4
O Sistema DSRT

Disponibilizado sob a forma de bibliotecas compartilhadas (ligáveis - *link*), o sistema DSRT (*Dynamic Soft Real Time*) [9] [11] (Figura 14) tem como objetivo chave suprir o nível de garantia dos recursos que um servidor compartilhado pode oferecer a diferentes usuários e suas aplicações, com exigências diversas de recurso e exigências variáveis de desempenho.

Trata-se de um sistema que, mesmo que não possa fornecer à aplicação um nível da garantia do recurso tão boa quanto se esta executasse em um *hardware* dedicado, pode oferecer um nível aceitável de garantia à maioria das aplicações que toleraram violações ocasionais na garantia do recurso computacional.

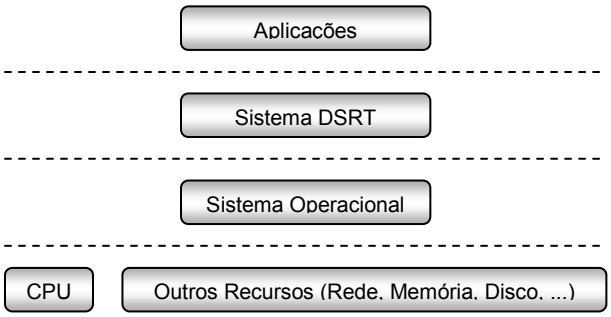


Figura 14 - Sistema DSRT

4.1.
Funcionalidades do sistema

A principal motivação no desenvolvimento do projeto DSRT, foram às dificuldades com a gerência de recursos necessários às aplicações, do tipo: (i) necessidade de execução sobre diversos sistemas operacionais e plataformas de *hardware*; (ii) necessidade de que as interfaces de programação nativas para

processos locais de sistemas operacionais, que monitoram a disponibilidade de recursos, sejam iguais nas diferentes plataformas; e (iii) dificuldade na alteração do mecanismo de escalonamento, desenvolvidos no nível do sistema operacional (*kernel level*).

Acrescentando a essas dificuldades, verifica-se ainda que, se por um lado os sistemas operacionais de propósito geral fornecem às aplicações interfaces padrão, capacidade de acesso a redes, e ferramentas de desenvolvimento, por outro não são projetados para fornecer em sua forma nativa, serviços com alguma QoS.

O sistema DSRT foi projetado para fornecer mecanismos flexíveis e escaláveis para diferentes tipos de recursos compartilhados (e.g., memória, processador, etc.) e plataformas operacionais, permitindo uma visão uniforme e portável, oferecendo negociação, admissão e reserva de recursos, de uma maneira integrada e acessível. Além disso, permite que parâmetros de QoS sejam obtidos para classes de aplicações variadas.

Trata-se de um mecanismo para estender sistemas operacionais de propósito geral. Como tal, permite que aplicações com alguma necessidade de QoS verifiquem exigências de recursos para serem executadas, reservem o recurso e adaptem a reserva. Permite, ainda, que o sistema operacional gerencie de maneira previsível os recursos para satisfazerem uma ampla gama de propriedades tais como evitar a postergação infinita (*starvation-free*) para aplicações da classe “melhor esforço” (*best-effort*), e propriedades específicas como garantias SRT (*Soft Real-Time*) para aplicações multimídia.

O sistema emprega o algoritmo preemptivo R-EDF (*Reservation-based Earliest Deadline First*) [10] para escalonar classes múltiplas de tarefas *soft real-time* e *best-effort*. Uma das principais características desse algoritmo é que fornece proteção e tratamento a excessos no uso de recursos (*overrun*) de modo que: (i) a tarefa que provoque o “excesso” não force com que outras esgotem seu tempo de execução; e (ii) a tarefa que provoque o “excesso” encerre suas atividades dentro de um limite aceitável e previsível de tempo. As aplicações passam a utilizar uma única interface de programação para solicitar o estado do recurso.

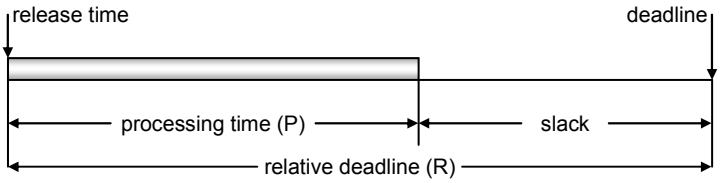
Através do escalonador de recursos (*scheduler*) o sistema faz o controle de admissão na reserva e o escalonamento das aplicações *soft real-time* e *best-effort*, adicionando um baixo custo (*overhead*). Além de escalonar, o sistema fornece garantias de tempo (QoS) para quatro classes de tarefas *soft real-time* (Figura 15):

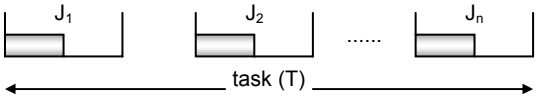
- PCPT (*Periodic Constant Processing Time*): escalonamento no qual os períodos de tempo e a utilização de recursos dentro desses períodos são constantes.

