POLITECHNIKA GDAŃSKA
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 skomplikowaności 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 konceptualne (pomysł, idea), formalnej (specyfikacje, definicje, założenia) i praktycznej (realizacje, implementacje). Zostaną też przeprowadzone eksperymenty praktyczne w celu pogłębienia i systematyzacji wiedzy. Na końcu autor przedstawi własne przemyślenia i wnioski bazujące się na poprzedniej pracę 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

Spis rysunków

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].

AMP ang. Asynchronous Multiprocessing, techniki zarządzania kilkoma jednostkami obliczeniowymi w sposób, gdy każdej jednoscę jest przydzielane oddzielny proces do wykonania [16].

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 Arhitecture, 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 uprawnieniamu

do poziomu z największymi): EL0, EL1, EL2 i EL3 [8].

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

HVC ang. Hypervisor Call, przerwanie systemowe, z którego może skorzystać system operacyjny, aby skomunikować się z hiperwizorem [8].

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

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

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

M ang. ARM M Profile, rodzina architektur ARM'owych stworzonych z myślą o małe energooszczędne 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 funkcjanolności 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 posida tylko mechanizmy zarządzania uprawnieniami prz dostępie 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 architektóry mikroprocesorowego układu oblczeniowego o skróconej funkcjanolności 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].

SMP ang. Synchronous Multiprocessing, techniki zarządzania kilkoma jednostkami obliczeniowymi w sposób, gdy proces jest dzielony na wątki które są przydzielane każdej jednostkie obliczeniowej, tzn. wszystkie jednostki obliczeniowe zostają na pewny okres czesu przepisane do jednego procesu [17].

SVC ang. Supervisor Call, przerwanie systemowe, z którego może skorzystać aplikacja, aby skomunikować się z jądrem systemu operacyjnego [8].

VA ang. Virtual Address, dosłownie adres wirtualny, pojęcie określające adresy pamięci przekazane aplikacji poprzez system operacyjny, nie są tą realne adresy w pamięci, tylko adresy w pewny sposób zmodyfikowane i przetwarzane na realne adresy podczas procesu translacji pamięci [3].

VCPU ang. Virtual CPU, wirtualna jednostka obliczeniowa.

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, mnogofunkcjonalny 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;
- Analiza wyodrębnionych części oprogramowania;

- Przedstawienie przykładów realizacji wyodrębnionych części oprogramowania;
- Sprawdzenie sposobów realizacji i weryfikacji wymagań czasu rzeczywistego dla wybranych przykładów;
- Sprawdzenie możliwości i sposobów analizy formalnej (np. analizy matematycznej);
- Przedstawić swoje rozwiązanie

2 Architektury i przykłady realizazcji

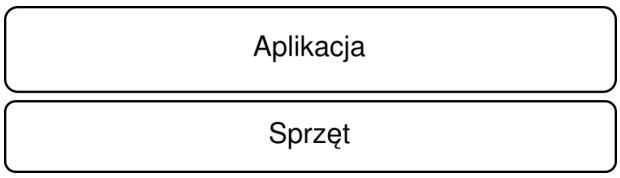
Ten rozdział przedstawia i opisuje architektury oprogramowania w systemach wbudowanych (używana podczas napisania tej pracy definicja pojęcia ”system wbudowany” jest zamieszczona w załączniku nr 1) bazujących się na jednostkach obliczeniowych skonstruowanych na podstawie architektury ARM’owej. Będą też 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 wspominany będzie termin ”wirtualizacja” i terminy pochodne (parawirtualizacja, wirtualizacja natywna), których wytłumaczenie 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 z zewnątrz. Zasoby sprzętowe są przydzielane statycznie podczas kompilacji, nie ma separacji środowisk lub uprzelewanego dostępu.

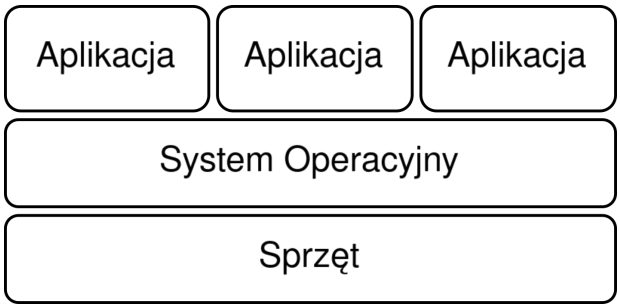
Cechy:

- 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 niewykorzystująca 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 zarządzającego 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 zarządzające pamięcią 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 A), 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 A), mający dostęp do wszystkiego. Dla większości systemów operacyjnych obecność podziału na stany ze strony sprzętowej (obecności pewnych rejestrów oraz instrukcji ISA) jest obowiązkowa.

