****

**CURSO SUPERIOR DE ENGENHARIA DE COMPUTAÇÃO**

**OSÉIAS KNEVITZ ROCHA**

**GERADOR DE CÓDIGO E ABSTRAÇÃO DE REDE PARA PROTOCOLOS PERSONALIZADOS (CNAG)**

**Porto Alegre**

**2021OSÉIAS KNEVITZ ROCHA**

**GERADOR DE CÓDIGO E ABSTRAÇÃO DE REDE PARA PROTOCOLOS PERSONALIZADOS (CNAG)**

**Trabalho apresentado para o Curso de Engenharia de Computação, da Faculdade de Tecnologia FTEC como parte dos requisitos para avaliação da unidade curricular de TCC.**

**Orientador (a): Prof. Esp. ou Me ou Dr.Xxxx Xxxx**

**Coorientador (a): Prof. Esp. ou Me ou Dr. Xxxx Xxxx**

**Porto Alegre**

**2021**

**OSÉIAS KNEVITZ ROCHA**

**GERADOR DE CÓDIGO E ABSTRAÇÃO DE REDE PARA PROTOCOLOS PERSONALIZADOS (CNAG)**

**Trabalho apresentado para o Curso de Engenharia de Computação, da Faculdade de Tecnologia FTEC como parte dos requisitos para avaliação da unidade curricular de TCC.**

**Aprovado em \_\_\_/\_\_\_\_/\_\_\_\_**

**BANCA EXAMINADORA**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Professor Orientador: Esp. ou Me ou Dr. Xxxx Xxxx**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Professor Avaliador: Esp. ou Me ou Dr. Xxxx Xxxx**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Professor Avaliador: Esp. ou Me ou Dr. Xxxx Xxxx**

**Porto Alegre**

**2021**

**DEDICATÓRIA**

(Opcional)

**AGRADECIMENTOS**

(Opcional)

**EPÍGRAFE**

(Opcional)

**LISTA DE FIGURAS**

[Figura 1 - Estrutura de uma aplicação em QT 14](#_Toc76841628)

[Figura 2 - Hello World em C 15](#_Toc76841629)

[Figura 3 - Hello World em C++ 16](#_Toc76841630)

[Figura 4 - Exemplo de classe em C++ 17](#_Toc76841631)

[Figura 5 - Chip Intel 8086 18](#_Toc76841632)

[Figura 6 - Chip Intel 80386 19](#_Toc76841633)

[Figura 7 - Evolução da arquitetura ARM 20](#_Toc76841634)

[Figura 8 - Placa STM32 Blue Pill com um ARMv7 STM32F103C8 e sua pinagem 20](#_Toc76841635)

[Figura 9 - Chip M1 da Apple 22](#_Toc76841636)

[Figura 10 – Interface gráfica do Ubuntu Linux 24](#_Toc76841637)

[Figura 11 – Shell do Ubuntu Linux 24](#_Toc76841638)

[Figura 12 - Anatomia de um sistema Linux embarcado 25](#_Toc76841639)

[Figura 13 - Shell do Nuttx 26](#_Toc76841640)

[Figura 14 - Objetivos da AUTOSAR 28](#_Toc76841641)

[Figura 15 - Arquitetura AUTOSAR 29](#_Toc76841642)

**LISTA DE TABELAS**

(Opcional para menos de três)

**LISTA DE ABREVIATURAS E SIGLAS**

(Opcional para menos de três)

**SUMÁRIO**

[1. INTRODUÇÃO 10](#_Toc76841832)

[2. OBJETIVOS E JUSTIFICATIVA 12](#_Toc76841833)

[2.1. OBJETIVOS GERAIS 12](#_Toc76841834)

[2.1.1. Objetivos específicos 12](#_Toc76841835)

[2.2. JUSTIFICATIVA 12](#_Toc76841836)

[3. FUNDAMENTAÇÃO TEÓRICA 14](#_Toc76841837)

[3.1. Framework Qt 14](#_Toc76841838)

[3.2. Linguagens suportadas 15](#_Toc76841839)

[3.2.1. Linguagem C 15](#_Toc76841840)

[3.2.2. Linguagem C++ 16](#_Toc76841841)

[3.3. Arquiteturas de Hardware suportadas 17](#_Toc76841842)

[3.3.1. Arquitetura x86\_64 18](#_Toc76841843)

[3.3.2. Arquitetura AARCH64 19](#_Toc76841844)

[3.4. Sistemas Operacionais suportados 22](#_Toc76841845)

[3.4.1. Linux 22](#_Toc76841846)

[3.4.2. NuttX 25](#_Toc76841847)

[3.5. Certificação e homologação de código 27](#_Toc76841848)

[3.5.1. AUTOSAR 27](#_Toc76841849)

[REFERÊNCIAS 30](#_Toc76841850)

1. INTRODUÇÃO

Com o desenvolvimento da tecnologia, hoje tem-se a oportunidade de ter dispositivos eletrônicos inteligentes(smart), o que significa que são capazes de se comunicar com um dispositivo central e entre si, em geral, utilizando redes baseadas em rádio frequência. Os exemplos são variados, desde eletrodomésticos como ar condicionado, até relógios inteligentes (smart watches), com o último tendo limitações de bateria e componentes para processamento para consumir a menor quantidade de energia possível.

A variedade de aplicações de dispositivos smart é imensa, no geral, as fabricantes os produzem compatíveis apenas com dispositivos que ela mesma produz, ou ainda, de linhas específicas de produtos. Isso se dá pelo desenvolvimento de protocolos de rede fechados e específicos, sejam eles de camada sete(aplicação) do modelo OSI, ou mais baixo. Esse desenvolvimento fechado geralmente está associado à necessidade de consumir a menor banda e energia possível, o que leva às fabricantes criarem esses protocolos tão específicos para trafegar apenas informações estritamente necessárias.

Todo esse desenvolvimento necessita de mão de obra especializada e se torna muito custoso em tempo de desenvolvimento. Pensando nisso, este trabalho propõe o desenvolvimento de um software capaz de gerar camadas de abstração para interfaces de rede ethernet e as respectivas mensagens associadas às mesmas, para agilizar e facilitar o desenvolvimento destes protocolos. Ele será licenciado sob a GNU GPLv3(GNU General Public License version 3) e sua interface gráfica utilizará o framework Qt que também está licenciado sob a GNU GPLv3 e é muito difundido no mercado para aplicações open source e, principalmente, embarcados. Empresas como Mercedes-Benz, Peugeot, Ford e SKY utilizam esse framework em seus projetos.

O código gerado por esse software seguirá as normas da AUTOSAR, que é uma parceria global de empresas e entidades interessadas em desenvolvimento embarcado para automóveis. O motivo para isso é que, o código gerado no nível AUTOSAR de exigência, pode embarcar aplicações safety critical (nível crítico de segurança) ou inferiores, o que abrange a maior parte do setor industrial.

Aplicações safety critical são aquelas que, quando em falha, podem colocar a vida e/ou a saúde das pessoas em risco. Existem diversos mercados que trabalham com aplicações safety critical, como na medicina, aeronáutica, automobilística, espacial e etc. Portanto, se fazem necessários padrões e guias para o desenvolvimento de tais aplicações, como a AUTOSAR, para que se possa tenha um processo de desenvolvimento contido, previsível, rastreável e testável.

Quando se trata de aplicações safety critical, o processo de desenvolvimento é extremamente importante, pois ele garante que os desenvolvedores estão implementando as capacidades necessárias da maneira mais segura possível, diminuindo o número de catástrofes. Um exemplo recente de caso onde o não seguimento de um processo levou a consequências catastróficas, é o acidente do avião 737 Max da Boeing que, em 2018 na Indonésia, após uma falha crítica em um subsistema que corrige a inclinação do avião, resultou na morte de 189 pessoas.

