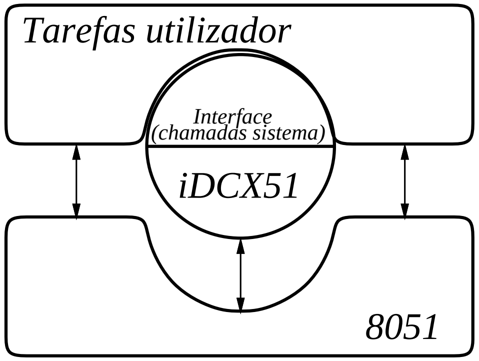
Sumário

- Arquitectura.
- Desenvolvimento de aplicações.
- Serviços do núcleo.

iDCX51: ilustração



Categorias de serviços

- Serviços de gestão de tarefas.
- Serviços de comunicação entre tarefas.
- Serviços de gestão de memória.
- Serviços de gestão das interrupções.
- Serviços de tempo.

Serviços de gestão de tarefas

- O executivo perimite escalonar até oito tarefas.
- A interface permite criar e escalonar uma tarefa:

  Estaticamente (durante a iniciação do sistema).

  Dinamicamente (em tempo de execução):`RQ$CREATE$TASK`.
- Eliminar uma tarefa: `RQ$DELETE$TASK`.
- Suspender a execução de uma tarefa: `RQ$WAIT`.

  Até que ocorra um evento (mensagens, interrupções e alarmes).

Serviços de comunicação entre tarefas

- Permite às tarefas comunicarem através da troca de mensagens.
- Reservar memória para a mensagem: `RQ$ALLOCATE`.
- Enviar uma mensagem: `RQ$SEND$MESSAGE`.
- Fazer uma tarefa esperar pela recepção de uma mensagem: `RQ$WAIT`.
- Devolver memória previamente reservada: `RQ$DEALLOCATE`.

Serviços de gestão de memória

- Serviços que permitem optimizar a utilizaçao do espaço disponível na memória RAM interna ao processador.
- Obter o endereço de um bloco de memória interna de um comprimento especificado: `RQ$GET$MEM`.
- Devolver um bloco de memória ao sistema: `RQ$RELEASE$MEM`.

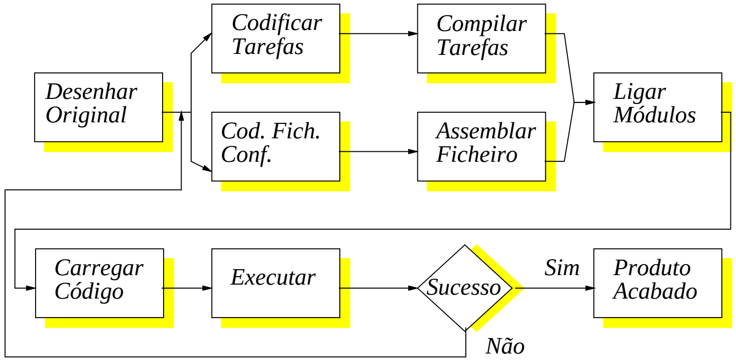
Serviços de gestão das interrupções

- Fornece uma interface assíncrona que permite às tarefas gerirem as interrupções.
- Atribuir a gestão de uma dada interrupção a uma tarefa específica.
- Inibir determinadas interrupções: `RQ$DISABLE$INTERRUPT`.
- Desinibir interrupções: `RQ$ENABLE$INTERRUPT`.
- Sincronizar tarefas com interrupções: `RQ$WAIT` .

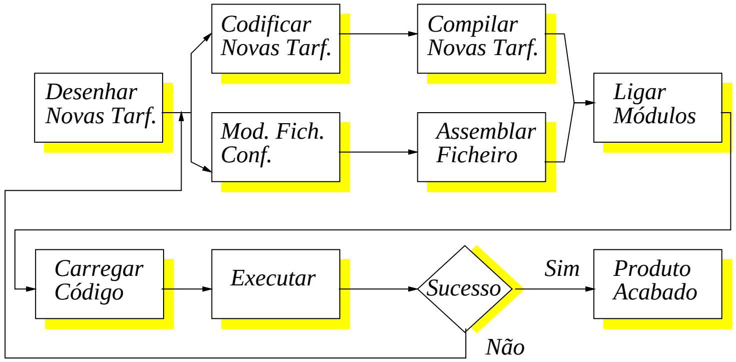
Serviços de tempo

- Utilização dos temporizadores internos para permitir medir intervalos de tempo e lançar alarmes.
- Definir um intervalo de tempo e sinalizar uma tarefa após esse intervalo: `RQ$SET$INTERVAL`.
- Esperar que um intervalo expire: `RQ$WAIT`.