Cechy:

- Skomplikowaność: w zależności od architektury systemu operacyjnego (monolityczne jądro, mikrojądro, nanojądro) system operacyjny może posiadać różną ilość narzędzi i funkcjonalności;
- Mniejsza wydajność: obecność abstrakcji w 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.

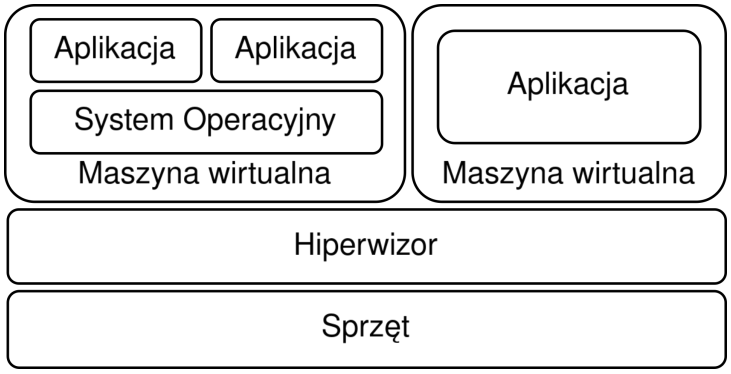
Taką architekturę oprogramowania można spotkać na wszystkich architekturach ARM’owych (tzn. na ARM M, ARM A i ARM R), popularne przykłady:

- 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 wirtualizacja



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 A), 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 niezależącego od sprzętu, tzn. niepotrzebującego odpowiedniego sprzętu do uruchomienia, jest to tzw. parawirtualizacja (ang. paravirtualization).

Cechy:

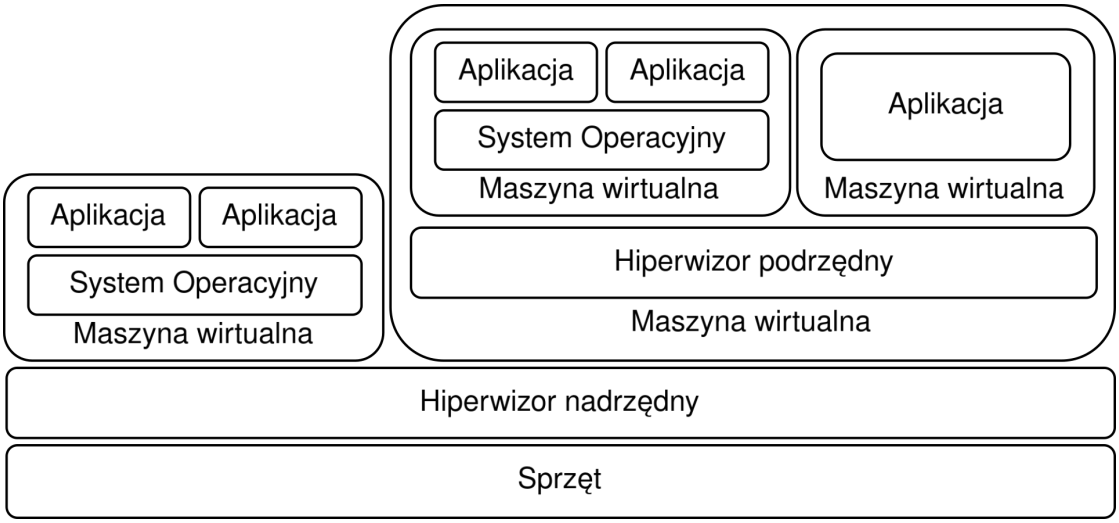
- Duża skomplikowaność: do faktu, że hiperwizor jest przeznaczony dla uruchomienia innych architektur w środowisku wirtualnym (co w przypadku parawirtualizacji może być dużym wezwaniem) dołączają się komunikacja pomiędzy środowiskami i przedzielenie dostępu do sprzętu poszczególnym środowiskom, w tym przerwań. Dodatkowo może być też wykorzystywana emulacja;
- Jeszcze mniejsza wydajność: dodatkowe zmniejszenie wydajności głównie jest spowodowane potrzebą translacji pamięci, przechwytywania i wirtualizacji przerwań, 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 dostępu do zasobów jednostki obliczeniowej.

2.4 Architektura z wirtualizacją zagnieżdżoną



Rysunek 4: Architektura z wirtualizacją zagnieżdżona

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

Aktualnie architekury ARM’owe wspierają tę architekturę częściowo, tzn. nie ma ani dodanego kolejnego EL, ani wsparcia dla trzypoziomowej translacji pamięci, ani dedykowanych mechanizmów przekierowywania przerwań, jest tylko podstawowe wsparcie w postaci rozszerzeń do pewnych rejestrów w architekturach ARM 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,

ta architektura nie będzie omówiona w ramach tej pracy.

