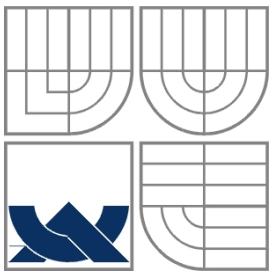
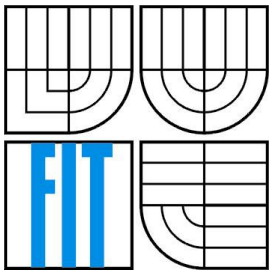


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

OPERAČNÍ SYSTÉM PRO ŘÍZENÍ VESTAVĚNÝCH APLIKACÍ

OPERATING SYSTEM FOR EMBEDDED APPLICATIONS CONTROL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — TOMÁŠ KOLARÍK
AUTHOR

VEDOUCÍ PRÁCE — Ing. VÁCLAV ŠIMEK
SUPERVISOR

BRNO 2009

Abstrakt

Práce se zabývá výběrem operačního systému pro vestavěnou řídící jednotku parního sterilizátoru obsluhující barevný displej s dotekovým panelem. Na teoretické úrovni se zabývá architekturou vestavěných systémů a operačních systémů. U nich je pozornost soustředěna na správu procesů. Dále se zabývá přístrojem, pro nějž je vybírán operační systém, důvody pro jeho zavedení, jeho výhodami a nevýhodami v řídícím systému. Pak je součástí práce přehled a rozdělení OS do kategorií podle způsobu použití. A blíže jsou popsány dva open source operační systémy eCos a FreeRTOS. Z nich byl následně kód FreeRTOSe upraven z překladače GCC na překladač IAR. Poslední část práce popisuje architekturu a funkcionalitu vytvořené ukázkové aplikace. Ta pomocí dotekového panelu a barevného displeje umožňuje kreslit barevné nákresy, které se dají pomocí sériové komunikace přenášet do počítače.

Abstract

This thesis deals with selection of operating system for embedded control unit of steam sterilizer with color touch screen. Theoretical part of the thesis deals with the architecture of embedded systems and operating systems with focus on process management. In addition, the thesis deals with a device for which the operating system is selected, the reason for its implementation, advantages and disadvantages in the control system. The thesis also contains categorization of operating systems by way of use. Of these, two open source operating systems eCos and FreeRTOS are described more thoroughly. The code of FreeRTOS was modified from GCC compiler to IAR compiler. The last part of the thesis describes architecture and functionality of the created demo application. Using the touch screen and color display, the application enables drawing of color schemes, which can be transferred to a computer using serial communication.

Klíčová slova

vestavěné systémy, operační systémy, parní sterilizátory, real-time systémy, dotekový panel, barevný displej, multitasking, plánování procesů

Keywords

embedded systems, operating systems, steam sterilizers, real-time systems, touch panel, colour display, multitasking, process scheduling

Citace

Kolarík Tomáš: Operační systém pro řízení vestavěných aplikací, bakalářská práce, Brno, FIT VUT v Brně, 2009

Operační systém pro řízení vestavěných aplikací

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Václava Šimka.
Další informace a prostředky mi poskytla firma BMT Medical Technology s.r.o.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Tomáš Kolarík
19.5.2009

Poděkování

Rád bych poděkoval firmě BMT Medical Technology s.r.o. za vstřícný přístup a za všechny prostředky, které mi k dokončení práce poskytla. Dále chci poděkovat vedoucímu bakalářské práce Ing. Václavu Šimkovi za pomoc a rady poskytnuté v průběhu psaní práce.

Obsah

1 Úvod

Ambiciózní projekt Multics v roce 1963, kdy započal svoji existenci v laboratořích MIT, neslavně odstartoval historický vývoj specializovaného softwaru nazvaného operační systém. Od jeho původního uplatnění v sálových počítačích se postupně přenášel na nižší systémy. Stal neodmyslitelnou součástí serverů a osobních počítačů. Poslední místo uplatnění našel operační systém ve vestavěných systémech, které nás dnes doslova obklopují. Vestavěné systémy najdeme doma v různé spotřební elektronice, v továrnách řídí výrobní stroje, na silnicích se starají o bezpečnou jízdu automobilů… S klesající cenou hardwaru a jeho stoupajícím výkonem je čím dál tím častější uplatnění vestavěných systému v různých oblastech života a spolu s jejich rozšiřováním se rozšiřuje používaní vestavěných operačních systémů.

Vzhledem k různorodosti uplatnění vestavěných systému je různorodé taky jejich hardwarové vybavení a periferie. Nejběžněji používanými periferiemi vestavěných systémů jsou spínače, přepínače, A/D a D/A převodníky. Z programátorského hlediska se tady pracuje s nižšími programovacími jazyky a často se programují obslužní rutiny přímo pro příslušný hardware, avšak u osobních počítačů je hardware dobře podporován operačními systémy, které poskytují efektivní nástroje pro jejich používaní. Proto je programování aplikací pro PC mnohem snadnější. Dalším rozdílem je, že u vestavěných aplikací není tak patrná snaha o zvyšování výkonu jako u osobních počítačích a serverů, spíše se při jejich výběru přihlíží na nízkou spotřebu energie, cenu, rozměry či hmotnost.

V této práci se podíváme na možnosti, které přináší operační systémy při programovaní a řízení vestavěných aplikací. Jaké jsou možnosti výběru vestavěných operačních systémů a implementujeme jeden z nich na konkrétní specifickou elektroniku určenou pro parní sterilizátory. Tyto přístroje jsou určeny hlavně pro zdravotnická zařízení a laboratoře, kde se vyžaduje kvalitní rychlá sterilizace. Dají se v nich sterilizovat předměty z různých materiálů a také kapaliny. Parní sterilizátory se vyrábí v různých velikostech. V této práci se budeme zmiňovat o řade malých parních sterilizátorů.

Úvodní část práce je věnovaná definici, popisu a jednotlivým částem vestavěných systému, kde je větší pozornost zaměřená na popis jejich softwaru. Další kapitola na tento softwarový úvod navazuje, jsou tam podrobněji rozebrány některé vlastnosti operačních systémů. Protože má být hlavním přínosem při zavedení operačního systému do softwaru parních sterilizátorů multitasking, je mu věnována značná pozornost. Tyto informace o operačních systémech jsou dál využity při hledání vhodného operačního systému a jeho pozdější úpravě při přenášení na hardware sterilizátoru. Zbylá část práce obsahuje popis funkčnosti a vlastností vytvořené ukázkové aplikace nad vybraným systémem. Aplikace byla navržena s úmyslem předvedení funkčnosti plně barevného displeje s analogovým dotekovým panelem. Tak byla pro tento účel vytvořená jednoduchá kreslící aplikace s možností změny barvy kreslícího pera. Byla také rozšířená o komunikaci umožňující vytvořené kresby přenášet do počítače a tam uchovávat. Další důvod implementace komunikace byl také v předvedení paralelního běhu procesů a v nezanedbatelné míře byla přínosná při ladění programu. Závěr je věnován hodnocení výsledků práce a návrhu vylepšení operačního systému FreeRTOS, jako námět na pokračovaní a rozšiřování této práce.

2 Vestavěné systémy

Oblast vestavěných systému je široká a různorodá a nemožno ji jednoduše definovat nebo popsat. V této kapitole bude uvedeno několik možných definicí vestavěných systému a bude představen model, který je možné použít na popis každého vestavěného systému.

Vestavěný systém je [2] aplikovaný výpočetní systém, odlišný od ostatních typů výpočetních systémů jako osobní počítače nebo superpočítače. Napříč této jasné odlišnosti se vestavěné systémy obtížně definují pro jejich neustály vývoj a pokrok v technologiích, který je způsoben dramatickým poklesem cen implementace hardwarových a také softwarových komponent. V posledních letech tak jejich možnosti přesáhly jejich tradiční definice. Dále jsou uvedené běžné definice vestavěných systémů:

1. Vestavěné systémy mají určitá omezení v rozsahu hardwarové anebo softwarové funkcionality než osobní počítače.
2. Vestavěný systém je navržen na vykonávaní určitého úkolu.
3. Vestavěný systém je výpočetní systém s vyššími nároky na kvalitu a spolehlivost než jiné výpočetní systémy.
4. Některé vestavěné systémy jako například PDA nejsou ve skutečnosti vestavěné systémy.

Každá z těchto definic je částečně pravdivá a definuje některou oblast vestavěných systému. Dále jsou uvedeny oblasti, kde jsou vestavěné systémy využívány:

- Automobilový průmysl: kontrola zapalování, řídící jednotky motoru, brzdové systémy…
- Spotřební elektronika: digitální a analogové televize, PDA, kuchyňské spotřebiče, GPS …
- Výroba: robotizované a kontrolní systémy
- Medicínská technika: infuzní pumpy, dialýzní zařízení, srdcové pumpy a monitory…
- Sítě: routery, huby, switche…
- Kancelářská technika: faxy, kopírky, tiskárny, monitory, skenery…

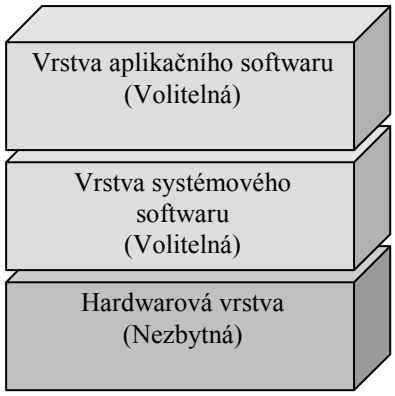
2.1 Architektura vestavěných systémů

Architektura vestavěného systému [2] je abstrakce funkcionality vestavěného zřízení. V tomto smyslu to je zevšeobecnění systému odstraněním detailní implementace jako například zdrojový kód programu nebo elektrické obvody hardwaru. Na architektonické úrovni jsou hardwarové a softwarové komponenty reprezentovány souborem navzájem provázaných prvků. Prvek nahrazuje hardware nebo software, jehož implementační detaily byly abstrahovány a bylo ponecháno jenom chováni prvku a informace o vnitřních vztazích. Architektonické elementy pak mohou být vnitřně včleněny do vestavěného zařízení, nebo existovat samostatně a jenom spolupracovat s vnitřními prvky systému. Ve zkratce vestavěná architektura obsahuje prvky vestavěného systému, elementy interagující s vestavěným systémem, vlastnosti každého elementu a interaktivní vztahy mezi elementy.

Informace v architektonickém návrhu je fyzicky reprezentována ve formě struktury, která je jednou z možných reprezentací architektury. Každá struktura je pak tvořena sadou elementů, vlastnostmi a informacemi o vnitřních vztazích. Tato struktura daná prostředím a sadou elementů tak tvoří náčrt systémového hardwaru a softwaru v čase návrhu stejně jako i v době běhu systému. Je velice obtížné jediným takovým náčrtem zachytit složitost celého systému, proto je architektura typicky tvořena více než jednou strukturou. Všechny struktury v architektuře jsou vnitřně propojeny navzájem a toto sjednocení struktur tvoří architekturu vestavěného systému.

2.2 Model vestavěného systému

Nejvyšší úroveň a zároveň základní architektonický nástroj používaný na znázornění hlavních elementů tvořících design vestavěného systému je model, který je znázorněn na obrázku 2.1.



Obrázek 2.1: Model vestavěného systému [2]

Tento model [2] naznačuje, že všechny vestavěné systémy mají společnou vlastnost na nejvyšší úrovni architektonického modelu. Každý vestavěný systém má nejméně jednu vrstvu a to hardwarovou, anebo má všechny: hardwarovou, vrstvu systémového softwaru a aplikačního softwaru. Hardwarová vrstva zahrnuje všechny hlavní fyzické komponenty umístněné na desce vestavěného zařízení, zatím co vrstva systémového a aplikačního softwaru obsahuje veškerý software, který je zpracováván pomocí hardwarové vrstvy.

Tento základní model je v podstatě vrstvová reprezentace architektury vestavěného systému, z které mohou být odvozeny standardní struktury vestavěné architektury. Bez ohledu na velké rozdíly mezi vestavěnými zařízeními je možné pochopit jejich architekturu pomocí zobrazení a sloučení komponentů těchto zařízení do vrstev. Zatím co vrstvový model není jednotný pro všechny vestavěné systémy, je užitečný na vyobrazení možných kombinací hardwarových a softwarových komponent, které mohou být použity v návrhu vestavěných systémů. Obecně existuje několik různých způsobů na vyjádření architektury a toto je jenom jeden z nich.

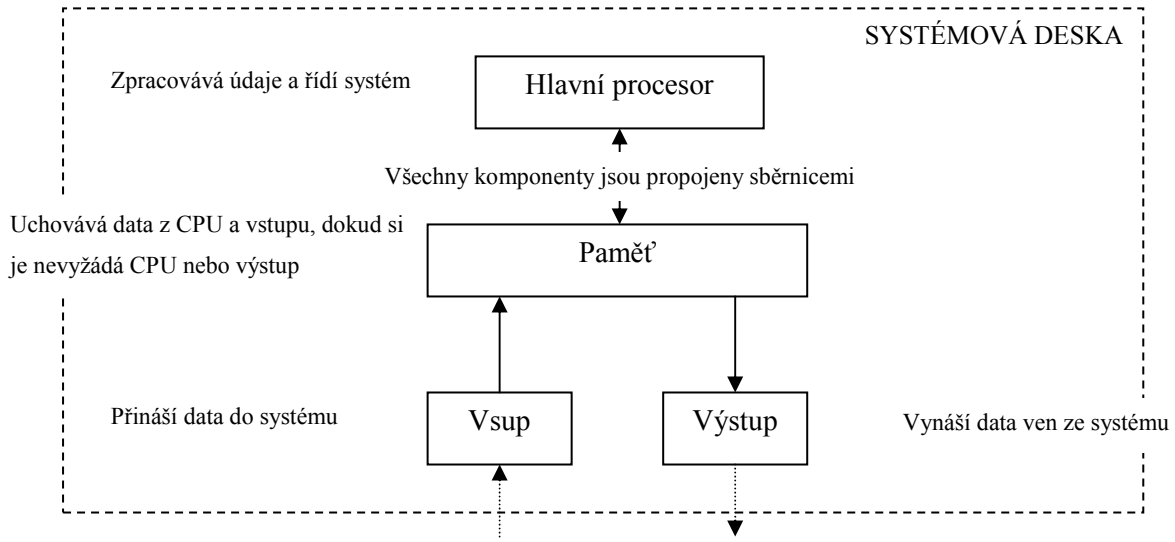
2.3 Vestavěný hardware

Ve vestavěných zařízeních je všechen hardware umístněn na jedné desce, často také označované jako plošný obvod (PO) nebo plošný spoj. PO [2] je často tvořen z tenkých vrstev sklolaminátu a elektricky vodivé části obvodu, která je tvořena vytlačenými měděnými cestami. Všechny elektrické součástky, které tvoří obvod jsou pak připojeny k této desce buď přiletováním, vložením do petice, nebo jiným přípojným mechanismem. Veškerý hardware na vestavěné desce je v modelu vestavěného systému umístěn v hardwarové vrstvě.

