O que é um sistema embarcado? É muito mais fácil dizer o que um sistema embarcado não é, do que tentar descrever tudo que um sistema embarcado pode ser. Enquanto um computador usual (desktop/notebook/PDA) serve aos mais diversos propósitos: checar seu orkut/email, usar seu editor de texto, usar a Internet, etc., um sistema embarcado é implementado para um propósito específico. Em geral, tais sistemas realizam uma única tarefa ou um número reduzido de tarefas relacionadas.

Em geral, um sistema embarcado:

* É um sistema construído para executar uma tarefa, completamente ou parcialmente independente de intervenção humana.
* É especialmente desenvolvido para concretizar pequenas ações da forma mais eficiente possível.
* Interage com elementos físicos do ambiente. Ex: um sensor de temperatura.

O software embarcado:

* Different software designed for general-purpose computers, embedded software cannot usually be run on other embedded systems without significant modification. This is mainly because of the incredible variety in the underlying hardware. The hardware in each embedded system is tailored specifically to the application, in order to keep system costs lo w. As a result, unnecessary circuitry is eliminated and hardware resources are shared wherever possible.
* By definition all embedded systems contain a processor and software, but what other features do they have in common? Certainly, in order to have software, there must be a place to store the executable code and temporary storage for runtime data manipulation. These take the form of ROM and RAM, respectively; any embedded system will have some of each. If only a small amount of memory is required, it might be contained within the same chip as the processor. Otherwise, one or both types of memory will reside in external memory chips.
* See Figure 1-1 for a general example of an embedded system.



**1. A generic embedded system**

Desta forma um sistema embarcado pode ser definido como um “computador de propósito único”.

Uma grande parte dos sistemas embarcados são sistemas de tempo crítico ou tempo real (time critical or real-time applications), isto é, são sistemas que funcionam em ambientes nos quais o controle de tempo (timing) é muito importante: os resultados de determinada operação só são relevantes se ocorrerem num determinado tempo. Um sistema de controle de vôo de um avião é um sistema embarcado de tempo crítico: em caso de algum problema durante o vôo, se o dispositivo de controle não detectar o problema prontamente e solucioná-lo de forma adequada, em milissegundos, haverá conseqüências catastróficas.

É interessante notar que um computador de propósito geral é composto de uma série de sistemas embarcados: teclado, mouse, HD, modem, sound card, etc. Cada um desses sistemas possui um processador e software embutidos, bem como tarefas específicas: o modem, por exemplo, envia e recebe dados de forma digital através da linha de telefone analógica.

Se um sistema embarcado for bem projetado, a existência de um processador e de software pode passar completamente despercebida por um usuário de tal dispositivo. Esse é o caso do forno micro-ondas, do DVD, do relógio digital, da cafeteira, da máquina de lavar, do controle remoto, entre outros. Em alguns casos, seria possível construir um dispositivo equivalente, que não contivesse um processador nem software. Isso poderia ser feito substituindo tal combinação por um circuito integrado que realizasse as mesmas tarefas em hardware puro. Entretanto, muita flexibilidade se perde, pois é muito mais fácil e mais barato mudar algumas linhas de código, do que projetar novamente uma peça de hardware.

**Programando SE:**

**INTRODUÇÃO:**

Programar sistemas embarcados não é como programar um PC comum. De fato programação para um sistema embarcado se assemelha a programação de computadores de 15 anos atrás. O hardware escolhido para o sistema é em geral escolhido para que o dispositivo seja o mais barato possível. Bem, isso significa que programador terá que lidar com processadores lentos e baixa memória, enquanto que ao mesmo tempo se esforçará para conseguir uma eficiência que não é vista na maioria dos aplicativos para PC.

Há um grande número de arquiteturas diferentes de sistemas embarcados e isso torna as ferramentas para programar mais caras. Além disso, essas ferramentas têm menos recursos e são menos desenvolvidas. Num projeto embarcado de grande porte, fatalmente você irá encontrar algum ‘’bug’’no compilador. Ferramentas de “debug” são outro problema, pois nem sempre você consegue rodar um “debugger” no seu processador embarcado devido à escassez de recursos e isso torna o controle de erros do programa difícil. Hardwares especiais como portas JTAG podem superar esse problema parcialmente. Debugging é de suma importância, pois se o sistema “travar” durante o funcionamento de um sistema de hardware real (como um motor, por exemplo), podem ocorrer danos permanentes ao equipamento.

Para um programador de sistemas embarcados, fazer uso dos algoritmos mais eficientes é essencial, bem como um entendimento da arquitetura de hardware sobre a qual você está trabalhando, de forma a facilitar a otimização do seu software.