- *Event*: caso particular do PCPT onde se têm apenas um período de tempo.

- PVPT (*Periodic Variable Processing Time*): escalonamento no qual o período de tempo é constante mas a utilização dentro deste período é variável, contudo, esta variação se repete a cada super-período.

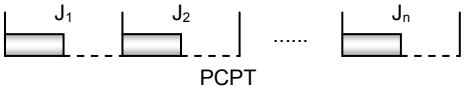
- ASCU (*Aperiodic/Sporadic Constant Utilization*): escalonamento no qual o período de tempo e a utilização dentro deste são variáveis, havendo uma relação de proporcionalidade nesta variação.

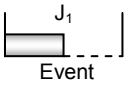




T = {J1, J2, ... , Jn}, onde n ≥ 1

utilização $\Theta(T) = \frac{1}{n} \sum_{i=1}^{n} \Theta(Ji)$





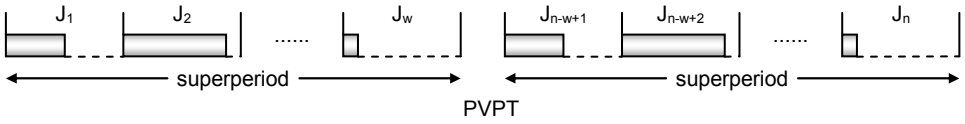


Figura 15 - Classes de tarefas *soft real-time*

4.2. Ambiente DSRT

O sistema DSRT fornece as seguintes características no cumprimento das exigências mínimas que permitem a coexistência de clientes SRT (*Soft Real Time*) e TS (*Time Sharing*), em um ambiente de sistema operacional de propósito geral:

- Reserva e garantia de tempo de processador:

  O cliente SRT executa uma reserva no sistema DSRT. A partir dela, o sistema executa um teste de controle de admissão para verificar se existem recursos suficientes de processador que satisfaçam o pedido de reserva. Em seguida, o sistema usa o algoritmo R-EDF que garante o tempo de processador reservado a todos os clientes;

- Proteção a excessos no uso de recursos (*overrun*):

  Situação na qual um cliente SRT necessita mais tempo de processador do que aquele que reservou para encerrar sua tarefa. O sistema DSRT fornece proteção a essa situação, considerando que excessos de um cliente não podem causar violações aos contratos de outros clientes;

- Garantia de alocação para processos TS:

  O sistema DSRT aloca uma porcentagem fixa de tempo de processador aos processos TS tradicionais, de modo que não ocorra postergação infinita (*starvation*) na presença de clientes SRT.

Conceitos e serviços utilizados pelo sistema DSRT:

- Classes de serviço de CPU:

  Clientes multimídia necessitam de uma ampla variedade de padrões no uso de processador. Por exemplo, alguns podem requerer um tempo de uso constante ou variável, podendo ter períodos fixos ou dinâmicos. O sistema DSRT define múltiplas classes de serviço de processador que podem atender clientes SRT com variados padrões de uso;

- Fluxo de execução:

  No sistema DSRT, um cliente percorre três fases distintas. A primeira fase é a avaliação (*probe*), quando os parâmetros de reserva são obtidos. A segunda fase é a reserva, quando um cliente submete seus parâmetros ao sistema. A terceira fase é a execução, quando um cliente é escalonado pelo servidor DSRT;

- Serviço de adaptação:

  Clientes multimídia podem mudar no tempo seus padrões de uso de processador. Por exemplo, a quantidade de tempo de processador usado por um decodificador MPEG, pode depender do cenário e da quantidade de ações na cena. O sistema DSRT fornece a adaptação necessária, podendo ajustar automaticamente os parâmetros de reserva de um cliente SRT, baseado em seu uso real do processador;

- Serviço de avaliação (*probing*):

  Representa um desafio para os desenvolvedores de clientes SRT, determinarem os parâmetros mais adequados de reserva. Muitas aplicações multimídia são escritas para independerem de plataforma. Como resultado, não é apropriado avaliar parâmetros específicos de reserva nos programas. Por exemplo, o tempo de processador para decodificar um quadro (*frame*) MPEG difere significativamente entre um Intel Pentium 4 e um SUN Ultra 2. O sistema DSRT fornece um serviço de avaliação inteligente (*smart probing*) que determina automaticamente a classe e os parâmetros mais apropriados, de modo que o processo SRT possa usá-lo para efetuar a reserva.