Desenvolvimento de aplicações



Manuntenção de aplicações



Ambiente de desenvolvimento

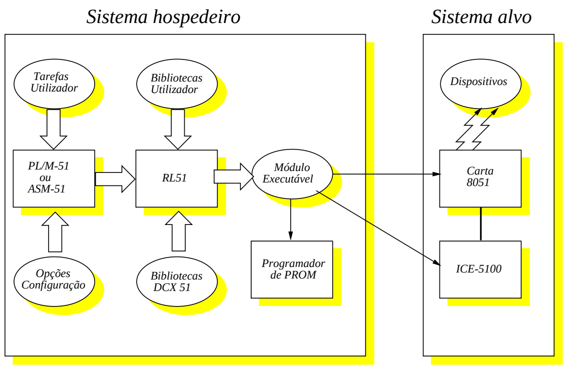
- Sistema hospedeiro ("host system"):

  Sistema no qual o código é desenvolvido.

- Sistema alvo ("target system"):

  Sistema no qual o código é executado.

Ambiente de desenvolvimento



Carregamento do executável

- Três alternativas disponíveis:

  Carregamento na memória do sistema alvo.

  Carregamento na memória do emulador.

  Programação de PROM, EPROM, ou EEPROM e instalação física da memória no sistema alvo.

Primitivas de gestão de tarefas

- Criar e escalonar uma tarefa: `RQ$CREATE$TASK`.
- Apagar uma tarefa: `RQ$DELETE$TASK`.
- Obter o identificador das tarefas do sistema: `RQ$GET$FUNCTION$IDS`.

Representação das tarefas

- Segmento de código: *define as instruções a executar pela tarefa.*
- Descritor da tarefa: *onde o núcleo guarada informação respeitante à tarefa.*
- Pilha privada: *para salvaguarda do contexto de execução.*
- Espaço partilhado de registos: *acumulador (ACC), registo de estado do processador (PSW), registo de usos genérico (B), apontador para os dados (DPH,DPL), apontador para a pilha (SP), contador de instruções (PC).*
- Banco de registos de utilização genérica: *conjunto de registos R0-R7 (podem ser utilizados quatro bancos de registos diferentes).*

Descritor de tarefa

- Designado por "Initial task Descriptor" (IDT).
- Padrão mágico: *sequência de dígitos que identifica a zona de memória como um válido IDT.*
- Endereço do código: *primeira instrução de código.*
- Tamanho da pilha.
- Nome: *código que permite identificar a tarefa.*
- Banco escolhido: *valor binário que indica se o banco de dados é escolhido pelo utilizador ou pelo sistema.*

Descritor de tarefa (continuação)

- Banco de dados: *qual o banco de dados utilizado.*
- Prioridade.
- Interrupções: *quais as interrupções associadas à tarefa.*
- Próximo: *usado para ligar os registos em listas.*

Estados das tarefas

- Existem três estados possíveis para uma tarefa:

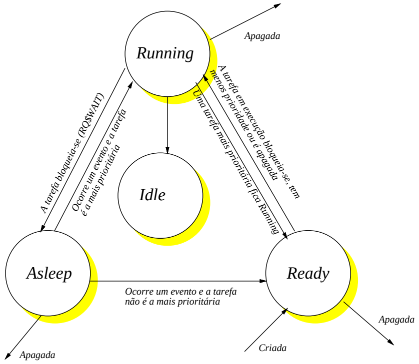
  Bloqueada (Asleep), quando espera por um evento após uma chamado do tipo. **RQ$WAIT**.

  Executável (Ready), quando estó pronta para ser executada. É inserida na lista de tarefas executáveis de acordo com a sua prioridade (1-4). Gestão FIFO na mesma prioridade.

  Em execução (Running).

- Quando não existem tarefas em execução ou executáveis o sistema fica inactivo (Idle).

