



WYDZIAŁ
ELEKTROTECHNIKI
I AUTOMATYKI

Imię i nazwisko studenta: Daniil Klimuk

Nr albumu: 187740

Poziom kształcenia: Studia pierwszego stopnia

Forma studiów: stacjonarne

Kierunek studiów: Automatyka, robotyka i systemy sterowania

Specjalność: Automatyka i Systemy Sterowania

PRACA DYPLOMOWA INŻYNIERSKA

Tytuł pracy w języku polskim: Procedury automatyzacji procesów w systemie mikroprocesorowym

Tytuł pracy w języku angielskim: Methods for implementing control algorithms using embedded systems

Opiekun pracy: dr inż. Robert Smyk

Streszczenie

W moment napisania pracy systemy cyfrowe i oprogramowanie zarządzające tymi systemami są bardzo rozbudowane posiadające od kilku do kilkunastu poziomów abstrakcji i podsystemów. Z powodu poważnego obniżenia ceny na technologię półprzewodnikową i duży skok progresu problemy małej pamięci i małej wydajności systemów cyfrowych są przeszłością. Z kolei stare problemy zostały zastąpione innymi, wynikającymi z coraz większej skomplikowości systemów i wzrostem zapotrzebowań: problem bezpieczeństwa i problem skalowalności. Aktualnie dużo zasobów jest spędzane na dobudowanie kolejnego poziomu na już istniejący system i rzadko jest zwracana uwaga, co leży pod spodem, chociaż jest tam logika nie mniej interesująca i definiująca właściwości całego systemu.

Centrum uwagi tej pracy jest logika zarządzająca zasobami obliczeniowymi systemu cyfrowego, nazywana ogólnie systemem operacyjnym, a szczególnie oprogramowanie planujące wykonywanie zadań i zarządzające zasobami jednostki obliczeniowej systemu — procesora. Praca zacznie się z podstawowych pojęć i architektur systemowych, i będzie coraz więcej pogłębiać się, aż nie dojdzie do pierwotnie postawionego celu. Następnie wspomniane przed tym programy zostaną przeanalizowane ze strony koncepcyjnej (pomysł, idea), formalnej (specyfikacje, definicje, założenia) i praktycznej (realizacja, implementacja). Zostaną też przeprowadzone eksperymenty praktyczne w celu pogłębienia i systematyzacji wiedzy. Na końcu autor przedstawi własne przemyślenia i wnioski bazujące się na poprzedniej pracy teoretycznej i praktycznej.

Słowa kluczowe: systemy operacyjne, systemy cyfrowe, zarządzanie zasobami procesora

Dziedzina nauki i techniki, zgodnie z wymaganiami OECD: smth

Abstract

smth **Keywords:** smth

Spis treści

Streszczenie

1

Abstract	1
Spis treści	1
Skróty i oznaczenia	3
1 Wstęp	5
1.1 Wprowadzenie	5
1.2 Cel i zakres pracy	6
2 Architektury i przykłady realizacji	6
2.1 Architektura "nagi metal"	7
2.2 Architektura z systemem operacyjnym	8
2.3 Architektura z wirtualizacją	10
2.4 Architektura z wirtualizacją zagnieżdżoną	11
3 Dekompozycja i wyodrębnienie	13
3.1 Dekompozycja i wyodrębnienie	13
4 Analiza	13
4.1 Analiza	13
Podsumowanie	13
4.2 Subsection	13
4.2.1 SubSubsection	14
Wykaz literatury	14
Załącznik nr 1: System wbudowany	16
Załącznik nr 2: ARM TrustZone	16
Załącznik nr 3: Wirtualizacja	18
Załącznik nr 4: Przełączanie stanu i kontekstu	18

Spis rysunków

1	Architektura "nagi metal"	7
2	Architektura z systemem operacyjnym	8
3	Architektura z wirtualizacją	10
4	Architektura z wirtualizacją zagnieżdżona	12
5	TrustZone dla architektury ARM v8-M	17
6	TrustZone dla architektury ARM v8-A	17
7	Tryby procesora zbudowanego na podstawie architektury ARMv8-M [7]	19
8	Przełączenie kontekstu w systemie operacyjnym uruchomionym na sprzęcie z jednostką obliczeniową zbudowanej na podstawie architektury ARMv8-M: deskryptor — struktura przechowująca meta-dane programu; meta-dane programu — dane opisujące program.	21

Skróty i oznaczenia

A ang. ARM A Profile, rodzina architektur ARM'owych stworzonych z myślą o układach o dużej mocy obliczeniowej i skomplikowanej architekturze oprogramowania [10].

ARM ang. Advanced RISC Machine, lub, poprzednio, Acorn RISC Machine; brytyjskie przedsiębiorstwo zajmujące się projektowaniem mikroprocesorów z siedzibą w Cambridge w Anglii [2].

BSA ang. Base System Architecture, termin wprowadzony przez ARM opisujący łączenie architektury sprzętowej i architektury oprogramowania, np. obecność MPU lub MMU [10].

EL ang. Exception Level, poziom uprawnień przypisywanych wykonującemu oprogramowaniu w architekturze ARM'owej, możliwe poziomy (od poziomu z najmniejszymi uprawnieniami do poziomu z największymi): EL0, EL1, EL2 i EL3 [8].

GIC ang. General Interrupt Controller, kontroler przerw skonstruowany dla wirtualizacji oraz obsługi kilku jednostek obliczeniowych na raz [9].

IDAU ang. Implementation Defined Attribution Unit, moduł zarządzający dostępem do pamięci przy obecności rozszerzenia bezpieczeństwa dla architektury ARMv8-M [15].

IPA ang. Intermediate Peripheral Address, dosłownie pośredni realny adres, pojęcie określające adresy pamięci widziane przez system operacyjny znajdujący się wewnątrz VM, te adresy są nazywane "pośrednio realnymi", dlatego, że realne adresy są ukrywane przez hiperwizor, z kolei system operacyjny uważa te adresy za realny, skoro nie wie że jest wewnątrz VM [3].

ISA ang. Instruction Set Architecture, architektura jednostki obliczeniowej z punktu widzenia programisty (rejstry, assembler i t.d.) [6].

M ang. ARM M Profile, rodzina architektur ARM'owych stworzonych z myślą o małe układy, lub tzw. mikrokontrolery [10].

