa mechanizmy kontroli przerwań zewnętrznych. Pierwszy to **poziom priorytetu przerwań** (ang. *interrupt priority level*) (§6.3.2) znajdujący się rejestrze stanu procesora `SR`. Drugi to rejestry «`INTREQ`» (ang. *interrupt requests*) i «`INTENA`» (ang. *interrupt enable*) układów specjalizowanych. Pierwszy z rejestrów przechowuje maskę przerwań oczekujących, a drugi maskę przerwań zgłaszanych procesorowi.

Twoim zadaniem jest zreferowanie w jaki sposób w systemie `FreeRTOS` zachodzi wywłaszczanie zadań. Procedura «`SystemClockTickHandler`» jest wywoływana w wyniku obsługi przerwania `VERTB`. Pierwsza instrukcja procedury obsługi przerwania leży pod adresem «`AmigaLvl3Handler`» (plik `intr.S`). Należy pokazać jak zostaje osiągnięty kod, który ustawia zmienną «`xNeedRescheduleTask`». Następnie należy dotrzeć do miejsca, w którym flaga ta jest sprawdzana i wołana jest procedura zmieniająca kontekst «`vPortYieldHandler`» (plik `portasm.S`).

[3] `http://amigadev.elowar.com/read/ADCD_2.1/Hardware_Manual_guide/node0160.html`