****

**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)

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

**Oséias Knevitz Rocha**

Autor

oseiaskr95@gmail.com

**Nome do professor (a)**

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

Endereço de e-mail

**Resumo:**É uma descrição sumária da totalidade do TCC utilizando-se 150 a 500 palavras, são destacados os objetivos, o método, os resultados e conclusões mais importantes. Deve ser utilizada fonte 10, espaçamento simples em um único parágrafo, de forma discursiva afirmativa e não apenas uma lista de tópicos. O verbo utilizado deve estar na terceira pessoa. A ideia central do texto deve aparecer logo na primeira frase e, sequência informar a forma da pesquisa.

**Palavras-chave:** Xxxxx. Xxxxx. Xxxxx. Xxxxx. Xxxxx.

**CODE AND NETWORKING ABSTRACTION GENERATOR (CNAG)**

**Oséias Knevitz Rocha**

Author

oseiaskr95@gmail.com

**Nome do professor (a)**

Teacher(Esp. ou Me ou Dr.) Advisor

Endereço de e-mail

***Abstract:***It is a summary of the entire TCC using 150-500 words, objectives, methodology, results and important findings are highlighted. It should be used font size 10, single spaced in a single paragraph, affirmative discourse and not just a list of topics. The verb used must be in the third person. The central idea of ​​the text should appear soon in the first sentence and, after informing the form of research.

***Key-words:***Xxxxx. Xxxxx xxxxx. Xxxxx. Xxxxx.

**LISTA DE FIGURAS**

[Figura 1 - Estrutura de uma aplicação em QT 14](#_Toc76324240)

[Figura 2 - Hello World em C 15](#_Toc76324241)

[Figura 3 - Hello World em C++ 16](#_Toc76324242)

[Figura 4 - Exemplo de classe em C++ 17](#_Toc76324243)

[Figura 5 - Chip Intel 8086 18](#_Toc76324244)

[Figura 6 - Chip Intel 80386 19](#_Toc76324245)

[Figura 7 – Interface gráfica do Ubuntu Linux 20](#_Toc76324246)

[Figura 8 – Shell do Ubuntu Linux 21](#_Toc76324247)

[Figura 9 - Shell do Nuttx 22](#_Toc76324248)

**LISTA DE TABELAS**

(Opcional para menos de três)

**LISTA DE ABREVIATURAS E SIGLAS**

(Opcional para menos de três)

SUMÁRIO

[1 INTRODUÇÃO 10](#_Toc76325001)

[2 OBJETIVOS E JUSTIFICATIVA 12](#_Toc76325002)

[2.1 OBJETIVOS GERAIS 12](#_Toc76325003)

[2.1.1 Objetivos específicos 12](#_Toc76325004)

[2.2 JUSTIFICATIVA 12](#_Toc76325005)

[3 FUNDAMENTAÇÃO TEÓRICA 14](#_Toc76325006)

[3.1 Framework Qt 14](#_Toc76325007)

[3.2 Linguagens suportadas 15](#_Toc76325008)

[3.2.1 C 15](#_Toc76325009)

[3.2.2 C++ 16](#_Toc76325010)

[3.3 Arquiteturas de Hardware suportadas 17](#_Toc76325011)

[3.3.1 X86 18](#_Toc76325012)

[3.3.2 ARMv8-A 19](#_Toc76325013)

[3.4 Sistemas Operacionais suportados 19](#_Toc76325014)

[3.4.1 Linux 19](#_Toc76325015)

[3.4.2 Nuttx 21](#_Toc76325016)

[3.5 Certificação e homologação de código 22](#_Toc76325017)

[3.5.1 AUTOSAR 22](#_Toc76325018)

[3.5.2 MISRA 22](#_Toc76325019)

[4 PROCEDIMENTOS METODOLÓGICOS (podendo ser chamado de METODOLOGIA ou MATERIAIS E MÉTODOS dependendo do caso) 23](#_Toc76325020)

[5 APRESENTAÇÃO E ANÁLISE DOS RESULTADOS 23](#_Toc76325021)

[6 CONSIDERAÇÃO FINAIS 23](#_Toc76325022)

[SUGESTÕES PARA TRABALHOS FUTUROS 23](#_Toc76325023)

[REFERÊNCIAS 24](#_Toc76325024)

# 

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, e também seguirá algumas normas mais específicas da MISRA, que é também um padrão de desenvolvimento de software, porém criado e mantido pela Motor Industry Software Reliability Association. O motivo para isso é que, o código gerado no nível AUTOSAR e MISRA 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 e a MISRA, 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.