Estados das tarefas



Escalonamento

- Escalonamento preemptivo, baseado em prioridades:
- São usados dois mecanismos elementares:

  A lista de tarefas executáveis.

  A preempção.

Lista de tarefas executáveis

- Uma tarefa é inserida á frente de todas as tarefas de menor prioridade.
- Uma tarefa à qual é retirado o processador é inserida á frente de outras tarefas de igual prioridade.
- Uma tarefa à qual não foi retirado o processador é inserida após todas as tarefas de igual prioridade.

Preempção

- Existem cinco acontecimentos que podem provocar a preempção (retirar o processador a uma tarefa).

  1. *Ocorre uma interrupção que desbloqueia uma tarefa de maior prioridade.*
  2. *Expira um alarme esperado por uma tarefa de maior prioridade.*
  3. *Expira uma espera condicionada de uma tarefa de maior prioridade.*
  4. *É enviada uma mensagem para uma tarefa de maior prioridade que se encontra à espera da mensagem.*
  5. *É criada uma tarefa de maior prioridade.*

Utilização das primitivas

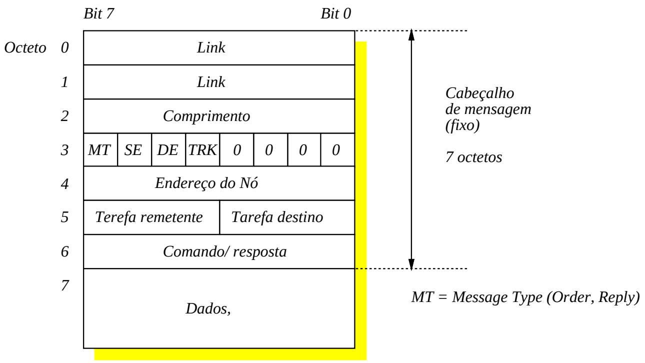
```
declare task_id byte;
declare status byte main;

DPL     = LOW  (.idx); /* Estrutura declarada noutro ficheiro */
DPH     = HIGH (.idx);
task_id = RQ\$CREATE\$TASK;
+
DPL     = LOW  (.table); /* Estrutura declarada noutro ficheiro      */
DPH     = HIGH (.table); /* (vector de 8 octetos, um para cada tarefa */
+
ACC     = task_id;
status  = RQ\$DELETE\$TASK;
```


Primitivas para comunicação entre tarefas

- Reservar espaço para uma mensagem: `RQ$ALLOCATE`.
- Libertar espaço de uma mensagem: `RQ$DEALLOCATE`.
- Enviar uma mensagem: `RQ$DSEND$MESSAGE`.
- Esperar por uma mensagem: `RQ$WAIT`.

Estrutura das mensagens



Gestão de memória para mensagens

- Durante a iniciação o sistema reserva um conjunto de blocos de memória para utilização na troca de mensagens.
- Este espaço designa-se por "System Buffer Pool (SBP)".
- As tarefas podem reservar e libertar blocos de memória deste espaço usando as chamadas:

  `RQ$ALLOCATE.`

  `RQ$DEALLOCATE.`

Reserva de memória para transmissão

- Quando uma tarefa pretende enviar uma mensagem, executa tipicamente as seguintes operações:

  Reserva um bloco de memória (a não ser que já disponha de uma bloco disponível).

  Preenche os campos da mensagem.

  Envia a mensagem (não liberta a memória).

Utilização de memória na recepção de uma mensagem

- Quando recebe uma mensagem, uma tarefa executa tipicamente as seguintes operações:

  Espera pela mensagem.

  Lê a informação da mensagem.

  Constroi uma mensagem de resposta ou liberta o bloco de memória.

Utilização das primitivas

```
msg_ptr = RQ$ALLOCATE;
preenche_a_mensagem;
DPL     = LOW (msg_ptr);
DPH     = HIGH (msg_ptr);
call RQ$SEND$MESSAGE;
+
B       = timeout;
ACC     = INTERVAL_EVENT or MESSAGE_EVENT;
msg_ptr = RQ$WAIT;
stat    = ACCl;
```


Utilização de mensagens (exemplo 1)

```
LOOP;
  msg_ptr = RQ$ALLOCATE;
  preencher a mensagem;
  RQ$SEND$MESSAGE (msg_ptr);
  RQ$WAIT(pela resposta);
  RQ$DEALLOCATE (msg_ptr);
ENDLOOP;
```

```
LOOP;
  RQ$WAIT(pedido);
  ler mensagem;
  preencher resposta;
  RQ$SEND$MESSAGE(msg_ptr);
ENDLOOP;
```