Na nejvyšší úrovni abstrakce je možné hlavní hardwarové komponenty většiny PO rozdělit na:

1. Centrální výpočetní jednotka (CPU) – hlavní procesor
2. Paměť – prostor pro uložení systémového softwaru a dat
3. Vstupní zařízení – vstupní pomocný procesor, nebo podobná elektrická komponenta
4. Výstupní zařízení – výstupní pomocný procesor, nebo podobná elektrická komponenta
5. Datové cesty/sběrnice – propojení dalších komponent včetně fyzických propojení, sběrnicových mostů a řadičů sběrnice.

Tyto kategorie jsou základem pro návrh systému postaveném na von Neumannovském modelu, který je na obrázku 2.2. Tento model je výsledkem publikované práce Johna von Neumanna z roku 1945, která definovala požadavky univerzálního výpočetního systému.



Obrázek 2.2: Schéma von Neumannovského modelu [1]

2.3.1 Procesor

Procesor je hlavní funkční jednotka vestavěného zařízení. Je odpovědný za zpracovávaní instrukcí a dat. Elektronické zařízení obsahuje nejméně jeden hlavní (master) procesor, chovající se jako hlavní řídící jednotka a může obsahovat přídavný (slave) procesor, který spolupracuje a je kontrolovaný master procesorem. Tento slave procesor může sloužit na rozšíření instrukční sady master procesoru, nebo mít na starosti správu pamětí, sběrnice a I/O zařízení. Slave procesor může být například ethernetový řadič.

Složitost master procesoru obvykle určuje, jestli patří do třídy mikroprocesorů, nebo mikrokontrolérů. Mikroprocesor [2] standardně obsahuje minimální integrovanou paměť a jen nezbytné I/O komponenty na čipu, avšak mikrokontrolér má většinu systémové paměti a většinu I/O komponent integrovaných přímo na čipu. Tento rozdíl se však čím dál tím víc zamlžuje kvůli zvyšovaní složitosti a komplexnosti nových mikroprocesorů.

Na trhu je dostupných několik set druhů vestavěných procesorů, ale žádný z nich nemá dominantní postavení. Všechny tyto procesory se dají rozdělit do několika skupin nazývaných také architektury. Rozdíly mezi architekturami jsou založeny na rozdílných instrukčních sadách procesorů. [2] Dva procesory jsou považovány za procesory stejné architektury, pokud jsou schopny zpracovat stejnou sadu strojových instrukcí viz. Tabulka 2.1.

Architektura	**Procesor**	**Výrobce**
AMD	Au1xxx	Advanced Micro Device, …
ColdFire	5282,5272, …	Motorola/Freescale, …
SuperH(SH)	SH3, SH4 (7750)	Hitachi, …
SPARC	UltraSPARC II	Sun Microsystems, …
x86	X86 (386,486 …)	Intel, Transmeta, Atlas, …
TriCore	TriCore1,TriCore2	Infineon, …

Tabulka 2.1: Architektury procesorů a jejich zástupci [2]

2.3.2 Paměť, I/O systém a sběrnice

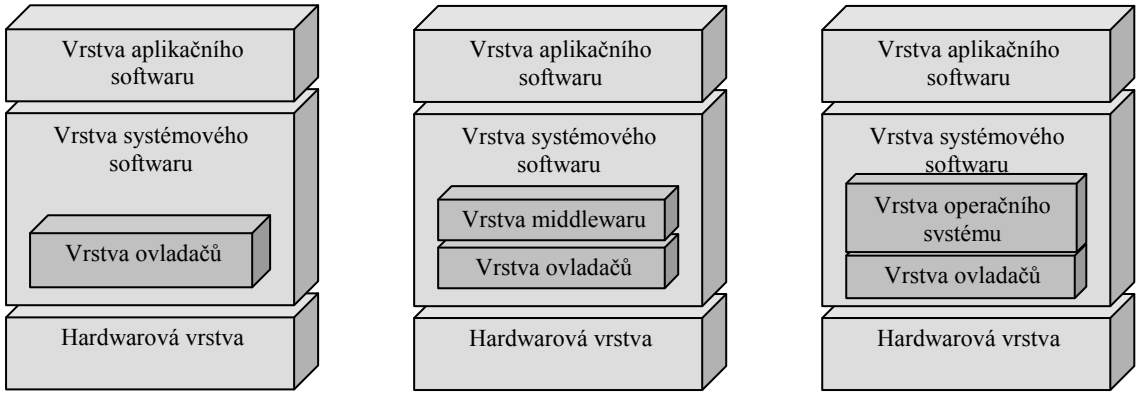
Zbylé částí sytému jako jsou paměť, I/O systém a také sběrnice tvoří podmínky pro zpracovávaní dat a tím pádem plnohodnotné fungování systému. Paměť slouží pro uchování programu a také zpracovávaných dat, zatím co I/O zařízení slouží k interakci s okolím a sběrnice umožňuje přenos informací uvnitř systému mezi mnoha prvky.

2.4 Firmware

Horní softwarové vrstvy modelu vestavěného systému mohou obsahovat různé kombinace softwarových podvrstev. Standardně se vestavěný software dá rozdělit do dvou obecných tříd: systémový software a aplikační software. Systémový software je každý software, který podporuje aplikaci jako řadiče zařízení, operační systém nebo middleware. Aplikační software [2] je vysokoúrovňový software, který definuje funkcionalitu a účel vestavěného sytému a zabezpečuje interakci s uživateli a administrátory.

2.4.1 Řadiče zařízení

Většina vestavěného hardwaru potřebuje ke svému fungování nějaký druh softwarové inicializace a řízení. Software, který přímo přistupuje k prostředkům hardwaru se jmenuje ovladač zařízení. Všechna vestavěná zařízení, která potřebují software mají ovladače svých zařízení ve vrstvě systémového softwaru. Ovladače zařízení [2] jsou softwarové knihovny, které inicializují hardware a řídí přístup k hardwaru pomocí vyšší vrstvy softwaru. Tyto ovladače tvoří spojení mezi hardwarem a operačním systémem, middlewarem nebo aplikační vrstvou jak je to znázorněno na obrázku 2.3.



Obrázek 2.3: Řadiče v architektuře vestavěných systémů [2]

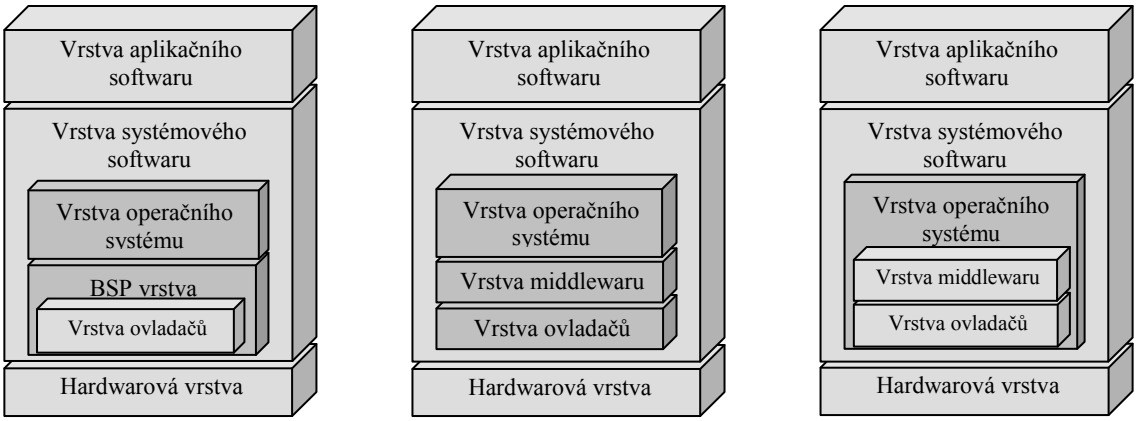
2.4.2 Middleware

Hranice mezi middlewarem a aplikací není úplně zřejmá. Middleware je software, který byl oddělen od aplikačního softwaru z několika příčin. Jeden důvod pro oddělení je, že může být integrován do operačního systému jako přídavný balík. Další důvod na oddělení od aplikační vrstvy je dosažení znovupoužitelnosti v jiných aplikacích a tím dosažení nižších nákladů, nebo ušetření času koupí tohoto softwaru, či kvůli zpřehlednění kódu.

Obecně se dá middleware definovat jako [2] veškerý software, který není jádro operačního systému, řadič zařízení, nebo aplikační software. Mnohokrát je middleware součástí spustitelné částí operačního systému. Middleware se tak typicky nachází buď nad ovladači zařízení, nebo nad operačním systémem, nebo je přímo do OS začleněn.

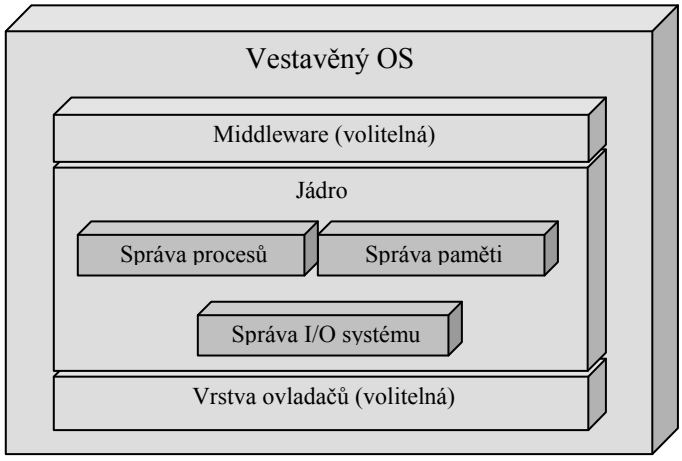
2.4.3 Operační systém

Operační systém je [1] volitelná část vestavěného softwaru, proto ne každý vestavěný systém jej má. OS může být použit na jakémkoli procesoru (ISA), na který byl operační systém programově přenesen. Na obrázku 2.4 je vyobrazeno možné umístnění OS v architektuře vestavěného systému.



Obrázek 2.4: Operační systém v architektuře vestavěných systémů [2]

OS je [1] software, který plní dva základní úkoly. Představuje abstrakční vrstvu pro vrchní část softwaru, aby byl co nejmíň závislý na použitém hardwaru, aby byl vývoj middlewaru a aplikačního softwaru umístněným nad OS snadnější. A zároveň spravuje hardwarové a softwarové prostředky a zajišťuje aby byly využívány efektivně a bezpečně. V oblasti vestavěných systému se vzhledem k různorodosti používaného hardwaru vyvinuly taky OS s hodně rozdílným rozsahem těchto výše popsaných vlastností. Jediná část, ve které se tyto systémy moc neliší je jádro. Některé OS jsou dokonce tvořeny pouze jádrem. Jádro je část, která obsahuje základní funkcionalitu OS. Pozůstává ze všech, nebo kombinaci některých z následujících funkcí. Jejich umístění v architektuře OS je znázorněno na obrázku 2.5.



Obrázek 2.5: Stavba operačního sytému [2]

Části jádra OS:

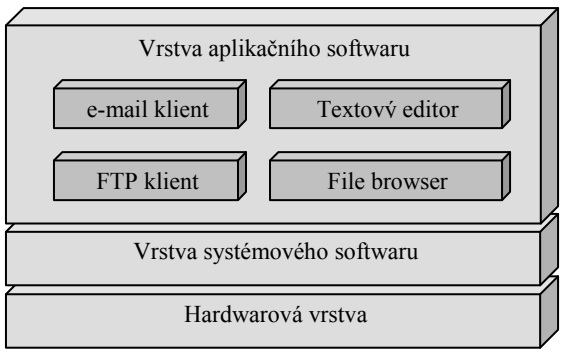
- Správa procesů – prostředek, kterým OS řídí a přistupuje k ostatnímu softwaru ve vestavěném systému a přiděluje mu procesor. Související funkce, která je často současí správce procesů, je detekce přerušení a chyb.
- Správa paměti – paměťový prostor vestavěného systému je sdílen všemi procesy a také operačním systémem. Proto přístup a také alokace paměti musí být řízená. Součástí správce paměti může být ochrana paměti, která zabezpečuje systém před nebezpečným, nebo špatně napsaným aplikačním softwarem.
- Správa I/O systému – vstupně/výstupní zařízení se sdílí mezi více procesy, a proto je nutno zabezpečovat řízení přístupu a alokace těchto prostředků.

S dělením na výpočetní, úložní a vstupně výstupní části jsme se setkali také u von Neumannovi architektury, která však popisovala fyzické části systému narozdíl od jádra OS. Tím, že jádro OS zabezpečuje správu všech těchto částí, umožňuje jejich bezpečné sdílení mezi několika procesy, z kterých může být tvořen aplikační software.

2.4.4 Aplikační software

Nejvrchnější částí architektury vestavěného zařízení je vrstva aplikačního softwaru neboli aplikace. Tato vrstva je umístěná nad systémovým softwarem, na kterém je aplikační vrstva závislá. Systémový software řídí aplikaci a vytváří podmínky pro její běh. Aplikační vrstva určuje, o jaké zařízení se vlastně jedná, protože funkcionalita firmwaru použitého ve vestavěném zařízení je dána právě aplikačním softwarem. Ten definuje chování systému na různé podněty a má na starosti většinu interakcí s uživatelem nebo administrátorem, pokud je systém určen pro interakci s člověkem.

Aplikační software [2] můžeme rozdělit stejně jako vestavěné systémy na aplikační software se specifickým zaměřením jako například aplikační software pro digitální televizi a univerzální aplikační software například FTP klient, který může být použitý u mnoha druhů zařízení. Příklad obsahu aplikační vrstvy je na obrázku 2.6.



Obrázek 2.6: Aplikace v architektuře vestavěných systémů [2]

3 Operační systémy

Operační systém [3] vytváří prostředí, ve kterém jsou spouštěny aplikace. Vnitřně se liší hlavně v uspořádaní vnitřních struktur. Design nových operačních systémů je zaměřen hlavně na správu úkolů. Na operační sytém se dá nahlížet z různých úhlů pohledu. Z jednoho úhlu je možno vidět výhody služeb, které systém poskytuje, z dalšího užitečné programátorské a uživatelské rozhraní, ze třetího vnitřní složky a jejich propojení.