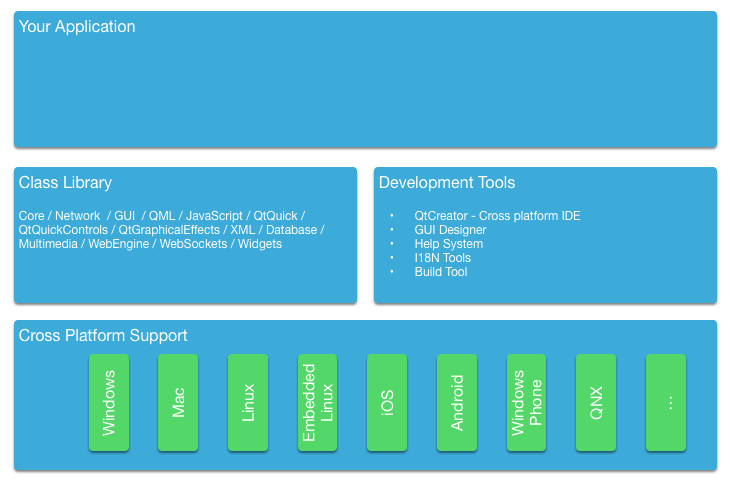
1. FUNDAMENTAÇÃO TEÓRICA

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 - 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).

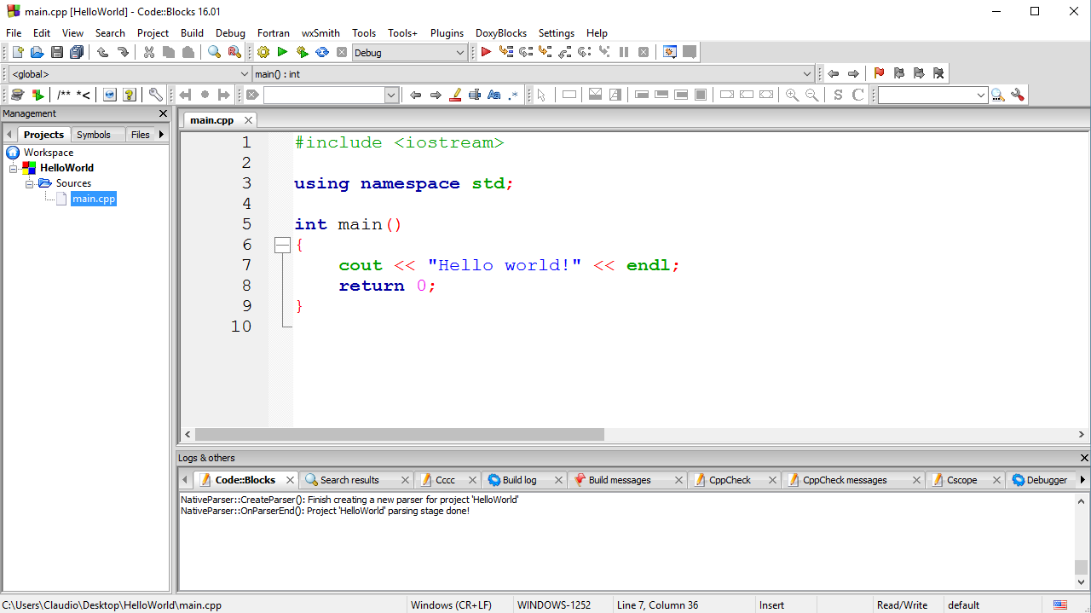
## Linguagens suportadas

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. 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/cpp/primeiro-programa-cpp.html, 2016