Memória também é um problema, por ser escassa: diferentemente de programas para PC, o programador deve usar algoritmos que sacrifiquem tempo do processador em favor da memória. Ou seja, nada de “memory leak” (consumo de memória, não intencional, quando um programa falha em liberar memória que não é mais necessária). Frequentemente, aplicativos embarcados usam técnicas determinísticas de memória e evitam o uso de funções “malloc” (função C de alocação dinâmica), facilitando a investigação e solução de possíveis “leaks”.

**C e Assembly**

Muitos programadores preferem programar em C a programar em assembly, por boas razões, já que C é de alto nível e portanto libera o profissional do conhecimento e controle de alguns detalhes de implementação a nível de máquina.

Contudo há alguns processos de baixo-nível que ou podem ser melhor implementados em assembly ou **somente** podem ser implementados em linguagem assembly. Por outro lado é freqüentemente útil para o programador analisar o código assembly produzido pelo compilador C e manualmente editá-lo a fim de otimizá-lo de formas que o compilador não consegue. Assembly também é útil para processos de tempo crítico (time-critical) porque diferentemente do que ocorre com linguagens alto-nível não há ambigüidade quanto a como o código será compilado. O “timing” pode ser controlado de forma mais eficiente, o que é útil para produzir “drivers” simples.

A maioria dos programadores de C são “mal-acostumados” porque eles programam em ambientes nos quais há uma série de bibliotecas prontas para uso. O fato cruel é que em sistemas embarcados raramente há tantas bibliotecas às quais o programador habituou-se a usar, ocasionalmente um sistema embarcado pode nem ter uma biblioteca padrão completa, se sequer houver uma. Frequentemente, devido a problemas de espaço, não é possível incluir uma biblioteca inteira, e programadores são muitas vezes forçados a implementar suas próprias bibliotecas. Enquanto algumas bibliotecas são muito grandes e não muito adequadas ao uso em microcontraladores, muitos sistemas de desenvolvimento incluem as bibliotecas padrão mais comuns para programadores C.

C continua a ser uma linguagem muito popular para microcontroladores devido à sua eficiência de código e reduzido tempo de desenvolvimento. C oferece controle de baixo nível e é mais humanamente legível que assembly. Muitos compiladores C estão disponíveis para uma grande gama de plataformas de desenvolvimento. Os compiladores são parte de IDEs com uma janela de assembly. Adicionalmente, usando C há um aumento na portabilidade, uma vez que o código em C pode ser compilado por diferentes tipos de processadores.

## Bootloader

A fim de simplificar muitas tarefas, programadores de muitos sistemas usam um tipo genérico de software chamado **bootloader** que é responsável por realizar algumas rotinas do sistema (como: liberar o modo protegido), e então carregar o kernel na memória, para a partir daí transferir o controle do sistema para o kernel. Bootloaders são usados em muitos microcontroladores e são em geral a forma mais rápida para atualizar um programa, com pequenas mudanças, em um microcontrolador. Isso torna o ciclo editar-compilar-baixar-testar um pouco mais rápido.

**Ferramentas de Desenvolvimento:**

**O proceso de construção**

Existem muitas coisas que as ferramentas de desenvolvimento de software podem fazer automaticamente quando a plataforma alvo está bem definida. Essa automação é possível porque as ferramentas podem explorar recursos do hardware e do sistema operacional nos quais o programa será executado. Dessa forma o compilador, esconde do programador certos aspectos do processo de construção do software.

O processo de conversão do código-fonte para do software embarcado em um arquivo binário executável envolve três passos distintos. Primeiro, cada arquivo-fonte precisa ser compilado ou montado em um programa-objeto. Depois, todos os os programas-objetos originados a partir do primeiro passo necessitam ser interconectados para produzir um objeto único, chamado programa-relocável (“ relocatable program”). Por último, o endereço físico da memória devem receber os “offsets relatives” do programa-relocável no processo chamado realocação. O resultado do terceiro passo é um arquivo que contém um um programa-executável em binário que está pronto para ser rodado no sistema embarcado.



Figura: Processo de conversão do código-fonte para um arquivo binário do Sistema Embarcado

Cada um dos passos no processo de construção do software embarcado é uma transfromação realizada por software rodando num computador de propósito geral. Para distinguir esse computador, onde se realiza o desenvolvimento ( PC ou Unix workstation), do sistema embarcado alvo (target), ele é referido como o computador host. Em outras palavras, o compilador, assembler, linker são softwares rodando no host, e não no sistema alvo. Ainda assim, apesar do fato de rodarem em outra plataforma computacional, essas ferramentas se combinam para produzir um executável binário que irá rodar apropriadamente no sistema embarcado alvo.