1. **OBJETIVOS E JUSTIFICATIVA**

Neste trabalho apresentam-se os seguintes objetivos.

## OBJETIVOS GERAIS

Desenvolver um software open source gerador de código que forneça camadas de abstração de rede e estruturas de mensagens para protocolos personalizados.

* + 1. Objetivos específicos
* Criar uma plataforma de desenvolvimento de alto nível de abstração seguindo as diretrizes open source de desenvolvimento de software;
* Pesquisar e determinar um padrão de desenvolvimento que garanta a conformidade com a AUTOSAR;
* Pesquisar e determinar as interfaces de rede que serão implementadas para a abstração;
* Desenvolver templates de código para a geração dinâmica de mensagens e suas dependências;
* Desenvolver arquiteturas de rede eficientes para cada interface determinada anteriormente;
* Garantir a expansão do suporte para diferentes arquiteturas de processador;
* Garantir a expansão do suporte para diferentes sistemas operacionais;
* Criar uma interface gráfica de fácil entendimento e utilização.

## JUSTIFICATIVA

Atualmente não há muitos softwares altamente difundidos no mercado que cumpram com os requisitos apresentados nos objetivos deste trabalho, e os que existem são projetos privados de custo alto para adquirir e treinar profissionais. Um exemplo de software no mercado é o SCADE da Ansys.

A ideia é que o software que será desenvolvido possa ajudar a todos que estudam e/ou trabalham com desenvolvimento de software embarcado e que o código gerado possa ser certificado de acordo com os padrões da AUTOSAR para sistemas críticos (automóveis, aeronaves, equipamentos médicos e etc.), e com isso facilitar os processos de desenvolvimento dos protocolos personalizados e certificação de projetos.

A utilização desse software poderá abranger estudantes, pesquisadores e profissionais que necessitam projetar desde sistemas simples de comunicação, até complexos sistemas distribuídos e protocolos de comunicação. Com esse software, os desenvolvedores terão um ganho de tempo e facilidade de integração das mensagens criadas, já que elas podem ser criadas via interface gráfica e atribuídas às interfaces de rede criadas também pelo usuário.

Também é interessante a utilização desse software sob a ótica do desenvolvedor que terá uma interface padrão para se comunicar com as interfaces de rede, sejam elas quais forem: sockets ethernet, serial, etc.

Em um projeto open source existem maiores chances de contribuições da comunidade para o crescimento do projeto e, por consequência, acaba gerando facilidade para os utilizadores do software que terão documentações e uma comunidade para tirar dúvidas.

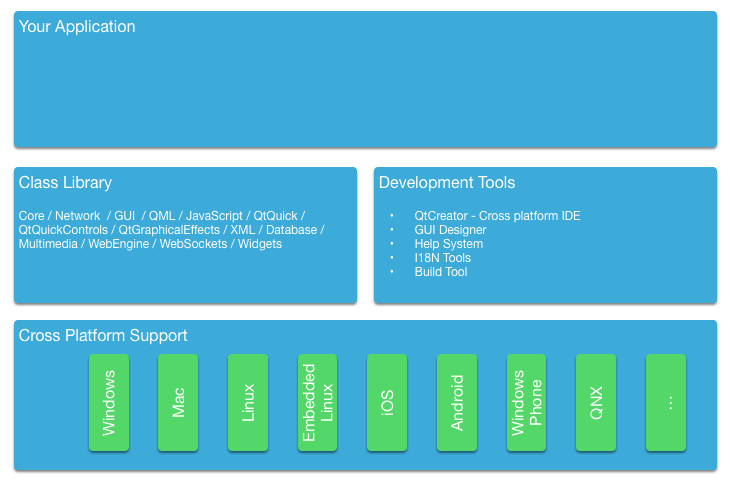
1. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os autores e bibliografias que formam a fundamentação teórica desta monografia. O sistema tem como objetivo dar ao usuário uma forma mais simples de implementar e controlar tipos de dados, mensagens e interfaces de comunicação, facilitando assim o desenvolvimento de aplicações que necessitam de mensageria específica. Este capítulo introduz a linguagem utilizada para o desenvolvimento do projeto e o framework para a interface gráfica com definições e opiniões de autores e desenvolvedores.

## Framework Qt

O framework Qt é um framework vasto que provê ferramentas em diversas áreas, como interfaces gráficas, threads, comunicação e etc. Ele é multiplataforma, podendo ser utilizado para computadores pessoais comuns rodando Linux, Windows, para smartphones com Android ou IOS, ou até mesmo em soluções embarcadas em Linux Embarcado, como pode ser visto na Figura 1 (LAZAR, 2016; PENEA, 2016).

Figura 1 - Estrutura de uma aplicação em QT



Fonte: QMLBook, 2020.

O Qt é muito utilizado devido a sua riqueza de recursos, a performance do C++, disponibilidade de código, a sua documentação, e, principalmente, porque os desenvolvedores gostam (BLANCHETTE, 2006; SUMMERFIELD, 2006).

O Qt faz sentido, é simples e permite ao programador se concentrar nos cumprimentos de suas tarefas. Quando os arquitetos originais o fizeram, e ainda fazem, pensaram não na solução mais simples ou uma boa solução, mas pensaram na solução certa e a documentaram. Claro que alguns erros foram cometidos, mas são corrigidos ao longo do tempo (BLANCHETTE, 2006; SUMMERFIELD, 2006).

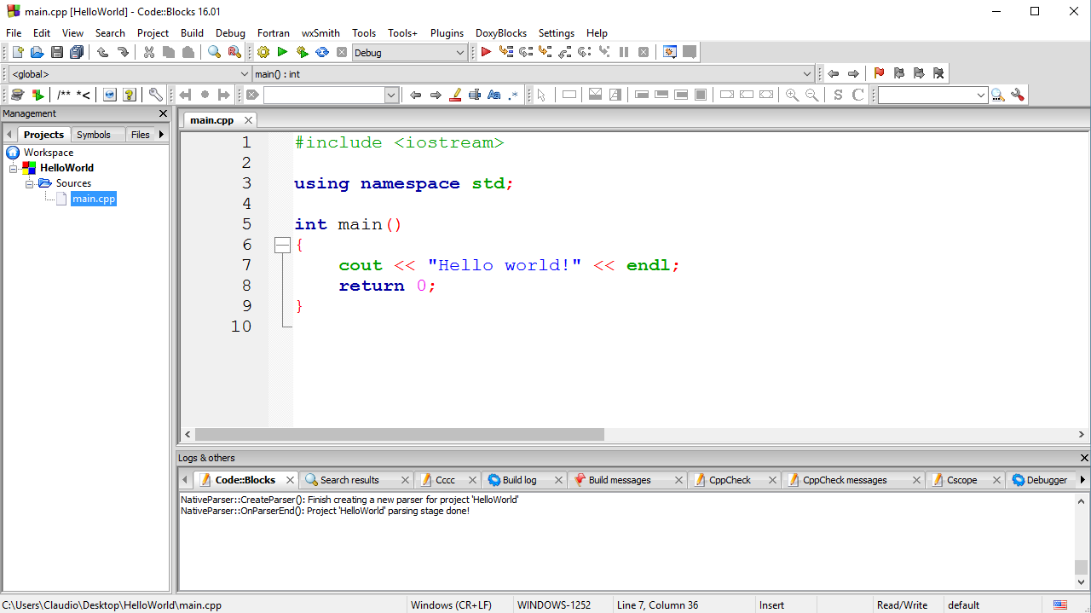
## Linguagens suportadas

Este capítulo introduz as linguagens que serão suportadas pelo sistema, ou seja, as linguagens em que ele gera código baseado nas definições do usuário.