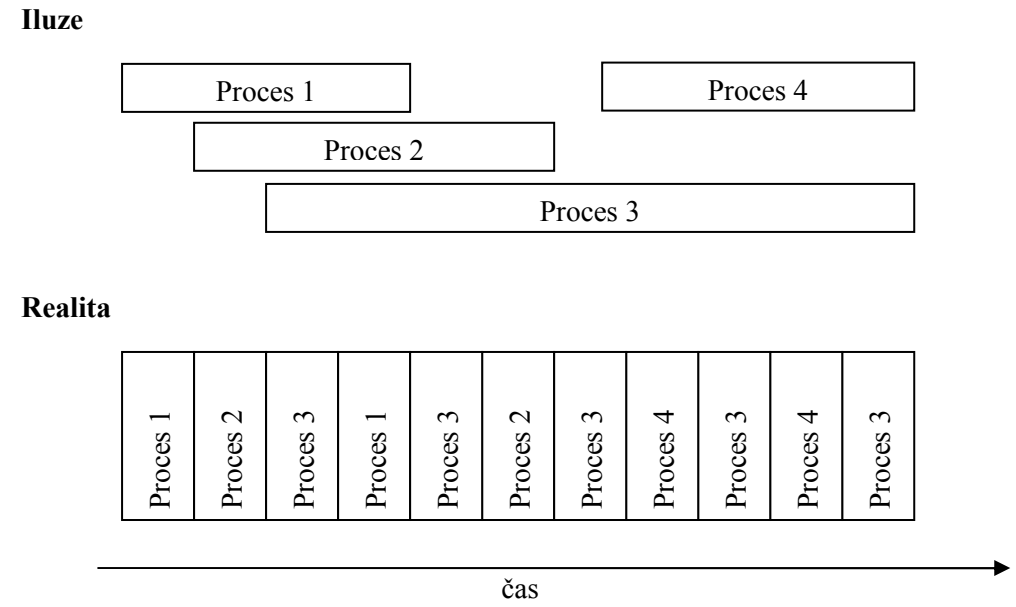
3.1 Proces

Operační systém rozlišuje mezi programem a chodem programu (executing). Program [1] je jednoduše pasivní statická sekvence příkazů. Až samotné spuštění programu je aktivní dynamická událost, při které se mění jednotlivé prvky v čase a vykonávají se instrukce. Proces, také u mnoha operačních systémů označován jako úkol, je vytvořen operačním systémem pro zapouzdření informací, které jsou zapotřebí pro spuštění programu jako například zásobník, programový čítač, zdrojový kód, data atd. To znamená, že program je pouze část úkolu.

Vestavěné operační systémy [1] spravují veškerý vestavěný software použitím úkolů, a proto je rozdělujeme na jednoúkolové a vícúkolové (multitasking). V jednoúkolových OS může existovat pouze jeden proces, nýbrž ve víceúkolových může současně existovat více procesů. Koexistence několika procesů najednou vyžaduje komplexnější správu procesů, aby byla zabezpečena nezávislost jednotlivých procesů až na případy, kdy to je žádoucí (komunikace procesů). Víceúkolové prostředí poskytuje lepší podmínky pro organizaci vývoje komplexních vestavěných systémů. V takovém systému jsou jednotlivé funkce separátně rozděleny na několik menších úkolů nebo stejná činnost může běžet najednou několikrát.

3.2 Multitasking

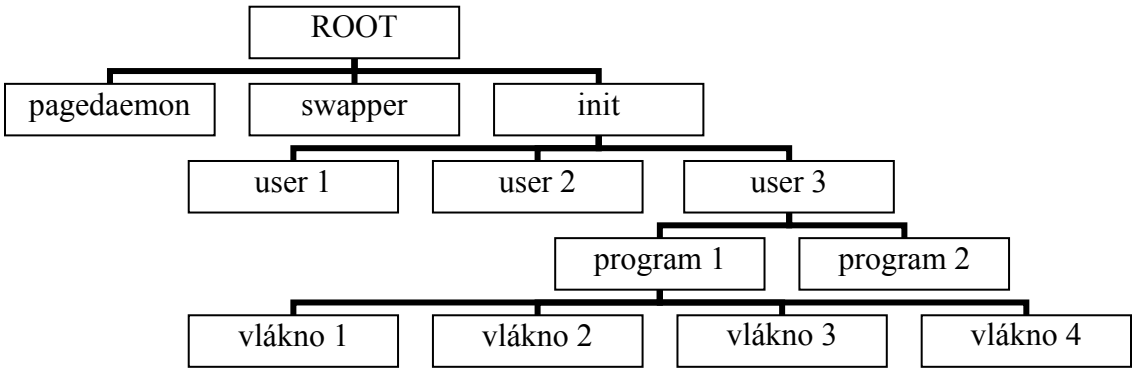
Víceúkolové (multitasking) operační systémy [1] vyžadují přídavný mechanismus na rozdíl od jednoúkolových systémům kvůli potřebě vytváření dojmu souběžně běžících procesů. Největším problémem je, že procesor dokáže najednou vykonávat pouze jeden proces, nebo vlákno, a tím pádem je zapotřebí hledat způsoby vytváření paralelních úkolů, rozdělování procesorového času mezi ně jako je to znázorněno na obrázku 3.1, jejich synchronizaci a umožnit komunikaci mezi nimi. Vyřešením těchto problémů dokáže OS vytvářet iluzi souběžně běžících úkolů na jednom procesoru.



Obrázek 3.1: Běh více procesů na jednom procesoru

Procesy ve většině operačních systémech mohou být spuštěny souběžně a mohou být za běhu vytvářeny a rušeny, proto musí OS mít mechanismus na vytváření a rušení procesů.

Každý proces za běhu může vytvořit několik dalších procesů přes systémové volání. Tvořící proces [1] je označován jako rodič (parent) a nově vytvořený proces se označuje jako potomek (child) tohoto procesu. Každý takový proces může vytvářet další procesy obdobným způsobem. Tak vzniká strom procesů, který v případě Uniových systému může vypadat jako je znázorněno na obrázku 3.2.



Obrázek 3.2: Strom procesů v operačním systému Linux

3.2.1 Vytváření procesů

Ve vestavěných OS jsou dvě nejčastější implementace tvoření úkolů. [1] První pomocí příkazů fork/exec, které vychází z POSIX 1003.1 standardu. Jiný způsob je použití příkazu spawn, který je odvozen od fork/exec modelu. Po použití jednoho z těchto systémových volání OS převezme kontrolu nad procesorem a vytvoří Task Control Block (TCB), označován také jako Process Control Block (PCB), čili strukturu pro řízení procesu. V některých OS tato struktura obsahuje systémové informace jako ID úkolu, stav úkolu, prioritu, indikátor chyb a CPU kontextové informace, jako jsou obsah registrů jednotlivých úkolů. V tomto momentu je paměť pro nový úkol alokována včetně jeho TCB,

jeho spouštěcích parametrů a samotného kódu, který bude tento proces vykonávat. Potom co je úkol připraven k běhu, systémové volání se ukončí a OS vrátí kontrolu zpátky přerušenému procesu.

Hlavní rozdíl mezi fork/exec a spawn modelem vytváření úkolů je ve způsobu alokace paměti pro nový proces. Při volání fork/exec systémových volání fork vytvoří kopii paměti rodičovského procesu. To umožňuje child procesu zdědit různé vlastnosti jako programový kód a proměnné od rodiče. Protože byla zkopírovaná celá paměť rodičovského procesu, existují v paměti také dvě kopie rodičovského programového kódu. Následně může child zavolat systémové volání exec, kterým se odloučí od programového kódu rodičovského procesu a „překryje“ ho programovým kódem pro potomka určeným.

Spawn model vytváření nových procesů na druhou stranu tvoří úplně nový adresný prostor pro child proces. Spawn systémové volání dává možnost definovat vstupní argumenty a také programový kód pro potomka. To umožňuje, aby byl programový kód childu vykonáván okamžitě po vytvoření.

Obě metody mají své výhody a nevýhody. Výhoda spawn metody je v neduplikování paměťového prostoru, který je následně nutné smazat a nahradit jiným programem. Avšak fork/exec řešení přináší snadnou možnost dědění vlastností od rodičovského procesu a následné širší možnosti změny programového prostředí potomka.

3.2.2 Rušení procesů

Proces se sám zruší po vykonání svého programu zavoláním služby operačního systému, která zabezpečí smazání procesu (např. systémovým voláním exit). V tomto momentu může proces svému rodiči předat návratovou hodnotu. Rodič si ji pak může převzít pomocí jiného systémového volání (např. wait). Všechny prostředky jako fyzická a virtuální paměť, otevřené soubory, komunikační buffery procesu jsou následně uvolněny, a tak jsou k dispozici jiným procesům. Ukončení procesu může také způsobit jiný proces, nejčastěji je to však v OS umožněno pouze rodičovským procesům, aby se zabránilo zneužití této funkce.

3.2.3 Stavy procesů

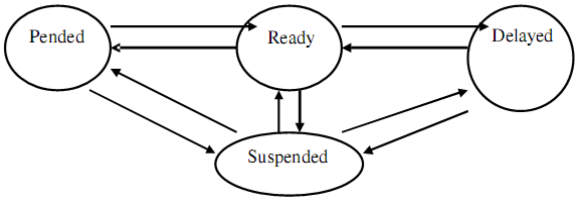
Během existence procesu a jeho spouštění se mění jeho stav. Stav procesu je definován jeho momentální aktivitou [1]. Každý proces může být v jednom z těchto stavů:

- Nový – proces byl právě vytvořen.
- Běžící – vykonává se programový kód tohoto procesu.
- Čekající – proces čeká na nějakou událost.
- Připraven – proces čeká na přidělení procesoru.
- Ukončen – proces skončil svoji činnost.

Tyto jména a počet stavů se může u různých OS lišit viz. Tabulka 3.1, 3.2. Důležité však je, že v jednom momentu může být běžící jen jeden proces na jednom procesoru, avšak běžících a čekajících procesů může být mnoho. Na obrazcích 3.3, 3,4 jsou stavové diagramy dvou OS.

State	Description
STATE + 1	The state of the task with an inherited priority
READY	Task in READY state
DELAY	Task in BLOCKED state for a specific time period
SUSPEND	Task is BLOCKED usually used for debugging
DELAY + S	Task is in 2 states: DELAY & SUSPEND
PEND	Task in BLOCKED state due to a busy resource
PEND + S	Task is in 2 states: PEND & SUSPEND
PEND + T	Task is in PEND state with a time-out value
PEND + S + T	Task is in 2 states: PEND state with a time-out value and SUSPEND

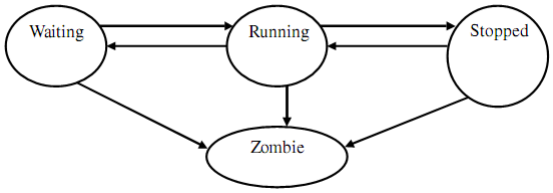
Tabulka 3.1: Popis stavů procesů v OS vxWorks [1]



Obrázek 3.3: Stavový diagram procesů OS vxWorks [1]

State	Description
RUNNING	Task is either in the RUNNING or READY state
WAITING	Task in BLOCKED state waiting for a specific resource or event
STOPPED	Task is BLOCKED, usually used for debugging
ZOMBIE	Task is BLOCKED and no longer needed

Tabulka 3.2: Popis stavů procesů v OS Linux [1]



Obrázek 3.4: Stavový diagram procesů Linuxu [1]

3.2.4 Vlákna

Některé operační systémy nabízí možnost vyváření tzv. vláken [1]. Jsou to odlehčené procesy přímo svázané se svým rodičovským procesem, a tak umožňují vyváření paralelismu také v rámci jednoho procesu. Vytvořené vlákno sdílí se svým rodičovským procesem některé zdroje (např. zdrojový kód, soubory, I/O zařízení, globální data, adresní prostor atd.), ale má svůj vlastní PC, zásobník a plánovací informace, což umožňuje pro instrukce, které vykonává, aby byly plánovány samostatně. Protože jsou vlákna vytvářeny v rámci stejného kontextu s úkolem a mohou sdílet stejnou paměť, přináší možnost snadnější komunikace, než je mezi samostatnými úkoly.

3.2.5 Plánování procesů

Víceúkolové OS obsahují mechanismus nazýván plánovač. Ten se stará o rozhodování, který z procesů bude následovat za aktuálním procesem při přidělování procesoru. V některých OS se plánovač také stará o přidělení CPU pro právě zavedené programy do paměti, v jiných se o to stará speciální plánovač.

V OS pro vestavěné systémy je implementováno mnoho druhů plánovacích algoritmů. Každý z nich má svoje silné a slabé stránky. Faktory mající vliv na efektivitu a kvalitu plánovacího algoritmu zahrnují:

- Odezvu – čas potřebný na vykonání změny kontextu na jiný připravený úkol ve fronte.
- Dobu obrátky – čas potřebný na zpracování každé dávkové úlohy.
- Režii – čas a data potřebné na zjištění, která úloha bude následovat.
- Spravedlivost – každý proces dostane spravedlivý díl času procesoru.
- Propustnost – množství spuštěných úkolů za daný čas.

Při zásadě spravedlivého přidělování procesoru musí plánovač dohlížet, aby žádný z procesů nehladověl – aby nebyly procesy, kterým se nedostává procesoru. To se většinou nestává pokud má plánovač vysokou propustnost.

Plánovače použité v dnešních OS můžeme z pohledu přístupu k úkolům rozdělit do dvou skupin: preemptivní a nepreemptivní.

Při nepreemptivním plánování OS nemůže násilně odebrat CPU aktuálně běžícímu procesu, ale čeká, až ho úkol uvolní dobrovolně bez ohledu na prioritu nebo čas strávený ostatními procesy ve frontě.