MMU ang. Memory Management Unit, układ, odpowiedzialny za dostęp do pamięci przez oprogramowanie wykonywane na CPU, jest bardziej złożony niż MPU (m. in. ma funkcjonalności wspierające wirtualizację) [12].

MPU ang. Memory Protection Unit, układ, odpowiedzialny za dostęp do pamięci przez oprogramowanie wykonywane na CPU, jest mniej złożony w porównaniu do MMU (typowo posiada tylko mechanizmy zarządzania dostępem do pamięci) [13].

PA ang. Peripheral Address, dosłownie sprzętowy adres, tzn. realne adresy pamięci widziane przez hiperwizor [3].

R ang. ARM R Profile, rodzina architektur ARM'owych stworzonych z myślą o skomplikowanych systemach będących też systemami czasu rzeczywistego [10].

RISC ang. Reduced Instruction Set Computing; typ architektury mikroprocesorowego układu obliczeniowego o skróconej funkcjonalności do potrzebnego minimum oraz z większością instrukcji wykonujących się w jednym okresie zegara [1].

RTOS ang. Real Time Operating System; System Operacyjny Czasu Rzeczywistego.

SAU ang. Secure Attribution Unit, moduł zarządzający dostępem do pamięci przy obecności rozszerzenia bezpieczeństwa dla architektury ARMv8-M [15].

SMC ang. Secure Monitor Call, instrukcja ISA generująca synchroniczne przerwanie, jest częścią SMCCC [14].

SMCCC ang. SMC Calling Convention, mechanizm ARM'owy dla komunikacji pomiędzy środowiskiem zaufanym a niezaufanym [14].

- VA** ang. Virtual Address, dosłownie adres wirtualny, pojęcie określające adresy pamięci przekazane aplikacji poprzez system operacyjny, nie są to realne adresy w pamięci, tylko adresy w pewny sposób zmodyfikowane i przetwarzane na realne adresy podczas procesu translacji pamięci [3].
- VM** ang. Virtual Machine, dosłownie - maszyna wirtualna, lub środowisko wirtualne tworzone poprzez VMM [5].
- VMM** ang. Virtual Machine Monitor, dosłownie monitor maszyn wirtualnych, często też zwany hiperwizorem; oprogramowanie pozwalające na tworzenie środowisk wirtualnych (VM) [5].

1 Wstęp

1.1 Wprowadzenie

Od momentu pojawienia się pierwszych urządzeń elektronicznych i maszyn obliczeniowych minęło dość dużo czasu, w ciągu którego progres pchał cele postawione przed systemami cyfrowymi, co powodowało ich szybką ewolucję. Od listy instrukcji napisanej na kartkach perforowanych, do skomplikowanych programów zajmujących duże obszary pamięci, dodaniu do mocy obliczeniowej mnóstwa urządzeń peryferyjnych i prób abstrahować się od ich implementacji, wprowadzenia skomplikowanych abstrakcji systemowych dla ułatwienia developmętu i bezpieczeństwa, i, w końcu, łączenia mnóstwa systemów w jeden, mnogofunkcyjny system.

Aktualnie, logika zarządzająca systemami cyfrowymi i sprawiająca, że system wykonuje postawione przed nim cele oraz spełnia określone oczekiwania, jest bardzo skomplikowana, posiada dużo poziomów abstrakcji i podsystemów, co znacząco utrudnia integrację i analizę. Dodatkowo, nieznajomość działania tych systemów coraz częściej powoduje błędy i problemy podczas implementacji rozwiązań lub realizacji celów bazujących się na tych systemach.

Z kolei, bardzo ważną dla automatyki częścią tego progresu jest też ewolucja systemów czasu rzeczywistego. Jeszcze można spotkać się z systemami cyfrowymi pełniącymi rolę systemu czasu rzeczywistego z oprogramowaniem wykonującym się wprost na sprzęcie (tzn. bez żadnych warstw abstrakcji pomiędzy sprzętem a aplikacją, np. bez systemu operacyjnego). Jednak coraz częściej są spotykane RTOS'y lub inne, bardziej skomplikowane oprogramowanie, które używa wzrost funkcjonalności i wydajności mikrokontrolerów, na przykład coraz bardziej złożoną wirtualizację. Wzbudzają zainteresowanie też sposoby i metody na realizację wymagań systemu czasu rzeczywistego, jak również i sprawdzenie tych wymagań przy coraz bardziej

skomplikowanym oprogramowaniu.

1.2 Cel i zakres pracy

Celem pracy jest analiza oprogramowania zarządzającego zasobami jednostki (lub jednostek) obliczeniowych w cyfrowych systemach wbudowanych i planującego wykonywanie zadań, w tym w systemach cyfrowych pełniących rolę systemów czasu rzeczywistego na sprzęcie bazującym się na architekturze ARM'owej.

Zakres pracy:

- Przedstawienie i omówienie głównych cech architektur oprogramowania uruchomianego na systemach wbudowanych skonstruowanych na podstawie systemów cyfrowych bazujących się na architekturze ARM'owej;
- Przedstawienie przykładów realizacji omówionych architektur oprogramowania systemowego;
- Dekompozycja wybranych architektur oprogramowania;
- Wyodrębnienie części oprogramowania odpowiadających za zarządzanie zasobami jednostki obliczeniowej i planujących wykonywanie zadań dla wybranych architektur, i ich analiza;
- Analiza wyodrębnionych części oprogramowania odpowiadających za zarządzanie zasobami jednostki obliczeniowej i planujących wykonywanie zadań dla wybranych architektur;
- Przedstawienie przykładów realizacji wyodrębnionych części oprogramowania odpowiadających za zarządzanie zasobami jednostki obliczeniowej i planujących wykonywanie zadań dla wybranych architektur;
- Sprawdzenie sposobów realizacji i weryfikacji wymagań czasu rzeczywistego dla wybranych przykładów wybranych architektur;
- Sprawdzenie możliwości i sposobów analizy formalnej (np. analizy matematycznej);
- Przedstawić swoje rozwiązanie

2 Architektury i przykłady realizacji

Ten rozdział przedstawia i opisuje architektury oprogramowania w systemach wbudowanych (używana podczas napisania tej pracy definicja pojęcia "system wbudowany" jest zamieszc-

zona w załączniku nr 1) bazujących się na jednostkach obliczeniowych skonstruowanych na podstawie architektury ARM'owej. Będą też omówione cechy przedstawionych architektur oraz właściwości je powodujące. Następnie są przedstawione przykładowe realizacje tych architektur.