* + 1. Linguagem C

A linguagem C foi criada por Dennis Ritchie, em 1972. É considerada uma linguagem de alto nível genérica e pode ser usada em diversos tipos de projeto, como drivers, sistemas operacionais, aplicativos, e etc. É uma linguagem estruturada e se tornou muito popular nos anos 1980 (NOLETO, 2020). A Figura 2 demonstra um programa simples e inicial em C.

Figura 2 - Hello World em C



Fonte: http://excript.com/, 2016

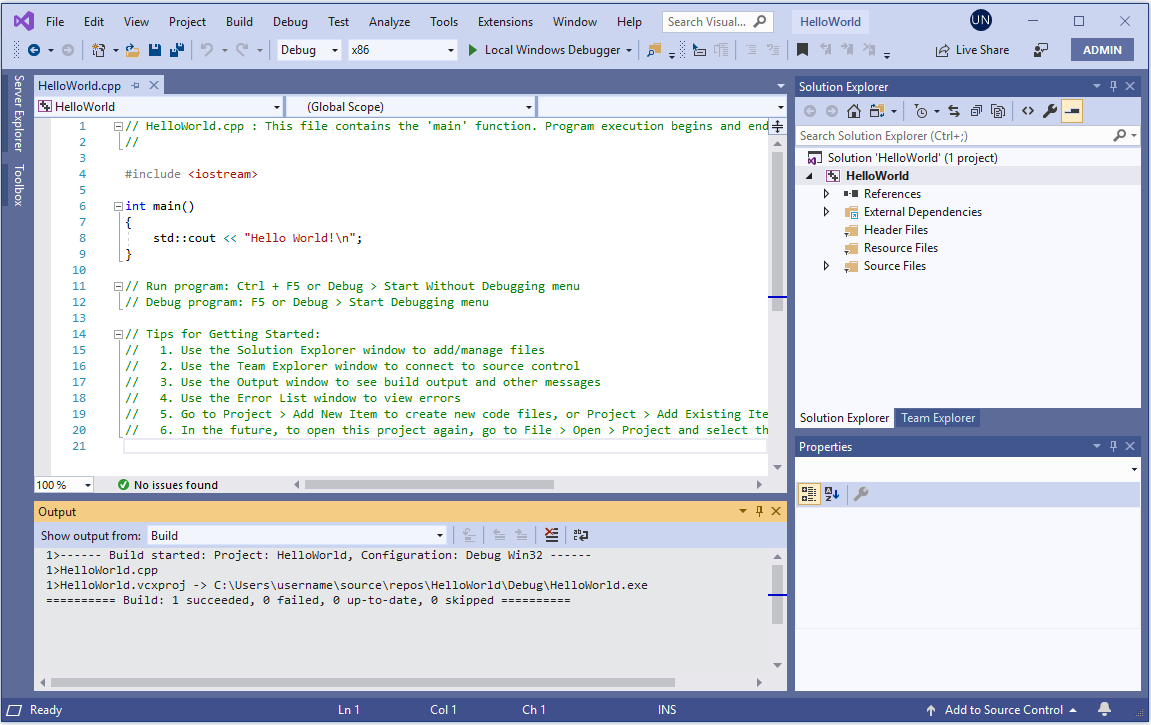
A linguagem C ficou muito conhecida como a linguagem de desenvolvimento do sistema operacional UNIX, mas hoje quase todos os sistemas operacionais são escritos em C e/ou C++. Atualmente o C está disponível para a maioria dos computadores e é independente de hardware (DEITEL, 2011, DEITEL, 2011).

Em 1989 foi aprovado um padrão para a linguagem C pelo ANSI (American National Standards Institute). Em 1999 este padrão foi atualizado e foi chamado de INCITS/ISO/IEC 9899-1999, ele aperfeiçoa e expande as capacidades da linguagem (DEITEL, 2011; DEITEL, 2011). O padrão mais recente disponível até a data de escrita deste documento é o ISO/IEC 9899:2018.

* + 1. Linguagem C++

Na década de 1980, Bjarne Stroustrup queria criar uma versão distribuída do kernel Unix e por ser uma tarefa muito complexa, ele queria escolher uma linguagem que trouxesse facilidades na hora de escrever o código, mas ao mesmo tempo tivesse boa performance (DIAS, 2021). Na Figura 3 está demonstrado um código inicial em C++.

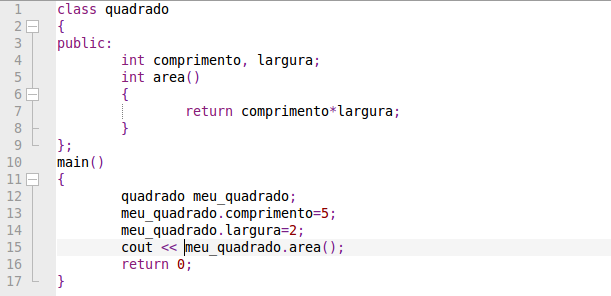
Figura 3 - Hello World em C++



Fonte: https://docs.microsoft.com/, 2020

Ainda segundo Dias (2021), linguagens como Simula já ofereciam facilidades para codificação, porém eram bastante lentas. Assim, o C++ surgiu a partir do C e inicialmente foi chamado de C with classes.

Figura 4 - Exemplo de classe em C++



Fonte: https://elias.praciano.com/, 2013

Segundo Noleto (2020), a linguagem C++ é orientada a objetos, enquanto a linguagem C é orientada a procedimentos e apesar de muito parecida com a linguagem C, a linguagem C++ pode ser considerada mais adaptável.

Segundo Dias (2021), a linguagem C++ é incrivelmente versátil e se destaca como líder nos seguintes cenários:

* Jogos;
* Mercado financeiro;
* Navegadores;
* Softwares multimídia;
* Sistemas Operacionais;
* Microcontroladores;
* Etc.

