

SMPL – Ferramenta de Simulação



M.H. MacDougall, *Simulating
Computer Systems Techniques and
Tools*, The MIT Press, 1987

SMPL



- ⌘ É uma extensão funcional de uma linguagem de programação de propósito geral (no caso, C).
 - ☒ Biblioteca de funções
- ⌘ Estas extensões tomam a forma de um conjunto de funções de biblioteca: o subsistema de simulação **SMPL**.
- ⌘ Este subsistema e a linguagem hospedeira compõem uma linguagem de simulação orientada a eventos.

SMPL

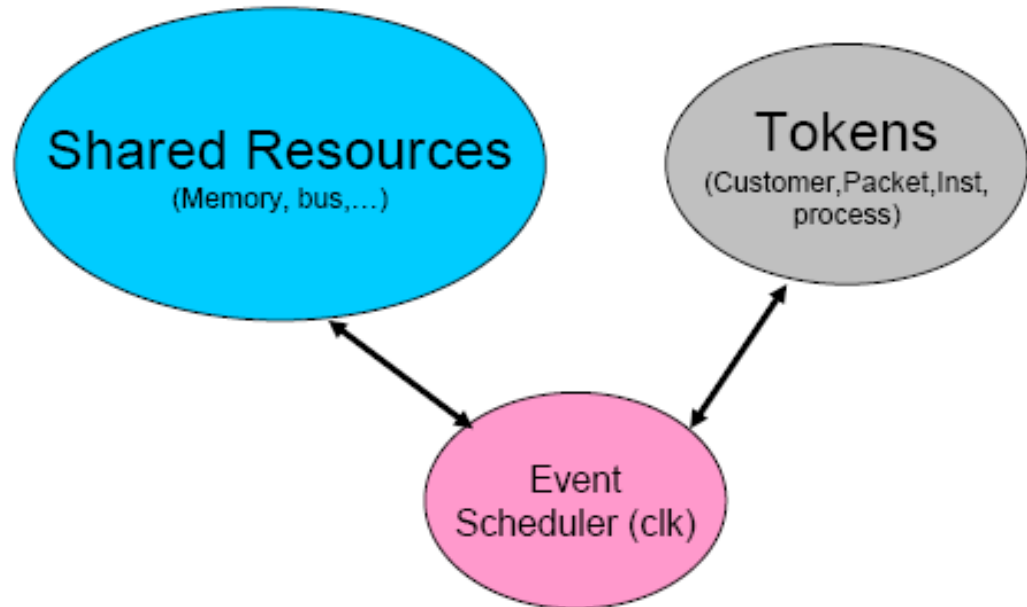


- ⌘ Um modelo de simulação é implementado como um programa da linguagem hospedeira: as operações de simulação são executadas através de chamadas das funções do subsistema de simulação.
- ⌘ Adequado para modelos de simulação pequenos ou médios.

A visão dos sistemas da simpl

⌘ Entidades:

- ☑ Recursos,
- ☑ fichas e
- ☑ eventos.



Recursos (*facilities*)

- ⌘ Um sistema é composto por uma coleção interconectada de recursos.
- ⌘ Os recursos podem ser, por exemplo: uma CPU; um barramento; chave de proteção num sistema operacional; caixa de banco; balcão de atendimento.
 - ☒ Prestam um “serviço”
- ⌘ Funções relacionadas a recursos: definição, reserva, liberação, preempção e status.
- ⌘ A interconexão entre recursos não é explícita, é determinada pelo roteamento (feito pelo modelo) das fichas entre os recursos.

Fichas (*tokens*)

- ⌘ Representam as entidades ativas do sistema; o comportamento dinâmico do sistema é modelado pelo movimento das fichas entre um conjunto de recursos.
- ⌘ Uma ficha pode representar, por exemplo:
 - ☒ uma tarefa
 - ☒ um pacote
 - ☒ um acesso à memória
 - ☒ um cliente

Fichas



- ⌘ Uma ficha pode reservar recursos e pode programar atividades com diversas durações. Se ela tentar reservar um recurso que já se encontra ocupado, ficará enfileirada até que o recurso fique disponível para ela.
- ⌘ O que a ficha representa depende do modelador; para o **SMPL**, ela é apenas um inteiro. O único atributo conhecido pelo subsistema é a sua prioridade.
- ⌘ Nenhuma decisão é tomada pelo **SMPL** apenas pelo número da ficha.

