

UNIVERSIDADE FEDERAL DE SANTA CATARINA

Bruno Fontana Canella

**OPENAUTOS:
UM SISTEMA OPERACIONAL VEÍCULAR**

Araranguá

2016

Bruno Fontana Canella

**OPENAUTOS:
UM SISTEMA OPERACIONAL VEÍCULAR**

Trabalho de Conclusão de
Curso submetido à Universi-
dade Federal de Santa Cata-
rina, como parte dos requisitos
necessários para a obtenção do
Grau de Bacharel em Engenha-
ria de Computação.

Orientador: Prof. Anderson
Luiz Fernandes Perez, Dr.

Araranguá, dezembro de 2016.

RESUMO

Afim de padronizar o desenvolvimento de sistemas e dispositivos para veículos automotivos, foram criados padrões abertos, regidos pelas principais montadoras de veículos, as quais os batizaram como AUTOSAR. Dentre estes padrões, foi estabelecido um para governar o desenvolvimento de sistemas operacionais, capazes de gerenciar os recursos eletrônicos de um veículo. A maioria das soluções existentes hoje no mercado, e que respeitam este padrão, são proprietárias. Com o intuito de criar uma alternativa de qualidade comercial, este trabalho visa desenvolver, documentar e validar um sistema operacional automotivo de código aberto, nas normas estabelecidas pelo padrão AUTOSAR.

Palavras-chave: Automação Veicular, Sistema Operacional Embarcado, AUTOSAR, OpenAUTOS.

LISTA DE FIGURAS

Figura 1	Evolução dos componentes eletrônicos.....	17
Figura 2	Trem de força de um veículo automotivo.....	19
Figura 3	Injeção Eletrônica e ECU.....	19
Figura 4	Componentes do sistema de ABS.....	20
Figura 5	Sistemas pertencentes ao domínio do <i>corpo</i> . Em destaque o mecanismo de abertura e fechamento dos vidros.....	21
Figura 6	Comunicação de uma ECU principal com as demais....	22
Figura 7	Painel de um carro.....	23
Figura 8	Visão dos sistemas de um carro automático.....	24
Figura 9	Relação entre tamanho do chicote e taxa de transferência.	26
Figura 10	Valores de transmissão do protocolo CAN.....	27
Figura 11	Rede CAN/LIN.....	27
Figura 12	Placa lógico e física de uma ECU.....	28
Figura 13	Modelo V.....	30
Figura 14	Processo de geração do código em C a partir de um arquivo OIL.....	31
Figura 15	Visão geral técnica do AUTOSAR.....	33
Figura 16	Visão geral da integração do SO com o sistema computacional.....	35
Figura 17	Serviços de um SO.....	36
Figura 18	Posição conceitual do <i>kernel</i>	37
Figura 19	Posição do <i>kernel</i> na memória.....	37
Figura 20	Estados de uma tarefa.....	39
Figura 21	Diagrama esquemático de funcionamento do algoritmo de escalonamento Round Robin.....	40
Figura 22	Diagrama esquemático de funcionamento do algoritmo de escalonamento baseado em prioridade.....	41
Figura 23	Diagrama esquemático de funcionamento do algoritmo de escalonamento baseado em prioridade com Round Robin.....	41
Figura 24	Transição de Estados para Semáforos de Contagem.....	42
Figura 25	Diagrama esquemático do funcionamento de uma fila de mensagens.....	43
Figura 26	Tratamento de sinal em uma tarefa.....	44

Figura 27 Subsistema de E/S e o modelo por camadas.....	46
Figura 28 Camada de abstração entre o aplicativo e o dispositivo.	47
Figura 29 Modelo em árvore de diretórios por usuário.	48
Figura 30 Sistemas embarcados em um veículo.	49
Figura 31 Resposta em Tempo-Real.	50
Figura 32 Processo de geração das configurações do kernel.....	52
Figura 33 Geração do cabeçalho de configuração.....	54
Figura 34 Módulos comuns a um SOE.....	54
Figura 35 FRDM-KL25Z.....	55
Figura 36 Comunicação entre ECUs e Sistemas.....	55

LISTA DE TABELAS

Tabela 1	Grupos de Protocolos.....	25
Tabela 2	Aplicações do barramento CAN.....	25
Tabela 3	Cronograma.....	56

LISTA DE ABREVIATURAS E SIGLAS

SO	Sistema Operacional
RAM	<i>Random Access Memory</i>
ROM	<i>Read-Only Memory</i>
ABS	<i>Antiblockier-Bremssystem</i>
IHM	Interface Homem-Máquina
ECU	<i>Eletronic Control Unit</i>
ESP	<i>Electronic Stability Program</i>
ASP	<i>Automatic Stability Control</i>
4WD	<i>Four-Wheel Drive</i>
LIN	<i>Local Interconnect Network</i>
CAN	<i>Controller Area Network</i>
ISO	Organização Internacional de Normalização
UART	<i>Universal Asynchronous Receiver/Transmitter</i>
LSD	<i>Low Side Driver</i>
HSD	<i>High Side Driver</i>
OIL	<i>OSEK Implementation Language</i>
RTOS	<i>Real Time Operating Systems</i>
E/S	Entrada e Saída
SOE	Sistemas Operacionais Embarcados
ISR	<i>Interrupt Service Routine</i>
ASR	<i>Asynchronous Signal Routine</i>
API	<i>Application Programming Interface</i>
XML	<i>eXtensible Markup Language</i>

SUMÁRIO

1	INTRODUÇÃO	13
1.1	JUSTIFICATIVA E MOTIVAÇÃO	13
1.2	OBJETIVOS	14
1.2.1	Geral	14
1.2.2	Específicos	14
1.3	ORGANIZAÇÃO DO TRABALHO	14
2	AUTOMAÇÃO VEICULAR	17
2.1	TRANSIÇÃO DA MECÂNICA PARA ELETRÔNICA	17
2.2	DOMÍNIOS FUNCIONAIS DE UMA ARQUITETURA VEICULAR	18
2.2.1	Trem de Força	18
2.2.2	Chassi	19
2.2.3	Corpo	20
2.2.4	IHM e Telemáticos	22
2.3	PROTOCOLOS DE COMUNICAÇÃO AUTOMOTIVOS	23
2.3.1	Protocolos Automotivos	24
2.3.1.1	Controller Area Network - CAN	24
2.3.1.2	Local Interconnect Network - LIN	27
2.4	UNIDADE DE CONTROLE ELETRÔNICA	28
2.5	PADRÕES DE DESENVOLVIMENTO DE SOFTWARE AUTOMOTIVO	29
2.5.1	Modelo V	29
2.5.2	OSEK/VDX	30
2.5.3	AUTOSAR	32
3	SISTEMAS OPERACIONAIS EMBARCADOS	35
3.1	SISTEMAS OPERACIONAIS	35
3.1.1	Kernel	36
3.2	ESTRUTURA DOS SISTEMAS OPERACIONAIS	37
3.2.1	Gerenciamento de Processos	37
3.2.1.1	Tarefas	38
3.2.1.1.1	<i>Fibras</i>	38
3.2.1.2	Escalonadores de Tarefas	39
3.2.1.2.1	<i>Escalonador Round Robin</i>	39
3.2.1.2.2	<i>Escalonador por Prioridade</i>	40
3.2.1.2.3	<i>Escalonador por Prioridade com Round Robin</i>	40
3.2.1.3	Semáforos	41
3.2.1.3.1	<i>Mutex</i>	42

3.2.1.4	Filas de Mensagens	43
3.2.1.5	Sinais	43
3.2.2	Gerenciamento de Memória	44
3.2.2.1	Alocação Dinâmica de Memória	44
3.2.3	Subsistemas de Entrada/Saída	45
3.2.3.1	Camada de Abstração	46
3.2.4	Sistemas de Arquivos	47
3.3	SISTEMAS EMBARCADOS	48
3.4	SISTEMAS DE TEMPO-REAL	49
3.5	ESTUDOS DE CASO	50
3.5.1	FreeRTOS	50
3.5.2	PICOS18	51
3.5.3	Trampoline	51
4	OPENAUTOS	53
4.1	PROPOSTA	53
4.2	ARQUITETURA	54
4.3	CRONOGRAMA	56

1 INTRODUÇÃO

Desde seu surgimento, popularização e evolução até os dias atuais, os veículos automotivos aumentaram em muito a sua complexidade, a ponto de que apenas o conhecimento mecânico do veículo não é mais suficiente. A quantidade de componentes eletrônicos presentes nos veículos automotivos aumentou consideravelmente passando, inclusive, a substituir sistemas puramente mecânicos. Dentre os fatores que alavancaram estas mudanças destacam-se o barateamento, miniaturização e popularização dos componentes eletrônicos, bem como a adequação da indústria automobilística às novas leis de trânsito, que passaram a dictar a emissão máxima de poluentes e a exigir mecanismos de segurança, como o ABS e o *Airbag*.

Com o propósito de padronizar e, assim, facilitar o desenvolvimento e intercambiabilidade de auto-peças por terceiros, as principais montadoras e fabricantes de veículos entraram em um consenso, estipulando um padrão de normas de desenvolvimento para veículos automotivos, chamada de AUTomotive Open System ARchitecture ou AUTOSAR, a qual se encontra em sua quarta revisão.

Para gerenciar os diversos módulos eletrônicos agora presentes em um veículo, bem como garantir a interoperabilidade entre eles, foram criadas normas referente ao desenvolvimento de sistemas operacionais. Estas normas visam o estabelecimento de padrões para o funcionamento, comunicação e especificações do sistema, sem sacrificar a liberdade criativa de desenvolvimento do sistema, como a seleção de hardwares e implementação de algoritmos.

1.1 JUSTIFICATIVA E MOTIVAÇÃO

A maioria das soluções em sistemas operacionais automotivos são exclusivamente comerciais e de código fechado. Embora existam soluções de código aberto para sistemas embarcados, não existe, na atualidade, um Sistema Operacional - SO de código aberto homologado nos padrões do AUTOSAR. O projeto que mais chega próximo deste cenário é o Trampoline, que se encontra em fase de homologação da norma (??).

Visando a criação de um SO embarcado, para uso em veículos populares, e que mantivesse um padrão de código aberto, surgiu a idealização do OpenAUTOS. Através do desenvolvimento do OpenAU-

TOS, deseja-se alcançar um SO nacional que seja referência na área, utilizando componentes e tecnologias com alta disponibilidade e de fácil acesso, além de agregar contribuições com a própria comunidade acadêmica.