3 Dekompozycja i wyodrębnienie

W tej części pracy będzie przedstawiona dekompozycja oprogramowania systemowego przedstawionego w poprzedniej części w architekturach z systemem operacyjnym i wirtualizacją, a dokładnie: systemu operacyjnego i hiperwizora. Następnie oprogramowanie odpowiadające za zarządzanie dostępem do zasobów jednostki obliczeniowej będzie wyodrębnione ze wspomnianego przed tym oprogramowania systemowego dla przeprowadzenia analizy w następnej części pracy.

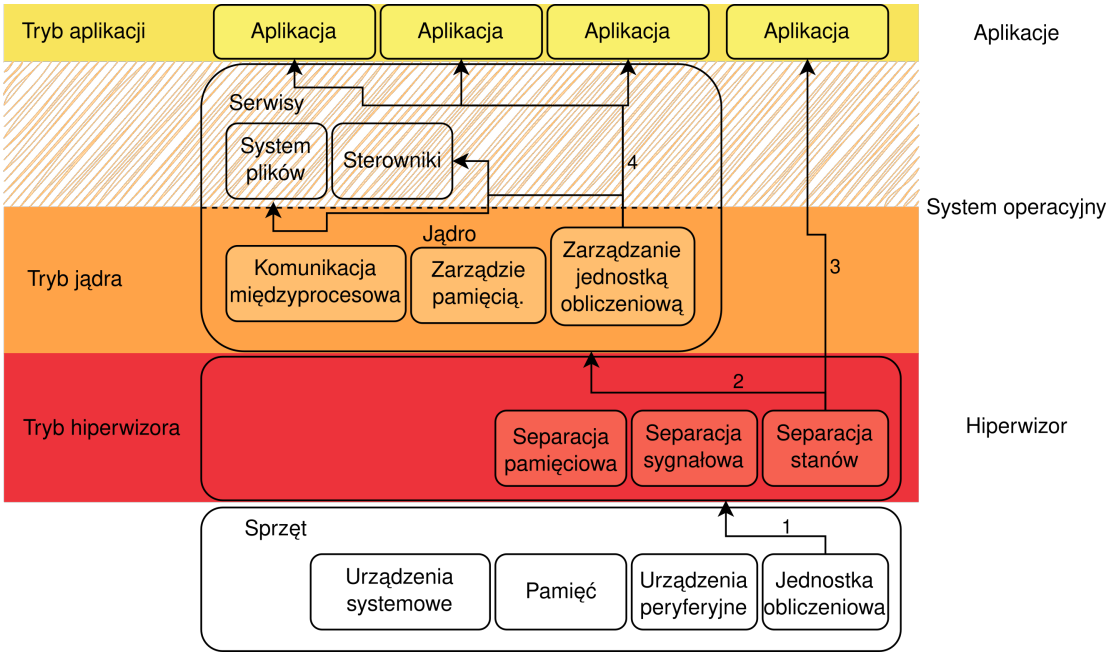
Należy zauważyć, że celem tej pracy nie jest omówienie wszystkich elementów oprogramowania systemowego. Pozostałe elementy oprogramowania systemowego będą wspomniane, jeżeli będzie na to potrzeba, tylko w sposób ogólny, pozwalający na prześledzenie udziału oprogramowania zarządzającego zasobami jednostką obliczeniową w ogólnej architekturze oprogramowania systemowego, oraz umożliwiający wyodrębnienie tej części dla dalszej analizy.

3.1 Dekompozycja

Jak wspomniano poprzednio, z punktu widzenia zarządzania zasobami jednostki obliczeniowej są ciekawe dwa rodzaje oprogramowania systemowego: system operacyjny i hiperwizor. Na rys. 5 jest pokazane przydzielenie zasobów jednostki obliczeniowej przy użyciu tego oprogramowania systemowego, kolorami są odznaczone tryby, w których znajduje się jednostka obliczeniowa podczas wykonywania instrukcji pewnego oprogramowania, zacieniony obszar wskazuje, że serwisy systemu operacyjnego mogą być wykonywane zarówno w trybie jądra jak i w trybie aplikacji (zależy od architektury systemu operacyjnego: mikrokernel lub monolityczny kernel). Strzałkami jest odznaczone przydzielenie zasobów jednostki obliczeniowej do pewnego oprogramowania przez pewną logikę w oprogramowaniu systemowym. Ten rysunek demonstruje architekturę z wirtualizacją, zaś architektura bez wirtualizacji wykorzystująca tylko system operacyjny może być pokazana usuwając z rys. 5 cały hiperwizor i strzałkę o numerze 3. W takim przypadku strzałka o numerze 1 będzie wskazywała wprost na system operacyjny.