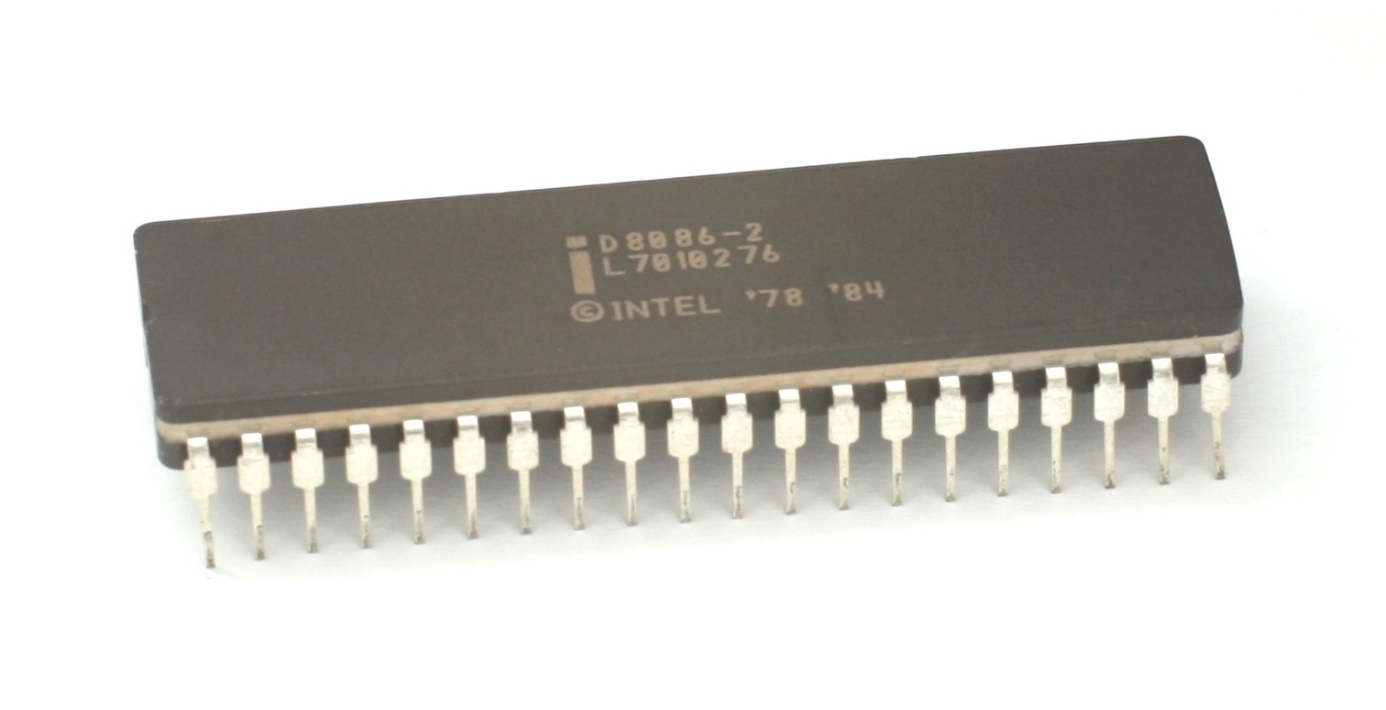
## Arquiteturas de Hardware suportadas

Este capítulo traz definições introdutórias sobre as arquiteturas de hardware suportadas, ou seja, os códigos gerados pelo sistema poderão ser compilados para estas arquiteturas sem necessidade de alteração de código gerado e/ou de bibliotecas utilizadas.

* + 1. Arquitetura x86\_64

Em 1978 o primeiro processador de 16 bits surgiu, o 8086 da Intel, e deu o nome à arquitetura de processadores utilizados, principalmente, em PCs hoje em dia (x86). Porém, o 8086 nem o 8088 (processador de 8 bits mais lento, porém mais barato que o 8086) conseguiam endereçar mais de 1 megabyte de memória. No início da década de 1980, isso se tornou um problema cada vez mais sério e, por isso, a Intel projetou o 80286, que era uma versão do 8086 compatível com os chips anteriores (TANENBAUM, 2013; AUSTIN, 2013).

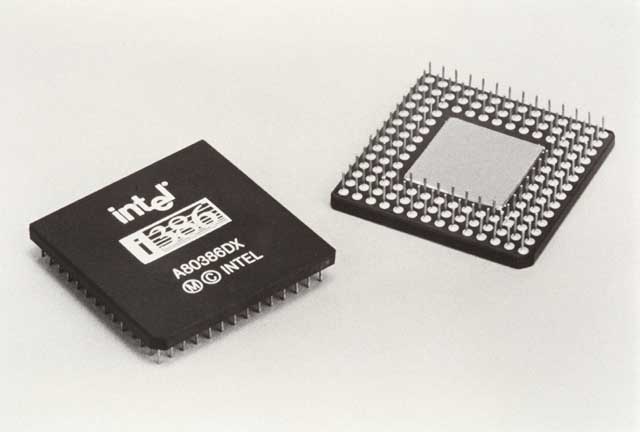
Figura 5 - Chip Intel 8086



Fonte: https://commons.wikimedia.org/, 2009

Em 1985 a Intel lançou o primeiro processador de 32 bits, o 80386, e assim como os seus antecessores, ele seguiu mais ou menos compatível com tudo que havia antes até o 8080 (TANENBAUM, 2013; AUSTIN, 2013).

Figura 6 - Chip Intel 80386



Fonte: https://www.cin.ufpe.br/, 2008

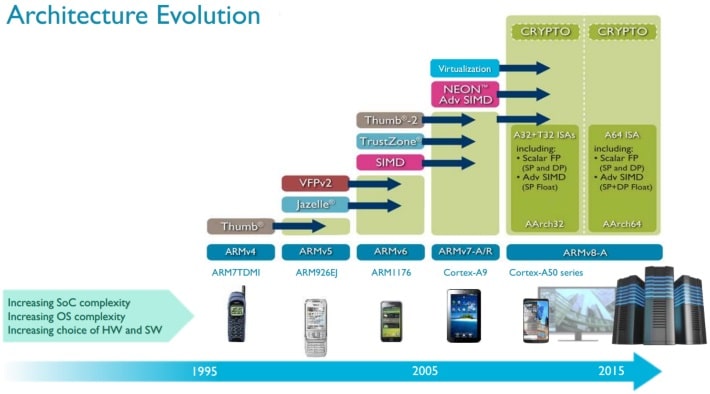
A arquitetura x86\_64 utilizada hoje em dia principalmente em PCs foi desenvolvida pela AMD e depois adotada pela Intel. É uma extensão e limpeza da arquitetura x86 introduzida pela Intel. Como o próprio nome indica, ela foi expandida para trabalhar com dados de 64 bits e endereçar 64 bits de memória. Também foi aumentado o número de registradores: 16 registradores de uso geral (GPR) e 16 registradores Streaming SIMD (XMM) (KLEINMAN, 2005).

A arquitetura provê dois modos de execução: long mode e modo legado. O long mode é o modo de execução que a arquitetura foi pensada para ser utilizada, é nele que código 64 bits é executado. Esse modo tem dois submodos: modo de compatibilidade, em que código 32 bits é executado sem perder desempenho, e modo 64 bits. Já o modo de execução legado executa como um processador x86 antigo (KLEINMAN, 2005).

* + 1. Arquitetura AARCH64

A linha de processadores ARM (Acorn RISC Machine ou Advanced RISC Machine, mais recentemente) começou em 1985 com um modelo denominado ARM-1 pela empresa Acorn Digital Computer e desde o início foi influenciado pela arquitetura RISC (Reduced Instruction Set Computer). Em 1994 foi lançado um projeto de grande sucesso chamado ARM7. Nessa época o ARM7 foi utilizado em larga escala, principalmente no celular com tecnologia GSM chamado Nokia 6110. Após esse período a arquitetura continuou evoluindo e alguns modelos clássicos se consolidaram: ARM7, ARM9 e ARM11 (GARCIA, 2017).