1.2 OBJETIVOS

Esta seção apresenta o objetivo geral e os objetivos específicos deste trabalho.

1.2.1 Geral

Desenvolver um sistema operacional embarcado de código aberto que atenda as normas estabelecidas pelo padrão AUTOSAR.

1.2.2 Específicos

1. Levantar o estado da arte com respeito a algoritmos para sistemas operacionais embarcados;
2. Estudar padrões de sistemas automotivos;
3. Levantar os requisitos para implementação de um SO de acordo com a norma AUTOSAR;
4. Estabelecer um projeto de código aberto em um repositório online;
5. Documentar o código do projeto;
6. Criar um modelo físico que utilize o SO desenvolvido;
7. Validar o OpenAUTOS em um veículo automotivo real.

1.3 ORGANIZAÇÃO DO TRABALHO

Este trabalho está dividido em 4 capítulos, contando com a introdução.

O **Capítulo 2** apresentará os domínios eletrônicos de funcionamento em veículos, seus meios de comunicação, unidades de controle e engenharia de software automotivo.

O **Capítulo 3** abordará os principais conceitos sobre sistemas operacionais. Ao final do capítulo serão relatados alguns estudos de caso a respeito de sistemas operacionais embarcados com foco para a automação veicular.

O **Capítulo 4** apresentará a proposta do SO OpenAUTOS, destacando as escolhas tanto do projeto do software bem como a arquitetura de hardware adotada.

2 AUTOMAÇÃO VEICULAR

A automação veicular faz um extenso uso de componentes eletrônicos em seus sistemas. Nos tópicos que seguem serão abordados os domínios eletrônicos de funcionamento em veículos, seus meios de comunicação, unidades de controle e engenharia de software automotivo.

2.1 TRANSIÇÃO DA MECÂNICA PARA ELETRÔNICA

De acordo com ??), os motivos para evolução automotiva variam entre ganhos de desempenho e segurança, barateamento de sistemas sem perda de qualidade e a adequação as leis de trânsito e meio-ambiente.

Contudo, independente da motivação, é notável que há cada vez mais a adoção de sistemas eletrônicos embarcados. A Figura 1 ilustra veículos de eras diferentes, a primeira dotada de recursos mecânicos, enquanto que a segunda destaca as centrais eletrônicas, as quais vem aumentando em número a cada ano que passa.

Figura 1: Evolução dos componentes eletrônicos.



(a) 1950

(b) 2000

Extraido de: ??) e ??).

Nos dias de hoje, recursos como injeção eletrônica e freios *Antiblockier-Bremssystem - ABS*¹ são exigências para que um novo modelo de carro possa entrar em circulação.

O crescente uso de recursos eletrônicos demanda também que os mesmos possam se comunicar e cooperar entre si, para que o máximo de desempenho possa ser extraído dos sistemas. Neste caso, os domínios

¹Traduzido como: Sistema de Anti-Bloqueio.

funcionais auxiliam na classificação e responsabilidade destes sistemas.

2.2 DOMÍNIOS FUNCIONAIS DE UMA ARQUITETURA VEICULAR

De modo a facilitar a classificação e identificação de sistemas eletrônicos que compõem um veículo, historicamente, foram estabelecidos cinco domínios de funcionalidade na arquitetura veicular, sendo eles o *Trem de Força*, *Chassi*, *Corpo*, Interface Homem-Máquina - IHM e *Telemática* (??).

Com o avanço das últimas décadas, recursos veiculares pertencentes a estes domínios que até então eram implementados via sistemas mecânicos e elétricos puderam ser substituídos por elementos da eletrônica, agregando confiabilidade, segurança e, em alguns casos, possibilitando o atendimento de requisitos que antes eram inviáveis de implementação.

Nas demais subseções, serão apresentados em mais detalhes cada um dos domínios funcionais e serão levantados exemplos de como eles se beneficiaram nos últimos anos com o uso de microeletrônicos e microprocessadores.

2.2.1 Trem de Força

O domínio do trem de força está relacionado aos sistemas que participam na propulsão longitudinal de um veículo. Conforme pode ser observado pela Figura 2, estes sistemas são compreendidos pelo *motor*, *transmissão* e demais componentes subsidiários, como o *trem de rodagem*².

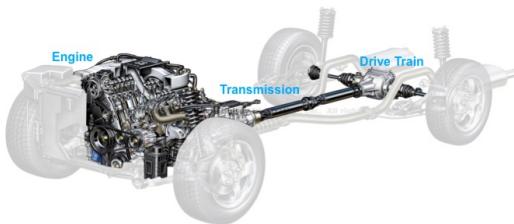
Os sistemas presentes neste domínio buscam otimizar componentes para condução, conforto, economia de combustível, dentre outros. Há uma grande quantidade de sistemas embarcados vinculados principalmente ao motor, onde desde 1939 a empresa Bosch já fazia pesquisas sobre o gerenciamento eletrônico de motores (??).

Um sistema bastante conhecido no ramo automobilístico, e que veio a substituir completamente o uso de carburadores, é a injeção eletrônica, implementado comercialmente pela primeira vez pela Bendix Corporation, em 1958³(??). Este sistema é responsável pela injeção

²Drive Train, no inglês.

³Na época batizado de *Electrojector* pela empresa.

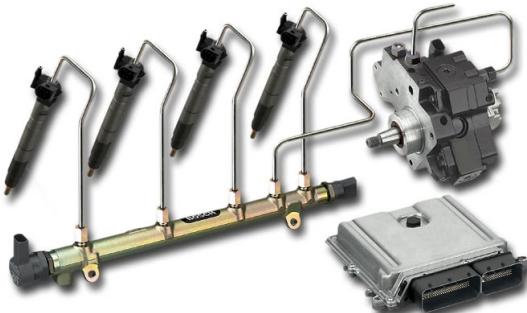
Figura 2: Trem de força de um veículo automotivo.



Extraido de: ??).

de combustível dentro do motor e controla as quantidades e proporção entre ar e combustível, bem como a produção da centelha pelas velas de ignição. O controle de tempos e proporções parte de uma *Electronic Control Unit* - ECU⁴, componente o qual será discutido em maior detalhes na Seção 2.4. A Figura 3 apresenta um modelo com os bicos injetores e sua respectiva ECU.

Figura 3: Injeção Eletrônica e ECU.



Extraido de: ??).

2.2.2 Chassi

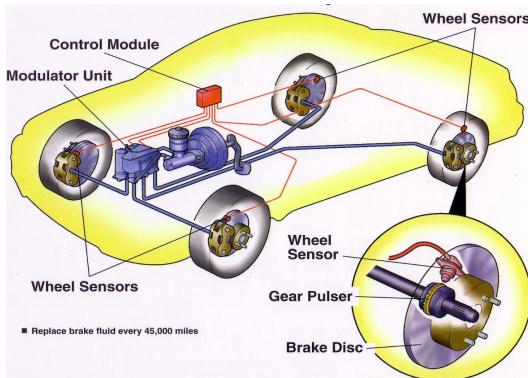
Os elementos que compõem o *Chassi* tem por objetivo controlar a interação do veículo com o ambiente ao qual ele irá circular. Isto é alcançado através de leituras e atuações nas rodas, suspensões, sistemas

⁴Traduzido como: Unidade de Controle Eletrônica.

de freio, direção, aceleração, dentre outros. Em carros mais modernos, também são levados em conta as condições da estrada, velocidade do vento, situação do clima e outros fatores externos ao veículo.

Dos sistemas que compõem este domínio, podem ser citados os de ABS, *Electronic Stability Program - ESP*⁵, *Automatic Stability Control - ASP*⁶ e o *Four-Wheel Drive - 4WD*⁷. A Figura 4 apresenta uma ilustração dos componentes do ABS.

Figura 4: Componentes do sistema de ABS.



Extraido de: ??).

Como o domínio do *chassi* tem por objetivo a segurança dos passageiros e do próprio veículo, seus sistemas utilizam recursos de alta qualidade e o mesmo é tratado como um setor crítico. Características desejáveis são similares a do *trem de força*, fazendo uso de uma técnica adicional de segurança chamada de *X-by-Wire*⁸. Este método consiste em manter um sistema mecânico (ergo, o "por fio" do nome) redundante ao eletrônico. Historicamente, o sistema mecânico é o sistema original, sendo mantido apenas por questões de segurança.

2.2.3 Corpo

O domínio do *Corpo* do veículo inclui os demais sistemas que não estão relacionados ao controle de suas dinâmicas. Mecanismos como

⁵Traduzido como: Programa Eletrônico de Estabilidade.

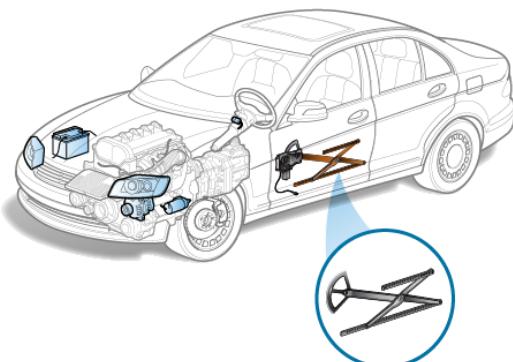
⁶Traduzido como: Controle de Estabilidade Automática.

⁷Traduzido como: Tração nas Quatro Rodas.

⁸Traduzido do inglês como *X-por-fio*

limpadores de vidro, faróis, portas e janelas, assentos e retrovisores hoje em dia são controlados diretamente por sistemas baseados em software. A Figura 5 provê uma ilustração de alguns destes sistemas e sua localização no veículo.

Figura 5: Sistemas pertencentes ao domínio do *corpo*. Em destaque o mecanismo de abertura e fechamento dos vidros.



Extraido de: ??).

De uma maneira geral, estes sistemas, embora necessários para o devido funcionamento do veículo e conforto dos passageiros, não representam um fator crítico de proteção para o usuário, salvo alguns sistemas de segurança quanto ao controle de acesso do veículo.

Este domínio geralmente envolve a comunicação de diversas funções entre si, o que, consequentemente, gera uma arquitetura complexa distribuída. Desta necessidade emerge o conceito de subsistemas baseados em redes de sensores-atuadores de baixo custo.