Jest to struktura hierarchiczna, tzn. (numery punktów są odpowiednie do numerów strzałek na rys. 5):

1. Hiperwizor otrzymuję kontrolę nad wszystkimi zasobami jednostki obliczeniowej i przy-



Rysunek 5: Przydzielenie dostępu do zasobów jednostki obliczeniowej przez oprogramowanie systemowe: system operacyjny i hiperwizor

dziela je do oprogramowania, które on uruchamia: system operacyjny (strzałka nr 2 na rys. 5) lub aplikacja (strzałka nr 3 na rys. 5), spędzając na ten proces pewną część otrzymanych zasobów;

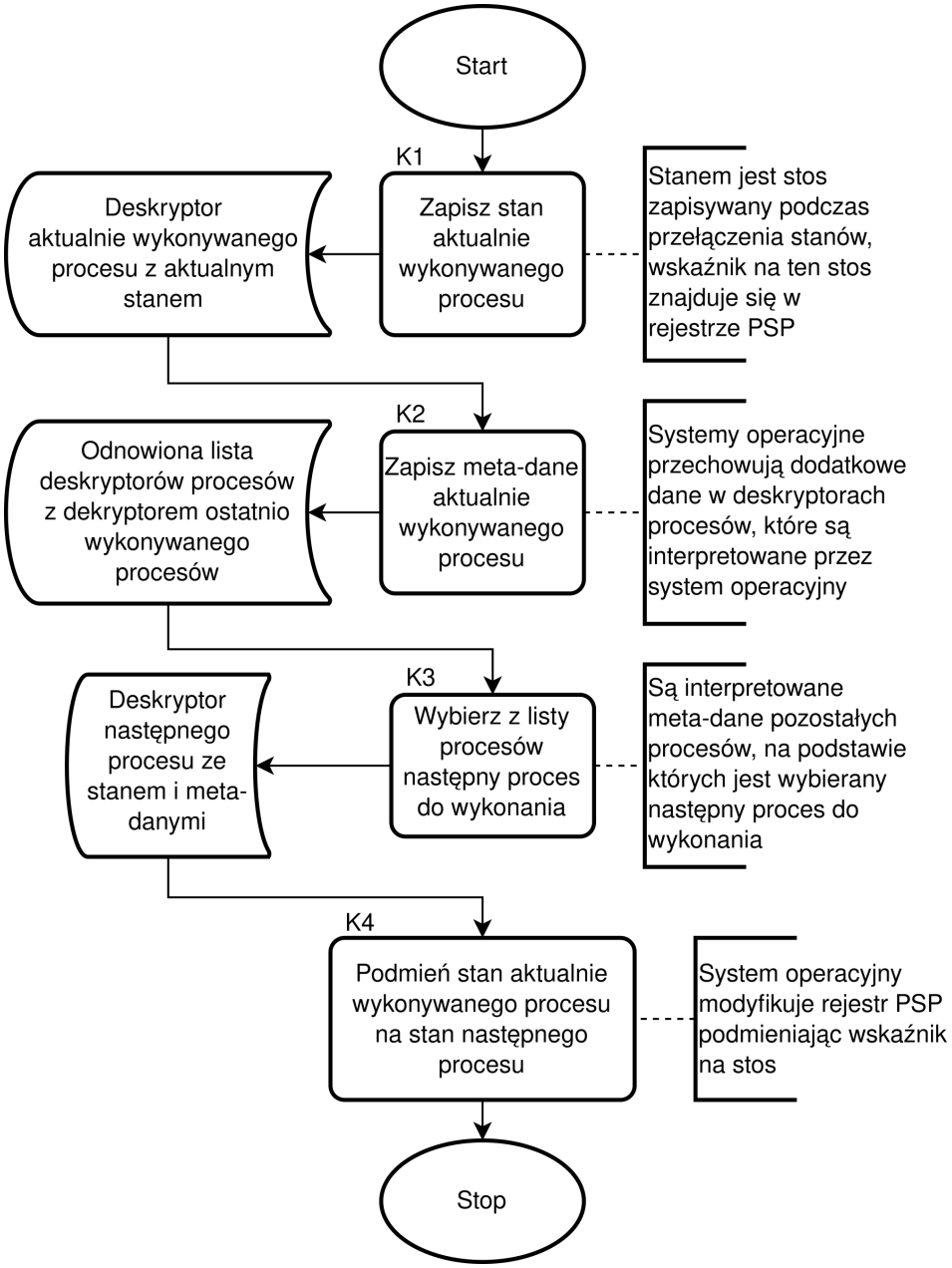
2. System operacyjny otrzymuję od hiperwizora pewną część zasobów jednostki obliczeniowej (strzałka nr 2 na rys. 5) i rozprowadza je do serwisów lub aplikacji (strzałki o wspólnym numerze 4 na rys. 5), spędzając na ten proces pewną część otrzymanych zasobów;
3. Aplikacja, która jest uruchomiona bez systemu operacyjnego otrzymuje przydzielone zasoby od hiperwizora (strzałka nr 3 na rys. 5) i zaczyna wykonywać swoją pracę;
4. Aplikację i serwisy, uruchomione przez system operacyjny, otrzymują przydzielone im zasoby przez system operacyjny (strzałki o wspólnym numerze 4 na rys. 5) i zaczynają wykonywać swoją pracę.