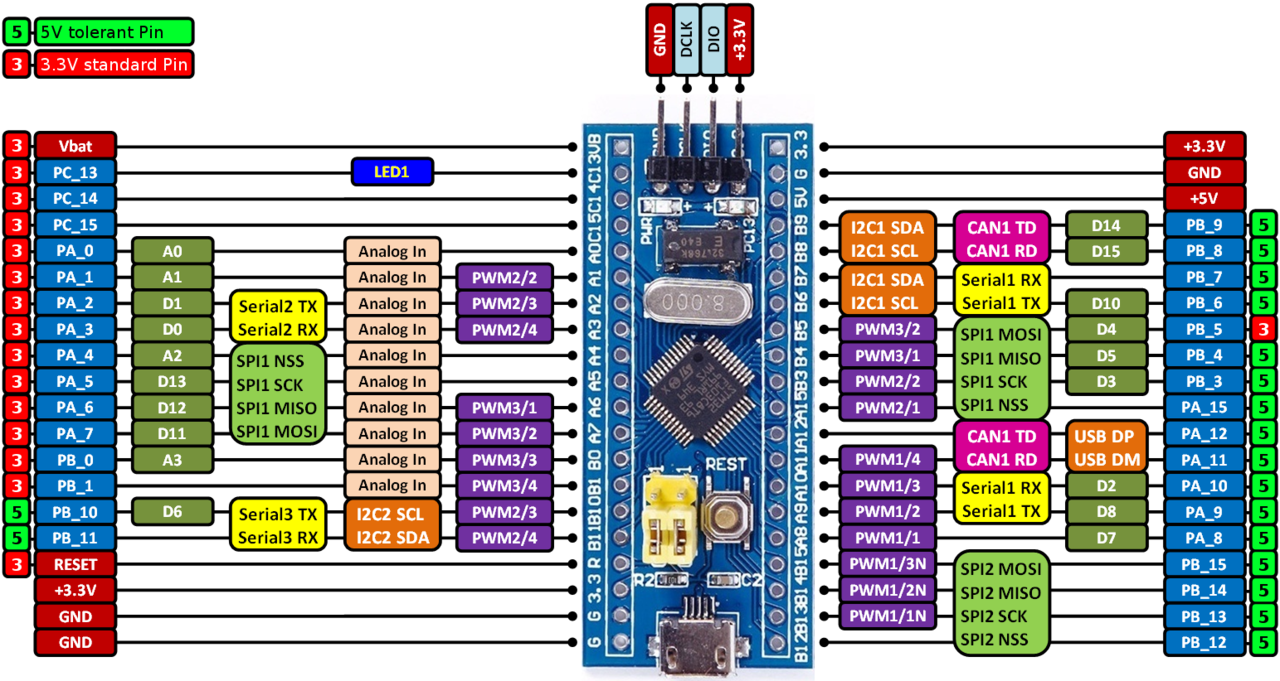
Figura 7 - Evolução da arquitetura ARM



Fonte: https://www.embarcados.com.br/, 2017

Em 2004 foi lançada a arquitetura ARMv7. Nesse período, a ARM diversificou seu portfólio de dispositivos com foco em aplicações industriais, criando a família Cortex, dedicada aos segmentos de aplicação, tempo real, críticos e sistemas embarcados de modo geral (GARCIA, 2017).

Figura 8 - Placa STM32 Blue Pill com um ARMv7 STM32F103C8 e sua pinagem



Fonte: https://os.mbed.com/, 2016

Hoje em dia os parceiros ARM vendem bilhões de processadores ARM a cada ano, mas ao contrário de muitas empresas de semicondutores, a ARM não fabrica processadores ou vende os chips diretamente, ao invés disso a ARM licencia os designs dos processadores para parceiros, incluindo a maioria das principais empresas de semicondutores do mundo. Com base nos projetos de processadores ARM de baixo custo e economia de energia, esses parceiros criam seus processadores, microcontroladores e soluções de sistema no chip. Esse modelo de negócios é chamado de licenciamento de propriedade intelectual (IP). Além dos designs de processador, a ARM também licencia IPs em nível de sistema e vários IPs de software. Para suportar esses produtos, a ARM desenvolveu uma base sólida de ferramentas de desenvolvimento, hardware e produtos de software para permitir que os parceiros desenvolvam seus próprios produtos (SILVA, 2020).

Os processadores ARM sempre apresentaram modelos com alta velocidade, die pequeno e de baixo consumo. Outra característica notável é o equilíbrio entre esses requisitos e sua capacidade de criar códigos compactos que tiram proveito do seu conjunto de instruções. Além disso, todos os processadores são desenvolvidos seguindo um modelo de arquitetura, conforme sua área de aplicação. Por exemplo, segundo GARCIA (2017) os processadores da família Cortex seguem a seguinte divisão:

* Cortex A: Linha de processadores de alto desempenho. É muito utilizado em aplicações mobile e plataformas.
* Cortex R: Linha de processadores para utilização em sistemas de tempo real em sistemas embarcados, automotivos, de controle industrial e aplicações críticas em geral que demandam muito processamento e processamento paralelo.
* Cortex M: Linha de processadores focado em microcontroladores, para sistemas embarcados de baixo custo e baixo consumo de energia.

Hoje os processadores ARM estão bastante espalhados pelo mercado, seja em smartphones, seja em dispositivos vestíveis ou single board computers (SBC), como o Raspberry Pi e muitos outros. A principal arquitetura utilizada hoje é a AARCH64, que é uma extensão para 64 bits da arquitetura já existente, e empresas como Facebook e Apple vêm tentando emplacar essa arquitetura em computadores de uso pessoal e em servidores, focando em economia de energia e desempenho (ENGELBERT, 2021).

Figura 9 - Chip M1 da Apple



Fonte: https://www.ifixit.com/, 2020

## Sistemas Operacionais suportados

Este capítulo traz um breve resumo dos sistemas operacionais e/ou kernels suportados pelo sistema. São importantes estas definições, pois as comunicações entre dispositivos são, geralmente, controladas pelo sistema operacional/kernel, o que faz o código ser muito atrelado ao sistema em que está rodando.

* + 1. Linux

Nos últimos anos, o Linux passou de parque de diversões de estudantes aficionados para um novo competidor no mercado de servidores, atingindo o estágio de sistema respeitado com lugar certo nas redes educacionais e corporativas (SIEVER *et al*, 2006, p.17).

O Linux foi desenvolvido inicialmente por Linus Torvalds, na Universidade de Helsinque, na Finlândia. De sua posição atual, no Vale do Silício, Linus continua a coordenar os aprimoramentos de forma centralizada.

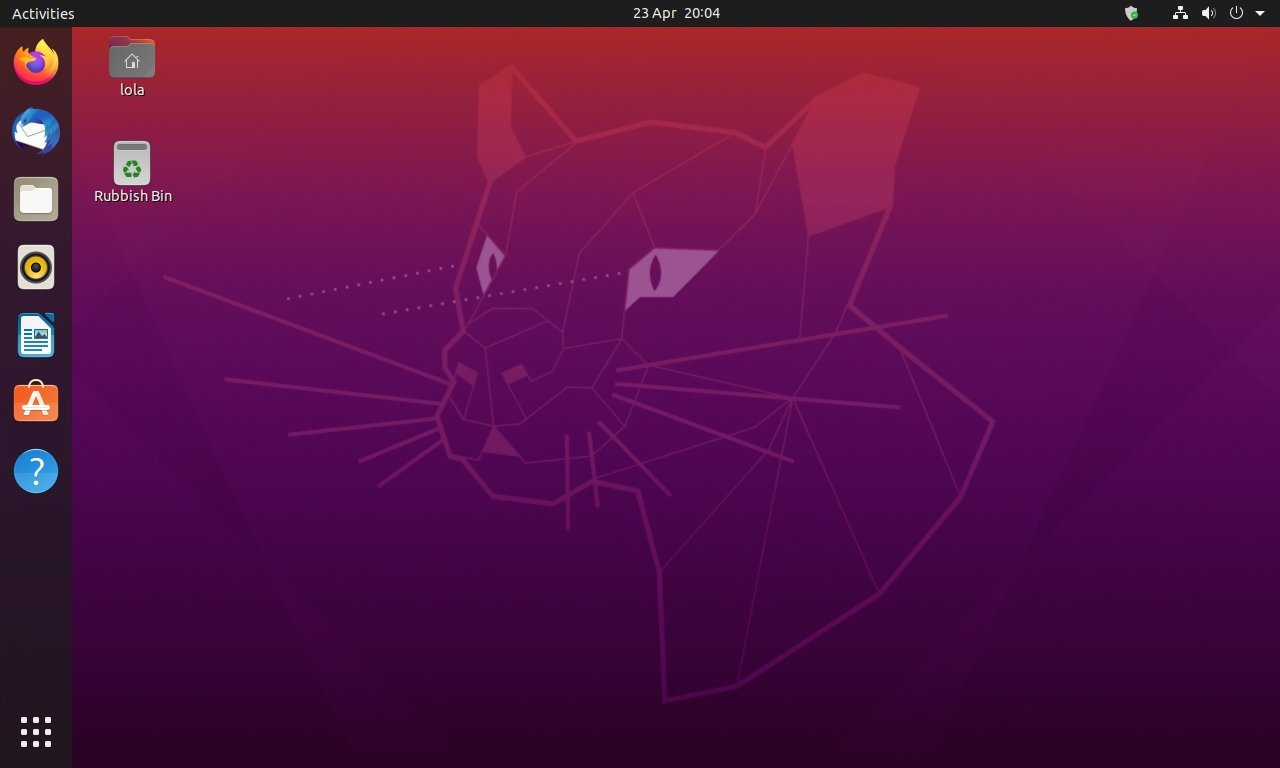
O kernel Linux continua a ser desenvolvido sob os auspícios de muitos outros programadores e aficionados de todo o mundo, acompanhados por membros de equipes de programação das principais companhias de informática, todos conectados pela Internet. Por kernel entendemos o núcleo do sistema operacional propriamente, não as aplicações (tais como compilador, shells, etc) que executam sobre ele.

