**SLIDE 8 – SISTEMA OPERACIONAL**

**INTRODUÇÃO**

* Nenhum programa começa do zero, uma parte do código foi reutilizada. Quem fez o “printf”?
* Não há tempo de reescreer
* Sempre ocorrerá a utilização da propriedade intelectual de alguém. Do mesmo jeito que no hardware você usa plataformas.
* Reuso:
  + SO, você utiliza chamadas a serviços do SO, o escalonador, gerenciador de memória.
  + Banco de dados
  + Middleware
  + Bibliotecas, exemplos clássicos de reuso.
* Software é o valor agregado ao hardware, em parte por você e em parte pelos outros. Tempo de escrever o software é o que mais consome num projeto, logo procura-se diminuí-lo o máximo possível.
* Existem coisas que realmente são necessárias que você escreva.

**PLATAFORMA**

* Software para reuso.
* IDE
* Protocolos
* Abstração do acesso ao hardware, primitivas de acesso ao hardware.
* Sistema operacional.
* Programas de exemplo.

**PORQUE USAR SO?**

* Pergunta difícil de responder. Existe um espaço de soluções para cada aplicação.
* Podemos analisar vários pontos:
  + Interface hardware e software:
    - Antigamente:
      * Hardware e software eram desenvolvidos pelos mesmos indivíduos.
      * Maior parte do produto era hardware e software era pequeno.
    - Avanço:
      * Hardware cresceu.
      * Pode-se usar times separados para hardware e software.
      * Software também cresceu.
      * Introdução de drivers: a camada de interface que sabe acessar o hardware escrita pelo pessoal do hardware e utilizada pelo pessoal do software.
    - Maiores aplicações:
      * Introdução de um SO.
      * Hardware se tornou mais complexo.
      * Features mais interessantes no software.
      * Padronização dos drivers entre as várias plataformas onde a aplicação pode ser usada. Promove uma migração mais facilitada.
      * Aplicações devem ser grandes os suficiente de forma a justificar os gastos de recursos que a introdução do SO promove.
  + Modelo de programação
    - Laço simples:
      * Trechos de código (serviços) sequenciais em um loop. Parece polling.
      * Cada serviço deve ser rápido.
      * Cada serviço não pode ser muito cricri com o tempo.
      * Aplicações simples.
      * Os serviços devem ser rápidos o suficiente de modo que o usuário não perceba que eles estão rodando sequencialmente.
    - Laço e ISR:
      * Utilizado quando cada serviço demora ou quando um serviço não pode esperar demais.
      * Um meio de fazer certos serviços furarem a fila.
      * Laço principal faz poucas coisas.
      * Trechos de código podem interromper o processador.
      * Usado quando se quer introduzir prioridade.
      * Número limitado de interrupções (serviços que podem interromper)
    - RTOS
      * Usado quando o número de serviços que podem interromper precisar ser muito grande
      * Utilizado quando se quer fazer o decoupling do código.
      * Quando o projeto exige uma gerência mais organizada.
      * Quando se quer ganhar um nível maior de isolamento entre as tarefas (de forma a proporcionar o trabalho independente entre membros de um time)
  + Complexidade do processador X do software
    - CPU simples + software simples: não usa RTOS ou um simples kernel.
    - CPU hard + software hard: usa um RTOS top
    - CPU hard + software simples: RTOS básico, não que gastar muito, hardware superdimensionado.
    - CPU simples + software hard: RTOS básico, não tem muito espaço

**RECURSOS DE UM SO PARA EMBARCADOS**

* Configuralibilidade:
  + Selecionar partes do SO para o produto final.
  + Acontece no desktop, mas é bônus. Nos sistemas embarcados é necessidade, para economia.
  + Necessário também por conta da diversidade de plataformas, não faz sentido o SO oferecer as mesmas coisas para todas as plataformas.
  + SO pode estar acoplado junto com o código da aplicação ou pode ir antes da aplicação.
* Mecanismos de proteção:
  + Desktop precisa pois tem muitos usuários.
  + SEMB não precisa pois um único usuário sabe e confia nas suas threads.
  + Pode-se retirar o código responsável por isso para economizar.
* Interrupções:
  + SO controla e faz intermédio entre aplicação e interrupções.
  + SO é quem trata as interrupções.
  + Pode desligar a influencia do SO sobre as interrupções deixando o gerenciamento das mesmas para a aplicação.
* Tempo real
  + Previsibilidade
  + Gerenciamento do tempo.
  + Influencia no escalonador de tarefas.

**ANALISANDO RECURSOS OFERECIDOS PELO SO E SUA INFLUENCIA EM FATORES COMO CUSTO E POTENCIA**

* Alto desempenho:
  + Fator dependente da CPU
  + Fazer um código mais eficiente diminuir a potencia e o custo pois posso usar CPU mais pobre.
  + Eficiência de SO:
    - Uso de um compilador melhor.
    - Poder diminuir o número de tarefas máxima pode otimizar o gerenciamento e melhorar a eficiência.
* Memória:
  + Mais memória significa custo maior
  + Economiza memória: diminuir o código e o SO (configurabilidade)
* Potência:
  + Menos memória menos potência
  + So mais eficiente ajuda:
    - Pode-se usar uma organização diferente de CPU, menor potencia e custo.
    - Pode-se usar menores frequências.
* Custo
  + Menos memória menor o custo.
  + CPU mais pobre menor o custo.
  + Cuidado com suporte e licenciamento:
    - Recursos a mais de otimização geralmente são pagos.
    - Recursos a mais podem ser de graça mas com suporte pago.
    - Recursos grátis podem exigir um tempo maior para aprendizado da ferramente.

**MIDDLEWARE**

* Conceito antigo
* Não era de sistemas embarcados
* Várias máquinas
* Framework usado pelos desenvolvedores para invocar cosias na rede em plataformas distribuídas.
* Parece um So distribuído.
* Provê abstração distribuída.