Algoritmy používané při nepreemptivním plánování:

- First Come First Serve (FCFS) / S během celého procesu najednou – Obsloužený je dřív vždy ten proces, který se dříve zařadil do fronty připravených procesů. Nevýhoda tohoto algoritmu je v malé odezvě systému při dlouhých procesech zařazených ve frontě.
- Shortest Process Next (SPN)/ S během celého procesu najednou – První obsloužený je vždy proces s nejkratším časem potřebným na vykonání celého programového úseku. Tento přístup má lepší odezvu při kratších procesech, avšak dlouhé procesy jsou tím postiženy. Mají horší přístup k CPU a hrozí jim vyhladovění. Zohledňování času běhu jednotlivých procesů se také negativně projevuje na režii plánování.
- Co-operative (spolupracující) – V tomto případě proces sdělí OS, kdy se má přepnout kontext. Tento algoritmus může být použit s FCFS i s SPN algoritmem. Tento přístup zlepšuje odezvu OS. V případě SPN pořád hrozí vyhladovění některých procesů.

Aby bylo možné použít nepreemptivní plánování, musí být zaručeno, že žádný proces nebude běžet v nekonečné smyčce, přičemž by vylučoval všechny ostatní úkoly. Výhoda je v jistotě, že proces nebude přerušen dokud se neukončí, nebo nebude připraven na změnu kontextu. Další výhodou je nižší režie OS.

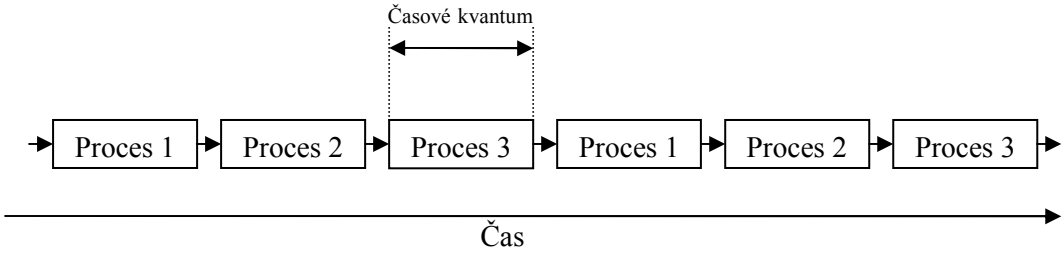
Na druhou stranu u preemptivního plánování si řídí čas změny kontextu samotný OS, a tak musí proces počítat se změnou kontextu v kterékoli chvíli, nebo ho může taky podnítit.

Základní algoritmy postaveny na preemptivním plánování jsou:

- Round Robin,
- Prioritní (preemptivní) plánování,
- Earliest Deadline First (EDF).

3.2.5.1 Round Robin

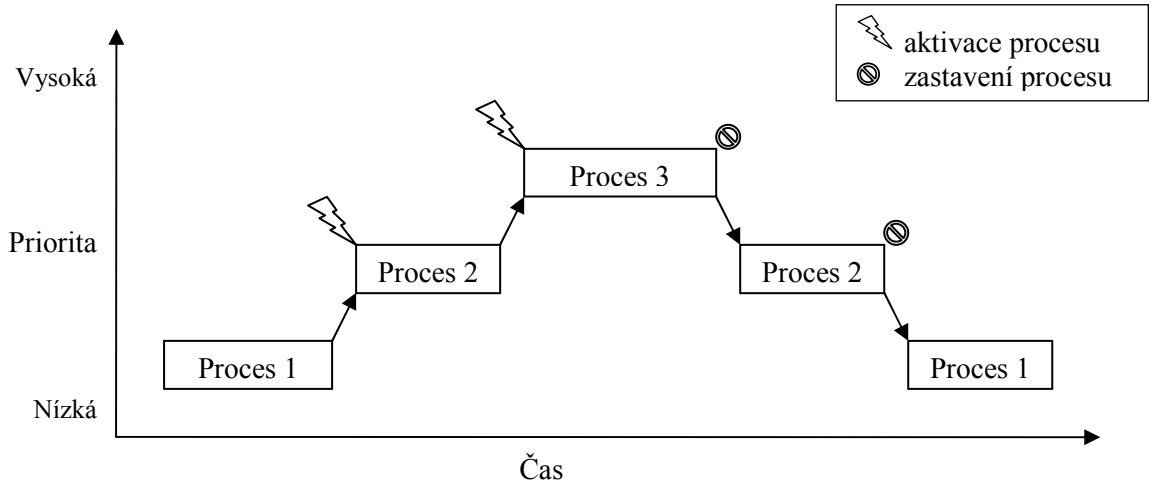
Round Robin [1] algoritmus implementuje FIFO frontu, do které jsou zařazovány procesy připravené pro spuštění. Procesy jsou zařazovány na konec fronty a vybírají se ze začátku a přiděluje se jim CPU. Nebere se v úvahu ani délka ani důležitost procesů. Každému procesu je přidělené stejné časové kvantum do další změny kontextu jako je to vidět na obrázku 3.5.



Obrázek 3.5: Příklad plánování Round Robin

3.2.5.2 Prioritní (preemptivní) plánování

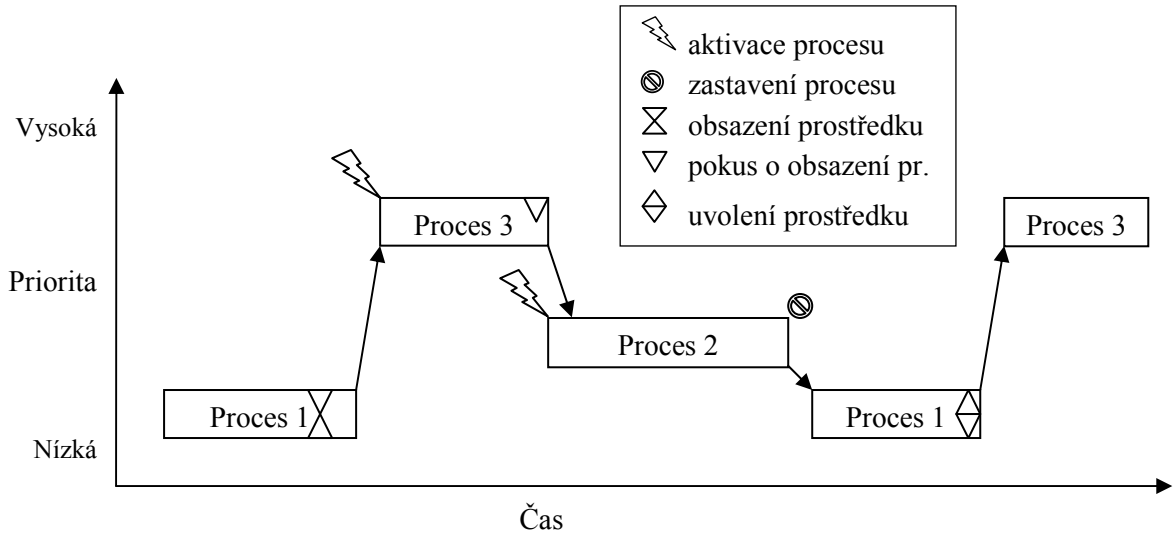
Při tomto plánování [1] se bere v úvahu přednostní postavení procesu vůči jiným procesům a sytému. Toto postavení je vyjádřeno jedinečnou hodnotou priority, která je přidělená každému z procesů. Při plánování tak úkol s vyšší prioritou vždy předběhne úkol s prioritou nižší, to taky znamená, že méně prioritnímu procesu může být odebrán CPU ihned po aktivaci více prioritního jako je to znázorněné na obázku 3.6.



Obrázek 3.6: Příklad prioritního preemptivního plánování

Problémy při tomto plánování představuje hlavně vyhladovění méně prioritních procesů a inverze priorit. Inverze priorit vzniká při obsazení prostředku málo prioritním procesem, který je před jeho uvolněním přerušen a v běhu pokračuje vysoce prioritní proces. V případě, že tento proces pak potřebuje přístup k tomu prostředku, který je obsazen a mezitím se stal aktivní proces se střední prioritou, bude tento vysoce prioritní proces pozastaven a bude běžet středně prioritní proces jako kdyby s vyšší prioritou, tento jev je znázorněn na obrázku 3.7.

Přidělené priority by měly vyjadřovat důležitost jednotlivých úkolů, ale může se stát, že některé úkoly mají stejnou důležitost, a tak se na přidělování priorit používají různé pomocné metody. Jednou z nich je RMS (Rate Monotonic Scheduling), který přiděluje priority na základě periody, s jakou jsou úkoly v systému spouštěny. Čím kratší je perioda úlohy, tím vyšší je priorita.



Obrázek 3.7: Příklad inverze priorit

3.2.5.3 Earliest Deadline First (EDF) plánování

U tohoto plánovacího algoritmu [1] se úkolům přiděluje procesor na základě mezní doby platnosti procesu a nevyžaduje, aby byly úkoly periodické. Obecně se u tohoto plánování pracuje s těmito parametry:

- deadline (kdy nejpozději potřebuje být úkol dokončen),
- frekvence (kolikrát bude proces běžet za určitý čas),
- trvání (čas potřebný na vykonání procesu).

Tento algoritmus má vyšší režii než RMS a vyžaduje možnost dynamického přidělování priorit. Výhodou je efektivnější využití procesorového času.

3.2.5.4 Real-time plánování

Real-time operační systémy [4] jsou klíčovým prvkem mnohých dnešních vestavěných systémů, kde vytváří softwarový základ pro běh aplikace. Ne všechny vestavěné systémy však používají RTOS. Některé systémy s relativně jednoduchým hardwarem nebo snadno implementovatelným softwarem nemusí vyžadovat RTOS. Avšak vestavěné systémy se středním až rozsáhlým aplikačním kódem většinou vyžadují nějakou formu plánování. Pro tyto systémy je vhodné použít RTOS.

Real-time vlastnosti OS závisí hlavně od plánování procesů, proto je dále uvedeno, co je při plánování důležité. Real-time systémy je možno rozdělit do dvou skupin a to hard real-time a soft real-time:

- Od hard real-time systémů se vyžaduje, aby vykonaly kritický úkol v přesně daném časovém úseku, nebo informovali, že splnění požadavku je neuskutečnitelné. Tyto systémy musí být deterministické, aby bylo možné zaručit tyto časy zpoždění.
- U soft real-time systémů už nejsou tak striktní požadavky na rychlost odezvy systému. Vyžaduje hlavně zajištění, aby úkoly, které mají být vykonány v reálném čase měly vyšší prioritu než ostatní méně důležité úkoly. Tento přístup však může vést k nespravedlivé alokaci prostředku a také vyhladovění některých procesů, proto je velice důležité věnovat přidělování priorit velkou pozornost.

Real-time systém je použit, pokud jsou dány požadavky na dobu zpracování vstupních dat a reakci na ně, proto je často používán v kontrolních systémech.

3.2.6 Meziúkolová komunikace a synchronizace

Jednotlivé úkoly ve vestavěném softwaru častokrát potřebují sdílet společné prostředky, ke kterým může najednou přistupovat pouze jeden úkol, nebo jejich omezený počet. Jindy si potřebují úkoly vyměňovat informace nebo synchronizovat svoji činnost, proto mají OS vyvinuty zvláštní mechanismy, které tyto činnosti bezpečně umožňují.

3.2.6.1 Fronty zpráv

Fronty zpráv jsou [1] často implementovaným prostředkem meziúkolové komunikace. Jejich prostřednictvím si mohou úkoly posílat balíky dat. OS zabezpečuje, aby byly zprávy doručeny správným úkolům a taky definuje formu a počet zpráv zařaditelných do jedné fronty.

OS s mikrojádrem typicky používají posílání zpráv jako jejich hlavní synchronizační mechanismus.

3.2.6.2 Semafory

Semafory [1] se používají při sdílení paměti (vzájemně výluční přístup) a na synchronizaci činnosti procesů. Jeden OS může podporovat současně více druhů semaforů jako např. binární semafory, mutexy a výčtové semafory.

3.2.6.3 Signály

Signály [1] informují úkol o proběhnutí asynchronní události. Jsou generovány buď externími, nebo interními událostmi. Když úkol přijme signál, ukončí vykonávání aktuálního programu a začne vykonávat obslužní rutinu pro obsluhu daného signálu. Po jejím ukončení se vrátí na původní místo, kde byl program přerušen. Signály jsou často používány na obsluhu přerušení pro jejich asynchronní povahu.

3.2.7 Deadlock

Deadlock je jeden ze závažných problémů, které se mohou vyskytnut za běhu programu, proto je důležité zohlednit některé postupy při navrhování procesů, aby se jim předcházelo.

Výpočetní systémy často obsahují prostředky, ke kterým může najednou přistupovat pouze jeden proces. Téměř všechny operační systémy poskytují prostředky pro výlučný přístup k požadovaným zdrojům jak hardwarovým tak softwarovým. Mnoho procesů však v jednu chvíli vyžaduje přístup nejen k jednomu prostředku, ale k mnoha. To může způsobit situaci, ve které si vzájemně dva procesy nealokují jeden z potřebných zdrojů a budou na sebe navzájem čekat, dokud ten jejich druhý zdroj neuvolní, což způsobí, že tyto dva procesy zůstanou v tzv. deadlocku a nepokračují dál ve své činnosti. Proto jsou součástí OS nástroje pro zamezování deadlocku a také nástroje na jejich zrušení v případě jejich výskytu.

Jedno z řešení je založeno na uvolnění nealokovaného prostředku procesem, kterému je znemožněno alokovat další prostředek pro pokračování ve své činnosti. Po následným čekáním o pseudonáhodné délce se proces pokusí znovu o alokaci všech potřebných prostředků.

4 OS pro parní sterilizátor

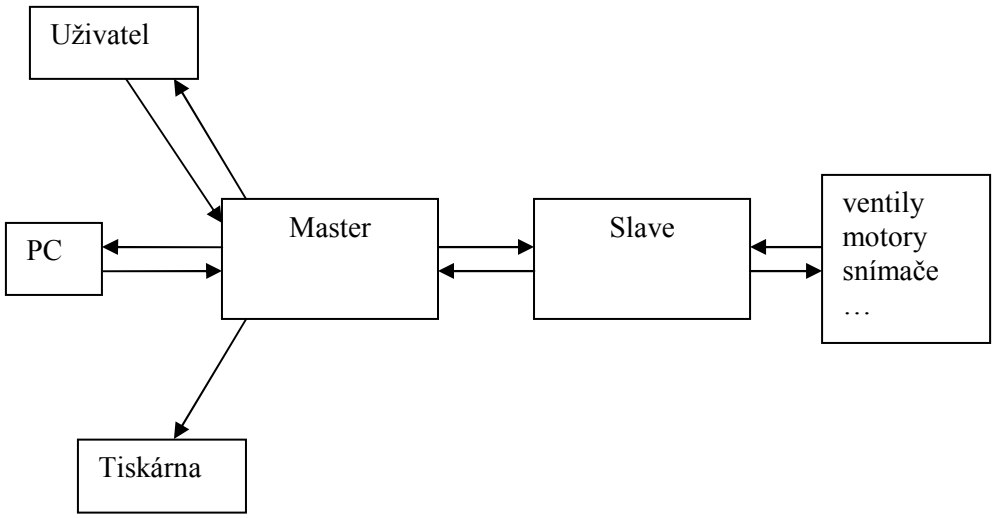
4.1 Popis zařízení

Malé parní sterilizátory firmy BMT [13] tvoří řadu výrobků určených především pro zdravotnictví, veterinární lékařství a laboratoře. Tyto sterilizátory jsou vhodné k sterilizaci všech nástrojů, materiálů, textilu, gumy a plastů určených ke sterilizaci parou. Tato řada se vyznačuje kompaktními rozměry, objemy komor v rozsahu 15 až 25 litrů, vysokou spolehlivostí a rychlostí sterilizace. Sterilizace parou je vysoce účinný způsob pro ničení všech mikroorganismů, avšak dosažení těchto vlastností vyžaduje přesné řízení napouštění a udržování páry v komoře. Nejmenší z rady je vidět na obrázku 4.1.