Um sistema de controle para portas, por exemplo, poderia utilizar uma ECU principal para realizar as operações da porta do motorista (trancar/destrancar a porta, fechar/abrir janela, ajustar o banco). Esta ECU se comunicaria com as ECUs da trava, janela e banco por uma rede *Local Interconnect Network - LIN*⁹¹⁰ de baixo custo. Como uma operação de trancar/destrancar uma porta pode influenciar nas outras, todas as ECUs das portas estariam conectadas por uma rede *Controller Area Network - CAN*¹¹¹², assim como com o painel, afim

⁹Traduzido como: Rede Local Interconectada.

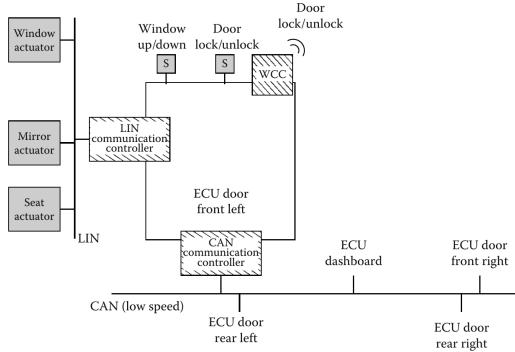
¹⁰vide tópico 2.3.1.2.

¹¹Traduzido como: Área de Rede Controladora.

¹²vide tópico 2.3.1.1.

de indicar o estado das portas. A Figura 6 apresenta uma ilustração destas interações.

Figura 6: Comunicação de uma ECU principal com as demais.



Extraido de: ??).

2.2.4 IHM e Telemáticos

O domínio da *interface homem-máquina* governa a interação do condutor e passageiros com as várias funções embarcadas do veículo. Resultados que podem ser apresentados pela IHM são:

- Estados do Veículo: velocidade, nível do óleo, situação das portas, estado das luzes;
- Estado de um dispositivo multimídia: rádio sintonizada, controle de volume;
- Reposta a uma requisição: visualização de um mapa no sistema de navegação.

Sistemas embarcados deste domínio geralmente estão instalados no painel de um veículo, conforme ilustração da Figura 7.

O domínio da *telemática*, em contraste à IHM, governa a troca de informações entre veículos ou entre o veículo e infraestruturas da estrada.

Exemplos de sistemas deste domínio são os coletores automáticos de tarifas para pedágios, no qual o carro pode passar por uma via

Figura 7: Painel de um carro.



Extraido de: ??).

sem precisar parar para realizar o pagamento em dinheiro. Outras aplicações incluem a comunicação com serviços de trânsito, os quais podem indicar a situação em uma rodovia, como acidentes ou trânsito intenso.

Uma das transições que a área está passando atualmente é no desenvolvimento de carros inteligentes, capazes de detectar riscos ao motorista e, até mesmo, dirigir automaticamente, sem que haja a necessidade de intervenção do condutor humano. Empresas como a Google e a Tesla estão com soluções comerciais em fase de avaliação no mercado.

A Figura 8 ilustra a visão do veículo para com seus meios externos, identificando veículos e limites de velocidades.

Ambos os domínios estão se aproximando cada vez mais do conceito de Internet das Coisas, onde os veículos serão um dos grandes beneficiados desta integração, devido aos novos tipos de serviços e informações que serão trocadas por eles (??).

2.3 PROTOCOLOS DE COMUNICAÇÃO AUTOMOTIVOS

Devido a grande quantidade de componentes que necessitam trocar informações em um veículo, é imperativo que existam meios de comunicação padronizados e que atendam aos requisitos dos sistemas, como taxa de transmissão e segurança. Os protocolos de comunicação permitem que estas integrações sejam realizáveis com relativa facilidade.

Figura 8: Visão dos sistemas de um carro automático.



Extraido de: ??).

Embora mais complexas, é comum o uso de redes distribuídas em veículos automotivos, onde mais de um protocolo de comunicação são empregados.

2.3.1 Protocolos Automotivos

Existem diversos protocolos automotivos, os quais estão listados em ??) e devidamente identificados quanto a fabricante, aplicação, tipo de barramento, dentre outros quesitos.

A classificação destes é feita por meio de grupos, os quais definem a taxa de transmissão e/ou aplicação dos mesmos. A Tabela 1 lista estes grupos de uma maneira resumida.

Dos protocolos citados na Tabela 1, existem dois que merecem destaque, devido a versatilidade e baixo custo de implementação, que são os barramentos CAN e LIN.

2.3.1.1 Controller Area Network - CAN

Foi desenvolvido pela empresa alemã Robert Bosch, entre 1983 e 1986, para uso em veículos de transporte. Na atualidade, o protocolo CAN é amplamente utilizado na comunicação veicular, além de estar presente em navios, tratores e outros ??).

Das aplicações que o CAN pode assumir em um veículo, é possível destacar as que estão presentes na Tabela 2.

Tabela 1: Grupos de Protocolos.

Grupo	Características & Uso	Protocolos
Classe A	Conforto 10Kbps	SINEBUS, I^2C , SAE, LIN
Classe B	Entretenimento 10Kbps a 125Kbps	CAN 2.0, Class 2, SCP, PCI
Classe C	Segurança 125Kbps a 1Mbks	CAN 2.0, ISO, SAE J139
Diagnóstico	Diagnóstico Embocado	J1850, Class 2, SCP, PCI
Mobile Media	PC-On-Wheels	IDB-C, MOST, MML, USB
Safety Bus	Airbag	BST, DSI, Byte Flight
Drive by-wire	Segurança	TTP, FlexRay, TTCAN

Tabela 2: Aplicações do barramento CAN.

Aplicação	Transmissão	Exemplos
Tempo Real	$>125\text{kbps}$ $<1\text{Mbps}$	Motronic, Câmbio, ESP
Multiplex	$>10\text{kbps}$ $<125\text{kbps}$	Ar Condicionado, Travas
Comunicação Móvel	$<125\text{kbps}$	Celular, Áudio, Navegação
Diagnóstico	500kbps	Diagnóstico das ECUs

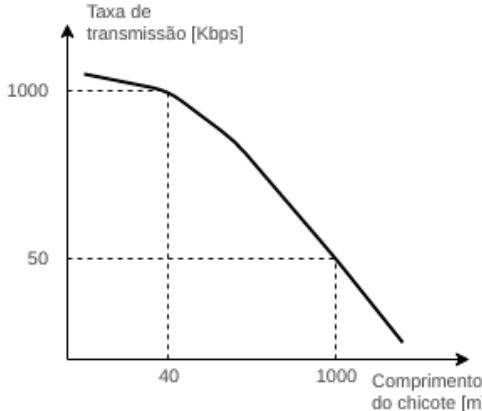
O protocolo define especificações tanto físicas quanto lógicas, e apresentam variações, o que o torna bastante flexível.

Conforme explica ??) a arquitetura de sua rede é a multimestre, na qual qualquer módulo pode assumir o papel de mestre enquanto que os demais se tornam escravos, a qualquer momento. As mensagens são transmitidas para todos os módulos da rede, via *broadcast*.

Cada rede CAN é formada por um único par trançado de fios, chamado de chicote, no qual a taxa de transmissão máxima varia conforme seu comprimento, conforme ilustrado na Figura 9. Vale ressaltar que um mesmo veículo pode possuir diversas redes CAN dentro dele.

Outro fator que determina seu desempenho é o tamanho das mensagens. Alguns bits são utilizados para identificar qual tipo de mensagem que está sendo enviada. A versão original do protocolo (CAN 2.0A) utiliza 11 bits para esta identificação, resultando em 2048 tipos de mensagem. Devido a necessidade de adicionar ainda mais tipos

Figura 9: Relação entre tamanho do chicote e taxa de transferência.



Adaptado de: ??).

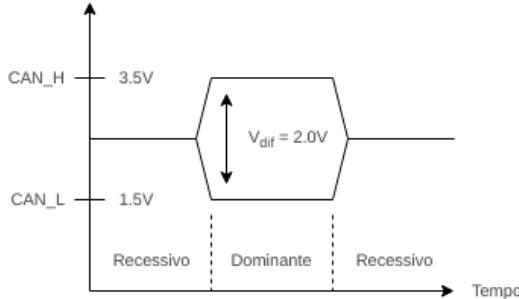
de mensagens a uma rede CAN, uma segunda versão foi criada (CAN 2.0B), com 29 bits utilizados para identificação. Este *overhead* afeta negativamente a taxa de transmissão da rede, mas virtualmente elimina o limite de mensagens, que passa a ser mais de 537 milhões.

Existem duas normas da Organização Internacional de Normalização - ISO que especificam os fundamentos do protocolo CAN. A ISO 11898 determina as características de uma rede CAN de alta velocidade, entre 125Kbps e 1Mbps, enquanto que a ISO 11519-2 possuí especificações para uma rede de baixa velocidades, entre 10Kbps e 125Kbps.

Outro ponto que merece destaque é quanto ao formato de envio dos dados na rede. Diferente do formato eletrônico tradicional, onde são utilizados níveis fixos de tensão para representar os valores 0 e 1, no protocolo CAN são utilizados *bits recessivos* e *dominantes*. Dois fios trançados, chamados de CAN_H e CAN_L, transmitem um diferencial de tensão, e é a partir desta diferença que se estipula os valores de 0 e 1. Caso o par trançado sofra ruídos externos em seu sinal, isto não afetará o valor final, pois ambos serão igualmente distorcidos, mantendo o diferencial original. A Figura 10 ilustra um exemplo desta transmissão.

O protocolo ainda conta com diversos mecanismos de tratamento de colisões, como a arbitragem *bit-a-bit* e detecção de falhas, como *bit monitoring*, *bit stuffing*, *cyclic redundancy check*, *frame check* e *acknowledgment error check*.

Figura 10: Valores de transmissão do protocolo CAN.



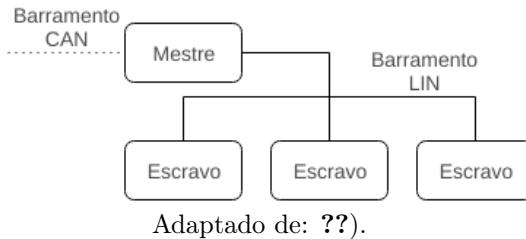
Adaptado de: ??).

2.3.1.2 Local Interconnect Network - LIN

Criado por um consórcio de empresas europeias do segmento automotivo, o protocolo LIN foi desenvolvido para servir como um sub-barramento de barramentos maiores. Conforme descrito em ??), ele foi idealizado para uso em aplicações de troca simples como assentos, travas nas portas, teto solar, sensores de chuva, dentre outras funções, em sua maioria, pertencentes ao domínio do corpo.