Atualmente o termo Linux é usado frequentemente com o significado de um ambiente de software com kernel Linux, junto com um grande conjunto de aplicativos e outros componentes de software. Nesse significado mais amplo, muitas pessoas preferem o termo GNU/Linux, que reconhece o papel central desempenhado pelas ferramentas do projeto GNU da Free Software Foundation como complementos ao desenvolvimento do kernel Linux. (SIEVER *et al*, 2006, p.17).

Hoje, o Linux pode ser descrito como um sistema operacional de código-fonte aberto tipo UNIX, que reflete uma combinação de conformidade de padrões com o SVID, o POSIX e o BSD. O Linux continua a apontar para a conformidade com o POSIX, bem como com as normas estabelecidas pelo proprietário da marca UNIX, The Open Group (NEGUS, 2014, p. 26).

Embora os sistemas Linux tenham uma interface de usuário gráfica, a maioria dos programadores e usuários gráficos ainda prefere uma interface de linha de comando, chamada de interpretador de comandos (shell) (TANENBAUM, 2016; BOS, 2016, p. 520). Abaixo um exemplo de interface de usuário gráfica (GUI) do Ubuntu Linux.

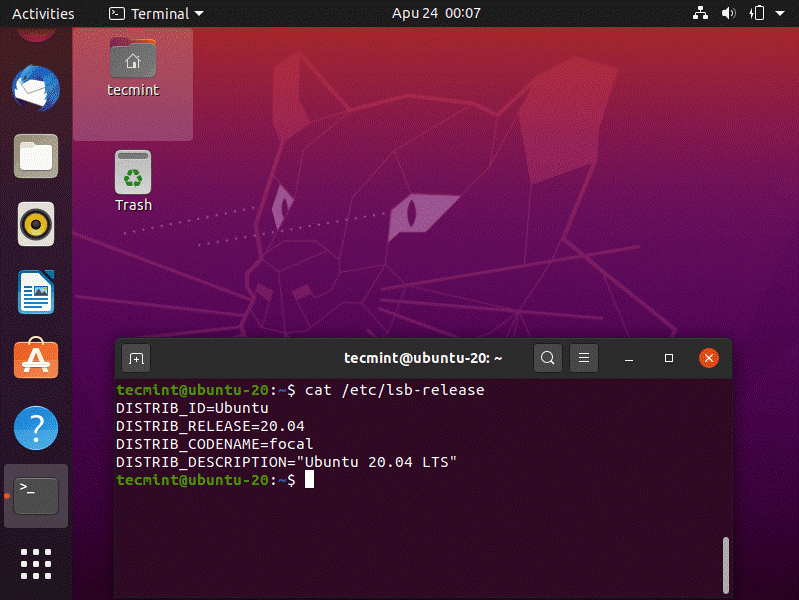
Figura 10 – Interface gráfica do Ubuntu Linux



Fonte: https://tecnoblog.net/, 2020

Na figura abaixo temos um exemplo de emulador de terminal do Ubuntu Linux. Um emulador de terminal faz o serviço do shell nas distribuições Linux com interface gráfica.

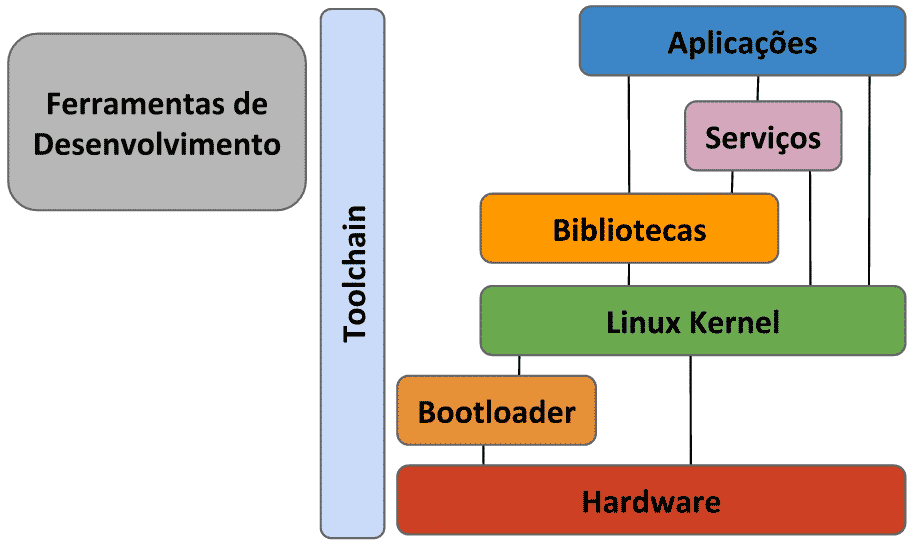
Figura 11 – Shell do Ubuntu Linux



Fonte: https://www.tecmint.com/, 2020

O Linux também é utilizado em sistemas embarcados e não se difere muito de um Linux usado em computadores desktop. As principais diferenças são relacionadas à customização e adaptações necessárias para embarcar no hardware específico e satisfazer, principalmente, os requisitos de desempenho, armazenamento e consumo de energia (SUEIRO, 2016).

Figura 12 - Anatomia de um sistema Linux embarcado



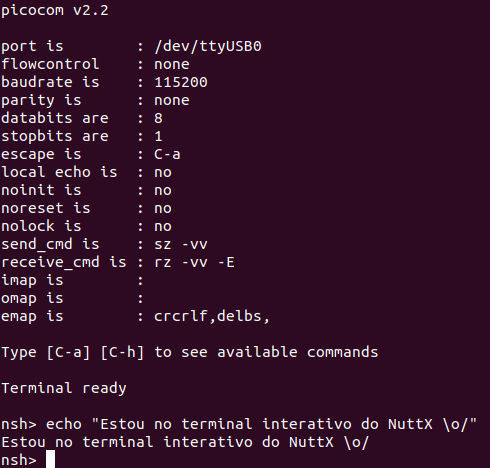
Fonte: https://www.embarcados.com.br/, 2016

Existe ainda a possibilidade de utilizar o Linux embarcado de forma real-time fazendo algumas alterações na forma em que compilamos o kernel o tornando totalmente preemptivo. Isso significa que todas as tarefas do kernel podem ser preemptadas para darem prioridade às aplicações real-time que devem rodar no sistema (SUEIRO, 2016).

* + 1. NuttX

O NuttX é um sistema operacional de tempo real (RTOS) com ênfase em conformidade com os padrões POSIX e ANSI, e possui um small-footprint, ou seja, ocupa pouco recurso de memória. Ele pode ser utilizado com microcontroladores e processadores de 8 até 64 bits e a sua conformidade com os padrões POSIX e ANSI colabora com a portabilidade e reutilização de código e suporte a aplicações que utilizam essa interface (MONTEIRO, 2021). O criador do NuttX é o senhor Gregory Nutt, daí a origem do nome NuttX (ASSIS, 2018).