W systemie operacyjnym funkcjonalność przydzielenia zasobów jednostki obliczeniowej jest jedna z trzech kluczowych (pozostałe dwie: komunikacja międzyprocesowa i zarządzanie pamięcią). Natomiast w hiperwizorze jest to jedna ze sposobów separacji (pozostałe dwie: separacja sygnałowa i separacja pamięciowa; załącznik nr 3).

Logikę lub kod, który odpowiada za zarządzanie zasobami jednostki obliczeniowej zarówno w systemie operacyjnym jak i hiperwizorze (pomiędzy strzałkami 1 i 2, oraz pomiędzy strzałkami 2 i 4, rys. 6) można nazwać nadzorcą (ang. supervisor). Takie nazewnictwo będzie stosowane w dalszej części pracy. Nadzorcę można także podzielić na dwie części: dyspozytora (ang. dispatcher) i planistę (ang. scheduler).

Centrum uwagi tej pracy jest planista.

3.2 Wyodrębnienie



Rysunek 6: Ogólna logika przełączenia kontekstu

Wspomniany w poprzedniej części nadzorca jest wyłowy łany tylko w pewnych, szczególnie zdefiniowanych przypadkach, aby wykonać przełączenie kontekstu. Wywoływany on może być

periodycznie, za pomocą periodycznego przerwania, lub ”na żądanie” za pomocą przerwania systemowego (w przypadku ARM’u jest to przerwanie generowane instrukcjami SVC i HVC).

Po wywołaniu nadzorcy następuje przełączenie kontekstu. Na rys. 6 są ogólnie opisane kroki przełączania kontekstu. Przy czym kroki K1, K2 i K4 (rys. 6) są wykonywane przez dyspozytora, i tylko krok K3 (rys. 6) jest wykonywany przez planistę.

Zadaniem dyspozytora jest otrzymywanie informacji i przekierowanie jej w odpowiednim formacie do planisty (kroki K1 i K2 na rys. 6), który podejmuje decyzję na podstawie zdefiniowanych reguł (krok K3 na rys. 6), i informuje o tym dyspozytora. Dyspozytor podejmuje odpowiednie dla podjętej decyzji działania (krok K4 na rys. 6).

Więc wejściem do planisty jest lista deskryptorów procesów oczekujących na przydzielenie zasobów obliczeniowych oraz ilość dostępnych zasobów obliczeniowych. Wyjściem jest deskryptor procesu z przydzielonymi zasobami jednostki obliczeniowej. Planista podejmuję decyzję na podstawie wstępnie zdefiniowanych reguł.

4 Analiza

W tej części pracy będzie szczególnie omówiona i przeanalizowana jedna część nadzorcy — planista, będą przedstawione i przeanalizowane algorytmy zarządzania zasobami jednostki obliczeniowe realizowane przez planistę.

Pojęcie zasobu jednostki obliczeniowej jest wyjaśnione w załączniku nr 5.

4.1 Klasyfikacja algorytmów

Podsumowanie

Lorem ipsum dolor sit amet, consectetuer 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, consectetuer 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.

[16] *Wieloprocesorowość asynchroniczna*. wikipedia.org. URL: https://pl.wikipedia.org/wiki/Wieloprocesorowo%C5%9B%C4%87_asynchroniczna.

[17] *Wieloprocesorowość symetryczna*. wikipedia.org. URL: https://pl.wikipedia.org/wiki/Wieloprocesorowo%C5%9B%C4%87_symetryczna.

Załącznik nr 1: System wbudowany

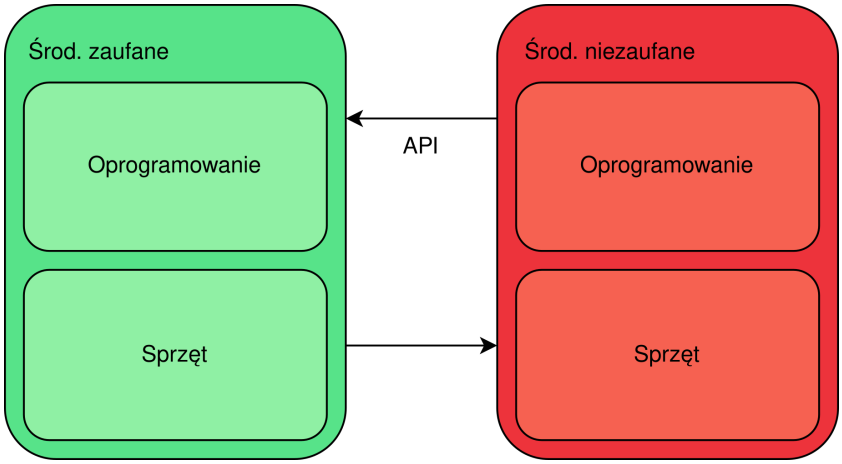
Skoro nie ma jednoznacznej definicji pojęcia ”system wbudowany” - należy określić definicję, która jest wykorzystywana w tej pracę.