O sub-barramento LIN é baseado em protocolos de comunicação serial. Ele faz uso da arquitetura mestre/escravos, utilizando um único fio para transmissão de dados, geralmente em conjunto com uma rede CAN, conforme ilustrado na Figura 11.

Figura 11: Rede CAN/LIN.



Adaptado de: ??).

Para redução de custos, os componentes podem ser guiados sem o uso de um cristal ou ressonador cerâmico, por utilizar sincronização temporal. É um sistema baseado em *Universal Asynchronous Recei-*

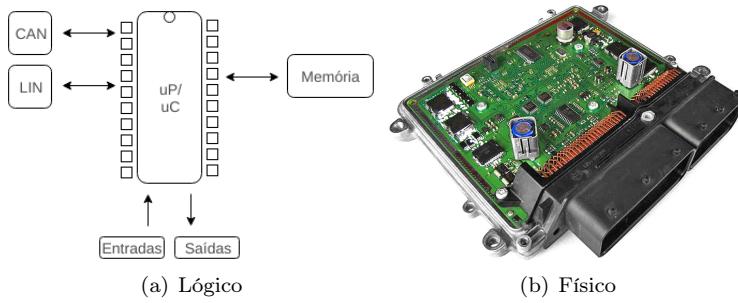
ver/Transmitter - UART¹³, disponível na maioria dos microcontroladores. O barramento também é capaz de detectar nós defeituosos na rede através do uso de técnicas para verificação de erros e *checksum* e paridade, detalhadas em ??).

2.4 UNIDADE DE CONTROLE ELETRÔNICA

Conforme ??), as unidades de controle eletrônicas são os dispositivos responsáveis por fazer a leitura de sensores, o acionamento de saídas, a comunicação entre módulos e pelo gerenciamento do funcionamento de sistemas. Uma ECU pode ser responsável por um ou mais sistemas em um veículo, geralmente pertencentes a um mesmo domínio.

Fisicamente, a ECU pode ser comparado a um computador. No geral, elas são compostos por um microprocessador ou microcontrolador, memórias e portas de entrada e saída, todos soldados em uma placa de circuito integrado. Também possuem *transceivers* para redes CAN e LIN, para que as ECUs possam trocar informações entre si e entre sensores e atuadores. Uma representação lógica dos seus componentes pode ser vista na Figura 13(a).

Figura 12: Placa lógico e física de uma ECU.



Adaptado de: ??).

As quantidade de entradas e saídas disponíveis está associada ao microprocessador/microcontrolador utilizado, e estas podem ser digitais ou analógicas, sendo que as saídas também podem ser do tipo *Low Side Driver* - LSD¹⁴ ou *High Side Driver* - HSD¹⁵.

¹³Traduzido como: Receptor/Transmissor Universal Assíncrono.

¹⁴Traduzido como: Driver do Lado Inferior.

¹⁵Traduzido como: Driver do Lado Superior.

Em um veículo, existem diversas ECUs, ligadas por uma ou mais redes CAN *bus*. Dependendo da quantidade e distribuição no veículo, podem ser que diversas ECUs pertençam a um mesmo domínio e que uma delas sirva como uma central para as demais, ou ainda que uma ECU seja responsável por repassar informações de uma rede CAN para outra.

Como cada ECU deverá executar um algoritmo específico, o qual varia conforme a função para a qual a mesma foi designada, surge a necessidade de que haja um gerenciamento dos vários módulos espalhados pelo veículo. Esta responsabilidade cai em uma das ECUs, a qual faz uso de um sistema operacional padronizado, que é capaz de se comunicar com as demais unidades de maneira intermitente e que dê prioridade para recursos de maior importância.

2.5 PADRÕES DE DESENVOLVIMENTO DE SOFTWARE AUTOMOTIVO

Com o crescente uso de componentes eletrônicos embarcados nos veículos automotivos, também surgiu a necessidade de padronizar as etapas do desenvolvimento de novos sistemas, afim de garantir a interoperabilidade e a intercambialidade entre eles.

Serão apresentados aqui um padrão relacionado a engenharia de software e outro dedicado a implementação de sistemas veiculares.

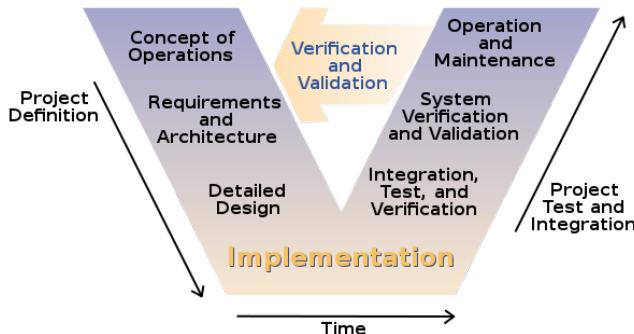
2.5.1 Modelo V

Segundo ??), o Modelo V, ou modelo de Verificação e Validação, pode ser considerado uma extensão do modelo Cascata, onde cada etapa deve ser concluída antes de avançar para a próxima. Ao invés de se deslocar para baixo de maneira linear, os passos do processo são curvados para cima após concluída a fase de codificação, tomando formato de um V, conforme ilustra a Figura 13.

Como vantagens, trata-se de um modelo de fácil utilização, que incentiva a criação de cenários de testes antes mesmo da produção do código. Permite também que defeitos sejam encontrados em fases iniciais e apresenta bom resultados em projetos de pequeno porte.

Entre suas desvantagens, pode-se destacar que trata-se de um modelo pouco flexível. A produção de código só tem seu início na fase de implementação, tornando impraticável a criação de um protótipo.

Figura 13: Modelo V.



Extraido de: ??).

2.5.2 OSEK/VDX

Conforme ??), o padrão OSEK/VDX foi idealizado como uma arquitetura aberta para ECUs veiculares. Ele surgiu a partir da união do padrões OSEK¹⁶, criado por um consórcio de fabricantes de veículos alemãs, com o padrão VDX¹⁷, criado pelas empresas francesas Renault e Peugeot.

A principal motivação para a criação deste padrão se decorreu por conta do número cada vez maior de sistemas eletrônicos nos veículos. Com a introdução do padrão, problemas recorrentes puderam ser corrigidos, como a incompatibilidade entre ECUs de diferentes fabricantes, bem como a contenção de custos com o desenvolvimento de protocolos para as ECUs.

O padrão utiliza uma abordagem estática para as configurações do sistema, forçando com que todas as configurações sejam definidas antes da inicialização do sistema. Isto impede que novas tarefas sejam criadas ou que a memória seja alocada dinamicamente durante a execução do programa.

Para auxiliar na definição destas configurações, o padrão OSEK/VDX utiliza um arquivo de configuração com uma linguagem própria, chamada de *OSEK Implementation Language - OIL*¹⁸. Uma vez definido, o arquivo é então processado para gerar o código em C de configuração

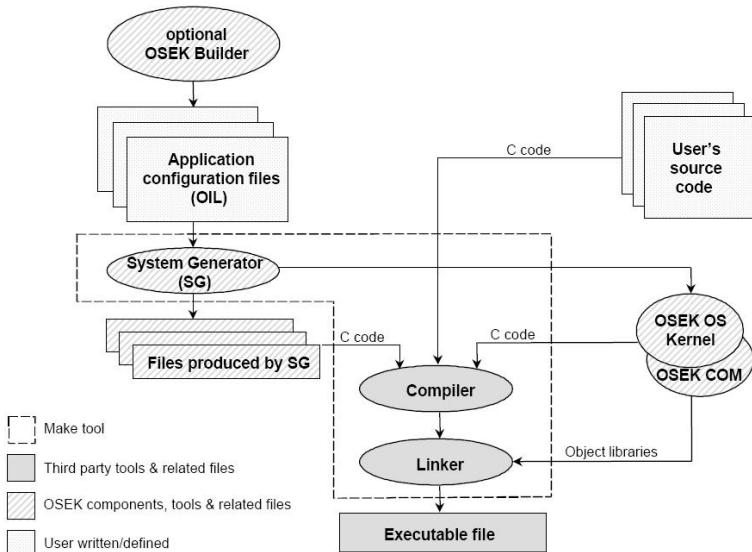
¹⁶Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen

¹⁷Vehicle Distributed eXecutive

¹⁸Traduzido como: Linguagem de Implementação do OSEK.

do *Real Time Operating Systems* - RTOS¹⁹. A Figura 14 provê uma ilustração do processo de geração envolvendo o arquivo de configuração OIL.

Figura 14: Processo de geração do código em C a partir de um arquivo OIL.



Extraido de: ??).

O padrão conta com sete especificações, cada qual atendendo a uma área diferente do sistema, sendo elas:

- **OSEK OS:** Sistema operacional;
- **OSEK Time:** Sistema operacional ativado por tempo;
- **OSEK COM:** Serviços de comunicação;
- **OSEK FTCOM:** Comunicação com tolerância a falhas;
- **OSEK NM:** Gerenciamento de rede;
- **OSEK OIL:** Configuração do kernel;

¹⁹Traduzido como: Sistema Operacional de Tempo-Real.

- **OSEK ORTI:** Configuração do kernel com suporte a debuggers;

Por ser um padrão mais antigo e restrito, atualmente existem soluções comerciais de código aberto, dentre elas: FreeOSEK²⁰, ERIKA Enterprise²¹ e Trampoline²².

2.5.3 AUTOSAR

Confome ??), o projeto AUTOSAR surgiu da cooperação entre manufaturadores e fornecedores de veículos e industrias de componentes eletrônicos e de software. Desde sua concepção, em 2003, foi idealizado como um projeto aberto, focado na padronização de arquiteturas de software para a industria automotiva. O AUTOSAR também incorpora a maior parte do OSEK/VDX, expandindo suas funcionalidades mas mantendo a compatibilidade.

A motivação por trás do desenvolvimento do padrão AUTOSAR visava atender as otimizações a nível de sistema, e não apenas em componentes individuais, obtendo assim a máxima reutilização de código possível. Para tal, uma arquitetura aberta, escalar e de módulos de software intercambiáveis era necessária, exigindo um esforço coletivo do consórcio de empresas envolvidas.

O principal objetivo do AUTOSAR é a padronização de funções básicas de sistemas e interfaces funcionais, garantindo a integração, intercambialidade e transferência de funcionalidades em uma rede veicular entre os parceiros de desenvolvimento, além de permitir que novas aplicações sejam agregadas durante o ciclo de vida do veículo.