Eventos



- ⌘ Mudança de estado em qualquer entidade do sistema.
- ⌘ Contém funções para programar a ocorrência de eventos e selecionar (causar) a sua execução.
- ⌘ Do ponto de vista do subsistema de simulação, um evento é identificado pelo seu número, o instante de tempo em que deve ocorrer e possivelmente a identidade da ficha envolvida com o evento.

Funções do *smpl*

⌘ Inicialização do modelo:

```
smpl(m, s);
```

```
int m; char *s;
```

Na implementação disponível, *m* deve ser sempre 0.

s é um ponteiro para o nome do modelo.

⌘ reset();

⊞ Inicializa todas as medidas acumuladas pelo **SMPL**, mas não o relógio de simulação

⌘ Usos:

⊞ Fazer múltiplas simulações dentro de um mesmo programa, mudando os parâmetros de entrada → `smpl(m, s)`

⊞ Fazer múltiplas simulações dentro de um mesmo programa, mudando a semente do gerador de números aleatórios → `smpl(m, s)`

⊞ Evitar o período de “aquecimento” (*warm-up*) ou transitório no início da simulação, quando as filas estão ainda vazias → `reset()`

Funções do *smpi*

⌘ Criação de um recurso:

```
f=facility(s,n);
```

```
char *s; int n, f;
```

s é o nome do recurso

f é um descritor (inteiro) do recurso utilizado em outras operações.

n é o número de servidores do recurso.

Funções do *smpl*

⌘ Pedido de reserva de um recurso:

```
r=request(f,tkn,pri);  
int f,tkn,pri,r;
```

- ☒ Quanto maior o valor de `pri`, maior a prioridade.
- ☒ Se houver um servidor livre, esta função retorna o valor 0.
- ☒ Cada recurso possui uma fila ordenada de acordo com as prioridades e ordem de chegada. Às vezes, as prioridades são utilizadas para implementar políticas diferentes para a fila.
- ☒ Se o recurso estiver ocupado, a ficha é enfileirada; ao liberar o recurso, o **SMPL** gera um novo evento "request" para esta ficha.
- ☒ Um pedido enfileirado devido a recurso ocupado é chamado *pedido bloqueado (blocked request)*.
- ☒ Observe que o tempo de serviço não aparece aqui! Ele é definido pelo escalonamento ou agendamento do evento "liberação do servidor".

Funções do *smpi*

⌘ Pedido de reserva de um recurso com preempção:

```
r=preempt(f,tkn,pri);  
int f,tkn,pri;
```

- ☑ O atendimento de um usuário com menor prioridade pode ser interrompido pelo pedido de reserva de um usuário com maior prioridade.
- ☑ O usuário (ficha) interrompido entra na fila com uma indicação do tempo restante de atendimento necessário. Esta ficha é chamada de *suspensa* (*suspended or preempted*).

Funções do *smpi*

⌘ Liberação do servidor de um recurso:

```
release(f, tkn);
```

```
int f, tkn;
```

⌘ Servidor dentro do recurso agora está livre.

⌘ Se houver fila para o recurso, o primeiro elemento desta fila é desenfileirado e um evento "request" é feito para ele pelo **SMPL**.

Funções do *smpi*

⌘ Funções de obtenção de *status* e medidas de desempenho:

`n=inq(f)`

⏏ retorna o número de fichas que se encontram na fila de um recurso (sem incluir as que se encontram em serviço).

`r=status(f)`

⏏ retorna 1 se o recurso estiver ocupado e 0 se pelo menos um servidor estiver livre.

Funções do *smpi*

⌘ Funções de obtenção de status e medidas de desempenho (cont.):

```
u=real U(f)
```

```
b=real B(f)
```

```
l=real Lq(f)
```

```
int f;
```

⏏ retornam, respectivamente, a utilização média, o período médio ocupado e o comprimento médio da fila.

⏏ `real` é tipicamente o tipo `double`

⏏ `typedef double real`

Funções do *smpi*

⌘ Escalonamento ou agendamento de eventos:

```
schedule(ev, te, tkn);
```

```
int ev, tkn; real te;
```

- ⏏ `te` é o intervalo de tempo a partir do instante atual, para o qual deve ser programado o evento.