Należy też wspomnieć, że ARM przedstawia też technologię TrustZone, która dodaje kilka możliwych architektur. Skoro celem tej technologii jest bezpieczeństwo — architektury, które ona wprowadza, nie zostaną tu omówione, są one jednak wkrótce opisane w załączniku nr 2.

W dalszej części pracy wspomniany będzie termin "wirtualizacja" i terminy pochodne (parawirtualizacja, wirtualizacja natywna), których wy tłumaczenie można znaleźć w załączniku nr 3, są w nim także omówione pojęcia separacji, translacji pamięci i przekierowywania przerwań.

Pojęcia przełączenia stanów i przełączenia kontekstu są wyjaśnione w załączniku nr 4.

2.1 Architektura "nagi metal"



Rysunek 1: Architektura "nagi metal"

Najprostsza ze względu na ilość poziomów abstrakcji architektura, składająca się ze sprzętu i aplikacji wykorzystującej ten sprzęt (rys. 1). Dla funkcjonowania (jako system wbudowany) jest potrzebna jednostka obliczeniowa dla wykonywania poleceń, pamięć dla przechowywania poleceń, moduły wejścia/wyjścia dla kontaktu ze środowiskiem zewnętrznym i kontroler przerwań dla poinformowania aplikacji o zdarzeniach na zewnątrz. Zasoby sprzętowe są przydzielane statycznie podczas kompilacji, nie ma separacji środowisk lub uprzedzającego dostępu.

Cechy:

- Prostota: architektura nie posiada poziomów abstrakcji, aplikacja kontaktuje wprost ze sprzętem, nie ma skomplikowanych modułów do zarządzania bezpieczeństwem (np. MMU lub GIC);
- Wydajność: architektura nie posiada rozbudowanego kodu systemu operacyjnego lub

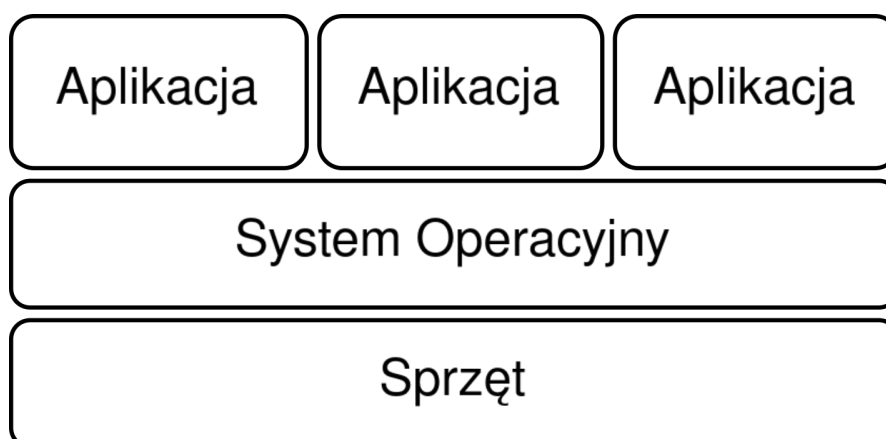
hiperwizora, nie ma translacji pamięci i przekierowania przerwań, nie ma przełączania kontekstu;

- Determinizm: nie ma konkurencji za zasoby jednostki obliczeniowej przez aplikacje, z powodu możliwej obecności tylko jednej aplikacji, pełna gwarancja reakcji na zdarzenie;

Uwzględniając powyższe cechy i właściwości, można zrobić wniosek, że taka architektura jest przeznaczona dla małych jednostek w systemie, np. czujniki, każda z których ma zdefiniowaną jedną cel. Przykładem realizacji danej architektury może być dowolna aplikacja niewykonywująca systemu operacyjnego lub hiperwizora na dowolnej architekturze ARM'owej, chociaż najczęściej będzie wybierana jedna z M architektur, ponieważ są one dla tego najlepiej przeznaczone.

Dana architektura nie stanowi zainteresowania w ramach tej pracy, skoro nie posiada oprogramowania sterującego dostępem do zasobów jednostki obliczeniowej.

2.2 Architektura z systemem operacyjnym



Rysunek 2: Architektura z systemem operacyjnym

Architektura posiadająca jedną warstwę abstrakcji — system operacyjny (rys. 2), oprogramowanie systemowe, które w pełni lub częściowo (zależy od architektury systemu operacyjnego) zachowuje kontrolę nad sprzętem. Opiera się o dodatkowe moduły sprzętowe mające na celu separację pamięci (najczęściej MPU, lub też MMU), aby rozdzielić kod systemu operacyjnego od kodu aplikacji, oraz bardziej złożony kontroler przerwań obsługujący nie tylko zdarzenia zewnętrzne, ale i zdarzenia wewnętrzne, pozwalające na zmianę stanu systemu.

Jednostka obliczeniowa ma wbudowany podział stanów systemu: stan aplikacji (EL0 w architekturze ARM'owej), w którym wykonywany kod nie ma dostępu do zasobów systemowych (definicja tych zasobów zależy od systemu operacyjnego) i stan systemu operacyjnego (EL1 w architekturze ARM'owej), mający dostęp do wszystkiego. Dla większości systemów operacyjnych obecność podziału na stany ze strony sprzętowej (obecności pewnych rejestrów oraz instrukcji ISA) jest obowiązkowa.