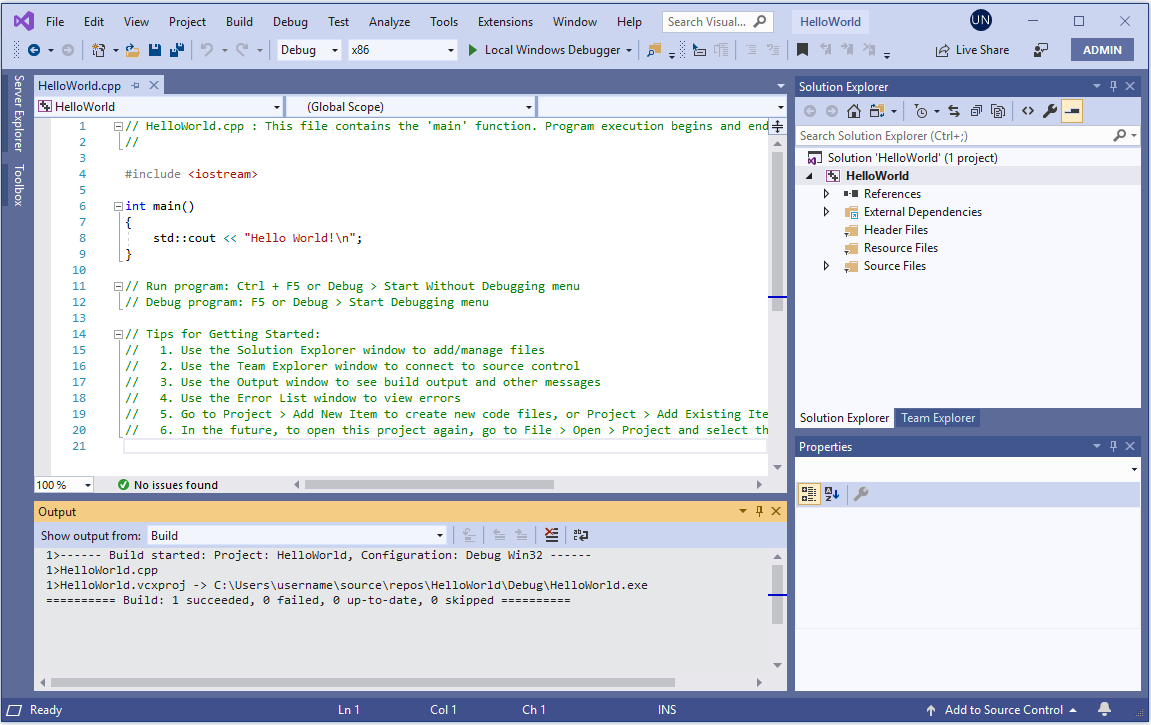
A linguagem C ficou muito conhecida como a linguagem de desenvolvimento do sistema operacional UNIX, mas hoje quase todos os sistemas operacionais são escritos em C e/ou C++. Atualmente o C está disponível para a maioria dos computadores e é independente de hardware (DEITEL, 2011, DEITEL, 2011).

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. 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++.

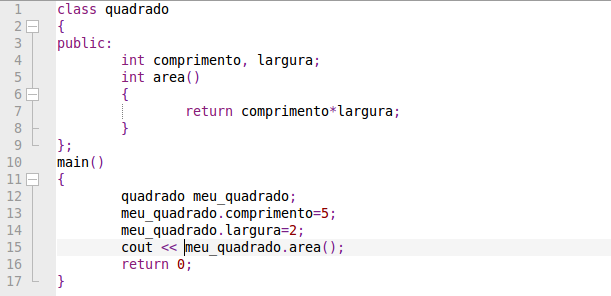
Figura 3 - Hello World em C++



Fonte: https://docs.microsoft.com/pt-br/cpp/build/vscpp-step-1-create?view=msvc-160, 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 - Exemplo de classe em C++



Fonte: https://elias.praciano.com/2013/09/um-introducao-a-programacao-orientada-a-objetos-em-c-parte-iii/, 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.