⌘ Recuperação do próximo evento da lista:

```
cause(ev, tkn); //endereços de ev e tkn
```

```
int *ev, *tkn;
```

- ⏏ Retira o próximo evento da cadeia de eventos; coloca o número (tipo) do evento em `ev` e o número da ficha em `tkn`.
- ⏏ O tempo de simulação é avançado automaticamente para o instante de ocorrência do evento recuperado.

Funções do *smpi*

⌘ Cancelamento de eventos:

```
tkn=cancel(ev);  
int tkn;
```

⊞ Esta função retorna o valor da ficha associada ao evento cancelado. Se o evento não for encontrado, ela retorna -1. Se houver múltiplas ocorrências do mesmo evento, apenas o próximo será cancelado.

⌘ Tempo atual de simulação:

```
t=real stime();
```

⊞ A unidade de tempo é estabelecida implicitamente pelos valores utilizados na simulação.

Funções do *smpl*

⌘ Geração de valores aleatórios

```
r=real ranf( );
```

☑ Retorna um valor pseudo-aleatório distribuído uniformemente entre 0 e 1.

☑ É a única função da **SMPL** que é dependente da máquina.

⌘ Seleção de seqüências aleatórias:

```
i=stream(n)
```

```
int n;
```

☑ O **SMPL** fornece 15 seqüências aleatórias, que na implementação do livro corresponde a uma distância de 100.000 amostras.

Funções do *smpi*

⌘ Geração de valores aleatórios

```
r=real expntl(x);
```

```
r=real erlang(x,s);
```

```
r=real hyperx(x,s);
```

```
r=real normal(x,s);
```

```
real x,s;
```

⌘ Distribuições uniformes:

```
r=real uniform(a,b);
```

```
real a,b;
```

```
k=random(i,j);
```

```
int i,j;
```

Recomendações para a simpl



- ⌘ Não utilize valores numéricos no código, use os recursos de macros (`#define`) em C.
- ⌘ Todas as definições de recursos devem ser feitas antes de qualquer outra operação com recursos ou operações com eventos; caso contrário, ocorrerá um erro.
- ⌘ Separe o evento de chegada do pedido de servidor, dado que a rotina (evento) correspondente ao pedido do servidor será reexecutada após a liberação do servidor.

Recomendações para a simul



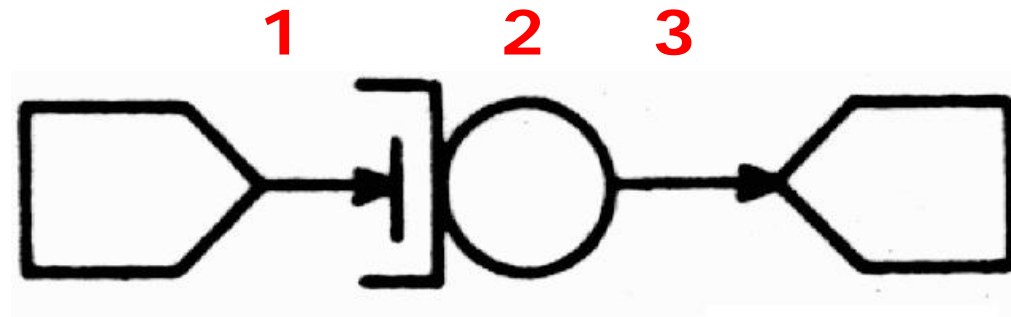
- ⌘ O programa de simulação não deve transferir o controle diretamente entre rotinas de tratamento de eventos, mesmo se os eventos devem ser executados no mesmo instante de tempo. O controle deve sempre ser transferido através de chamadas de `schedule()`.
- ⌘ Fichas que utilizam recursos que podem ser "preemptados" devem possuir apenas um evento agendado por vez para elas.

Recomendações para a simpl



- ⌘ O relatório deve não só apresentar os resultados, mas também os parâmetros de entrada.
- ⌘ Deve-se ter cuidado especial com a modelagem de estruturas de dados: freqüentemente ocorrem erros na alocação e liberação dinâmica de atributos das fichas e, em particular, no gerenciamento de índices e ponteiros.