System wbudowany jest systemem, to znaczy ma cechy systemu: jest złożony z więcej niż jednego elementu, jest funkcjonalnie niezależny od 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.

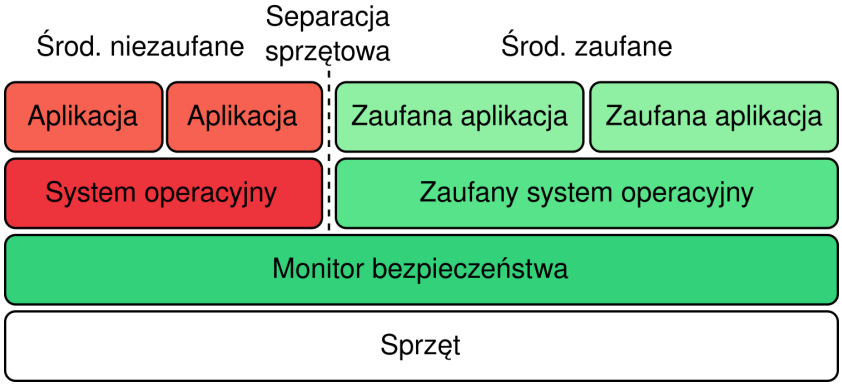
Natomiast przełożyć tą definicję na dziedzinę elektroniki i informatyki można w następujący sposób: system wbudowany — jest to system komputerowy składający się z jednostki obliczeniowej i modułów wejścia/wyjścia (tzn. może reagować na bodźce zewnętrzne), mający wszystkie narzędzia (w tym oprogramowanie)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

Załącznik nr 2: ARM TrustZone

Koncept technologi TrustZone bazuję się na podziale sprzętowym środowiska (pamięci i innych zasobów) na środowisko zaufane i niezaufane (rys. 7 i rys. 8), co jest robione za pomocą rozszerzeń do architektur ARMv8-A i ARMv8-M dotyczących BSA i ISA.



Rysunek 7: TrustZone dla architektury ARM v8-M. Kolorami są dodatkowo odznaczone środowiska: zielony — środowisko zaufane, czerwony — środowisko niezaufane



Rysunek 8: TrustZone dla architektury ARM v8-A. Kolorami są dodatkowo odznaczone środowiska: zielony — środowisko zaufane, czerwony — środowisko niezaufane

Dla architektury ARMv8-A został dodany EL3, w którym jest zamieszczony monitor bezpieczeństwa (rys. 8), 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. 7) za pomocą modułów SAU i IDAU, oraz instrukcji dla komunikacji pomiędzy środowiskami: ISA SG, BXNS i BLXNS. [15]

Załącznik nr 3: Wirtualizacja

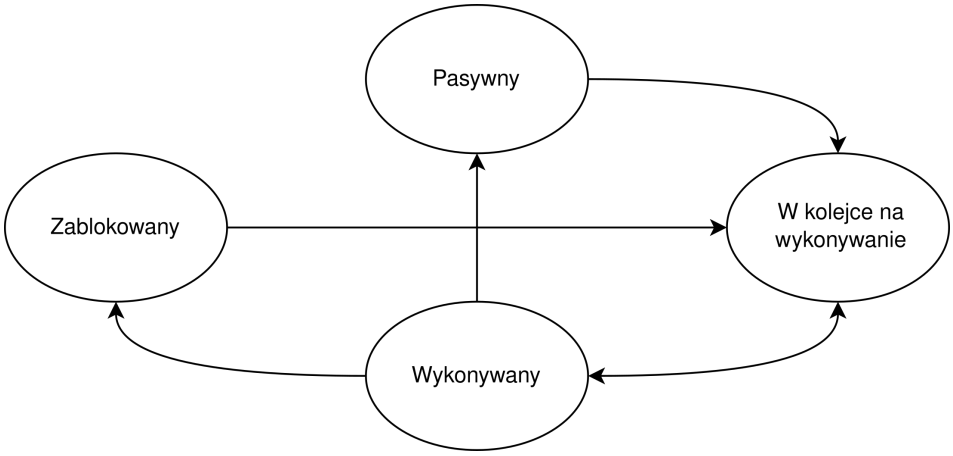
Wirtualizacja jest technologią wprowadzającą separację środowisk z oprogramowaniem poprzez tworzenie tzw. VM zarządzanych przez VMM (rys. 3). Typy separacji używane w wirtualizacji: separacja pamięciowa (poprzez translację pamięci), separacja sygnałowa (przekierowywanie przerwań) i separacja stanów (przełączanie 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 przerwań sprzętowych pomiędzy VM. Każda VM ma przepisany podzbiór urządzeń peryferyjnych za pomocą separacji pamięci, te urządzenia mogą generować przerwania, za przekierowywanie których jest odpowiedzialny hiperwizor i kontroler przerwań (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.