Současná inovace řady malých parních sterilizátorů přinese také změněnu jejich elektroniky a periferií. Monochromatický displej osazen maticovým dotekovým panelem bude nahrazen barevným dotekovým displejem. Kvůli barevnému displeji bude do elektroniky přidán řadič displeje a paměť pro uchovávaní či zpracovávaní zobrazovaných dat. Dále bude přidán konektor a řadič USB pro připojení k počítači. Všechny tyto změny si vyžádají také změnu v řídícím softwaru.

Obrázek 4.1: Parní sterilizátor Steridient [13]

Elektronika přístroje je tvořena dvěma řídícími jednotkami na nezávislých plošných spojích, které navzájem hlídají svoji činnost. Jedna řídicí jednotka má přednostní postavení, proto je označována jako master, podřízená jako slave. Master má na starosti interakci s uživatelem přístroje a s okolním prostředím (PC, tiskárna), zatímco slave se stará o řízení procesu sterilizace. Obě řídící jednotky jsou propojeny komunikačním kanálem, který je v textu označován jako vnitřní komunikace. Komunikační kanály sterilizátoru jsou znázorněny na obrázku 4.2. Zbytek práce bude pojednávat pouze o elektronice a firmware pro elektroniku master.



Obrázek 4.2: Schéma komunikace malého parního sterilizátoru

Současný firmware je tvořen stavovým automatem, který se staral o přepínání zobrazování jednotlivých obrazovek uživatelského prostředí, zatímco ostatní funkce byly řízeny přes přerušení. U vyšší řady přístrojů s obdobnou elektronikou se se zvětšováním programové části a tím také prodlužováním a zvyšováním četností přerušení začala zhoršovat propustnost komunikace. Jedním z možných řešení je zavedení operačního systému do řídícího softwaru.
Výhody, které by přineslo zavedení OS do firmwaru:

- Pseudoparalelismus procesů,
- zkrácení času stráveného v přerušení,
- zpřehlednění architektury firmwaru (rozdělení programu do procesů),
- zrychlení odezvy na události (RTOS),
- usnadnění udržování a rozšiřování firmwaru (vlákna možno přidávat i odebírat).

Jaké nevýhody má zavedení OS:

- zvětšení programového kódu,
- zabrání místa v paměti RAM,
- jistou spotřebu výpočetního výkonu,
- může zvýšit pořizovací náklady zařízení svoji cenou,
- potřeba zaškolení programátora na práci s daným OS,
- potřeba zachovávání jistých zásad při programování (kritické sekce, sdílení zařízení).

Vybraný OS by měl mít pro nás vhodné vlastnosti a měly by u něj být eliminovány nežádoucí projevy. Z toho vyplývají následující body:

- OS podporující multitasking
- program OS tvořen ve snaze minimalizovat čas, kdy jsou zakázána přerušení,
- neměl by být příliš komplexní, nad rámec potřebných vlastností,
- mohl by podporovat real-time plánování,
- mohl by podporovat dynamické linkování aplikací s OS,
- neměl by mít velké paměťové nároky na RAM i ROM,
- cena by měla být co nejnižší,
- měl by mít snadně použitelné API.

4.2 Výběr operačního sytému

Pro systémy jako jsou servery nebo PC je na trhu docela úzký okruh OS, které se užívají. Zatím u vestavěných systému je výběr široký a každé z řešení má jisté výhody a nevýhody. V dnešní době jsou jich na trhu desítky, jak je vidět v příloze 4, proto není výběr OS pro vestavěný systém úplně snadný. Pokud v tomto případě budeme považovat za vestavěné systémy vše, co není počítač nebo server, tak je možné OS pro vestavěné systémy rozdělit do několika kategorií, a to na OS určené pro:

- PDA (Palm OS, Inferno, Windows CE, Symbian OS, ...)
- Hudební přehrávač (DSPnano RTOS, ipodlinux, iPhone OS, ...)
- Smartphone (BlackBerry OS, iPhone OS, Embedded Linux, Windows Mobile, ...)
- Router (Cisco IOS, AlliedWare, Inferno, CatOS, ...)
- Mikrořadič, Real-time (VxWorks, XMK, Nucleus, µClinux, µC/OS-II, ...)

4.2.1 Vestavěné operační systémy

Představíme si blíž některé operační systémy hlavně z pohledu podporovaných architektur, z kterých budeme následně vybírat jeden, na kterém vytvoříme ukázkovou aplikaci. Vzhledem k použití OS v parním sterilizátoru jsem se věnoval především operačním systémům obecného užití, například OS pro PDA a OS pro mikrořadiče a RTOS.

4.2.1.1 Operační systémy bez real-time jadra

- Palm OS – pracuje na Dragonball procesoru a také na procesoru Intel XScale. Vývojové prostředky jsou volně dostupné na oficiálních stránkách nebo od třetích stran.
- EPOC/Symbian – podporuje ARM procesory. Má vedoucí postavení na trhu s mobilními zařízeními.
- µClinux (čte se *ju sí linux*) – byl původně vytvořen z Linuxového jádra 2.0 pro mikrořadiče bez MMU. Dnes je tento projekt rozšířený na mnoho různých procesorových architektur a značek. V dnešní době je k dispozici verze s jádrem 2.0, 2.4 a 2.6 tak jako i sada knihoven, aplikací a vývojových prostředí.

4.2.1.2 Komerční RTOS

- CMX-RTX – Real-timef víceúkolový operační systém, který podporuje většinu vestavěných mikrokontrolérů, mikroprocesorů a DSP procesorů. Systém je konfigurovatelný pro vývojová prostředí.
- RTXC – real-time OS s integrovaným middlewarem jako USB, FS, GUI … Vývojové prostředí RTXC Bridže a VisualRTCX. Podpora: ARM, ColdFire a Blackfin.
- VxWorks, Wind River – nejrozšířenější operační systém ve světe vestavěných aplikací.
- XMK – real-time operační systém, který je navržen pro minimální paměťové nároky. Podporuje AVR, H8 a mnoho dalších 8 bitových mikrořadičů.
- QNX – vysoce spolehlivý vývojový systém zahrnující OS, který je vhodný pro aplikace, u kterých by v případě chyby mohly být v ohrožení lidské životy nebo by mohly vzniknout velké škody na majetku jako to je v případě leteckých systémů, chirurgických nástrojů či jaderných reaktorů.

4.2.1.3 Open source RTOS

- BeRTOS – volně šiřitelný operační systém s možností použití na komerční účely. Podporuje architektury ARM, Atmel AVR.
- FreeRTOS – přenositelný, open source operační systém s mini real-time jádrem. Podporuje preemptivní plánování. Mimo standardních procesů dokáže používat korutiny.
- µC/OS-II – vysoce přenositelné, upravitelné, preemptivní real-time víceúkolové jádro pro mikrořadiče a mikroprocesory. Volně použitelné pro edukační účely.
- eCos – open source, real-time operační systém bez nároku na licenční poplatky. Je vysoce konfigurovatelný čím se dosahují nízké paměťové nároky přizpůsobením systému aplikačním nárokům. Podporuje širokou škálu mikrořadičů.
- RTEMS – operační systém na úrovni komerčních produktů je však šířen pod upravenou licencí GNU GPL. RTEMS je OS podporující aplikace s nejstriktnějšími podmínkami pro real-time. Hlavní výhodou tohoto systému je však podpora více-procesorových systémů.

4.2.2 Použitý hardware

Použitý hardware ve vestavěném systému může omezit možnosti použití některých operačních systému. Buď je problém v nedostatku výkonu procesoru, nebo malé paměti, jindy nemusí být OS přenesen na danou architekturu.

Dále je uveden popis důležitých prvků použitého hardwaru pro výběr OS:

- MCU – jejich řídícím prvkem je [5] mikrokontrolér H8S/2633f od firmy Hitachi. Jeho jádrem je CPU H8S/2600, který je možné charakterizovat jako vysokorychlostní CPU s vnitřní 32 bitovou architekturou, která je zpětně kompatibilní s H8/300 a H8/300H. CPU obsahuje šestnáct 16-bitových registrů obecného užití, může adresovat 16-Mbyte lineárního adresného prostoru a je ideální pro real-time řízení.
- Paměť RAM – použitý čip H8S/2633f obsahuje 16 kB vestavěné vysokorychlostní RAM paměti. [5] Paměť je připojena k CPU pomocí 16-ti bitové datové sběrnice, která umožňuje CPU přistupovat do paměti v jednom cyklu k uloženému bytu, nebo wordu. Tato výhoda rychlého přístupu však má jedno omezení. Sudé paměťové adresy používají horních 8 bitu sběrnice a liché 8 spodních bitů. To má za následek, že přenášený word musí začínat na sudé adrese. Tuto vlastnost v našem případě bude hlídat použitý překladač. V registru SYSCR je možnost zapnutí, nebo vypnutí vnitřní RAM paměti, paměť je implicitně zapnuta. I když budeme mít k dispozici dostatek vnější paměti využijeme této paměti hlavně při spouštění systému. Systém také obsahuje paměť RAM připojenou na vnější sběrnici. Byla použitá paměť R1LV1616H o velikosti 16 Mbit se sníženou spotřebou energie. To umožňuje zálohování napájení paměti pomocí baterie.
- Flash paměť – přímo na čipu mikrokontroléru je umístěná paměť o velikosti 256 kB se stejným zapojením jako to bylo u vnitřní paměti RAM. Také se dá tato paměť vypnout stejně jako vnitřní RAM. Mikrokontrolér obsahuje přímo v sobě mechanismus na programování této flash paměti, což umožňuje její přehrávání i za chodu systému. Dále je součástí hardwarové výbavy také vnější flash paměti M29W160EB o velikosti 16 Mbit s vestavěným programovacím mechanismem a s možností dočasného zamykání paměti kvůli bezpečnosti dat.

Všechny tyto součástky jsou osazeny na plošném spoji dle schématu zapojení viz. příloha 3. Použitá elektronika úplně neodpovídá elektronice, která bude v konečných přístrojích použita, protože v současné době není vyrobena a je ve stádiu návrhu. Na schématech je proto náhradní elektronika, na které byl program vyvíjen a testován. Ta se v konečné podobě nebude moc lišit od momentálně vyvíjené elektroniky.

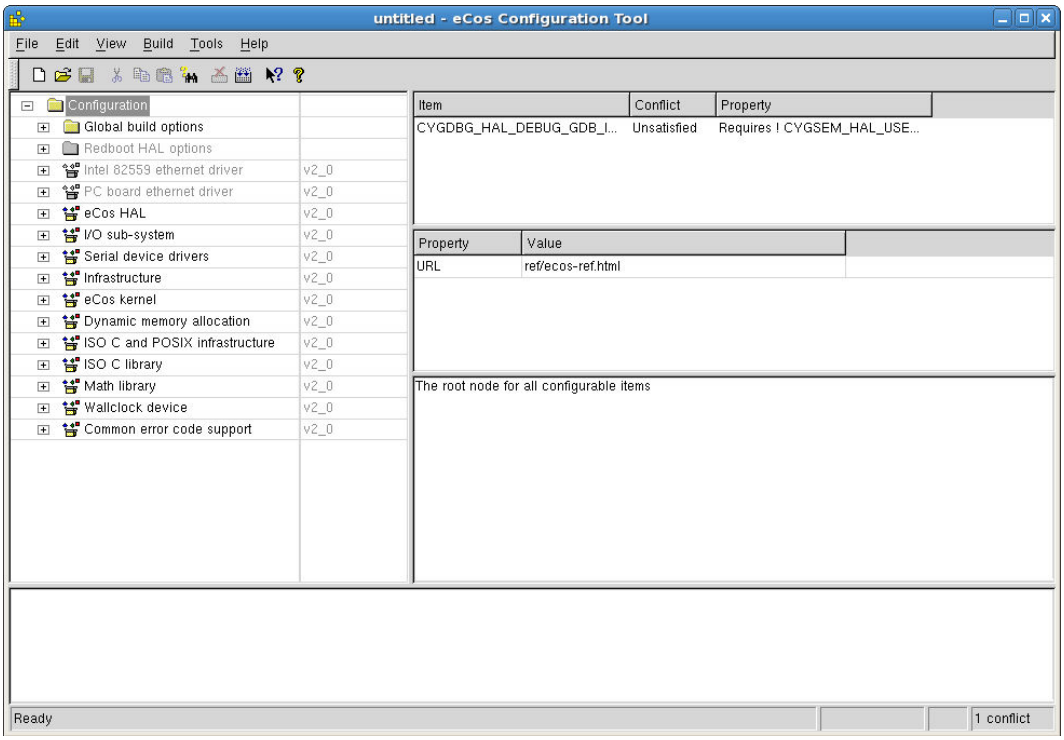
4.2.3 Open source OS vhodné pro architekturu H8S

Ze zmíněných operačních systému byly vybrány ty, které nejlépe splňovaly požadavky kladené na OS a také podporují použitou architekturu H8S. Při výběru byla také hodně důležitá cena, proto jsem se rozhodl pro použití volně šiřitelného operačního systému, který se může použít v komerčních aplikacích bez nároků na jakékoli poplatky a bez potřeby zveřejňování veškerého kódu.