Simulação de uma fila M/M/1



Eventos:

- 1: Chegada de cliente
- 2: *Request* servidor
- 3: *Release* servidor

Simulação de uma fila M/M/1

- ⌘ Evento 1 deve chamar evento 2 e gerar novo evento 1
- ⌘ Evento 2 deve chamar evento 3 (se servidor livre)
- ⌘ Evento 3 é o último, não gera novos eventos
- ⌘ As estatísticas de interesse devem ser atualizadas nos eventos adequados

/ Simulação de uma fila M/M/1*

2004 by Marcos Portnoi

**/*

#include <stdio.h>

#include "simpl.h"

#include <stdlib.h>

void main()

{

real Ta=0.4,Ts=0.1,te=200000.0;

int customer=1,event,server;

int n_tot_cheg=0, tam_max_fila=0, num_job_serv=0, n_jobs_gerados=0,
maxjobs=600000.0;

float tx_cheg;

```

smp(0,"M/M/1 Queue");
server=facility("server",1);
schedule(1,0.0,customer);
while (stime()<te && n_jobs_gerados < maxjobs)
{
  cause(&event,&customer);
  switch(event)
  {
    case 1: /* arrival */
      schedule(2,0.0,customer);
      n_tot_cheg++; //incrementa acumulador de jobs que chegaram para a fila de destino
      n_jobs_gerados++; //incrementa acumulador de jobs gerados
      schedule(1,expntl(Ta),customer);
      break;
    case 2: /* request server */
      if (request(server,customer,0)==0)
        schedule(3,expntl(Ts),customer);
      else
      {
        if (inq(server) > tam_max_fila) tam_max_fila = inq(server); //atualiza tamanho máximo
        de fila
      }
      break;
  }
}

```

case 3: */* release server */*

num_job_serv++; *//incrementa acumulador de jobs servidos*

release(server,customer);

break;

}

}

report();

printf("Fila M/M/1");

printf("\n\n\nTempo Simulado: %.2f\nNum. Clientes Gerados: %d", stime(),
n_jobs_gerados);

printf("\n\n *** FONTES ***\n\n");

printf("\nFONTE - NUM.CLIENTE.GER. - TAXA DE GERACAO");

printf("\n [clientes] [uts/cliente]");

printf("\n-----");

printf("\n 1 %13d %8.4f", n_jobs_gerados, Ta);

printf("\n\n\n *** SERVIDORES ***\n\n");

printf("\nSERV. - UTILIZ. - CLI.SERV. - TX SERVICO - TEMPO SERVICO");

printf("\n [clientes] [cliente/uts] [uts/cliente]");

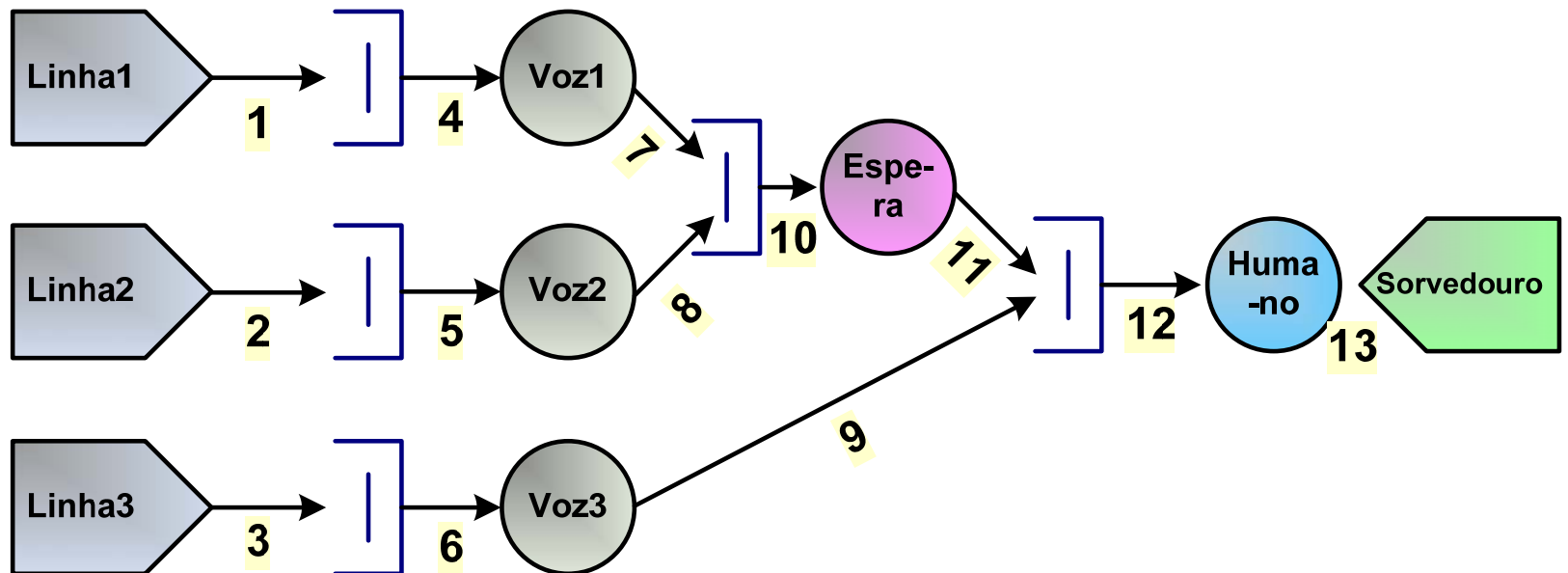
printf("\n-----");