****

**Divisão entre host e alvo**

Vale ressaltar o fato de os kernel’s monolíticos vem se tornando cada vez mais populares por seus vários benefícios. É um kernel relativamente grande com capacidades sofisticadas é adaptado para um ambiente embarcado. Isso dá ao programador um ambiente similar ao de um sistema operacional desktop como o [Linux](http://en.wikipedia.org/wiki/Linux) ou o [Microsoft Windows](http://en.wikipedia.org/wiki/Microsoft_Windows), e, portanto o desenvolvimento é muito produtivo, por outro lado, tal kernel requer consideravelmente mais recursos de hardware, e por causa da sua complexidade esse tipo de kernel é menos previsível e menos seguro/confiável (maior chance de erros). Alguns exemplos de kernel’s monolíticos embarcados são [Embedded Linux](http://en.wikipedia.org/wiki/Embedded_Linux) e o [Windows CE](http://en.wikipedia.org/wiki/Windows_CE). Apesar de aumentar o custo do hardware, esse tipo de sistema embarcado está cada vez mais popular, especialmente em dispositivos mais poderosos como [Wireless Routers](http://en.wikipedia.org/wiki/Routers) e [GPS Navigation Systems](http://en.wikipedia.org/wiki/Automotive_navigation_system).

Há diferentes tipos de arquitetura de software em uso:

**SUPER LOOP:**

Um superloop é uma estrutura de programa composto de um loop infinito, com todas as tarefas do sistema contidas nesse loop. Aqui está um pseudocódigo de um superloop.

Function Main\_Function()

{

Initialization();

Do\_Forever

{

Check\_Status();

Do\_Calculations();

Output\_Response();

}

}

Nós colocamos as rotinas de inicialização antes do superloop, porque nós somente queremos inicializar o sistema uma única vez. Assim que o loop infinito seja iniciado, nós não queremos “resetar” os valores, porque precisamos manter estado persistente no sistema embarcado. Dessa forma, torna-se imprescindível o uso do superloop nos sistemas embarcados. Sistemas embarcados não são os únicos a utilizar tal arquitetura, jogos, por exemplo, constantemente usam um loop similar, o chamado *(tight) (main) game loop*.

## Power Save Superloop:

Digamos que tenhamos um sistema embarcado que tem um tempo de loop de 1ms, e só precisa checar certa entrada de dados uma vez por segundo. É um evidente gasto continuar a repetir o loop ininterruptamente, especialmente se não precisamos fazer nada na maior parte do tempo. Nessa situação, por exemplo, o programa irá executar 1000 loops antes de ler a entrada, e os outros 999 loops terão sido apenas uma contagem regressiva para a próxima leitura. Nesse caso, é muito ineficiente ter o processador rodando a 100% todo o tempo. Seria muito melhor implementa um atraso (delay) no programa, tornando esse sistema embarcado mais eficiente. O superloop extendido seria:

Function Main\_Function()

{

Initialization();

Do\_Forever

{

Check\_Status();

Do\_Calculations();

Output\_Response();

**Delay\_For\_Next\_Loop();**

}

}

Se tal atraso for de 999ms, não precisamos de 1000 loops e podemos ler a entrada em cada loop.

Desenvolvedores de sistemas embarcados usam [compiladores](http://en.wikipedia.org/wiki/Compiler), [assemblers](http://en.wikipedia.org/wiki/Assembly_language#Assembler) e [debuggers](http://en.wikipedia.org/wiki/Debugger) para desenvolver softwares de sistemas embarcados. Contudo, eles também usam ferramentas mais específicas. Entre elas estão ferramentas , matematicas como o MATLAB (software voltado para o calculo numerico), ou ainda se pode personalizar uma linguagem de programação para otimizar o seu uso (Java, Pascal ).

DEBUGGING

Como um sistema embarcado é geralmente composto por uma série de elementos, a estratégia de debugging pode variar. Por exemplo, debugar um sistema embarcado centrado em software-microprocessador é diferente de debugar um sistema no qual a maior parte do processamento é realizada por periféricos.

Sistemas embarcados freqüentemente residem em máquinas que têm expectativas de funcionamento contínuo por anos, sem erros, ou em alguns casos que se recuperem automaticamente de algum erro que ocorra. Portanto o software é geralmente desenvolvido e testado mais cuidadosamente do que se este fosse destinado a computadores pessoais.

Os softwares embarcados são muitas vezes construídos de forma que um erro possa ser recuperado automaticamente, pois o sistema está inacessível (exemplo: sistemas espaciais), ou precisa estar sempre ligado (sistema de controle de reatores nucleares), ou ainda o mau funcionamento da maquina pode acarretar grandes prejuízos (ex: caixas automáticos).