4.2.3.1 eCos

eCos (Embedded Configurable Operating Systém) je jako již název napovídá škálovatelný open source RTOS určen pro vestavěné aplikace. Pro svou konfiguraci používá samostatní konfigurační nástroj viz. Obrázek 4.3. Je určen hlavně pro použití, kdy je vestavěný Linux příliš náročný na prostředky [6].
Jeho hlavními výhodami jsou [11]:

- Široká podpora platforem a 16, 32 a 64 bitových architektur.
- Open source vývojové nástroje a překladač GCC.
- Výběr mezi RT prioritním plánováním round robin, bitmapovým, nebo kooperativním plánováním. Taky je tady možnost připojení jiného plánovače procesů.
- Jádro nezakazuje přerušení za normálního běhu, což přináší krátké časy odezvy.
- Podpora vláken.
- HAL izolující zbytek systému od hardwaru.
- Podpora časovačů, čítačů a alarmů.
- Podpora standardních knihoven C a C++.
- Podpora file systemů, síťových protokolů, komunikačních protokolů.
- Testovací a debagovací prostředky.



Obrázek 4.3: Okno konfiguračního nástroje operačního systému eCos

4.2.3.2 FreeRTOS

FreeRTOS je real-time jádro operačního systému s upravitelným rozsahem funkcí podle potřeb aplikace. Je vyvinutý pro malé vestavěné systémy. Má přehledné stránky se všemi potřebnými informacemi pro použití poskytnutých kódů a také návod na úpravy jádra systému pro přenos na jinou elektroniku.

Hlavní výhody FreeRTOS [7]:

- Volně šířitelné jádro s preemptivním, kooperačním, nebo hybridním nastavením.
- Kompaktní, jednoduchý a snadný na použití.
- Vysoce přenosný kód psaný převážně v jazyce C.
- Podpora úkolů a taky korutin současně.
- Užitečná možnost trasování běhu systému.
- Detekce přetečení zásobníku.
- Žádna softwarová omezení na počet vytvořených úkolů.
- Žádna softwarová omezení na počet priorit, které mohou být použity.
- Fronty, několik druhů semaforů a mutexy pro komunikaci a synchronizaci.
- Mutexy s děděním priorit.
- Široka podpora překladačů a různých hardwarových architektur.
- Volně dostupné vývojové nástroje (Cortex-M3, ARM7, H8/S, …).
- Bez licenčních poplatků.
- Volně dostupné zdrojové kódy pod upravenou GNU GPL licencí.
- Využití paměti RAM operačním systémem od 4,3 kB.

Příklad obsahu konfiguračního souboru FreeRTOSConfig.h:

```
#define configUSE_PREEMPTION            1
#define configUSE_IDLE_HOOK             0
#define configUSE_TICK_HOOK             0
#define configCPU_CLOCK_HZ         ( ( unsigned portLONG ) 22118400 )
#define configTICK_RATE_HZ         ( ( portTickType ) 600 )
#define configMAX_PRIORITIES       ( ( unsigned portBASE_TYPE ) 4 )
#define configMINIMAL_STACK_SIZE   ( ( unsigned portSHORT ) 220 )
#define configTOTAL_HEAP_SIZE      ( ( size_t ) ( 20 * 1024 ) )
#define configMAX_TASK_NAME_LEN          ( 8 )
#define configUSE_TRACE_FACILITY         0
#define configUSE_16_BIT_TICKS           1
#define configIDLE_SHOULD_YIELD          1

/* Co-routine definitions. */
#define configUSE_CO_ROUTINES            0
#define configMAX_CO_ROUTINE_PRIORITIES  ( 1 )

/* Set the following definitions to 1 to include the API function, or zero
to exclude the API function. */

#define INCLUDE_vTaskPrioritySet         0
#define INCLUDE_uxTaskPriorityGet        0
#define INCLUDE_vTaskDelete              0
#define INCLUDE_vTaskCleanUpResources    0
#define INCLUDE_vTaskSuspend             0
#define INCLUDE_vTaskDelayUntil          0
#define INCLUDE_vTaskDelay               1
```

4.3 Implementace operačního systému

Jako nejvhodnější OS pro účel implementace na parní sterilizátor byl zvolen FreeRTOS. Z hlediska funkcionality je dostačující stejně jako eCos. Pro FreeRTOS jsem se nakonec rozhodl díky jeho podpoře překladače IAR používaného firmou BMT viz. příloha 2, dále pro jeho snadné použití v aplikacích (stačí do projektu přidat 4 zdrojové soubory) a pro dostupné a přehledné informace o funkčnosti a používaní OS.

Vybraný operační systém FreeRTOS bude upraven a případně rozšířen, aby bylo možné na jeho základe vytvořit ukázkovou aplikaci pro barevný display s dotekovým panelem.

Jádro OS se dá dobře otestovat pomocí aplikace. Využijeme to, že nám ukázkové aplikace nabízí přímo poskytovatel OS s podrobným návodem. Použijeme aplikaci přenesenou na vývojový kit EDK 2329 osazený mikrořadičem H8S/2329. Jeho parametry jsou téměř stejné s mikrořadičem H8S/2633. Jsou postavené na stejné architektuře, mají stejné taktovací frekvence. Liší se například ve velikostech vestavěných pamětí.

Námi vybraná aplikace je určena pouze pro překladač GCC, proto ji budeme muset upravit, aby ji bylo možné překládat pomocí IAR překladače. Pomocí zdrojových textů určených pro jiné architektury pro překladač IAR upravíme potřebné částí kódu. Takové úpravy jsou usnadněné tím, že celý kód závislý na překladači nebo architektuře je oddělen v samostatných souborech. V případě přenášené ukázkové aplikace jsou to soubory port.c a portmacro.h. Tyto dva soubory byly zásadně změněny. Všechny vložené kódy psané v jazyce assembler byly upraveny a přeneseny do samostatného souboru portMacroAsm.asm.

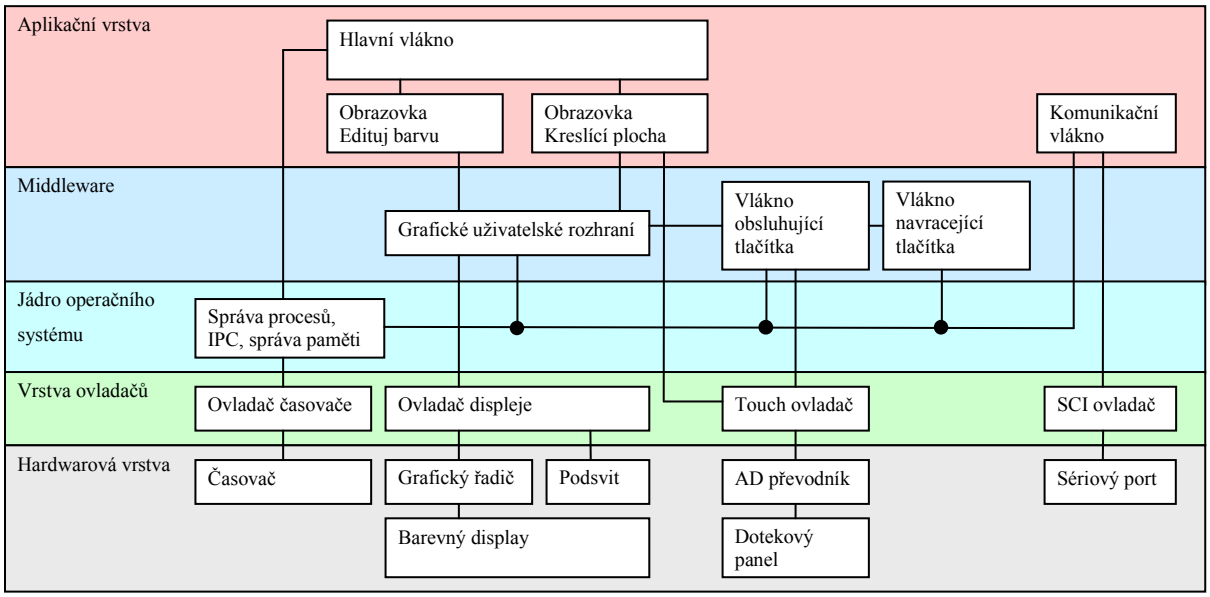
Další změny byly provedeny na funkcích pro obsluhu kritických sekcí pomocí zákazu přerušení. Jejich původní implementace při každém vnoření kritické sekce zaplnila dva byty paměti na zásobníku, což mohlo v budoucnu u rozsáhlejších aplikací způsobovat protékání paměti. Vytvořená implementace uchovává hloubku vnoření v jedné dvoubytové proměnné. Ta se při každém vstupu do kritické sekce inkrementuje a zakážou se přerušení a při vynoření dekrementuje, ale přerušení se povolí až po úplném vynoření. Tato implementace je sice výpočetně náročnější, ale spotřebuje pouze dva byty paměti.

Největším problémem přenosu byla náhrada atributů funkcí typických pro překladač GCC uvozovaných klíčovým slovem `__attribute__`, za kterým následovaly v závorkách jednotlivé atributy funkcí. S atributem `interrupt_handler` byly označeny funkce obsluhy přerušení, ten byl nahrazen klíčovým slovem `__interrupt` překladače IAR. Ale atribut `saveall`, který zabezpečoval při změně kontextu uložení registrů na zásobník, nemá svůj ekvivalent u překladače IAR. Problém se týkal důležitých funkcí přepínání kontextu. Nahrazení tohoto atributu inline assemblerem spolu s různými atributy IAR překladače zbytečně znepřehledňovalo kód. Dále toto řešení vyžadovalo dodržování nestandardních pravidel při psaní obslužných funkcí přerušení pro aplikace s možností změny kontextu. To totiž vyžadovalo používaní nestandardního atributu raw a dvou API funkcí, jedné na začátku a druhé na konci funkce. Problém byl nakonec vyřešen přepsáním všech částí jádra pro změnu kontextu do assembleru spolu s ukládáním všech registrů na zásobník. Také bylo usnadněno používání obslužných funkcí přerušení aplikace pro změnu kontextu, do kterých teď postačuje na konec vložit volání jedné API funkce.

Do operačního systému byla připojena původní knihovna h8s2633f firmy BMT pro snadné ovládání mikrořadiče a jeho některých periferií. Také bylo přeneseno GUI z velkých parních sterilizátorů, které po několika úpravách a po odhalení přetékání zásobníku začalo spolehlivě fungovat. Na něm se pak začala vyvíjet ukázková aplikace a událostí řízené stláčení tlačítek místo náročného cyklického kontrolování všech tlačítek a jejích příznaků.

5 Implementace ukázkové aplikace

Ukázková aplikace demonstruje funkčnost LCD displeje s dotekovým panelem a rychlost jeho odezvy. Ukazuje možnosti využití dotekového panelu jako vstupního zařízení uživatelského rozhraní. Samotná aplikace umožňuje kreslení pomocí dotekového panelu a výběr barvy použitého pera.



Obrázek 5.1: Architektura ukázkového vestavěného sytému

Architektura celého firmwaru je znázorněná na obrázku 5.1. Tet budě pozornost věnovaná jeho obsluze. Po spuštění se zobrazí kreslící okno, odkud se pomocí tlačítka Barva dá přepnout do režimu výběru barvy. Zde si lze pomocí tlačítek namíchat libovolnou barvu z použité 16-ti bitové barevné škály. Míchaní je prováděno za pomocí tlačítek pro přidání nebo ubrání jedné z barvených složek RGB nebo jasu. Aktuální barva je zobrazována nad tlačítky. Následným zmáčknutím tlačítka Kreslit se vrátíme do kreslící obrazovky. Doteková plocha rozlišuje, jestli je kreslena spojitá čára, při přerušení kontaktu s plochou se také přeruší čára a začíná se kreslit na místě dalšího doteku. Pomocí tlačítka Smaž se maže kreslící plocha. V pravém horním rohu této obrazovky jsou souřadnice místa posledního doteku. Pomocí programu PrnScr.exe lze kdykoli získat obsah aktuálně zobrazené obrazovky a uložit je jako rastrový obrázek.

5.1 Obsluha Dotekového panelu

V první řade bylo potřebné obsloužit A/D převodník, na který je připojen analogový dotekový panel. Externí A/D převodník je připojen na volné piny mikrokontroléru a pomocí časovače se sériově přenášejí data do a z převodníku. Tato část obsluhy je umístěna v obsluze přerušení časovače. Po přenesení dat z převodníku se ověřuje, jestli hodnota patří do rozsahu hodnot odpovídající doteku, nebo je panel uvolněn. Jakmile je zjištěn dotek na panelu, uloží se souřadnice doteku a posílá se zpráva o zaznamenání doteku.

Zprávy o přijetí doteku primárně přijímá vlákno obsluhující stisknutí tlačítek GUI. Vlákno je do přijetí zprávy zablokované. Přijetím zprávy se stane aktivním a pokud právě neprobíhá komunikace mající vyšší prioritu, je toto vlákno aktivované na konci obslužní rutiny časovače A/D převodníku. Vlákno po aktivaci přijme souřadnice doteku a zkontroluje, jestli odpovídají některému z definovaných tlačítek. Jestli souřadnice patří některému z tlačítek, nastaví se mu příznak stisknuto a čeká se na další příchozí souřadnice z převodníku na potvrzení stisku. Po potvrzení stisku se zavolá

funkce PressFce s číslem tlačítka, která vykoná stisknutí tlačítka. Jistou dobu po stisku je tlačítko nestisknutelné. O následné odmáčknutí tlačítek se stará další nezávislé vlákno, které se spouští v pravidelných časových intervalech a po jisté době jednotlivá tlačítka uvolňuje. Vlákno obsluhující stisknutí tlačítek také ošetřuje případy nechtěného stisknutí tlačítka při přepínaní obrazovky, kdy jsou všechny stisky zakázány, pokud se neuvolní dotekový panel.