printf("\n 1 %11.4f %10d %12.4f %8.4f", U(server), num_job_serv,
num_job_serv/(stime()*U(server)), Ts);

```
printf("\n\n\n *** FILAS ***\n\n");
printf("\nFILA - TX.CHG. - TAM.MED.FIL - TEM.MED.FIL - TAM.MAX");
printf("\n [cli/uts] [clientes] [uts] [clientes]");
printf("\n-----");
tx_cheg=n_tot_cheg/stime();
printf("\n 1 %10.4f %10.4f %10.4f %10d", tx_cheg, Lq(server), Lq(server)/tx_cheg,
tam_max_fila);
```

```
printf("\n\nTaxa de Chegada: %f\n", tx_cheg);
printf("Tamanho medio de fila: %f\n", Lq(server));
printf("Tempo medio em fila: %f\n", Lq(server)/tx_cheg);
printf("Tamanho Maximo de Fila: %d\n", tam_max_fila);
printf("Utilizacao do servidor: %f\n", U(server));
printf("Taxa de Servico: %f\n", num_job_serv/(stime()*U(server)));
}
```

Outro Exemplo: *call center*



Depuração de Programas de Simulação

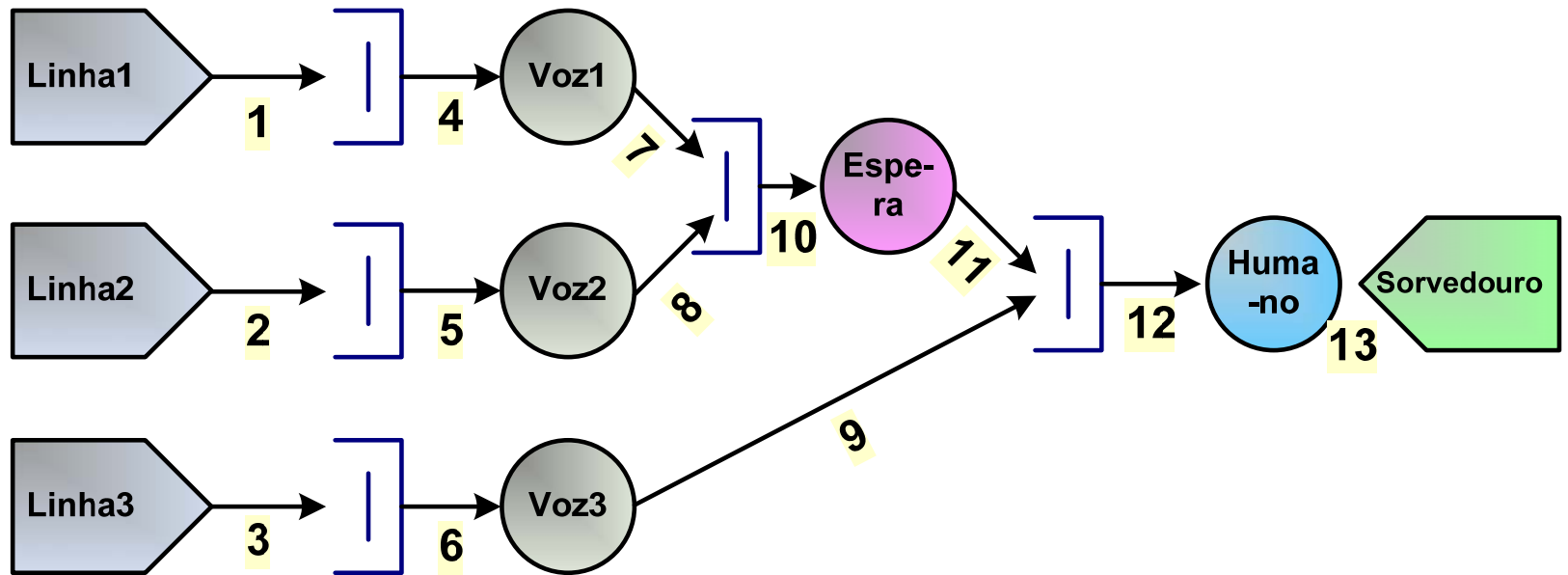
- ⌘ Colocar o SMPL para rodar.
- ⌘ Sugestão: testar antes o funcionamento de um dos exemplos fornecidos (mm1.c).
- ⌘ Problemas comuns de compilação:
 - ⊞ Uso de nomes reservados de rotinas:
 - ⊞ random() → irandom();
 - ⊞ time() → stime()
- ⌘ Problemas comuns de execução:
 - ⊞ Erro: “Empty Element Pool”
 - ⊞ Arquivo smpl.c: #define nl 256 /* element pool length */
 - ⊞ Verifique se seu modelo está em *equilíbrio*
 - ⊞ Ponteiros!