4.3. Fases de execução

No sistema de DSRT a aplicação percorre três fases diferentes, como se pode verificar na seqüência de execução da Figura 16. A primeira é a fase de avaliação (*probing phase*), em que o sistema efetua algumas iterações na aplicação, sem nenhuma reserva. Paralelamente, o sistema monitora e grava o uso do processador pela aplicação, a cada iteração. No fim da fase de avaliação, o sistema analisa o histórico de uso do processador e configura uma

"reserva", com a classe de serviço e os parâmetros mais apropriados para utilização do processador. Em seguida, o sistema retorna a "reserva" ao usuário da aplicação.

A segunda fase é a fase de reserva (*reservation phase*), em que o usuário submete a reserva ao sistema DSRT. O sistema executa um teste de controle de admissão (*admission control test*) para verificar se existe o volume de processador necessário para satisfazer a reserva. Se a reserva for aceita, transforma-se em um contrato. Em sistemas multiprocessados, o teste de controle de admissão executa ainda um mapeamento dos processadores (*processor binding*), identificando que processador tem a capacidade de escalonar a aplicação.

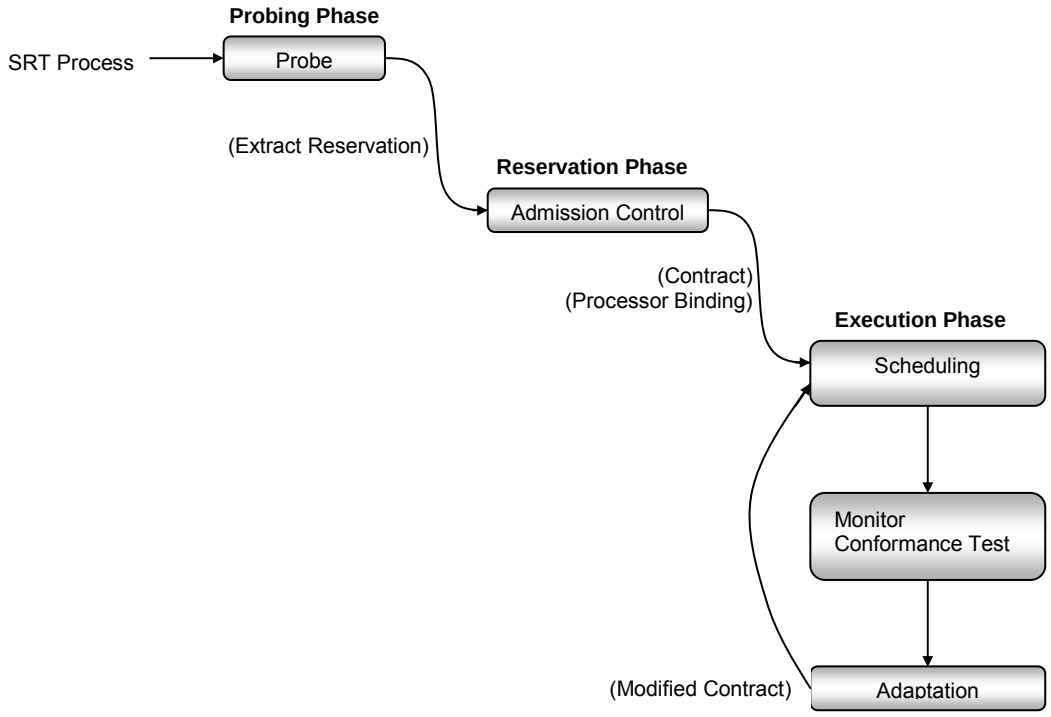


Figura 16 - Seqüência de execução (fases) dos serviços DSRT

A terceira fase é a fase de execução (*execution phase*), em que a aplicação inicia sua execução. O sistema de DSRT escalona a aplicação e executa regularmente um teste de conformidade (*conformance test*) para verificar se o uso real do processador pela aplicação, está dentro do contrato

estabelecido. Se houver uma não conformidade entre o uso e o contrato (sobre-reserva ou sub-reserva), o sistema executa as adaptações necessárias para ajustar os parâmetros no contrato, a fim de solucionar esta não conformidade.