Para cumprir seus objetivos técnicos de prover uma infraestrutura comum de software para sistemas automotivos, quatro características foram tidas como essenciais, sendo elas:

- **Modularidade:** permite que pedaços de código sejam agregados conforme requerimento do software;
- **Escalabilidade:** promove a adaptação de módulos de software comuns para diferentes veículos;
- **Transferibilidade:** otimiza o uso de recursos disponíveis através da arquitetura eletrônica de um veículo;

²⁰<https://github.com/ciaa/Firmware/>

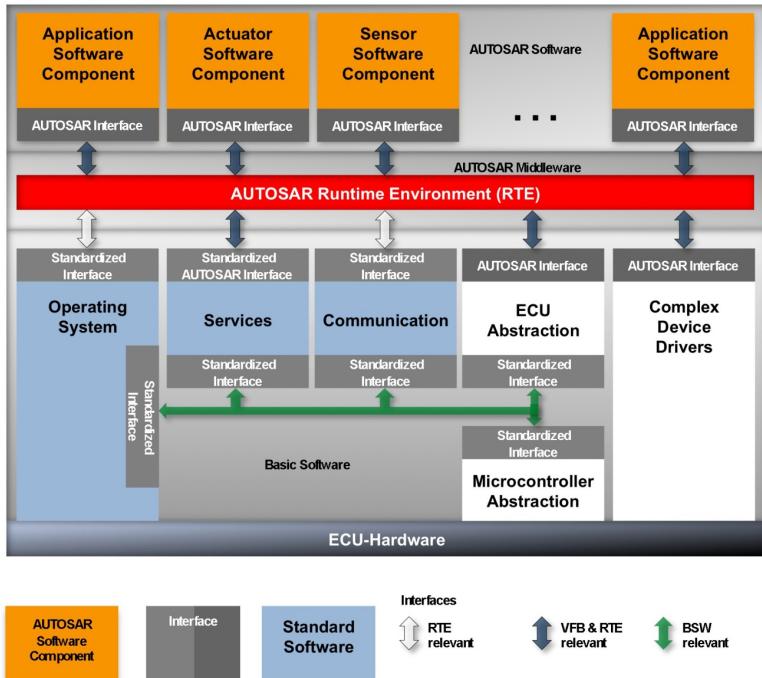
²¹<http://erika.tuxfamily.org/drupal/>

²²<http://trampoline.rts-software.org/>

- **Reusabilidade:** aumenta a qualidade do produto, utilizando códigos já testados;

Com base nestes conceitos, a arquitetura básica pode ser ilustrada na Figura 15. Ela também demonstra como os módulos do AUTOSAR estão associados.

Figura 15: Visão geral técnica do AUTOSAR.



Extraido de: ??).

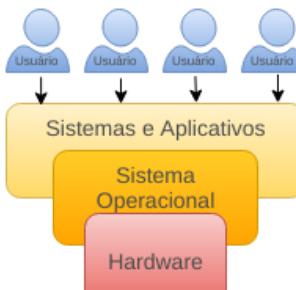
3 SISTEMAS OPERACIONAIS EMBARCADOS

Neste capítulo são abordados os principais conceitos sobre sistemas operacionais. Ao final do capítulo são relatados alguns estudos de caso a respeito de sistemas operacionais embarcados com foco para a automação veicular.

3.1 SISTEMAS OPERACIONAIS

O objetivo de um SO é o de gerenciar os recursos de um sistema computacional, tornando transparente seu funcionamento para o usuário final. O SO é o componente de software que faz a união e abstração dos recursos de hardware e os oferece de maneira simplificada para as aplicações utilizadas pelo usuário final. A Figura 16 ilustra a visão abstrata de um SO, ou seja, o SO como um provedor de serviços para as aplicações de usuários.

Figura 16: Visão geral da integração do SO com o sistema computacional.



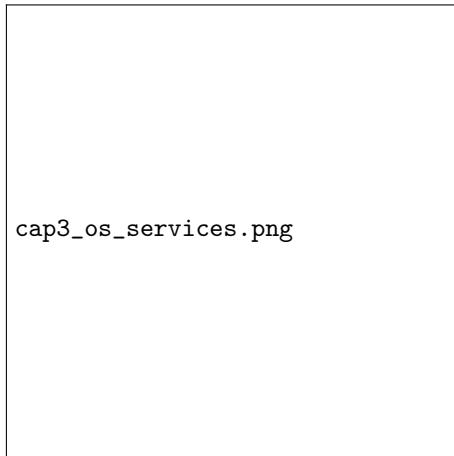
Adaptado de: ??).

Segundo ??), um SO pode oferecer um número variável de serviços. Os serviços do SO que estão sempre presentes na memória principal fazem parte do kernel, que é descrito na Seção 3.1.1.

Para que os aplicativos tenham acesso aos serviços oferecidos sem que haja um comprometimento de sua integridade, o SO oferece rotinas de acesso aos serviços. A quantidade de serviços varia conforme implementação do SO, sendo o mínimo oferecido as operações de gerenciamento de processos, memória e Entrada e Saída - E/S. A Figura

17 apresenta uma hierarquia de chamada dos serviços do SO.

Figura 17: Serviços de um SO.



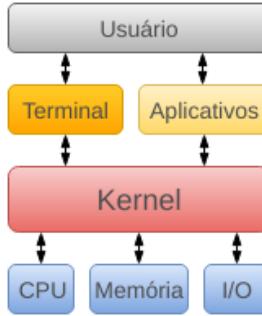
Adaptado de: ??).

3.1.1 Kernel

??) define o *kernel* como sendo o programa que constitui o núcleo central de um sistema operacional. Ele é responsável por fazer o gerenciamento dos recursos de hardware bem como sua abstração para os aplicativos do usuário. A Figura 18 ilustra onde se encontra a camada de abstração do *kernel*.

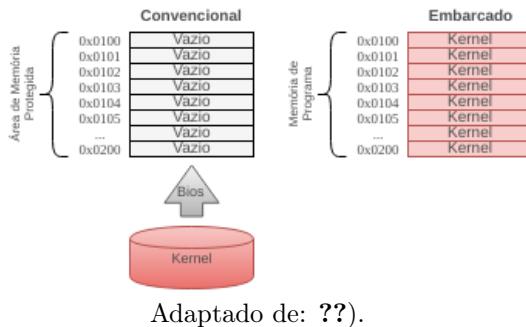
Em sistemas computacionais convencionais, o *kernel* é a primeira parte do SO a ser carregada na memória durante o processo de inicialização. Uma vez carregado, o *kernel* permanece na memória principal do sistema até que o mesmo seja desligado. Para Sistemas Operacionais Embarcados - SOE, o *kernel* sempre está presente na memória de programa. Após uma instrução de *reset*, é sempre o primeiro código a ser executado. A Figura 19 ilustra ambos os modelos.

Figura 18: Posição conceitual do *kernel*.



Extraido de: ??).

Figura 19: Posição do *kernel* na memória.



3.2 ESTRUTURA DOS SISTEMAS OPERACIONAIS

Independente dos serviços oferecidos, um SO pode ser separado em quatro serviços principais: gerenciamento de processos, de memória, de entrada e saída de dados e sistemas de arquivos.

3.2.1 Gerenciamento de Processos

O gerenciamento de processos consiste no tratamento de interrupções, controle de tarefas e acesso a recursos do sistema, de uma maneira que nenhum dos processos entre em conflito ou pare sua execução permanentemente, que não de maneira espontânea.

A quantidade de elementos que compõem um SO varia conforme implementação, mas geralmente incluem: tarefas, semáforos, filas de mensagem, entre outras.

3.2.1.1 Tarefas

Em um SO convencional, cada programa que é executado no computador toma a forma de um processo, que pode ser definido como uma atividade que possui sua própria pilha, área de memória privada, e que pode alocar memória dinamicamente conforme a necessidade, muitas vezes sem precisar se preocupar com a falta do recurso. Mais programas podem ser adicionados a qualquer momento em um SO para computadores.

Em sistemas embarcados, o conceito de processo é substituído pelo de *tarefas*, onde a principal diferença está no fato de que todas as tarefas já estão definidas no momento em que o SO inicia. Para criar novos tipos de tarefas é necessário parar o SO e adicioná-las manualmente.

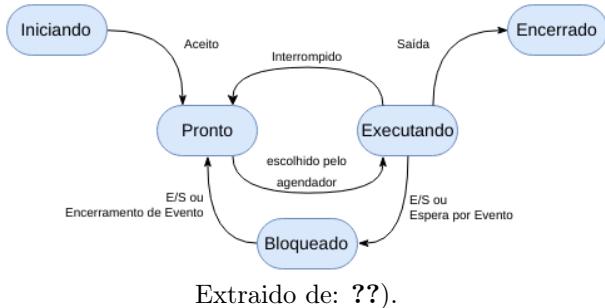
Uma tarefa pode assumir um número de estados pré-determinado, os quais variam conforme o SO utilizado, mas geralmente incluem os estados presentes na Figura 20, conforme visto em ??). Estes estados podem ser descritos como:

- **Iniciando:** etapa inicial, onde são executadas instruções de preparo para a tarefa;
- **Pronto:** indica que a tarefa está pronta para ser executada, mas que ela não é a tarefa ativa, no momento;
- **Executando:** estado da tarefa que está em execução no SO;
- **Bloqueado:** a tarefa está aguardando a ocorrência de algum evento externo para que ela possa voltar a executar;
- **Finalizado:** a tarefa libera recursos alocados, antes de ser encerrada definitivamente.

3.2.1.1.1 *Fibras*

Uma variante das tarefas são as fibras, que são threads de execução leves e não-preemptivas, normalmente utilizadas para executar

Figura 20: Estados de uma tarefa.



porções de código responsáveis pelos drivers de dispositivos e outros trabalhos de desempenho crítico (??).

3.2.1.2 Escalonadores de Tarefas

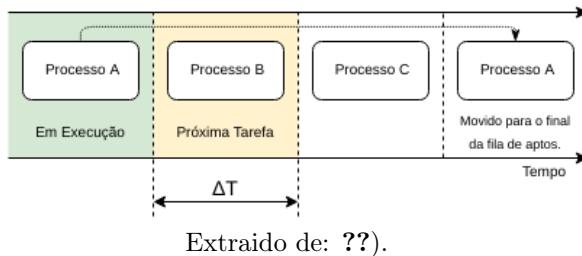
O escalonamento de tarefas surgiu para permitir que mais de uma tarefa fique em execução no processador, sem que ela tenha que concluir seu funcionamento para que isso aconteça. Na prática, isso gera uma situação na qual várias tarefas aparentam estar em execução ao mesmo tempo, em arquiteturas com um único processador.