Cechy:

- Skomplikowość: w zależności od architektury systemu operacyjnego (monolityczne jądro, mikrojądro, nanojądro) system operacyjny może posiadać różną ilość narzędzi i funkcjonalności;
- Mniejsza wydajność: obecność abstrakcji w samym systemie operacyjnym (sterowniki, interpretery aplikacji), przełączanie stanów pomiędzy systemem operacyjnym a aplikacją, przełączenie kontekstu pomiędzy aplikacjami;
- Większe bezpieczeństwo: możliwość "narzucania" zasad dla aplikacji z poziomu systemu operacyjnego, separacja aplikacji;
- Skalowalność: obecność oprogramowania zarządzającego zasobami jednostki obliczeniowej pozwala na uruchomienie wielu aplikacji oraz zdefiniowanie priorytetów dla każdej z nich, więc jeden system wbudowany może pełnić wiele celów;
- Mniejszy determinizm: przy obecności wielu aplikacji pojawia się pewne prawdopodobieństwo, że zadania postawione dla każdej z tych aplikacji zostaną wykonane w inne odcinki czasu, niż to było przewidziane, skoro jest konkurencja za dostęp do jednostki obliczeniowej przez pozostałe aplikacje.

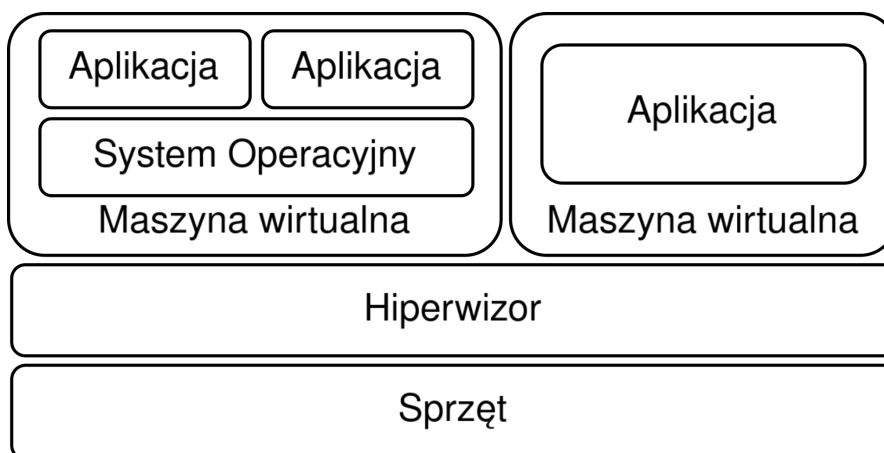
Taką architekturę oprogramowania można spotkać na wszystkich architekturach ARM'owych (tzn. na ARM M, ARM A i ARM R), popularne przykłady:

- FreeRTOS;
- Zephyr RTOS;
- Linux;
- FreeBSD, NetBSD;
- Windows;
- seL4.

Ta architektura stanowi zainteresowanie w ramach tej pracy, ponieważ pojawia się oprogramowanie zarządzające zasobami jednostki obliczeniowej, zwane także schedulerem. Jest

ono schowane wewnątrz systemu operacyjnego.

2.3 Architektura z wirtualizacją



Rysunek 3: Architektura z wirtualizacją

Architektura z wirtualizacją jest osiągana za pomocą dodania kolejnego poziomu abstrakcji do poprzednio omówionych architektur (rys. 3; EL2 w architekturach ARM'owych), tzn. hiperwizor może uruchamiać VM z aplikacją w architekturze "nagi metał" jak również i z systemem operacyjnym, który już będzie odpowiedzialny za uruchomienie aplikacji.

Typowo hiperwisory potrzebują odpowiedniego sprzętu dla uruchomienia (ang. virtualization support; głównie chodzi tu o rozszerzenia do obsługi przerwań oraz separacji pamięci, a także dodaniu oddzielnego stanu jednostki obliczeniowej), w takim przypadku wirtualizacja nazywa się natywną (ang. native virtualization, także ang. full virtualization, lub ang. hardware-accelerated virtualization).

Istnieje także możliwość stworzenia hiperwizora niezależnego od sprzętu, tzn. niepotrzebującego odpowiedniego sprzętu do uruchomienia, jest to tzw. parawirtualizacja (ang. paravirtualization).

Cechy:

- Duża skomplikowość: do faktu, że hiperwizor jest przeznaczony dla uruchomienia innych architektur w środowisku wirtualnym (co w przypadku parawirtualizacji może być dużym wezwaniem) dołączają się komunikacja pomiędzy środowiskami i przedzielenie dostępu do sprzętu poszczególnym środowiskom, w tym przerwań. Dodatkowo może być

też wykorzystywana emulacja;

- Jeszcze mniejsza wydajność: dodatkowe zmniejszenie wydajności głównie jest spowodowane potrzebą translacji pamięci, przechwytywania i wirtualizacji przerw, oraz dodatkowego przełączania stanów i kontekstu;
- Jeszcze większe bezpieczeństwo: do separacji oprogramowania w pamięci dochodzi jeszcze dokładne przydzielenie używanego sprzętu dla każdego ze środowisk wirtualnych.
- Konfigurowalny determinizm: to oznacza, że na jednym sprzęcie można uruchomić dwie i więcej zupełnie różne architektury, np. "nagi metal" i system operacyjny, cechujące się różnymi właściwościami deterministycznymi i rozłożyć priorytety, można też zmienić sposób zarządzania dostępem do zasobów jednostki obliczeniowej: z dynamicznego zarządzanie na zarządzanie statyczne.

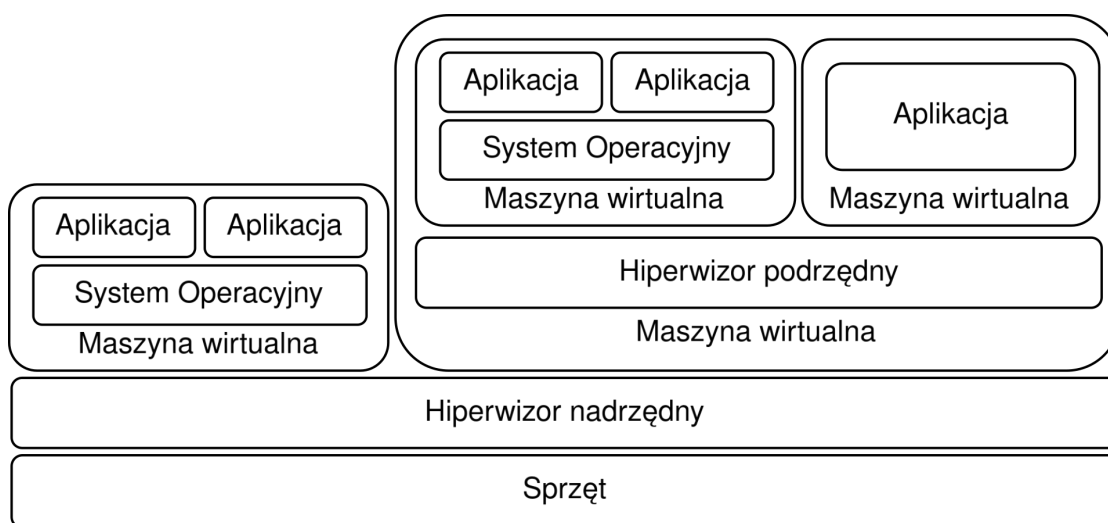
Tę architekturę oprogramowania systemowego można spotkać na architekturach sprzętowych ARM A i ARM R, skoro one mają sprzęt specjalnie przeznaczony dla wirtualizacji. Możliwe jest także uruchomienie na architekturach nieprzeznaczonych dla wirtualizacji, w takim przypadku niezbędne będzie użycie parawirtualizacji. Przykłady realizacji:

- Jailhouse;
- Xen;
- Bao;
- seL4 CAMkES;
- KVM;
- Crosscon Hypervisor.

Główną notatką tu jest fakt, że hiperwizor jest dodatkową warstwą zarządzającą zasobami jednostki obliczeniowej, to oznacza, że jest on też punktem zainteresowania w ramach tej pracy. W przypadku, gdy w środowisku wirtualnym jest realizowana architektura z systemem operacyjnym, można mówić o zagnieżdżonym (dwupoziomowym) zarządzaniem dostępem do zasobów jednostki obliczeniowej.

2.4 Architektura z wirtualizacją zagnieżdżoną

Wirtualizacja zagnieżdżona (ang. nested virtualization) polega na stworzeniu VM wewnątrz VM (rys. 4), co oznacza też dodanie dodatkowego hiperwizora, którego można nazwać "hiperwizorem podrzędnym".



Rysunek 4: Architektura z wirtualizacją zagnieżdżona

Aktualnie architektury ARM'owe wspierają tę architekturę częściowo, tzn. nie ma ani danego kolejnego EL, ani wsparcia dla trzypoziomowej translacji pamięci, ani dedykowanych mechanizmów przekierowywania przerwań, jest tylko podstawowe wsparcie w postaci rozszerzeń do architektury ARMv8-A [4].

Autor nie miał doświadczeń z tą architekturą, więc nie będzie przedstawiono jej cech. Należy też dodać, że nie jest ona często wykorzystywana i wygląda raczej jako architektura eksperymentalna [4].

Ta architektura dodaje kolejną warstwę z oprogramowaniem, która zarządza dostępem do zasobów jednostki obliczeniowej, oprogramowanie to znajduje się w hiperwizorze podrzędnym, co, w przypadku, gdy w VM stworzonej przez hiperwizor podrzędny jest wykorzystywana architektura z systemem operacyjnym, skutkuje trzema poziomami zarządzania dostępem do jednostki obliczeniowej: oprogramowanie w hiperwizorze nadrzędnym, oprogramowanie w hiperwizorze podrzędnym i oprogramowanie w systemie operacyjnym, co by stanowiło ciekawy punkt dla badań, gdyby nie słabe wsparcie dla tej architektury w społeczeństwie systemów wbudowanych, co powoduje dużą ilość problemów niebędących centrum uwagi tej pracy. Więc nie będzie to omówione w tej pracy.

3 Dekompozycja i wyodrębnienie

3.1 Dekompozycja i wyodrębnienie

4 Analiza

4.1 Analiza

Podsumowanie

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

4.2 Subsection

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper,

felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

4.2.1 SubSubsection

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

RTOS

Wykaz literatury

- [1] Samuel O. Aletan. "An Overview of RISC Architecture". In: *SAC '92: Proceedings of the 1992 ACM/SIGAPP Symposium on Applied computing: technological challenges of the 1990's* (Apr. 1992).
- [2] *Arm Holdings*. wikipedia.org. URL: https://en.wikipedia.org/wiki/Arm_Holdings.
- [3] *Armv8-A Address Translation*. Version 1.1. ARM. July 2024. URL: <https://developer.arm.com/documentation/100940/0101/>.
- [4] Christoffer Dall. *Nested Virtualization on ARM*. Linaro. URL: <https://www.youtube.com/watch?v=cuXyGkZRUKU>.
- [5] Robert P. Goldberg Gerald J. Popek. "Formal Requirements for Virtualizable Third Generation Architectures". In: *Communications of ACM* (July 1974). URL: <https://dl.acm.org/doi/10.1145/361011.361073>.
- [6] *Instruction set architecture*. wikipedia.org. URL: https://en.wikipedia.org/wiki/Instruction_set_architecture.
- [7] *Introduction to the Armv8-M Architecture and its Programmers Model User Guide*. Version 1.1. ARM. July 2023. URL: <https://developer.arm.com/documentation/107656/0101/>.
- [8] *Learn the architecture - AArch64 Exception Model*. Version 1.3. Arm. Dec. 2022. URL: <https://developer.arm.com/documentation/102412/0103>.
- [9] *Learn the architecture - Generic Interrupt Controller v3 and v4*. Version 3.2-01. ARM. Dec. 2021. URL: <https://developer.arm.com/documentation/198123/0302>.
- [10] *Learn the architecture - Introducing the Arm architecture*. Version 2.1. Arm. Sept. 2024. URL: <https://developer.arm.com/documentation/102404/0201>.
- [11] *Learn the architecture - TrustZone for AArch64*. Version 1.1. Arm. Feb. 2024. URL: <https://developer.arm.com/documentation/102418/0101>.
- [12] *Memory management unit*. wikipedia.org. URL: https://en.wikipedia.org/wiki/Memory_management_unit.
- [13] *Memory protection unit*. wikipedia.org. URL: https://en.wikipedia.org/wiki/Memory_protection_unit.

- [14] *SMC Calling Convention*. Version G ALP1. ARM. Oct. 2024. URL: <https://developer.arm.com/documentation/den0028/galp1>.
- [15] *TrustZone technology for Armv8-M Architecture*. Version 2.1. Arm. Oct. 2018. URL: <https://developer.arm.com/documentation/100690/0201>.