4.4. API DSRT

O sistema DSRT oferece API's para as linguagens de C++ e Java. Estas interfaces tem como finalidade básica iniciar a conexão com o servidor que efetua a gerência do processador (*CPUSvc*). Neste trabalho utilizou-se apenas a interface para a linguagem C++. A Figura 17 apresenta esta interface.

```
// each RT thread has an instance of CCPUApi
class CCPUApi
{
private:
        DWORD           m_dwThreadId;           //thread id of the RT
thread, system uniqe
        HANDLE          m_hYieldSemaAtApi;//handle of yield semaphone
in the context of CPUApi
        HANDLE          m_hContiSemaAtApi;  //wait after yield, continue
only after released by server

private:
        BOOL        sendRecvMsg(CMessage& msg);

public:
        CCPUApi();  //CPUApi and RThread are in same thread
        CCPUApi(DWORD dwPid, DWORD dwThrdId, HANDLE hThrdAtCln);
        virtual ~CCPUApi();

        // Following methods are used to send message to CPU server
        BOOL Probe(LONG lPeriod);    //probe to determine the reservation
parameters
        BOOL ProbeEnd(CCPUReserve& resv); //end probe to get the parameters

        BOOL Reserve(CCPUReserve resv); //reserve CPU with parameters from
resv
        BOOL StartRTRun();  //start to run as a realtime thread
        BOOL StopRTRun();   //stop running as a realtime thread
        BOOL YieldCPU();    //yield CPU to mark the end of current iteration
        BOOL KillRThread(); //exit the RT thread
```

```
        BOOL UpdateReservation(CCPUReserve); //update the reservation
        BOOL FreeReservation();      //finish realtime run and free the reservation

        BOOL KillServer(); //kill cpu server
};
```

Figura 17 - Interface C^{++} do sistema DSRT (arquivo CPUApi.h)

À seguir, são descritas as operações oferecidas por esta interface:

- BOOL Probe(LONG lPeriod); - marca o inicio da fase de avaliação da aplicação. Esta fase é muito útil quando se deseja determinar a classe do serviço e quanto tempo de processador utilizar na reserva. A aplicação não tem que ter qualquer outra reserva válida. Se a aplicação for periódica, o parâmetro lPeriod necessita ser especificado em milissegundos (μs).

- BOOL ProbeEnd(CCPUReserve& resv); - marca o fim da fase de avaliação da aplicação processo.

- BOOL Reserve(CCPUReserve resv); - efetua a reserva especificada em resv para a aplicação. A execução é iniciada por StartRTRun.

- BOOL StartRTRun(); - inicia o escalonamento para a aplicação. Um Reserve tem que ter sido efetuado antecedendo a esta operação.

- BOOL StopRTRun(); - pára temporariamente o escalonamento SRT para a aplicação. Neste intervalo de parada, a aplicação torna-se um processo com prioridade de escalonamento “melhor esforço”. Entretanto, sua reserva permanece válida. Um StartRTRun tem que ter sido efetuado antecedendo a esta operação A aplicação pode ser re-iniciada quando executar a operação StartRTRun novamente.

- BOOL YieldCPU(); - marca o fim de uma iteração de execução (período) para a aplicação. Aplica-se tanto para reservas periódicas quanto aperiódicas. É necessário que a aplicação tenha sido escalonada anteriormente através da operação StartRTRun. A aplicação é bloqueada até o início do próximo período.

- BOOL KillRThread(); - encerra a *thread* SRT.

- BOOL UpdateReservation(CCPUReserve); - modifica a reserva corrente especificada em CCPUReserve para a aplicação.

- BOOL FreeReservation(); - encerra o escalonamento SRT e libera a reserva corrente da aplicação.

- BOOL KillServer(); - encerra o servidor que efetua a gerência do processador (*CPUSvc*).

4.5. Limitações do sistema DSRT

Há diversas características nos sistemas operacionais e no *hardware* de propósito geral que podem potencialmente provocar violações às garantias que o sistema DSRT faz aos clientes SRT, mesmo se estes clientes passarem no teste de controle de admissão. A interrupção de *hardware* é uma dessas limitações.

A prioridade no atendimento de uma interrupção de *hardware* é mais alta do que a prioridade que o servidor DSRT pode tratar em nível de usuário (*user-level*). Se uma interrupção de *hardware* ocorrer no mesmo tempo que o servidor DSRT está executando o algoritmo de escalonamento, o servidor DSRT irá temporizar, aumentando adversamente a latência de liberação e a precisão do despacho. Se uma interrupção de *hardware* ocorrer ao mesmo tempo em que um processo despachado está executando, o processo despachado irá temporizar, retardando adversamente o tempo de processamento deste processo.

Resolver os problemas de interrupção de *hardware* requer mudanças não triviais na estrutura do sistema operacional. Ao invés de atender as interrupções sempre que ocorrem, o sistema operacional poderia periodicamente apurá-las. Adicionalmente, o sistema operacional necessitaria distinguir as interrupções geradas por aplicações TS, que seriam atendidas somente quando houvesse um tempo livre, das interrupções geradas por processos de clientes SRT, que seriam atendidas imediatamente.