??) descreve alguns tipos de algoritmos para escalonamento. Serão apresentados aqui apenas os escalonadores mais utilizados em RTOSes, sendo eles Round Robin, Prioridade e Prioridade com Round Robin.

3.2.1.2.1 Escalonador Round Robin

No escalonador Round Robin, uma tarefa permanece em execução apenas por um valor ΔT de tempo, geralmente na casa dos *us*, após a qual ela é movida para o estado de *Pronto*, colocando a próxima tarefa na fila em execução, conforme ilustrado na Figura 21.

Figura 21: Diagrama esquemático de funcionamento do algoritmo de escalonamento Round Robin.



3.2.1.2.2 Escalonador por Prioridade

Neste tipo de escalonador, a tarefa é associada a um atributo numérico, o qual indica o nível de sua prioridade. A tarefa de mais alta prioridade que não estiver bloqueada será sempre a que estará em execução no SO. Ela permanece em execução até que se auto-bloqueie, seja por uma rotina de *delay*, por tentar acessar um recurso indisponível no momento, ou ainda porque uma tarefa de maior prioridade ficou disponível.

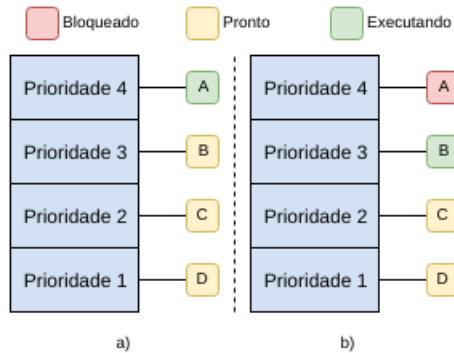
A Figura 22 apresenta um escalonador por prioridade em dois momentos. No momento (a), a tarefa em execução chama uma rotina que faz seu auto-bloqueio. No instante de tempo (b), a tarefa em execução passa a ser a próxima tarefa de prioridade mais alta, disponível naquele momento.

3.2.1.2.3 Escalonador por Prioridade com Round Robin

Faz uma junção dos dois tipos de escalonadores apresentados anteriormente. Funciona da mesma maneira que o escalonador de prioridades, exceto que agora é possível ter mais de uma tarefa com o mesmo nível de prioridade. Quando este cenário ocorrer, as tarefas de mesma prioridade ficarão alternando a posição de *em execução*, através do algoritmo Round Robin.

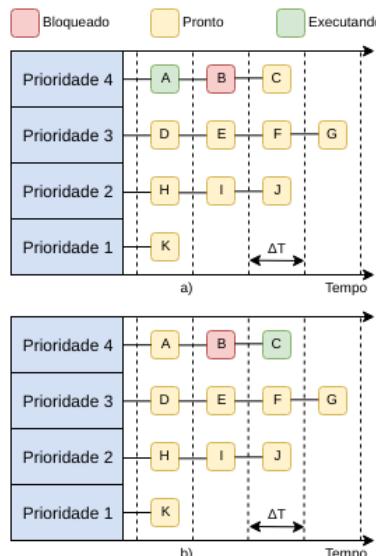
A Figura 23 apresenta um exemplo deste escalonador, onde após um intervalo de tempo ΔT , ocorre a troca de contexto para a próxima tarefa de mesma prioridade que não se encontra bloqueada.

Figura 22: Diagrama esquemático de funcionamento do algoritmo de escalonamento baseado em prioridade.



Extraido de: ??).

Figura 23: Diagrama esquemático de funcionamento do algoritmo de escalonamento baseado em prioridade com Round Robin.



Extraido de: ??).

3.2.1.3 Semáforos

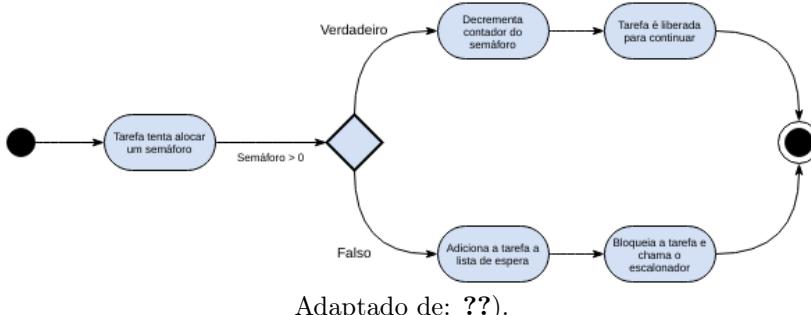
Múltiplas tarefas em execução concorrente devem ser capazes de sincronizar suas ações e coordenar o acesso mutuamente exclusivo a

recursos compartilhados. Para atender a estes requisitos, o SO pode prover um objeto chamado semáforo.

Um semáforo funciona como uma espécie de chave que permite a uma tarefa acessar algum tipo de operação ou recurso. Se a tarefa puder adquirir um semáforo, ela poderá continuar sua execução normalmente. Do contrário, a tarefa poderá ser bloqueada até que o recurso esteja disponível novamente.

Segundo ??), existem semáforos de contagem, binários e de exclusão mútua. Semáforos de contagem e binários apresentam comportamentos similares, tendo como única diferença que um semáforo binário possui seu valor máximo igual a 1. Um diagrama de atividades, demonstrando o funcionamento de um semáforo pode ser visto na Figura 24.

Figura 24: Transição de Estados para Semáforos de Contagem.



3.2.1.3.1 Mutex

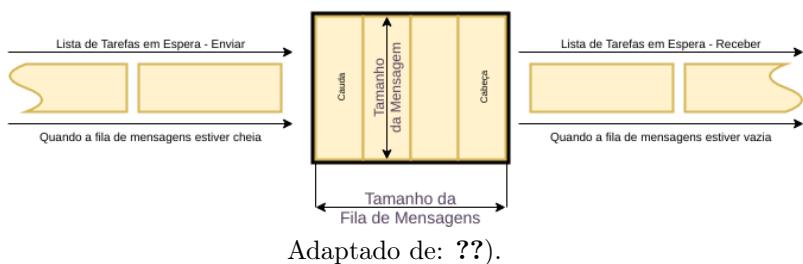
O semáforo do tipo *mutex* atende a um caso especial do semáforo binário, chamado de semáforo de exclusão mútua ou *mutex*. Um *mutex* pode suportar propriedades de posse, trava recursiva, deleção segura de tarefas, dentre outros serviços, afim de evitar problemas inerentes a exclusão mútua.

3.2.1.4 Filas de Mensagens

Uma fila de mensagens é um objeto através do qual as tarefas e *Interrupt Service Routine* - ISR¹ enviam e recebem mensagens para comunicação e sincronização de dados. Elas armazenam temporariamente as mensagens do remetente até que o destinatário esteja pronto para receber-las. Isto garante o desacoplamento entre a tarefa emissora e receptora.

Uma fila de mensagens é composta por objetos chamados de *elementos*, dos quais cada um pode armazenar uma única mensagem, a qual pode estar vazia. O número total de elementos na fila equivale ao seu comprimento. A Figura 25 apresenta um esquema para as filas de mensagens.

Figura 25: Diagrama esquemático do funcionamento de uma fila de mensagens..



Adaptado de: ??).

3.2.1.5 Sinais

Um sinal é uma interrupção gerada por software, a qual dispara quando ocorre um evento. Assim como numa interrupção, um sinal faz com que o processo em execução seja interrompido para executar uma outra rotina assíncrona.

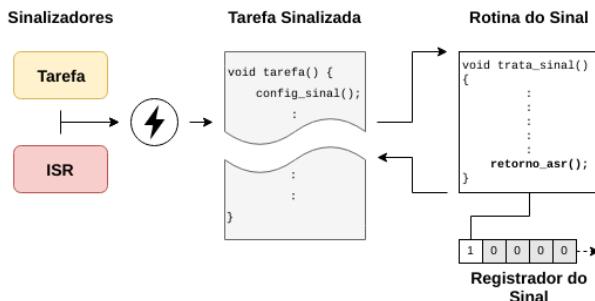
Na essência, os sinais notificam as tarefas de eventos que ocorreram durante a execução de outras tarefas ou ISRs. Assim como as interrupções, estes eventos são assíncronos para a tarefa notificada e não ocorrem em nenhum ponto pré-determinado durante sua execução. A principal diferença entre uma interrupção e um sinal é que o primeiro é gerado por hardware, como quando um pino de um microcontrolador

¹Traduzido como: Rotina de Serviço para Interrupções.

passa de 0V para 5V, enquanto o último é gerado por software.

Quando há a chegada de um sinal, a tarefa desvia de seu fluxo normal de execução, e a *Asynchronous Signal Routine* - ASR² correspondente é invocada, conforme ilustrado na Figura 26.

Figura 26: Tratamento de sinal em uma tarefa.



Adaptado de: ??).

3.2.2 Gerenciamento de Memória

Em muitos dos sistemas embarcados (tais como celulares, câmeras digitais, *tablets*) há um número limitado de aplicações (tarefas) que podem estar em execução em um dado momento. Parte do motivo é que estes dispositivos apresentam uma quantidade limitada de memória física. Dispositivos maiores, como roteadores de rede, possuem uma maior quantidade de memória física instalada, mas também fazem maior uso dela e precisam de um gerenciamento ainda maior deste recurso.

Independentemente do tipo de sistema embarcado, os requisitos comuns a sistemas de gerenciamento de memória são a mínima fragmentação, mínima sobrecarga em operações de gerenciamento e tempos de alocação determinísticos.

3.2.2.1 Alocação Dinâmica de Memória

O código do programa, seus dados e a pilha do sistema ocupam a memória física do sistema embarcado, uma vez carregados e inicia-

²Traduzido como: Rotina de Sinal Assíncrona.

lizados. A memória restante é utilizada para alocação dinâmica pelo RTOS ou pelo *kernel*. Esta área recebe o nome de *heap*³.

Um gerenciador de memória mantém informações sobre a *heap* em uma área reservada, chamada de bloco de controle. Informações típicas sobre o controle incluem:

- O endereço inicial do bloco de memória física utilizado para alocação dinâmica;
- O tamanho total de memória disponível;
- A tabela de alocações, a qual indica quais áreas da memória estão em uso, quais estão vagas e o tamanho de cada região que ainda está livre.