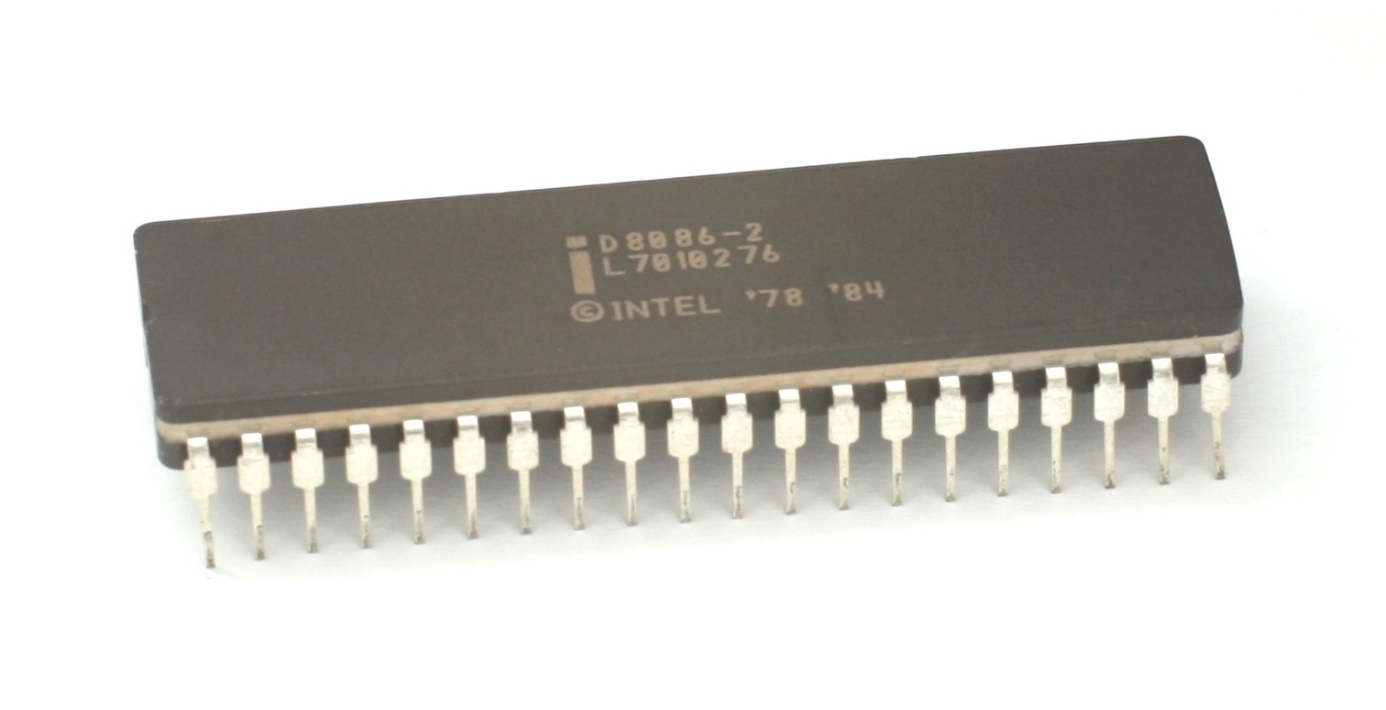
## Arquiteturas de Hardware suportadas

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. X86

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. 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).

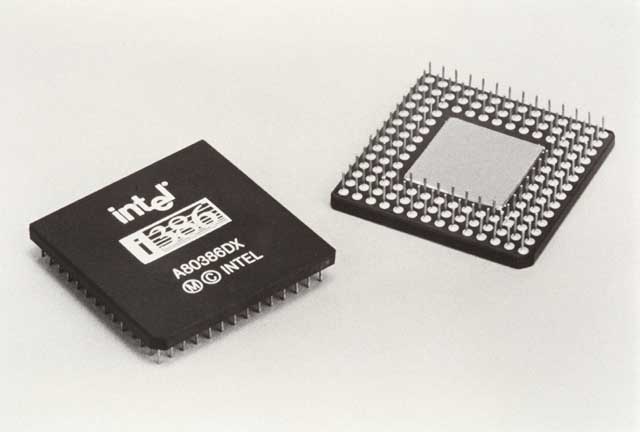
Figura 5 - Chip Intel 8086



Fonte: https://commons.wikimedia.org/wiki/File:KL\_Intel\_D8086.jpg, 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 - Chip Intel 80386



Fonte: https://www.cin.ufpe.br/~rcaf/ihs/familiax86.html, 2008

* + 1. ARMv8-A

TODO:

## 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

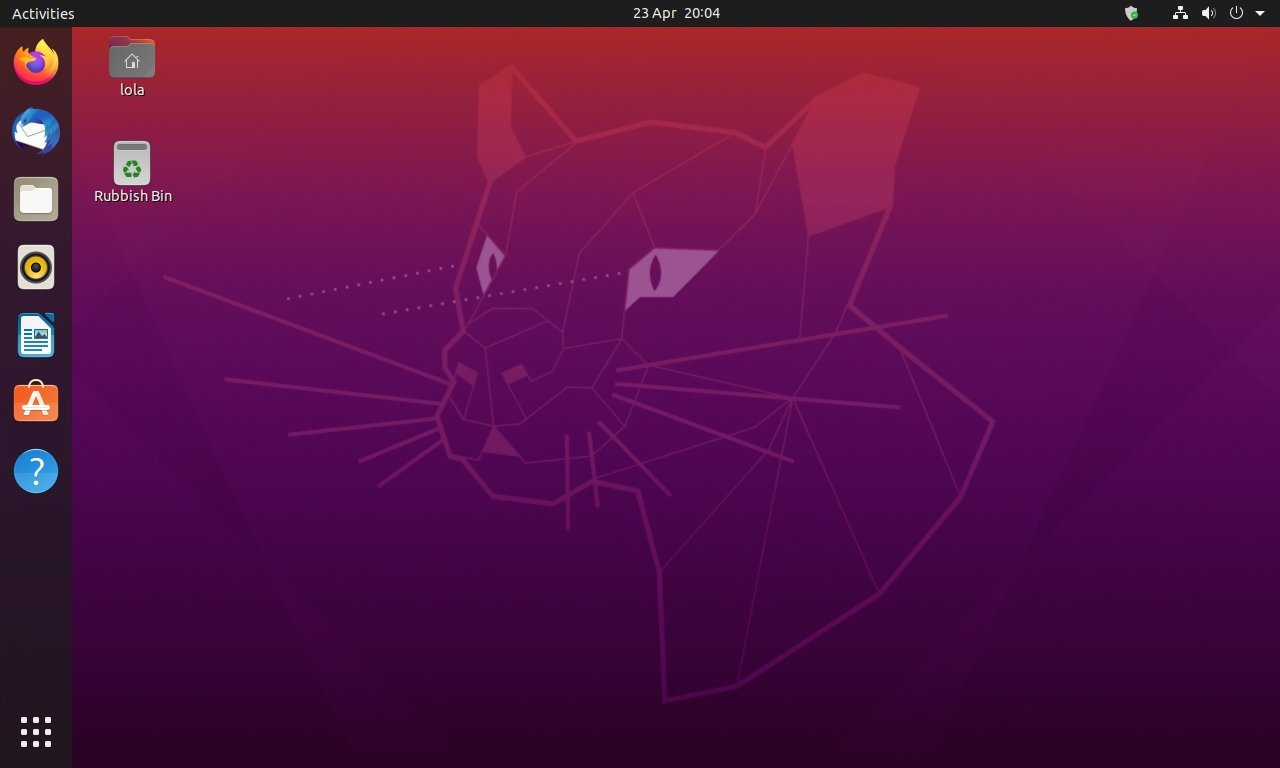
Nos últimos anos, o Linux passou de parque de diversões de estudantes aficionados para um novo competidor no mercado de servidores, atingindo o estágio de sistema respeitado com lugar certo nas redes educacionais e corporativas (SIEVER *et al*, 2006, p.17).

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).

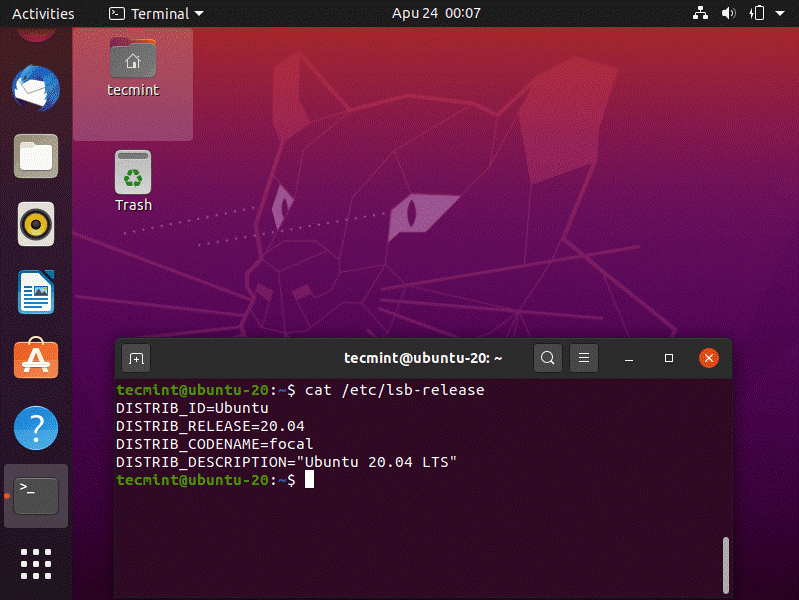
Embora os sistemas Linux tenham uma interface de usuário gráfica, a maioria dos programadores e usuários gráficos ainda prefere uma interface de linha de comando, chamada de interpretador de comandos (shell) (TANENBAUM, 2016, p. 520).