Załącznik nr 1: System wbudowany

Skoro nie ma jednoznacznej definicji pojęcia "system wbudowany" - należy określić definicję, która jest wykorzystywana w tej pracy.

System wbudowany jest systemem, to znaczy ma cechy systemu: jest złożony z więcej niż jednego elementu, jest funkcjonalnie niezależny od środowiska (tzn. może być wyodrębniony ze środowiska) i posiada możliwość reagowania (tzn. odbierania, przetwarzania i odpowiedzi) na bodźce zewnętrzne. "wbudowany" oznacza, że system jest częścią innego systemu fizycznie i/lub funkcjonalnie.

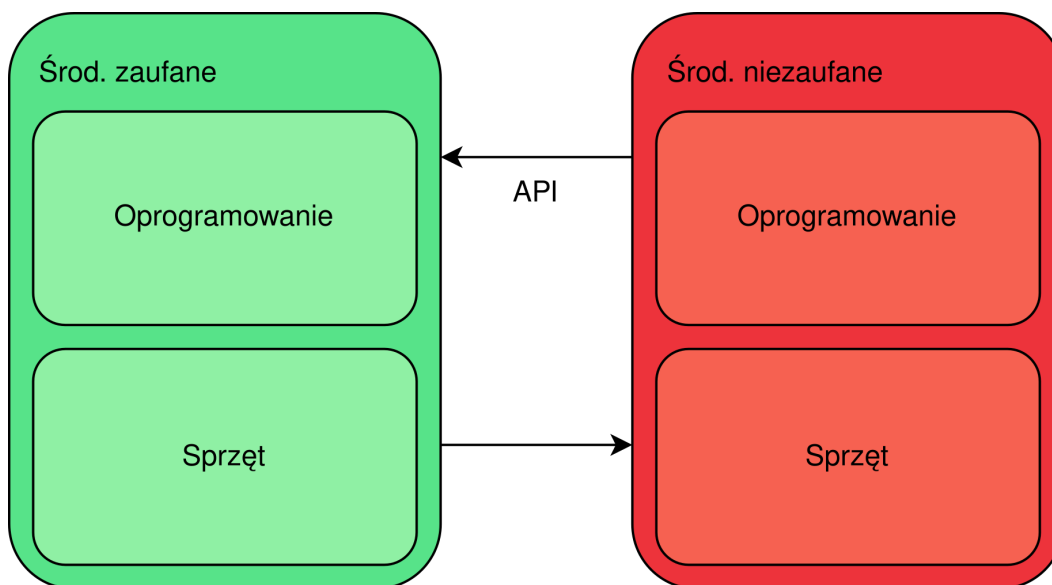
Natomiast przełożyć tę definicję na dziedzinę elektroniki i informatyki można w następujący sposób: system wbudowany — jest to system komputerowy składający się z jednostki obliczeniowej i modułów wejścia/wyjścia (tzn. może reagować na bodźce zewnętrzne), mający wszystkie narzędzia (w tym oprogramowanie) dla spełnienia pewnej, zdefiniowanej funkcji (więc może być wyodrębniony) i będący częścią większego systemu.

Załącznik nr 2: ARM TrustZone

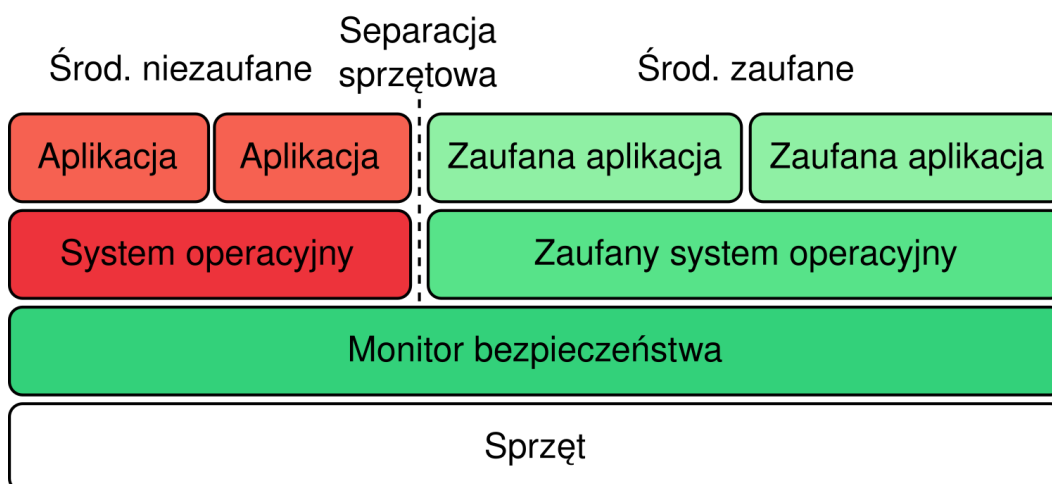
Koncept technologii TrustZone bazuje się na podziale sprzętowym środowiska (pamięci i innych zasobów) na środowisko zaufane i niezaufane (rys. 5 i rys. 6), co jest robione za pomocą rozszerzeń do architektur ARMv8-A i ARMv8-M dotyczących BSA i ISA.

Dla architektury ARMv8-A został dodany EL3, w którym jest zamieszczony monitor bezpieczeństwa (rys. 6), realizowany przez oprogramowanie ARM'owe Trusted Firmware A (referencyjna implementacja monitoru bezpieczeństwa). Faktycznie ten monitor bezpieczeństwa jest pewnym mostem przekazującym polecenia ze środowiska niezaufanego do środowiska zaufanego oraz przełączający procesor w bezpieczny tryb za pomocą SMCCC. [14][11]

Dla architektury ARMv8-M zostały dodane rozszerzenia pozwalające na separację środowiska zaufanego od niezaufanego (rys. 5) za pomocą modułów SAU i IDAU, oraz instrukcji dla komu-



Rysunek 5: TrustZone dla architektury ARM v8-M



Rysunek 6: TrustZone dla architektury ARM v8-A

nikacji pomiędzy środowiskami: ISA SG, BXNS i BLXNS. [15]

Załącznik nr 3: Wirtualizacja

Wirtualizacja jest technologią wprowadzającą separację środowisk z oprogramowaniem poprzez tworzenie tzw. VM zarządzanych przez VMM (rys. 3). Typy separacji używane w wirtualizacji: separacja pamięciowa (poprzez translację pamięci), separacja sygnałowa (przekierowywanie przerw) i separacja stanów (przełączanie kontekstu).