Um sistema de memória deve ser capaz de executar eficientemente as seguintes operações:

- Determinar se há um bloco livre que comporta a alocação requisitada;
- Manter as informações internas atualizadas;
- Mesclar um ou mais blocos, assim que estes forem liberados;

A estrutura da tabela de alocação é a chave para um gerenciamento de memória eficaz. Esta estrutura gera um *overhead*, já que ocupa espaço de memória que, outrora, poderia ser utilizado para armazenar dados dos programas. Minimizar a tabela de alocações e maximizar o desempenho das operações anteriores é um dos principais desafios no gerenciamento de memórias.

3.2.3 Subsistemas de Entrada/Saída

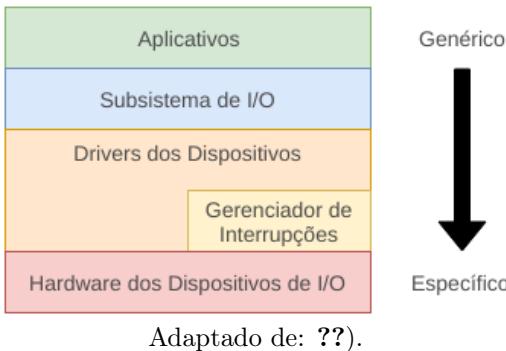
Em sistemas embarcados, um sistema de entrada/saída são a combinação dos dispositivos de E/S, *drivers* de dispositivos associados e subsistemas de E/S.

O propósito de um subsistema de E/S é o de esconder do *kernel* as informações específicas de um dispositivo, assim como do desenvolvedor de aplicações, e prover uma método de acesso uniforme aos periféricos de E/S do sistema embarcado.

³Tanto a *Stack* quanto a *Heap* são áreas de memória para um programa. A diferença entre elas é que a sua alocação é, respectivamente, estática e dinâmica

A Figura 27 ilustra o subsistema de E/S em relação ao resto do sistema em um modelo de camadas de software. Conforme indicado, cada camada descendente agrega mais informações à arquitetura necessária para gerenciar um dado dispositivo.

Figura 27: Subsistema de E/S e o modelo por camadas.



3.2.3.1 Camada de Abstração

Cada dispositivo de E/S pode oferecer um conjunto específico de interfaces de programação para os aplicativos. Este arranjo requer que cada aplicativo esteja ciente da natureza do dispositivo de E/S subjacente, incluindo as restrições impostas pelo dispositivo. O conjunto da *Application Programming Interface - API*⁴ é específico à implementação, o que torna difícil a portabilidade das aplicações utilizando esta API. Para reduzir esta dependência, é implementado no sistema embarcado um subsistema de E/S, a qual atua como uma camada de abstração.

Esta camada de abstração define um conjunto padrão de funções para operações de entrada e saída, de forma a esconder as peculiaridades dos dispositivos da aplicação. Todos os *drivers* de E/S passam a se conformar e a suportar este conjunto de funções, já que o objetivo é o de prover uma camada uniforme de E/S para as aplicações.

Para se alcançar estas operações de E/S uniformemente no nível de aplicação, os seguintes procedimentos devem ser seguidos:

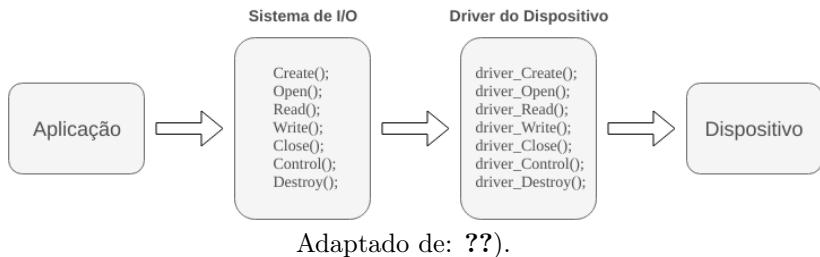
1. Definir o conjunto de APIs do subsistema de E/S.

⁴Traduzido como: Interface de Programação de Aplicação.

2. Implementar cada função do conjunto para o driver do dispositivo.
3. Exportar este conjunto de funções do driver do dispositivo para o subsistema de E/S.
4. Encarregar ao driver do dispositivo o preparo do mesmo para uso.
5. Carregar o dispositivo pelo driver do mesmo e informar o subsistema de E/S.

A Figura 28 ilustra como a camada de E/S abstraí o dispositivo de hardware, garantindo a flexibilidade do sistema.

Figura 28: Camada de abstração entre o aplicativo e o dispositivo.



3.2.4 Sistemas de Arquivos

Segundo ??), um arquivo é uma coleção nomeada de informações relacionadas que são gravadas em uma unidade secundária e persistente de armazenamento. Pela perspectiva do usuário, dados não podem ser salvos em uma unidade secundária, senão pela alocação de um arquivo nela.

Os arquivos são utilizados para guardar informações referentes a dados ou programas. Arquivos de dados podem ser numéricos, alfabéticos, alfanuméricos ou binários. Eles podem também ser estruturados ou não. De maneira geral, um arquivo é uma sequência de bits, no qual o significado varia conforme o criador e usuário do arquivo.

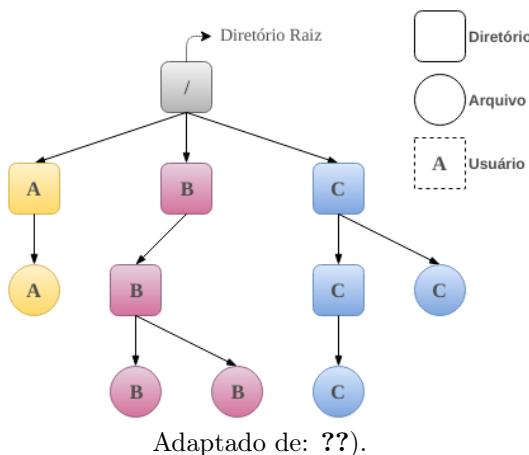
Existem diversos componentes capazes de persistirem estes dados, sendo alguns deles discos e fitas magnéticas, *flash cards* e *Compact Disks*. Para que o sistema computacional possa utilizar estes dispositivos, o SO deve abstrair as propriedades físicas dos dispositivos de

armazenagem e definir uma unidade lógica de armazenagem, capaz de armazenar e localizar facilmente os dados em forma de arquivos.

Um sistema de arquivos provê os meios para organizar os dados, de forma que eles possam ser armazenados, resgatados e atualizados, além de gerenciar o espaço disponível no dispositivo que o contém. Um sistema de arquivos organiza os dados de maneira eficiente e é otimizado para características específicas de um dispositivo (??).

Internamente, um sistema de arquivos funciona de maneira similar a alocação dinâmica da memória principal, onde o sistema armazena informações adicionais sobre a área de dados, como seu tamanho e ponto de origem no dispositivo. Contudo, um sistema de arquivo oferece mais níveis de organização, afim de facilitar o encontro de informações posteriormente, como diretórios, ou ainda controle de permissões, para limitar o acesso de usuários. A Figura 29 apresenta um modelo de organização de diretórios com controle de permissão.

Figura 29: Modelo em árvore de diretórios por usuário.



3.3 SISTEMAS EMBARCADOS

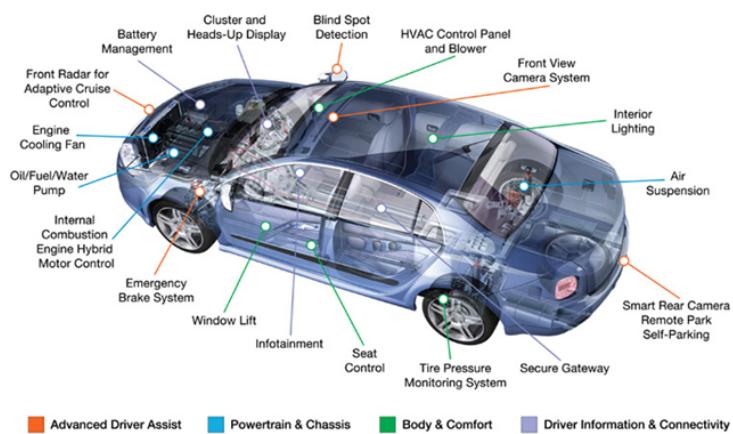
Um sistema embarcado é um sistema computacional cujo propósito é bem definido e específico. Ele é geralmente conceituado para atender a uma situação particular, com hardware bem definido e, muitas vezes, imutável.

Diferente de um sistema computacional convencional, como os

Desktops e Laptops, um sistema embarcado possuí recursos de hardware bem mais restritos. Por ser projetado para um propósito específico, o mesmo muitas vezes não requer tecnologia de ponta para executar sua tarefa, o que permite a criação de sistemas de baixo custo.

Um sistema embarcado pode ser composto, por exemplo, de sensores e atuadores, bem como uma central de processamento. Em veículos, um exemplo equivalente seria o dos sensores e atuadores para travamento das portas, bem como a central eletrônica responsável por sua operação. A Figura 30 demonstra diversos sistemas embarcados presentes em um automóvel.

Figura 30: Sistemas embarcados em um veículo.



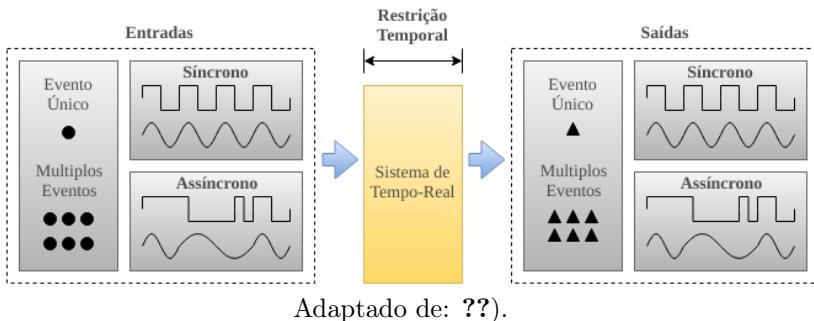
Extraido de: ??).

3.4 SISTEMAS DE TEMPO-REAL

Segundo ??), uma aplicação de tempo-real é aquela que é composta por múltiplas tarefas independentes em execução, as quais competem pelo tempo de processamento em um processador.

Um sistema de tempo-real pode ser definido como um sistema que responde a eventos externos em tempo hábil. Ou seja, onde o tempo de resposta é uma garantia do sistema. A Figura 31 ilustra o conceito de sistema de tempo real, onde independente da entrada, há um tempo limite de resposta, o qual deverá sempre ser respeitado.