5.2 Implementace obrazovek aplikace

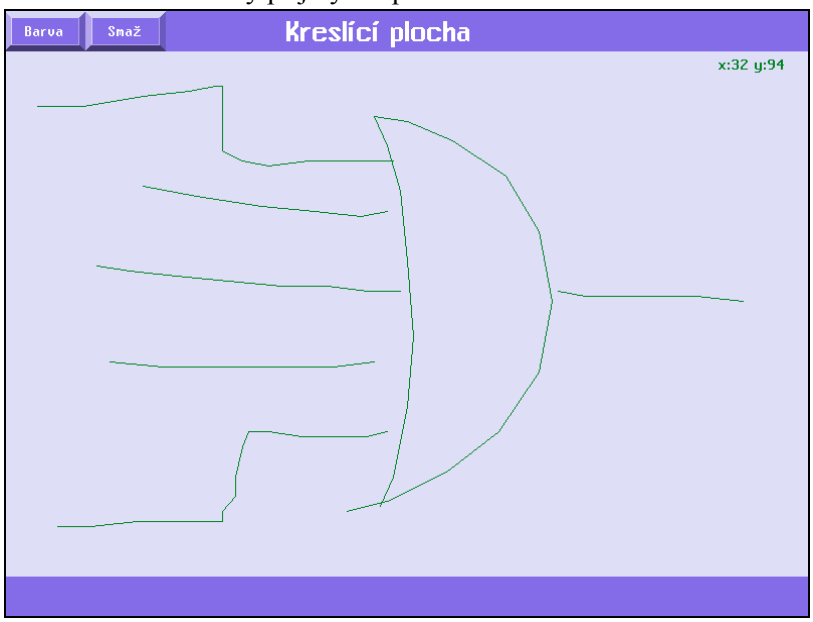
Obě vytvořené obrazovky pro testovací aplikaci jsou programovou součástí hlavního vlákna main. Vlákno main podle aktuálního stavu systému spouští kreslící, nebo nastavovací obrazovku. Jednoduchým přidáním nových stavů do přepínače se dá rozšířit počet obrazovek.

Obsluha obrazovky je implementována ve smyčce opakující se, dokud je aktuální obrazovka platná, čili odpovídá aktuálnímu stavu. Před cyklem se mohou v GUI nastavit grafické a funkcionální popisy tlačítek. V této smyčce pak stačí volat kontrolní funkci GUI a obsluhu WDT časovače a máme vytvořené uživatelsky plnohodnotné prostředí. Tímto způsobem jde lehce implementovat jakoukoli obrazovku. Implementace dvou ukázkových bude popsána v následujících podkapitolách.

5.2.1 Kreslící plocha

Kreslící plocha je obrazovka testovací aplikace umožňující za pomocí dotekového panelu kreslení čar podle nastavené aktuální barvy. U této obrazovky je možné nejlépe otestovat běh víc vláken najednou, kdy mohou souběžně běžet komunikační vlákno přenášející zobrazované informace, pak hlavní vlákno, které se stará o vykreslování čar na plátně podle doteku na panelu a pak další dvě vlákna obsluhující tlačítka GUI.

Po vstupu do hlavní smyčky obsluhy okna se nahlédne do fronty zpráv o přijetí doteku. Pokud je fronta prázdná, čeká se jistou krátkou dobu, jestli se ve frontě za tuto dobu nic neobjeví, ukončí se kreslená souvislá čára. Pokud do fronty přijde zpráva, informuje to o vstupu z dotekového panelu, vlákno se stane aktivním a začíná zpracování informací: výčet souřadnic, otestování, jestli spadají do kreslící plochy a jestli existuje bod, na který se dá navázat přímka, nebo se kreslí nová čára. Po tomto zpracování se uloží souřadnice aktuálního bodu, aby se od něj dalo dál pokračovat a vrací se smyčka znovu do náhledu do fronty přijatých zpráv.

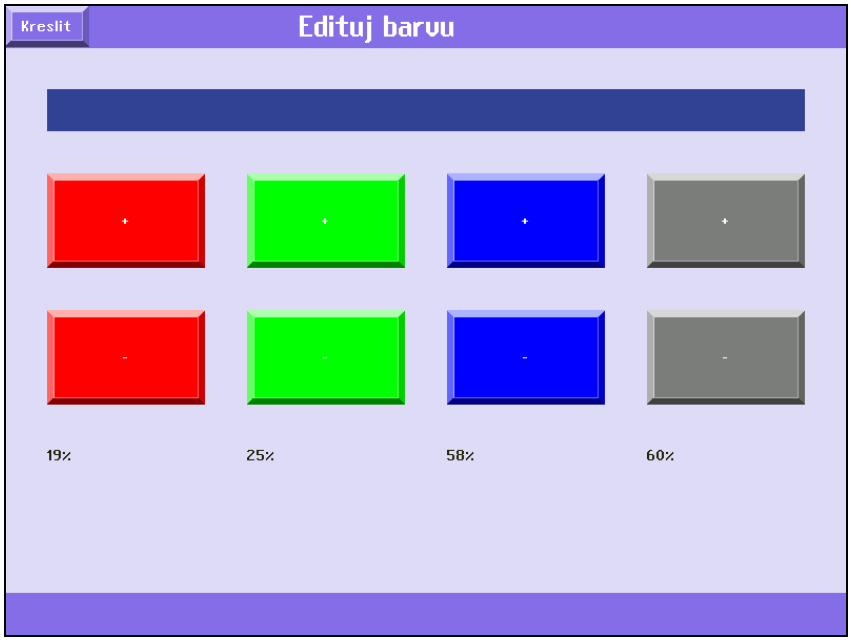


Obrázek 5.2: Kreslící plocha ukázkové aplikace

5.2.2 Edituj Barvu

Obrazovka Edituj barvu slouží ke změně barvy kreslícího pera pro kreslící obrazovku. Hlavní prvky obrazovky jsou tlačítka pro nastavovaní barvy a barevný pás nad tlačítky zobrazující aktuálně namíchanou barvu.

Okno v hlavní smyčce neobsahuje žádné funkční částí mimo funkce OS na pozastavení vlákna. Všechny funkce okna jsou umístěný v obslužných rutinách jednotlivých tlačítek. Barva se může upravovat změnou podílu červené, zelené a modré barevné složky nebo úpravou jasu pomocí šedých tlačítek, které upravují všechny tři složky najednou. Display je schopen zobrazovat barvy 16-ti bitové barevné hloubky. Proto se při každé změně vytváří ze tří složek hodnota barvy na 16-ti bitech, kde zelená barva má vyhrazených 6 bitů, protože je na ni lidský zrak nejcitlivější. Zbylé složky mají po 5 bitech. Po stisknutí tlačítka Kreslit se změní obrazovka na kreslící plochu.



Obrázek 5.3: Obrazovka editace barev ukázkové aplikace

5.3 Obsluha komunikace

Implementovaná komunikace umožnila pořízení snímků obrazovek použitých v této práci a sloužila také při testování jádra OS. Komunikace byla jeden z důvodu proč zavést do systému OS. Umožňuje totiž, aby komunikace probíhala na pozadí a přitom mohla být povolena přerušení a systém mohl normálně fungovat.

Komunikace je vytvořena nad sériovým rozhraním řízeným pomocí přerušení. O komunikaci na nejnižší úrovni se starají obslužné rutiny přerušení, které jsou schopny pomocí front zpráv přenášet data do a z úkolu starajícího se o komunikaci na vyšší úrovni a zároveň jsou schopny udělat změnu kontextu na komunikační úkol. Ten nepřistupuje k frontám přímo, ale prostřednictvím pomocných funkcí zapouzdřujících celou komunikaci po sériové lince. Toto zapouzdření vytváří ovladač sériové linky. Funkce určené ke komunikaci se sériovým portem dávají programátorovi možnost posílání dat po bytech přímo přes funkci pomocí zásobníku, nebo posílání a přijímání dat s přímým přístupem do paměti, kdy přijímací funkce nabízí uživateli kontrolu přijatých dat kontrolním součtem.

Pro obsluhu komunikace a vyřizování příchozích požadavků byl vytvořen samostatný úkol. Úkol je opět tvořen nekonečnou smyčkou. Hned na začátku smyčky přijímá data ze sériové linky.

Když nejsou žádné k dispozici úkol je pozastaven do přijetí dat, nebo do vypršení časového limitu. Komunikace umožňuje vyčítat kteroukoliv adresovatelnou část paměti včetně videopaměti a z ní je schopen přiložený program vyčítat a interpretovat data ve správném tvaru. Pak vyčtená data umožňuje uložit jako bitmapu. Tato funkce komunikace však měla u vývoje mnohem větší úkol a to ten, že se pomocí ní dokázal kontrolovat obsah paměti, či už paměti RAM, ve které nás zajímali především zásobníky nebo flash, do které se ukládá program.

6 Závěr

Úspěšným přenesením operačního systému na elektroniku parního sterilizátoru se otevírají nové možnosti návrhu a vývoje řídícího softwaru pro tyto zařízení. Možnost znovupoužití dosavadního kódu zaručuje rychlou integraci OS také do jiných řídících vestavěných systémů vyráběných ve firmě BMT. Práce dále poukazuje na široký výběr OS s různými vlastnostmi, proto je vhodná v případě hledání vestavěného operačního systému pro jakoukoli jinou cílovou architekturu či zařízení. Moje implementace obsluhy dotekového panelu udělaná v rámci této práce, zaručuje rychlou odezvu systému se sníženými výpočetními nároky. Také implementace kreslícího plátna přináší nové možnosti využití dotekového panelu jako vstupního zařízení.

V současnosti se již v projektu pokračuje a plánuje se rozšíření OS o možnost nezávislého linkování od aplikace, což by umožnilo nahrávání OS nezávisle na aplikaci a usnadnilo by to vývoj a údržbu systému. To se sebou přináší také potřebu spravování flash paměti, kde by tím pádem bylo vhodné zavést souborový systém a implementovat jeho podporu do OS. Dále by bylo vhodné rozšířit tento OS o podporu čítačů, který jsem při vývoji postrádal, hlavně pro navracení zmáčknutých tlačítek GUI do původní polohy.

Celá aplikace byla testovaná a její funkčnost je taky demonstrovaná použitými obrazovkami, které jsou vidět na obrázcích 5.2 a 5.3. Samotné testování operačního systému je načrtnuto v kapitole 4.3 kde je popis jeho implementace. Vizuálně byla funkčnost OS kontrolována rozsvěcováním LED diod indikujících komunikaci. Po implementaci obsluhy dotekového panelu a displeje se testování usnadnilo. Testování dotekového panelu probíhalo vytvářením tlaku na dotekový panel pomocí tenkého tvrdého předmětu. Tým bylo možné zjistit rozdíly souřadnic mezi obrazovkou a dotekovým panelem. Takto bylo možné zjistit ofsety souřadnic a dosáhnout, aby místo doteku odpovídalo místu na obrazovce. Při implementaci obsluhy tlačítek se zase vyskytl problém s nechtěným mačkáním tlačítek. Nastávalo to při změně obrazovky, kdy pro stejnou polohou tlačítka na změnu obrazovky se jeho zmáčknutím začali rychle přepínat obrazovky. Problém byl vyřešen zakázáním zmačknutí tlačítek po změně obrazovky až do uvolení dotekového panelu.

Práce je dobrým začátkem pro další vývoj aplikací používajících ke komunikaci s okolím dotekový displej. Samotný přínos operačního systému z hlediska zefektivnění využití CPU a snížení času stráveného v kritických sekcích není v tak malé aplikaci znatelný, je však možné předpokládat jeho přínos v tomto směru u rozsáhlejších aplikací.

Literatura

[1] Labrosse, J., et. al. *Embedded software*. 1st ed. Amsterdam: Newnes, 2008. xxi, 770 s. ISBN 978-0-7506-8583-2

[2] Noergaard, T. *Embedded system architecture*. Amsterdam: Newnes, 2005. xiv, 640 s. ISBN 0-7506-7792-9

[3] Silberschatz, A. *Operating system concepts with Java*. 6th ed. New York: John Wiley, 2004. xxiii, 952 s. ISBN 0-471-45249-1

[4] Li, Qing. *Real-time concepts for embedded systems*. 1st ed. New York: CMP Books, 2003. 294 s. ISBN 1-57820-124-1

[5] Hitachi Kodaira Semiconductor. *H8S/2633 Series, H8S/2633 F-ZTAT™ Hardware Manual*. 3rd ed. [Japan]: Hitachi, 2001. 1160 s.

[6] Wikimedia Foundation. e*Cos* [online]. [cit. 2009-5-10]. Dostupné z: <http://en.wikipedia.org/wiki/ECos/>

[7] Barry, Richard. *Features Overview* [online]. [cit. 2009-5-11]. Dostupné z: <http://www.freertos.org/>

[8] Wikimedia Foundation. *List of operating systems* [online]. [cit. 2009-5-10]. Dostupné z: <http://en.wikipedia.org/wiki/List_of_operating_systems>

[9] QNX Software Systems. *Medical and Life Science Technology* [online]. [cit. 2009-5-10]. Dostupné z: <http://www.qnx.com/solutions/industries/medical/>

[10] Haigh, Thomas. *Multicians.org and the History of Operating Systems* [online]. [cit. 2009-5-13]. Dostupné z: <http://www.cbi.umn.edu/iterations/haigh.html>

[11] eCosCentric. *Runtime Functionality* [online]. [cit. 2009-5-10]. Dostupné z: <http://www.ecoscentric.com/ecos/features.shtml>

[12] Hawley, G. *Selecting a Real-Time Operating System* [online]. [cit. 2009-5-10]. Dostupné z: <http://www.embedded.com/1999/9903/9903sr.htm >

[13] BMT Medical Technology. *Sterident* [online]. [cit. 2009-5-13]. Dostupné z: <http://www.bmt.cz/default.asp?nDepartmentID=25&nLanguageID=1>

[14] Dionne, J., Durrant, M. *What is uClinux?*[online]. [cit. 2009-5-8]. Dostupné z: <http://www.uclinux.org/description/>

Seznam příloh

Příloha 1.

Seznam použitých zkratek

A/D	Analog/Digital
API	Application Program Interface
BMT	BMT Medical Technology s.r.o.
BSP	Board Support Package
CPU	Central Processing Unit
D/A	Digital/Analog
DSP	Digital Signal Processor
EDF	Earliest Deadline First
EW	Embedded Workbanch
FCFS	First Come First Serve
FIFO	First In First Out
FS	File System
FTP	File Transfer Protocol
GPS	Global Positioning System
HAL	Hardware Abstraction Layer
I/O	Input/Output
ID	Identifikátor
ISA	Instruction Set Architecture
LCD	Liquid Crystal Display
MCU	Micro Controller Unit
MMU	Memory Management Unit
OS	Operating System
PC	Personal Computer
PC	Program Counter
PCB	Process Control Block
PDA	Personal Digital Assistant
PO	Plošný Obvod
RAM	Random Access Memory
RGB	Red Green Blue
RMS	Rate Monotonic Scheduling
ROM	Read Only Memory
RT	Real-Time
RTOS	Real-Time Operating system
SPN	Shortest Process Next
TCB	Task Control Block
USB	Universal Serial Bus
WDT	Watch Dog Timer

Příloha 2.