Przy czym, typowo hiperwizor jest zaprojektowany w taki sposób, że przy użyciu wszystkich tych rodzai separacji, aplikacja (lub system operacyjny), uruchomiana na tym hiperwizorze, wcale nie musi wiedzieć że jest uruchomiana nie wprost na rzeczywistym sprzęcie. Taki rodzaj wirtualizacji jest nazywany wirtualizacja natywną (ang. native virtualization, także ang. full virtualization, lub ang. hardware-accelerated virtualization).



Rysunek 9: Ogólny graf stanów procesu

W przypadku, kiedy sprzęt nie posiada dedykowanych narzędzi do zapewnienia jakiegoś z rodzajów separacji, można skorzystać z tzw. parawirtualizacji (ang. paravirtualization). W takim przypadku aplikacja, która jest uruchamiana na hiperwizorze jest modyfikowana w sposób taki, aby ona wiedziała, że jest uruchomiona nie na realnym sprzęcie a na wirtualnym i wiedziała o obecności hiperwizora. Przykładem może być separacja pamięciowa. Typowo próba otrzymania dostępu do zabronionej sekcji pamięci RAM jest wykrywana przez MMU, w przypadku nieobecności tego modułu, uruchamiana aplikacja jest modyfikowana i wszystkie instrukcje odczytujące coś z pamięci są podmieniane na zestaw instrukcji, który powoduje komunikat do hiperwizora ze sprawdzeniem dostępów.

Załącznik nr 4: Przełączanie stanu i kontekstu, definicja procesu

Na początku należy oddzielić pojęcie programu od pojęcia procesu. Programem, w przypadku oprogramowania, jest lista instrukcji do wykonania przechowywana w jakimkolwiek miejscu. Program nie posiada, żadnego stanu, on po prostu istnieje. Natomiast procesem jest program z przydzielonym stanem. Każdy stan procesu może zawierać dodatkowe dane, np. priorytet, zasoby, połączenia z innymi procesami i tak dalej. Na rys. 9 jest pokazany najbardziej popularne stany procesu i przejścia pomiędzy nimi.

Stan wykonywanego oprogramowania jest określany poprzez następujące elementy:

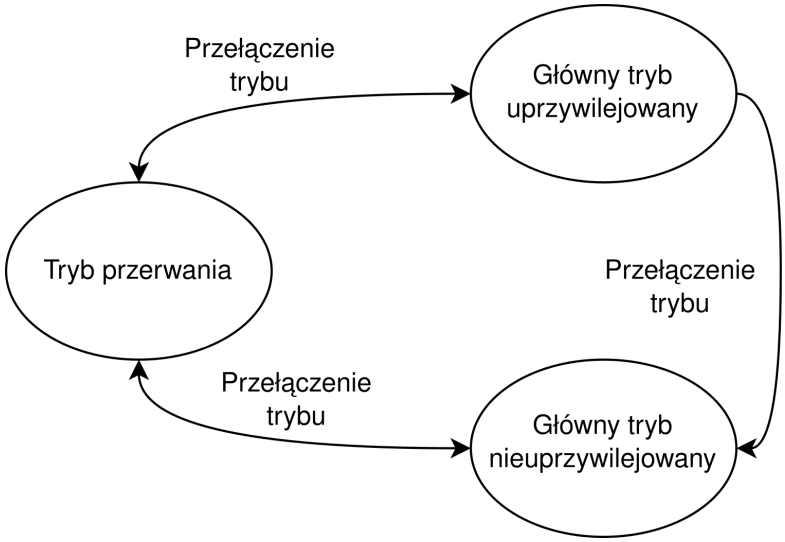
- Zawartości rejestrów ogólnego przeznaczenia: zawierają wyniki wykonywania ostatnich instrukcji;
- Rejestry wskazujące na adres następnej komendy (PC w architekturach ARM'owych), adres powrotu (LR w architekturach ARM'owych) i adres aktualnie używanego stosu (MSP lub PSP w architekturach ARM): określają "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.

Na rys. 10 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).