Figura 31: Resposta em Tempo-Real.



Uma restrição temporal pode ser rígida ou flexível. Restrições rígidas causam uma falha do sistema quando não respeitadas, geralmente causando danos físicos ao equipamento e possível risco humano. Um sistema de restrição flexível não sofre consequências tão graves quando o mesmo falha, mas ainda assim pode causar invalidação do sistema ou de processos. Um veículo possui diversos sistemas embarcados, muitos dos quais possuem restrições temporais, tanto rígidas quanto flexíveis.

3.5 ESTUDOS DE CASO

A seguir são apresentados alguns estudos de casos sobre sistemas operacionais embarcados já existentes no mercado.

3.5.1 FreeRTOS

Conforme ??), o FreeRTOS é uma solução padronizada para sistemas operacionais embarcados. Por separar os códigos do sistema operacional dos códigos específicos para a arquitetura do sistema embarcado, ele é considerado um SO extremamente portável, sendo disponibilizado nas principais plataformas do mercado.

Dentre as características oferecidas pelo SO, podem-se destacar:

- Escalonador de tarefas baseado em prioridades com *Round Robin*;
- Ocupa pouco espaço da memória do microcontrolador;
- Oferece recursos como semáforos, filas de mensagens e co-rotinas;

- Timers e interrupções por softwares.

Sua interface é resumida a três arquivos de código fonte. Os demais arquivos atendem a adaptação para o hardware alvo.

Em desenvolvimento a mais de 12 anos, o FreeRTOS é o SOE mais utilizado no mercado. Além de sua versão de código aberto, ele também possui licenças para versões comerciais (??).

3.5.2 PICOS18

Conforme ??), o PICOS18 é considerado um kernel de tempo-real. Ele surgiu como uma implementação de um SOE para a arquitetura PIC18, utilizando o compilador C18 da Microchip®.

Embora descontinuado, se destaca por ser uma das primeiras soluções de código aberto para sistemas operacionais dedicados a automação veicular, apresentando uma implementação não-homologada da norma OSEK/VDX.

3.5.3 Trampoline

Segundo ??), o Trampoline⁵ é uma plataforma aberta para sistemas embarcados de pequeno porte com restrições de tempo-real. Sua inspiração veio inicialmente do padrão OSEK/VDX e, atualmente, busca ser compatível com o padrão de SO do AUTOSAR.

Ele é composto por:

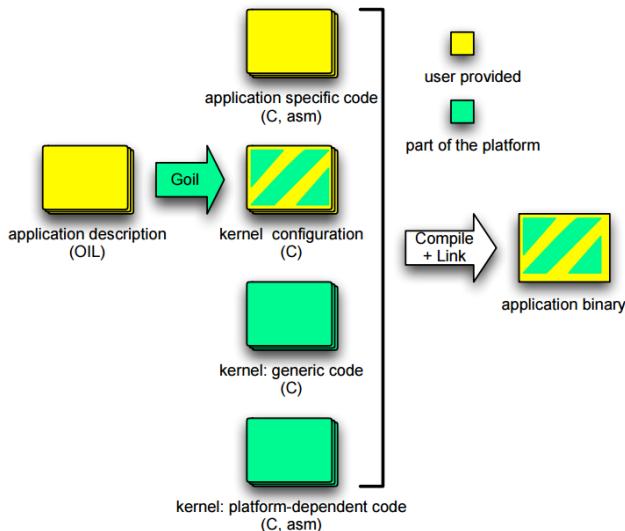
- Um kernel de tempo-real - trampoline;
- Uma ferramenta de configuração do kernel - GOIL;
- Um ambiente virtual para prototipação de aplicações em estações de trabalho.

Conforme especificado pelo padrão OSEK/VDK, a linguagem OIL é utilizada para gerar o código fonte em C para configuração do SO, neste caso, processada pela ferramenta GOIL, conforme ilustra a Figura 32. É possível observar também que o código do kernel é dividido duas partes: uma genérica e a outra específica para a arquitetura alvo.

Por consequência do padrão AUTOSAR OS, todos os objetos de sistemas devem ser estáticos. Isto implica que threads, mutexes, semáforos, etc, devem ser declarados em tempo-de-compilação.

⁵<http://trampoline.rts-software.org>

Figura 32: Processo de geração das configurações do kernel.



Extraido de: ??).

O foco do SO é voltado para hardwares de baixo custo, com especificações de:

- **Arquitetura:** entre 16 ou 32 bits;
- **Clock:** até 20MHz;
- **RAM:** até 32KB;
- **ROM:** até 128KB;

Sua implementação oferece suporte a modelos regido temporalmente e regidos por eventos, ambos necessários à premeditabilidade em tempo-real. Também oferece suporte para computações regidas por eventos.

4 OPENAUTOS

Este capítulo apresenta a proposta do SO OpenAUTOS, destacando as escolhas tanto do projeto do software bem como a arquitetura de hardware adotada.

4.1 PROPOSTA

O sistema OpenAUTOS é um SOE de código aberto, que busca atender a norma AUTOSAR SO, ou seja, realizar a implementação do sistema operacional proposto pelo padrão. Para tal, serão seguidas e respeitadas as guias e restrições definidas no relatório de especificações definidos em ??).

Por se tratar de um SO estático, não serão implementados gerenciadores de memória e sistemas de arquivos. Porém, isto implica que todas as tarefas, semáforos e demais objetos fornecidos pelo kernel sejam definidos em tempo de compilação. Segundo a norma, é obrigatório que um parser da linguagem OIL ou, alternativamente, *eXtensible Markup Language - XML*¹ seja fornecido, para que este faça a geração do arquivo de cabeçalho da linguagem C que será utilizado durante a compilação do SO. A Figura 33 ilustra este cenário.

Na Figura 34, é apresentada uma lista dos módulos e objetos mais comuns em um SOE. O OpenAUTOS se propõe a implementar estes módulos, com exceção da alocação dinâmica de memória e sistema de arquivos pois, por se tratar de um SO estático, todas as alocações e definições deverão ser realizadas em tempo de compilação. Filas de mensagens e áreas de memória reservadas poderão ser definidas estaticamente.

Em termos de escalonador, o sistema deverá oferecer suporte para as tarefas normais e estendidas do padrão. As tarefas normais fazem uso de um escalonador de prioridades com Round Robin. Já as tarefas estendidas fazem uso de *tabelas de escalonamento*, as quais garantem que restrições temporais serão atendidas e que uma operação não será interrompida antes de sua conclusão (??).

Segundo a norma ??), é necessário que a interface das funções sejam realizadas em C. Logo, está será a linguagem de implementação primária do projeto. Em seções críticas de performance, poderá ser utilizado a linguagem Assembly, mas nunca para a definição de métodos.

¹Traduzido como: Linguagem de Marcação Extensível.

Figura 33: Geração do cabeçalho de configuração.

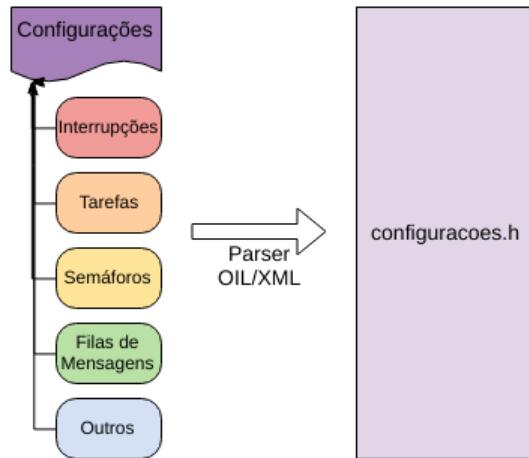
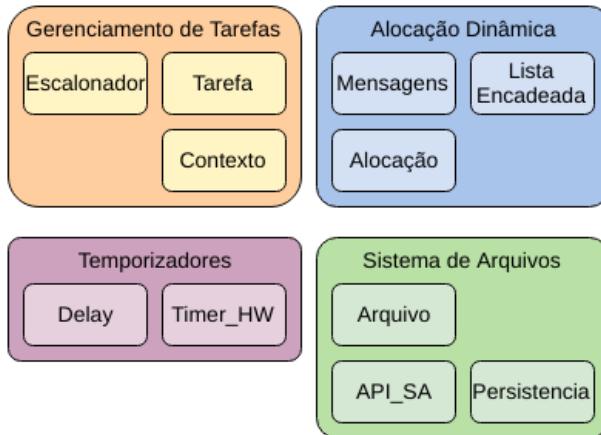


Figura 34: Módulos comuns a um SOE.



4.2 ARQUITETURA

Devido a disponibilidade, bem como o suporte as linguagens escolhidas, serão utilizadas as placas FRDM-KL25Z da *freescale*, com processador *ARM Cortex-M0*, ilustrado em Figura 35. Com base neste hardware, a implementação do OpenAUTOS utilizará uma arquitetura

de processador único, centralizando todas as responsabilidades do SO em uma única ECU.

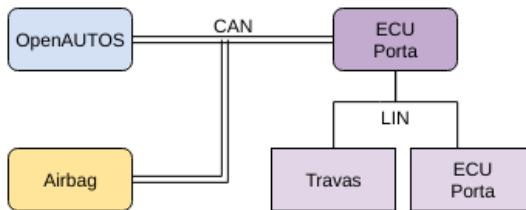
Figura 35: FRDM-KL25Z.



Extraido de: ??).

Para fins de simulações e demonstração, cada placa FRDM-KL25Z representará um sistema ou ECU de um veículo. Uma ECU será responsável por executar o SO OpenAUTOS, enquanto que as demais farão a manipulação de sistemas automotivos, como ABS, travamentos de portas, dentre outros, conforme ilustra a Figura 36

Figura 36: Comunicação entre ECUs e Sistemas.



Na comunicação entre os diferentes módulos eletrônicos serão utilizados os protocolos CAN e LIN. Será necessário o uso de dois pares de pinos TX/RX, um para cada protocolo. O protocolo LIN não precisa de nenhum tratamento especial, pois é baseado na UART. O protocolo CAN, porém, não é disponibilizado nativamente. Sendo assim, serão utilizados *transceivers* MPC2551 para comunicação entre ECUs.

A comunicação entre os diferentes módulos também deverá usar as normas do AUTOSAR para aplicativos, utilizando o interfaceamento definido na norma.

4.3 CRONOGRAMA

Com base no trabalho proposto, a Tabela 3 propõe um cronograma para estudo e implementação do projeto.

Tabela 3: Cronograma.