Figura 13 - Shell do Nuttx



Fonte: https://www.embarcados.com.br/, 2020

Um sistema operacional de tempo real (RTOS) é mais especializado do que um sistema operacional de propósito geral, onde o tempo de resposta é mais importante do que executar centenas de tarefas “simultaneamente”. O tempo de resposta não precisa ser o mais rápido possível, mas deve ser previsível, ou seja, real-time pode ter uma resposta de vários segundos ou nano segundos. As tarefas de um RTOS têm um tempo limite para serem executadas, por isso, normalmente é chamado de time-critical (MORAIS, 2018).

O NuttX possui muitos subsistemas que se assemelham aos subsistemas do Linux. Por exemplo: Virtual File System (VFS), Memory Technology Device, subsistema de áudio, subsistema USB com suporte a USB Composite e entre outros. A principal diferença do NuttX para o Linux é em relação ao consumo de alguns recursos de hardware. O NuttX é muito pequeno, pode rodar com menos de 32KB de flash e menos de 8KB de RAM. Se forem adicionados mais recursos como Ethernet, Wifi, USB e etc, ele pode acabar consumindo um pouco mais de recursos, mas ainda menos que o Linux (ASSIS, 2018).

Esse sistema operacional tem sido muito utilizado pode diversas fabricantes. Ele foi utilizado nas capas especiais chamadas Snaps de uma série de smartphones da Motorola chamada Moto Z. Essas capas podiam adicionar diversas funcionalidades tais como: Projetor multimídia, câmera com zoom ótico, sistema de som e entre outros. A Sony também utiliza o NuttX em diversos aparelhos, como gravadores digitais e headphones Bluetooth. Outra grande empresa que está utilizando o NuttX é a Samsung, que criou um fork do NuttX e adicionou alguns recursos. Hoje o sistema da Samsung é chamado de TizenRT (ASSIS, 2018).

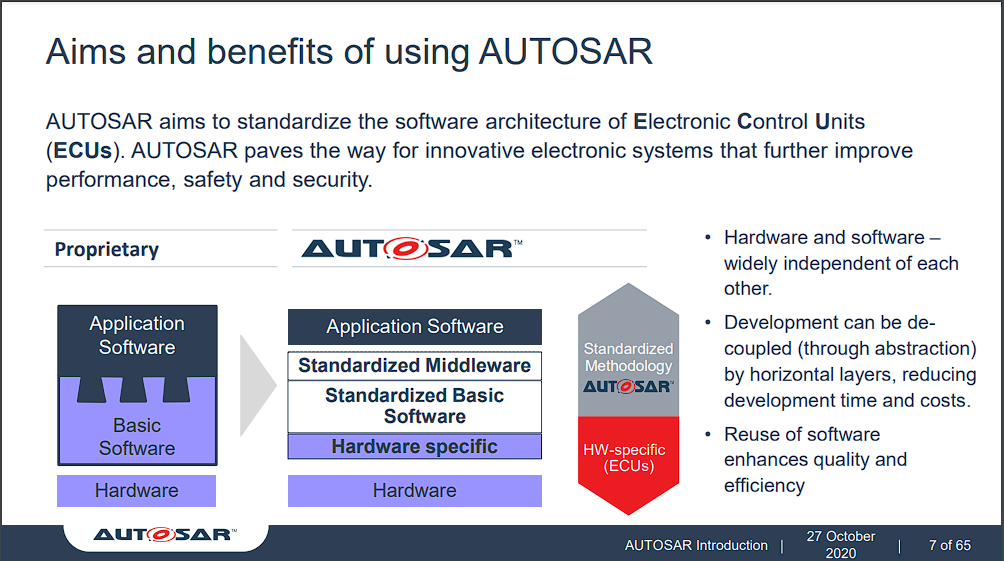
## Certificação e homologação de código

Este capítulo traz uma introdução à uma das mais conhecidas normas para aplicações safety critical do mundo, a AUTOSAR. Ela traz, além de normas para o processo de desenvolvimento, guidelines para escrever código seguro e certificável dentro dos seus parâmetros.

* + 1. AUTOSAR

O AUTOSAR (Automotive Open System Architecture) é uma iniciativa de padronização dos principais fabricantes, fornecedores e prestadores de serviços e empresas da área de eletrônica automotiva fundada em 2003, com o intuito de gerenciar a complexidade da arquitetura elétrica e eletrônica integradas por meio do aumento de reutilização e trocas de módulos de software OEM (Original Equipment Manufacturer) entre fornecedores, padronizando assim a arquitetura de software dos módulos (TEIXEIRA, 2019).

Figura 14 - Objetivos da AUTOSAR



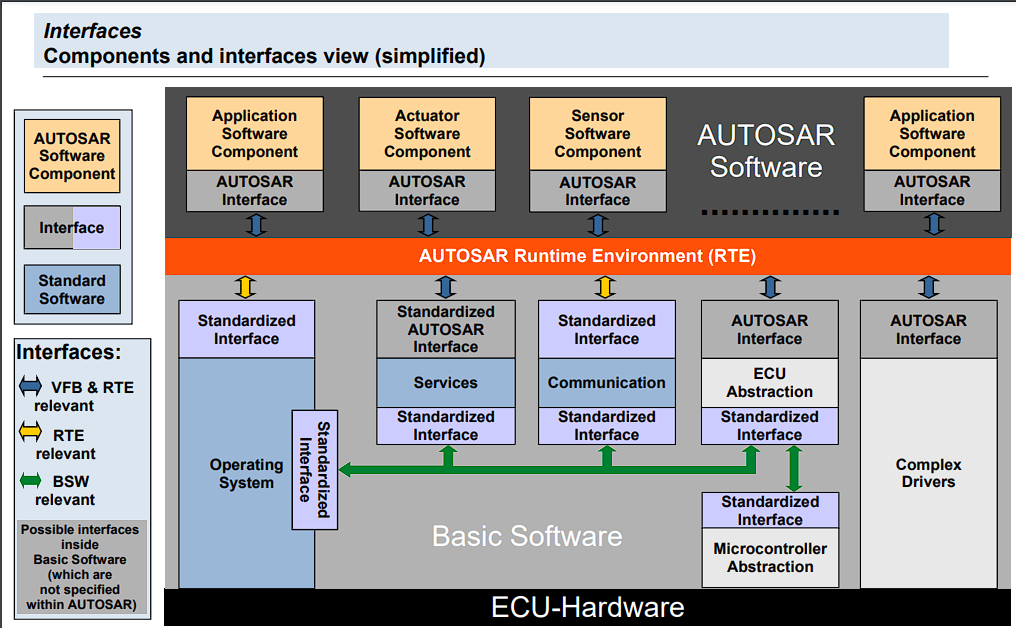
Fonte: AUTOSAR, 2020

Na arquitetura AUTOSAR, as funcionalidades do automóvel são encapsuladas em componentes de software separando aplicações de infraestrutura. Uma aplicação na AUTOSAR consiste em um conjunto de componentes de software interconectados através do VFB (Virtual Funcional Bus) (TEIXEIRA, 2019).