Celą separacji pamięciowej jest przypisanie do każdej VM przedziału adresu pamięci (kodu programu i urządzeń peryferyjnych, ponieważ rejestry tych urządzeń są zmapowane do pamięci; ang. memory mapped peripherals) i wychwytywanie wszystkich prób manipulacji pamięcią nieprzypisaną do tej VM. W architekturach ARM tym się zajmuje MMU, który, zaczynając od ARMv8, posiada dwa poziomy translacji: z poziomu aplikacji (VA), do poziomu systemu operacyjnego (IPA) i do poziomu hiperwizora (PA [3]).

Celą separacji sygnałowej jest podział i przekierowywanie przerw sprzętowych pomiędzy VM. Każda VM ma przepisany podzbiór urządzeń peryferyjnych za pomocą separacji pamięci, te urządzenia mogą generować przerwy, za przekierowywanie których jest odpowiedzialny hiperwizor i kontroler przerw (w przypadku ARM'u to jest GIC).

Celą separacji stanów jest sprzętowy podział wykonywanych programów i przypisanie do nich pewnych poziomów uprawnień. Dokładniej to jest opisane w załączniku nr 4.

Załącznik nr 4: Przełączanie stanu i kontekstu

Stan wykonywanego oprogramowania jest określany poprzez następujące elementy:

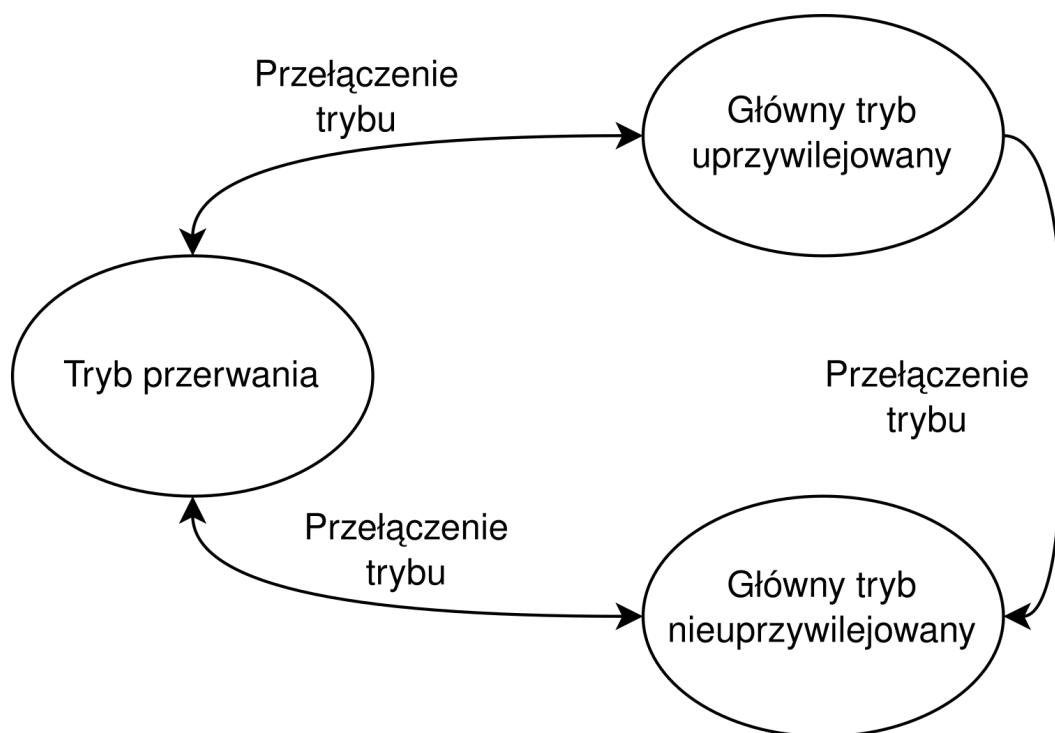
- Zawartości rejestrów ogólnego przeznaczenia: zawierają wyniki wykonywania ostatnich instrukcji;
- Rejestry wskazujące na adres następnej komendy (PC w architekturach ARM'owych), adres powrotu (LR w architekturach ARM'owych) i adres aktualnie używanego stosu (MSP lub PSP w architekturach ARM): określają "kierunek" (tzn. co będzie wykonane następne, gdzie wrócić po wykonaniu i tak dalej) wykonywanych instrukcji oprogramowania;
- Zawartość aktualnie używanego stosu: tak samo jak i poprzedni punkt, określa "kierunek" wykonywania instrukcji oprogramowania, przechowuje historie wykonywanych instrukcji

oprogramowania (np. przy przerwaniu, do niego są zapisywane pewne rejestry, które po zakończeniu przerwania są odczytywane z powrotem, pozwalając na wznowienie wykonywanego programu);

- Rejestry systemowe (np. PSR i rejestry masek przerwań w architekturach ARM'owych): określają między innymi tryb jednostki obliczeniowej, w którym program jest wykonywany, co z kolei określa poziom dostępu do zasobów sprzętowych.

Z powyższego wynika, że stan wykonywanego oprogramowania — to jest zbiór pewnych wartości przechowywanych w rejestrach jednostki obliczeniowej, co oznacza, że przełączenie stanów jest robione sprzętowo, przez jednostkę obliczeniową (ponieważ oprogramowanie nie ma dostępu zapisu lub odczytu do wszystkich rejestrów jednostki obliczeniowej).

Z kolei przełączenie stanów — to jest proces, kiedy stan aktualnie wykonywanego oprogramowania jest zapisywany (w stos przynależącego do tego oprogramowania) i podmieniany na stan przynależący do innego oprogramowania (który jest odczytywany ze stosu uruchamianego oprogramowania). Występuje to podczas zmiany trybu jednostki obliczeniowej, ponieważ każdy tryb jednostki obliczeniowej ma własny stos.