Figura – Interface gráfica do Ubuntu Linux



Fonte: https://tecnoblog.net/wp-content/uploads/2020/04/ubuntu-20-04.jpg, 2020

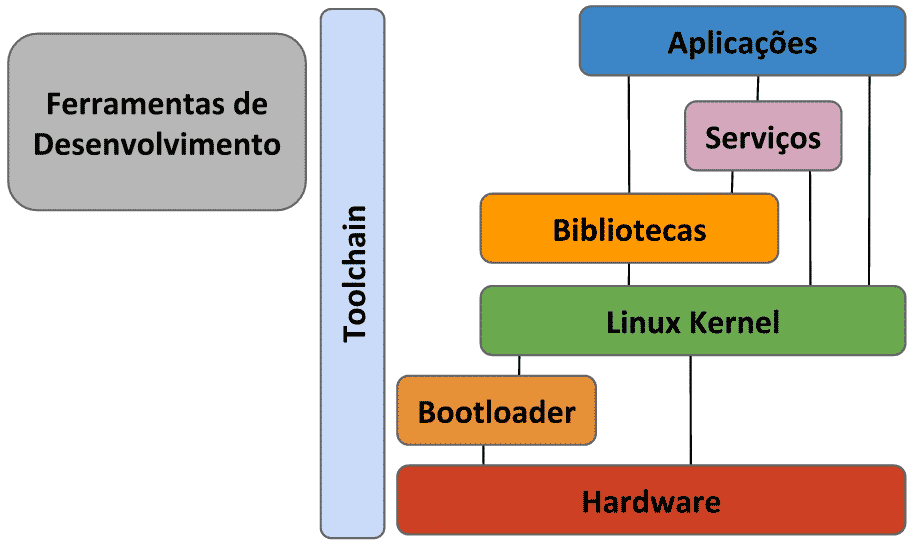
Figura – Shell do Ubuntu Linux



Fonte: https://www.tecmint.com/wp-content/uploads/2012/10/ubuntu-20.04-LTS-desktop.png, 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 - Anatomia de um sistema Linux embarcado



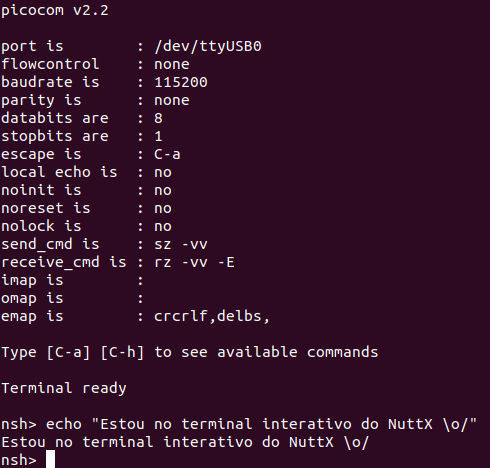
Fonte: https://www.embarcados.com.br/yocto-project-introducao/, 2016

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 preemptívo. 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 - Shell do Nuttx



Fonte: https://www.embarcados.com.br/primeiros-passos-com-o-esp32-e-o-nuttx-parte-1, 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 duas das mais conhecidas normas para aplicações safety critical do mundo, a AUTOSAR e a MISRA. Ambas trazem, além de normas para o processo de desenvolvimento, guidelines para escrever código seguro e certificável dentro dos seus parâmetros.

* + 1. AUTOSAR

TODO:

* + 1. MISRA

TODO:

# REFERÊNCIAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6023**: informação e documentação: referências, elaboração. Rio de Janeiro, 2002.

\_\_\_\_\_\_\_. **NBR 6027**: informação e documentação: sumário, apresentação. Rio de Janeiro, 2003.

\_\_\_\_\_\_\_. **NBR 6028**: informação e documentação: resumo, apresentação. Rio de Janeiro, 2003.

\_\_\_\_\_\_\_. **NBR 10520**: informação e documentação: citação em documentos, apresentação. Rio de Janeiro, 2002.

\_\_\_\_\_\_\_. **NBR 14724**: informação e documentação, trabalhos acadêmicos, apresentação. Rio de Janeiro, 2011.

FURASTÉ, Pedro Augusto. **Normas Técnicas para o Trabalho Científico:** Explicação das Normas da ABNT. 17 ed. Porto Alegre: Dáctilo Plus, 2015.

Oliveira, M.M. **Como fazer projetos, relatórios, monografias, dissertações e teses.** 5. ed. São Paulo: Elsevier. 2010.

**APÊNDICE**

**ANEXOS**