O VFB é o mecanismo de comunicação que permite aos componentes trocarem informações. Um componente tem entradas e saídas bem definidas que vão ser usadas para comunicação com outros componentes. Ele pode implementar funcionalidades muito simples ou muito complexas e contar com poucas ou muitas “portas” de comunicação e é justamente o VFB que possibilita todo o design ser baseado em componentes separados e independentes. As especificações do VFB devem incluir todos os conceitos para todos os serviços de infraestrutura que podem ser necessários por um componente para implementar uma aplicação automotiva (AUTOSAR, 2020). Isso inclui:

* Comunicação com outros componentes no sistema;
* Comunicação com sensores e atuadores no sistema;
* Acesso a serviços padrão, como ler ou escrever de uma memória;
* Responder a mudanças de operação;
* Interagir com sistemas de calibração e medições;

Figura 15 - Arquitetura AUTOSAR



Fonte: AUTOSAR, 2020

O Runtime Enviroment é o elo de ligação entre os componentes de software com o Basic Software. Basicamente é o RTE que implementa as funcionalidades, de fato, do VFB. Já o Basic Software é o responsável por implementar as abstrações das ECUs (Engine Control Unit) (AUTOSAR, 2020).

# REFERÊNCIAS

**A família 80x86.** Disponível em: <<https://www.cin.ufpe.br/~rcaf/ihs/familiax86.html>>. Acesso em: 08 Jun. 2021.

ALECRIM, Emerson. **Ubuntu 21.04 terá recurso de segurança pedido em 2006**. Disponível em: <<https://tecnoblog.net/403364/ubuntu-linux-21-04-mudanca-configuracao-permissoes/>>. Acesso em: 26 Jun. 2021.

ASSIS, Alan Carvalho de. **O que é o RTOS NuttX e por que você deve se importar?.** Disponível em: <<https://www.embarcados.com.br/o-que-e-o-rtos-nuttx/>>. Acesso em: 26 Jun. 2021.

AUTOSAR. **AUTOSAR Introduction**. Disponível em: <<https://www.autosar.org/fileadmin/ABOUT/AUTOSAR_EXP_Introduction102020.pdf>>. Acesso em: 08 Jul. 2021.

AUTOSAR. **Virtual Functional Bus**. Disponível em: <<https://www.autosar.org/fileadmin/user_upload/standards/classic/20-11/AUTOSAR_EXP_VFB.pdf>>. Acesso em: 08 Jul. 2021.

AUTOSAR. **Foundation Release Overview**. Disponível em: <<https://www.autosar.org/fileadmin/user_upload/standards/foundation/20-11/AUTOSAR_TR_FoundationReleaseOverview.pdf>>. Acesso em: 08 Jul. 2021.

AUTOSAR. **Layered Software Architecture**. Disponível em: <<https://www.autosar.org/fileadmin/user_upload/standards/classic/20-11/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf>>. Acesso em: 08 Jul. 2021.

**Criar um projeto de aplicativo de console C++**. Disponível em: <<https://docs.microsoft.com/pt-br/cpp/build/vscpp-step-1-create?view=msvc-160>>. Acesso em: 05 Jun. 2021.

DEITEL, Paul; DEITEL, Harvey. **C Como Programar**. 6 ed. São Paulo: Pearson Education do Brasil, 2011.

DIAS, Vinicius. **C++: Por que aprender em 2021?** Disponível em: <<https://www.alura.com.br/artigos/formacao-linguagem-c-plus-plus>>. Acesso em: 05 Jun. 2021.

ENGELBERT, Chris. **The Future Is Now:** ARM’s AARCH64 on the Rise. Disponível em: <<https://www.instana.com/blog/the-future-is-now-arms-aarch64-on-the-rise-with-instana/>>. Acesso em: 06 Jul 2021.

FILHO, Claúdio Rogério Carvalho. **PRIMEIRO PROGRAMA EM C++**:

CURSO C++. Disponível em: <<http://excript.com/cpp/primeiro-programa-cpp.html>>. Acesso em: 05 Jun. 2021.

GARCIA, Fernando Deluno. **Breve Histórico da ARM**. Disponível em: <<https://www.embarcados.com.br/breve-historico-da-arm/>>. Acesso em: 06 Jul. 2021.

GOLDHEART, Sam. **M1 MacBook Teardowns: Something Old, Something New** Disponível em: <<https://www.ifixit.com/News/46884/m1-macbook-teardowns-something-old-something-new>>. Acesso em: 06 Jul. 2021.

HUDAK, Zoltan. **STM32F103C8T6 board, alias Bluepill**. Disponível em: <<https://os.mbed.com/users/hudakz/code/STM32F103C8T6_Hello/>>. Acesso em: 03 Jul. 2021.

KILI, Aaron. **How To Install Ubuntu 20.04 Desktop**. Disponível em: <<https://www.tecmint.com/install-ubuntu-20-04-desktop/>>. Acesso em: 24 Jun. 2021.

KLEINMAN, Alan. **Uma visão geral da Arquitetura x86-64 e de processadores que a implementam**. Disponível em: <<https://www.ic.unicamp.br/~rodolfo/Cursos/mo401/2s2005/Trabalho/041438-x86.pdf>>. Acesso em: 08 Jul. 2021.

LANZET, Konstantin. Disponível em: <<https://commons.wikimedia.org/wiki/File:KL_Intel_D8086.jpg>>. Acesso em: 08 Jun. 2021.

LAZAR Guillaume; PENEA Robin. **Mastering Qt 5**. Birmingham: Packt, 2016.

MONTEIRO, Sara. **Primeiros Passos com o ESP32 e o NuttX**. Disponível em: <<https://www.embarcados.com.br/primeiros-passos-com-o-esp32-e-o-nuttx-parte-1/>>. Acesso em: 26 Jun. 2021.

MORAIS, José. **RTOS: Um ambiente multi-tarefas para Sistemas Embarcados**. Disponível em: <<https://www.embarcados.com.br/rtos-sistema-operacional-de-tempo-real/>>. Acesso em: 26 Jun. 2021.

NEGUS, Chistopher. **Linux – A Bíblia.** 8 ed. Rio de Janeiro: Alta Books, 2014

NOLETO, Cairo. **Linguagem C: o que é e quais os principais fundamentos!** Disponível em: <<https://blog.betrybe.com/linguagem-de-programacao/linguagem-c/>>. Acesso em: 05 Jun. 2021.

PRACIANO, Elias. **Um introdução à Programação Orientada a Objetos em C++ (parte III)**. Disponível em: <<https://elias.praciano.com/2013/09/um-introducao-a-programacao-orientada-a-objetos-em-c-parte-iii/>>. Acesso em: 05 Jun. 2021.

RYANNEL, Jürgen. **A Book about Qt5**. Disponível em: <<https://qmlbook.github.io/ch17-qtcpp/qtcpp.html>>. Acesso em: 05 Jun. 2021.

SIEVER, C. et al. **Linux: o guia essencial**. 5 ed. Porto Alegre: Bookman, 2006

SILVA, Ismael Lopes da. **Introdução ao microcontrolador ARM Cortex M3**. Disponível em: <https://www.embarcados.com.br/introducao-ao-microcontrolador-arm-cortex-m3/>. Acesso em: 06 Jul. 2021.

\_\_\_\_\_\_. **Conheça a placa Blue Pill (STM32F103C8T6).** Disponível em: <<https://www.embarcados.com.br/blue-pill-stm32f103c8t6/>>. Acesso em: 06 Jul. 2021.

SUEIRO, Diego. **Yocto Project: Introdução**. Disponível em: <<https://www.embarcados.com.br/yocto-project-introducao/>>. Acesso em: 26 Jun 2021.

SUMMERFIELD, Mark; BLANCHETTE, Jasmin. **C++ GUI Programming with Qt 4**. Stoughton: Prentice Hall, 2006

TANENBAUM, Andew; AUSTIN, Todd. **Organização estruturada de computadores**. 6 ed. São Paulo: Pearson Prentice Hall, 2013.

TANENBAUM, Andrew. BOS, Herbert. **Sistemas Operacionais Modernos**. 4 ed. São Paulo: Pearson Education do Brasil, 2016.

**APÊNDICE**

**ANEXOS**