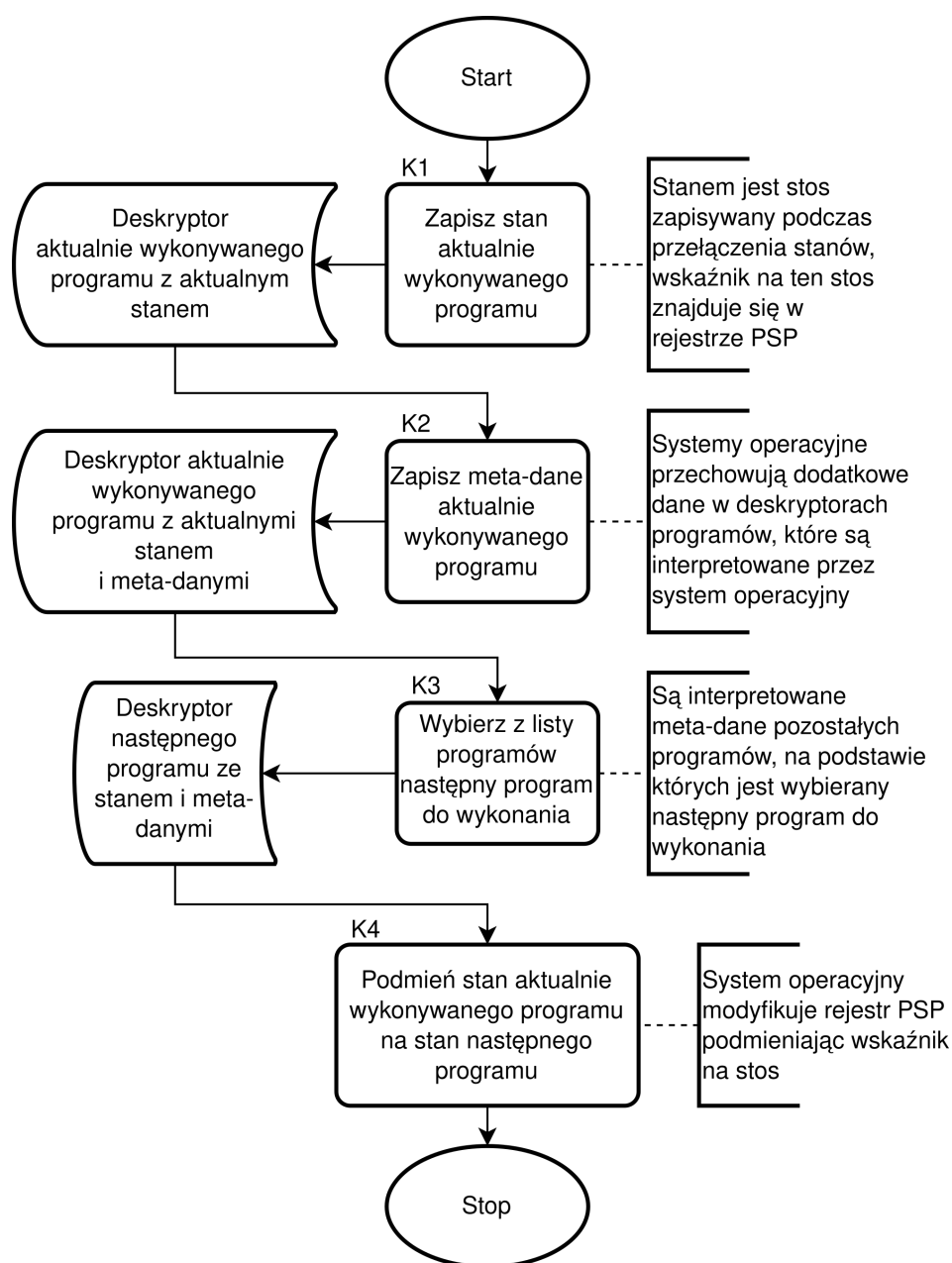
Rysunek 7: Tryby procesora zbudowanego na podstawie architektury ARMv8-M [7]

Na rys. 7 są pokazane tryby jednostki obliczeniowej zbudowanej na podstawie architektury ARMv8-M. Taka jednostka posiada trzy tryby z dwoma poziomami dostępu: tryb główny (główny uprzywilejowany i główny nieuprzywilejowany) i tryb przerwania. Każdy z tych trybów ma swój stos, w którym jest przechowywany stan aktualnie wykonywanego oprogramowania. Adres na każdy ze stosów jest przechowywany w rejestrach MSP i PSP [7].

Natomiast kontekst programu jest pojęciem definiowanym przez poszczególną implementację oprogramowania systemowego (system operacyjny bądź hiperwizor) i zawiera w sobie opisany wyżej stan programu oraz dodatkowe dane, interpretowane przez oprogramowanie systemowe (np. czas jednostki obliczeniowej, który był użyty przez ten program, lub priorytet).

Stąd przełączenie kontekstu oznacza, że nie jednostka obliczeniowa jest odpowiedzialna za wybór i uruchomienie pewnego programu, tylko oprogramowanie systemowe.

Na rys. 8 jest pokazany przykładowy przebieg takiego przełączenia dla architektury ARMv8-M. Przełączenie kontekstu następuje podczas wykonywania kodu systemu operacyjnego, tzn. kiedy jednostka obliczeniowa znajduje się w trybie głównym uprzywilejowanym (rys. 7), co oznacza, że przed przełączeniem kontekstu następuje przełączenie stanów, podczas którego jednostka obliczeniowa przełącza się z wykonywania kodu aktualnego programu na wykonywanie kodu systemu operacyjnego (tzn. przechodzi z trybu głównego nieuprzywilejowanego do trybu głównego uprzywilejowanego, rys. 7). Natomiast po przełączeniu kontekstu następuje przełączenie stanów, podczas którego jednostka obliczeniowa przełącza się z wykonywania kodu systemu operacyjnego na wykonywanie kodu programu zaproponowanego przez system operacyjny podczas przełączenia kontekstu programu (tzn. przechodzi z trybu głównego uprzywilejowanego do trybu głównego nieuprzywilejowanego).



Rysunek 8: Przełączenie kontekstu w systemie operacyjnym uruchomionym na sprzęcie z jednostką obliczeniową zbudowanej na podstawie architektury ARMv8-M: deskryptor — struktura przechowująca meta-dane programu; meta-dane programu — dane opisujące program.