Utilização de mensagens (exemplo 2)

```
msg_ptr = RQ$ALLOCATE;
LOOP;
  preencher a mensagem;
  RQ$SEND$MESSAGE (msg_ptr);
  RQ$WAIT(pela resposta);
ENDLOOP;
```

```
LOOP;
  RQ$WAIT(pedido);
  ler mensagem;
  preencher resposta;
  RQ$SEND$MESSAGE(msg_ptr);
ENDLOOP;
```


Utilização de mensagens (exemplo 3)

```
LOOP;
  msg_ptr = RQ$ALLOCATE;
  preencher a mensagem;
  RQ$SEND$MESSAGE (msg_ptr);
ENDLOOP;
```

```
LOOP;
  RQ$WAIT(pedido);
  RQ$SEND$MESSAGE(msg_ptr);
  RQ$DEALLOCATE (msg_ptr);
ENDLOOP;
```

Primitivas de gestão de memória

- Reservar memória: `RQ$GET$MEM`.
- Libertar memória: `RQ$RELEASE$MEM`.

Gestão da memória sistema

- O núcleo gere a memória interna do processador mantendo uma lista ligada de blocos livres de tamanho variável.
- Região de memória designada por System Memory Pool (SMP)

Primitivas de gestão de interrupções

- Inibir uma interrupção: `RQ$DISABLE$INTERRUPT`.
- Desinibir uma interrupção: `RQ$ENABLE$INTERRUPT`.
- Esperar pela ocorrência de uma interrupção: `RQ$WAIT`.

Gestão de interrupções

- Cada nível de interrupção (0-7) pode ser inibido através da activação do dígito correspondente no "Interrupt Enable Register":
- Uma tarefa pode ficar associada a um nível de interrupção:

  Quando a interrupção ocorre a tarefa em execução perde o processador, o qual é dado à tarefa que trata a interrupção.

  Se o gestor de interrupção tem menor prioridade que a tarefa em execução, a interrupção é inibida.

Latência da interrupção

- Intervalo de tempo entre a ocorrência da interrupção e o início de execução da tarefa que a gere.
- Fontes de atraso:

  Latência simples: devido à execução do sistema (gestão de listas, salvaguarda de registos, etc).

  Tempo de inibição da interrupção por tarefas mais prioritárias.

  Tempo de inibição da interrupção por tarefas menos prioritárias.

  Tempo de inibição da interrupção para actualização do relógio sistema.

Latência da interrupção: valores

- Latência simples:

  Entre 5 μs (só vectorização) e 174 μs (com mudança de contexto).
- Devido ao relógio do sistema:

  Depende do número de tarefas que usem o relógio (mínimo 50 μs, máximo centenas de μs).
- Outros valores:

  Dependem do código das tarefas.

Primitivas de gestão do tempo

- Estabelecer um intervalo de tempo: `RQ$SET$INTERVAL`.
- Esperar por um fim de um intervalo de tempo: `RQ$WAIT`.

Temporizadores

- Enviam periódicamente um sinal para a tarefa.
- Esta pode esperar pelo sinal invocando `RQ$WAIT`.
- O temporizador é lançado de novo de modo automático.

Temporizadores: utilização

```
declare timer_tick byte;
declare dummy word;
+
ACC = timer_tick;
call RQ$SET$INTERVAL
+
DO WHILE (1);
  +
  B     = WAIT_FOREVER;
  ACC   = INTERVAL_EVENT;
  dummy = RQ$WAIT;
END;
```


Espera temporizada

- Permite esperar por um acontecimento por um período pré-determinado.
- A temporização só é feita durante a espera.

Espera temporizada: utilização

```
declare timeout byte;
declare dummy word;
+
B     = timeout;
ACC   = TIMEOUT_EVENT;
dummy = RQ$WAIT;
+
```


Resumo

- Arquitectura.
- Ambiente de desenvolvimento.
- Serviços de gestão de tarefas.
- Serviços de comunicação entre tarefas.
- Serviços de gestão de memória.
- Serviços de gestão das interrupções.
- Serviços de tempo.