Použitá verze vývojového prostředí a jeho součástí

IAR CSpyBat
4.7 (4.7.0.0)
22.5.2006 14:57:34, 131072 bytes

IAR Embedded Workbench IDE
4.7 (4.7.0.0)
22.5.2006 15:42:34, 741376 bytes

IAR XLINK
4.59Z (4.59.26.0)
10.5.2006 12:23:42, 1363968 bytes

IAR Assembler for H8
V2.10A/W32 (2.10.1.7)
13.6.2006 12:50:28, 630784 bytes

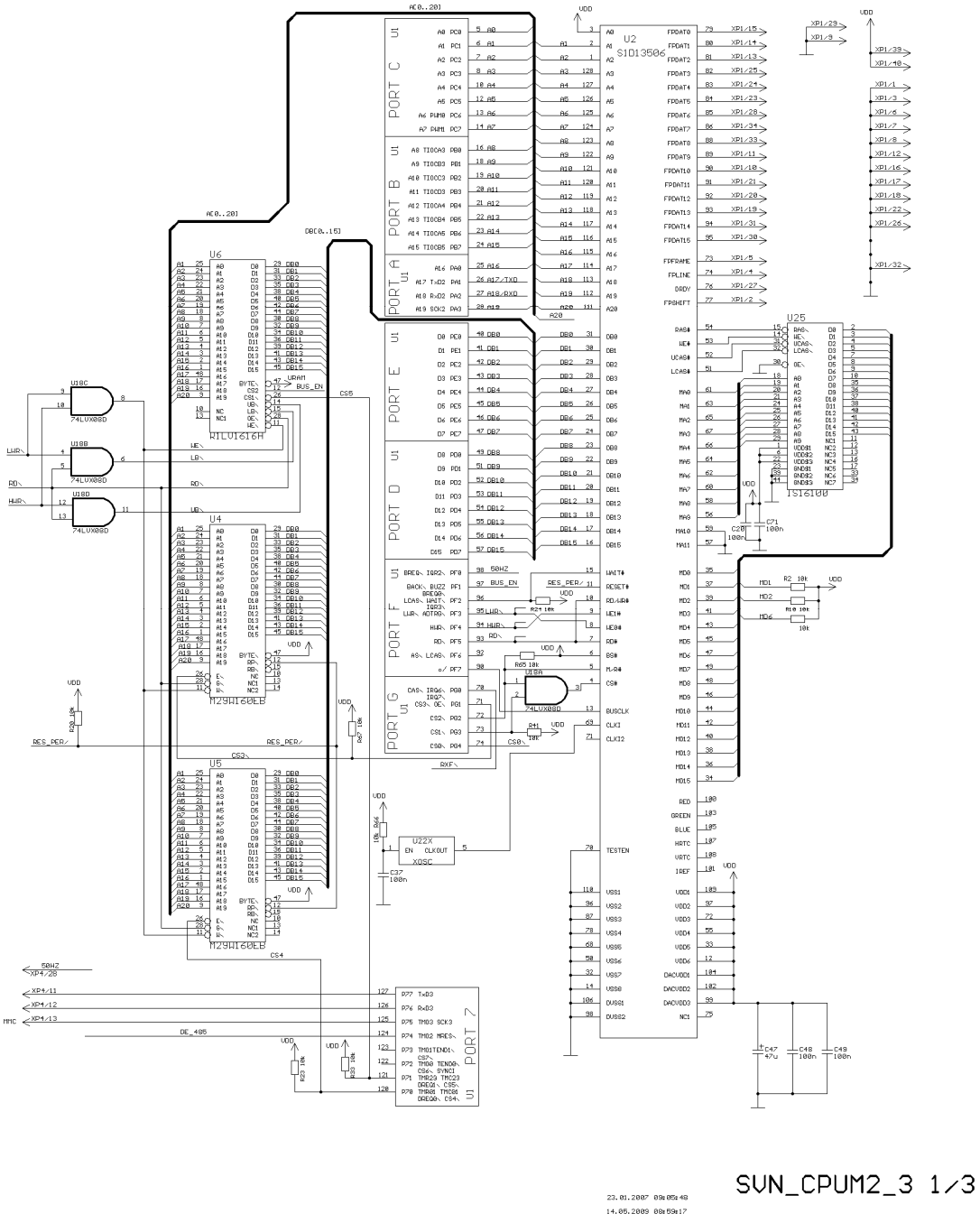
IAR C/C++ Compiler for H8
V2.10A/W32 (2.10.1.7)
13.6.2006 13:01:26, 9740288 bytes

IAR C-SPY Debugger GUI
4.7 (4.7.0.0)
29.5.2006 13:33:04, 958464 bytes

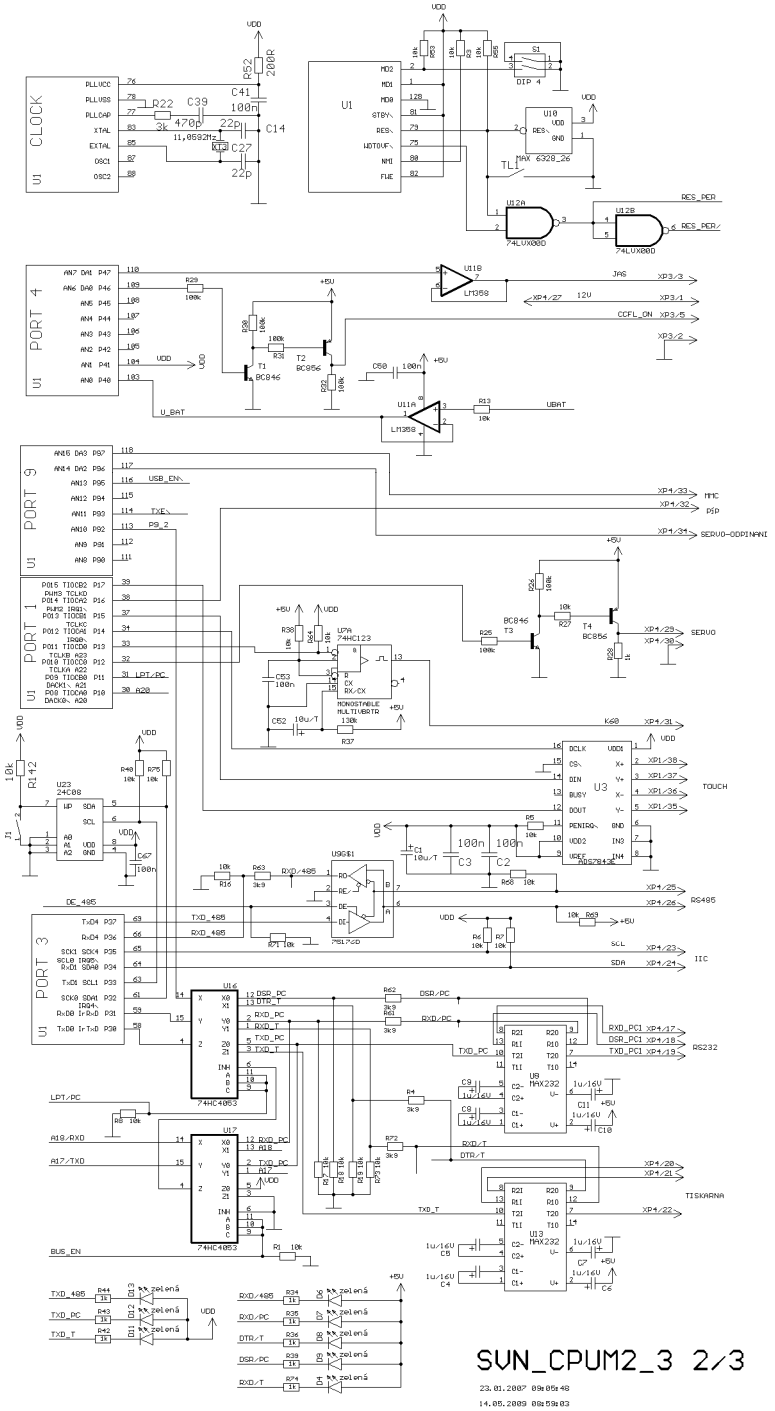
IAR C-SPY Debugger Kernel
4.7 (4.7.0.0)
22.5.2006 14:38:46, 1101824 bytes

Příloha 3.

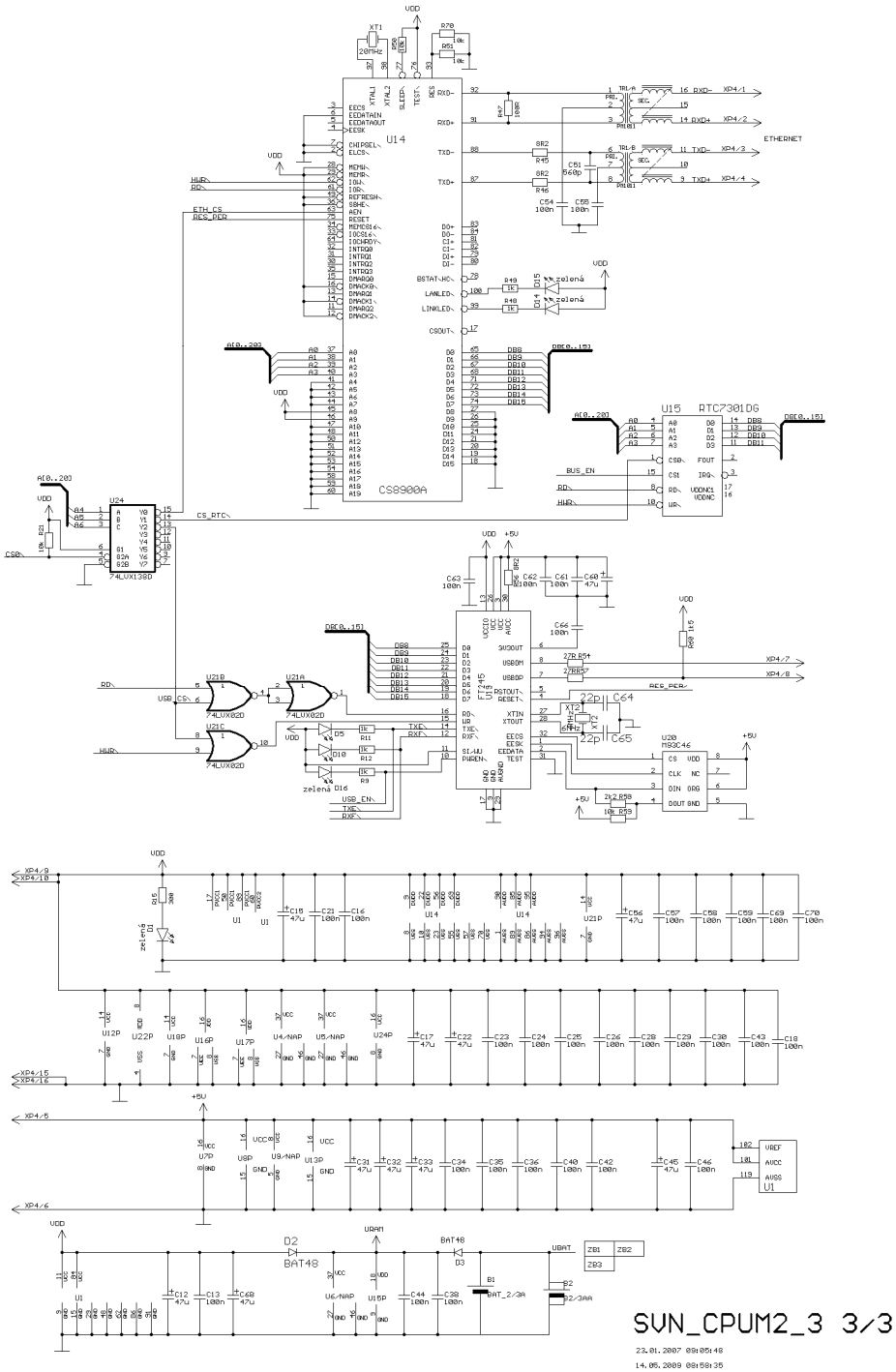
Schémata zapojení použité elektroniky



Obrázek přílohy 3.1: Schéma zapojení pamětí master elektroniky parního sterilizátoru



Obrázek přílohy 3.2: Schéma zapojení periferií master elektroniky parního sterilizátoru



Obrázek přílohy 3.3: Schéma zapojení master elektroniky parního sterilizátoru a napájení

Příloha 4.

Seznam vestavěných a real-time operačních systémů

A

A/ROSE
AmiQNX
AMX
Ardence RTX
B

BeRTOS
BlackBerry OS
BrickOS operating system
C

Cisco IOS
CMX RTOS
Contiki
Cosmos (operating system)
D

DNIX
DrRtos
DSOS
DSP RTOS
DSPnano RTOS
E

eCos
Embedded Linux
embOS
Erika Enterprise
EROS
F

Femto OS
FreeDOS
FreeOSEK
FreeRTOS
Fusion RTOS
Helium
HP-1000/RTE
ChibiOS/RT
ChorusOS
I

Iguana (operating system)
Inferno (operating system)
Integrity (operating system)
iRMX
J

Java Card OpenPlatform
JavaOS
JUNOS
L

LeJOS
LINX (IPC)
LynxOS
M

MaRTE OS
MenuetOS
MERT
Mobilinux
Multiuser DOS
N

Nano-RK
NetBSD
Neutrino
Nokia OS
Nucleus OS
NuttX RTOS
NxtOSEK
O

Open AT OS
OpenBSD
OpenBSD security features
Operating System Embedded
OS2000
OS-9
OSE
OSEK
OS-X
P

Palm OS
Passport Carrier Release
PaulOS
Phar Lap ETS
Phoenix-RTOS
pico]OS
PICos18
PikeOS
Plan 9 from Bell Labs
Prex
pSOS
Q

QNX
R

RMX
Rockbox
RSX-11
RT-11
RTAI
RTCX
RTEMS
RTLinux
S

Salvo
Scout (operating system)
Series 40 (software platform)
Series 80 (software platform)
SHaRK
SimpleAVROS
SINTRAN III
SOOS Project
Symbian OS
Symobi
T

Talon DSP RTOS
THEOS
ThreadX
Timos
TinyOS
TNKernel
Trampoline Operating System
Transaction Processing Facility
TRON Project
TUD:OS
U

Ubuntu Studio
Unison Operating System
UNIX-RTR
u-velOSity
V

velOSity
Versatile Real-Time Executive
VMX kernel
VRTX
VxWorks
W

Windows CE
Windows Embedded
X

Xenomai
XMK (operating system)
µ

µC/OS-II
µClinux
µnOS