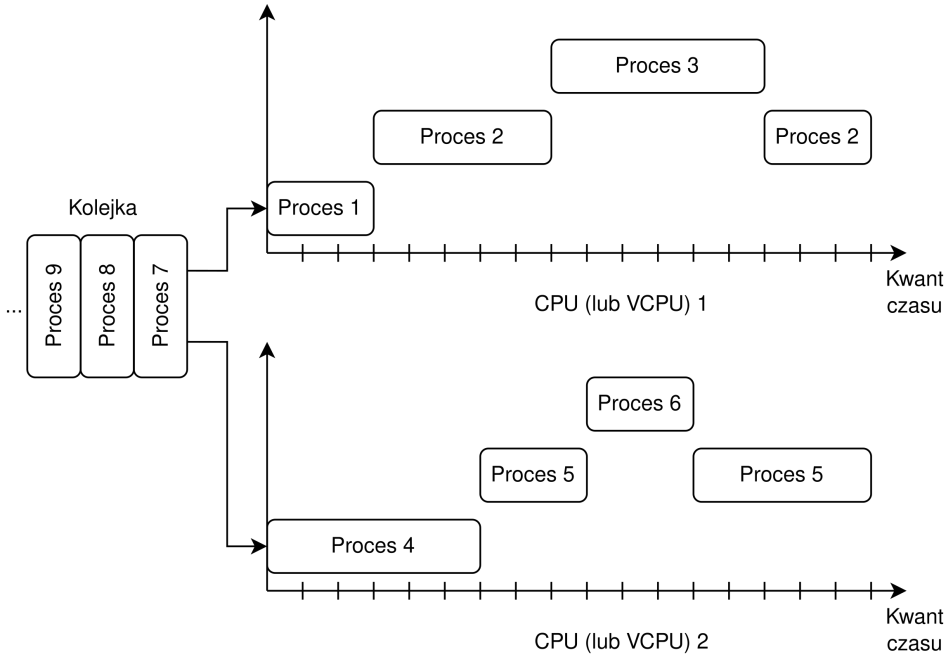
Rysunek 10: Tryby procesora zbudowanego na podstawie architektury ARMv8-M [7]

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

Na rys. 6 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. 10), 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. 10). 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).

Załącznik nr 5: Zasób jednostki obliczeniowej

Jednostka obliczeniowa jest pewnym systemem przyjmującym instrukcje, wykonującym je i podającym wynik. Więc podstawową wielkością będzie czas wykonania jednej instrukcji (w przypadku architektury RISC większość instrukcji są wykonywane w przedziale jednego okresu



Rysunek 11: Współdielenie zasobów jednostek obliczeniowych przez wiele procesów w technologii AMP

zegara zasilającego jednostkę obliczeniową).

Każdy program wykonywany przez jednostkę obliczeniową jest listą instrukcji, i każdemu programu zależy, aby ta list instrukcji była wykonana jak najszybciej, czyli idealnie jakby jedna jednostka obliczeniowa wykonywała tylko jeden program. Z tego wynika, że programy walczą o czas, na który im pozwolono używać jednostki obliczeniową do wykonania swoich instrukcji. Finalnie mamy — zasobem jednostki obliczeniowej jest czas.

Czas ten można podzielić pomiędzy programami w różny sposób, za to akurat odpowiada nadzorca w oprogramowaniu systemowym.

W przypadku kiedy mamy kilka jednostek obliczeniowych dostępnych dla wszystkich programów, zorganizować zarządzaniem ich zasobów można na dwa sposoby: przydzielać każdemu programu czas oddzielnej jednostki obliczeniowej (techniki AMP) lub połączyć zasoby wszystkich jednostek obliczeniowych i przydzielać jednemu programu pewną część (techniki SMP).

W przypadku wirtualizacji programom są udostępniane VCPU. Są to w pewny sposób przerobione przez hiperwizor zasoby realnej jednostki obliczeniowej.

Na rys. 11 jest pokazany przykład współdzielenia zasobów jednostek obliczeniowych przez

wiele procesów technologii AMP. Każdemu procesowi jest przypisana pewna część zasobów obliczeniowych reprozentowana przez ilość kwantów czasu (ang. time slice; najmniejsza możliwa ilość czasu jednostki obliczeniowej, którą można przypisać procesu w niektórych realizacja oprogramowania systemowego).

Stąd można zrobić trzy wnioski:

1. Zasobem jednostki obliczeniowej jest jej czas;
2. Czas może być przypisywany bezpośrednio, lub on może być podzielony przez oprogramowanie systemowe na pewne minimalne odcinki: kwanty czasu;
3. W przypadku obecności wielu jednostek obliczeniowych, zarządzanie ich zasobami odbywa się zgodnie z technikami AMP lub SMP.