Depuração de Programas de Simulação

⌘ Problemas com o modelo

- ☒ Encadeamento incorreto dos eventos
- ☒ Verifique se todos os eventos estão encadeados corretamente
- ☒ O tratamento de um evento deve gerar outro evento, de forma que o *token* caminhe pelo sistema
- ☒ Número incorreto de servidores em um recurso (*facility*)

Encadeamento de Eventos



Depuração de Programas de Simulação

- ⌘ Na primeira compilação com sucesso do programa, tente executá-lo.
- ⌘ Faça com que o programa de simulação execute apenas uma ficha, ligue o depurador, e acompanhe a execução desta ficha. Reveja os dados do trace e verifique se o programa executou corretamente.

Depuração de Programas de Simulação

⌘ Modifique o programa para que execute duas fichas seqüencialmente (de modo que a execução da segunda ficha só inicie após o término da execução da primeira) e acompanhe a execução delas. O objetivo é o de detectar problemas deixados pela primeira ficha, tais como recursos que não foram liberados, elementos de dados que não foram desalocados, etc. A impressão de valores de ponteiros e de índices podem ajudar a identificar estes problemas.

Depuração de Programas de Simulação

- ⌘ Revise mais uma vez o programa para que execute duas fichas concorrentemente, acompanhe a execução das fichas, e observe problemas na interação das duas fichas. Pode ser necessário iniciar ao mesmo tempo a execução das duas fichas para garantir a ocorrência de enfileiramento, preempção e outros conflitos de recursos.

Depuração de Programas de Simulação

⌘ Continue neste processo de execução controlada se o modelo for complexo. Por exemplo, o programa pode ser modificado para dirigir as fichas para caminhos específicos de execução, ou para examinar condições limites tais como o bloqueio de nós de redes de comunicação.

Depuração de Programas de Simulação

- ⌘ Quando a execução seletiva não produzir mais erros, tente a execução completa de um número modesto de fichas. Observe o trace para verificar se as filas crescem porque os recursos não estão sendo liberados. Se for identificado um erro, faça com que o trace seja ligado antes da ocorrência do erro.

Depuração de Programas de Simulação

- ⌘ Os erros encontrados neste estágio são freqüentemente causados por problemas de "fim de jogo": condições atingidas pela primeira vez na execução do programa, tais como a alocação do último elemento numa lista de elementos livres.

Depuração de Programas de Simulação

⌘ Quando o modelo executar até o final sem erros, verifique o relatório de simulação. Observe as utilizações, comprimentos das filas e distribuição dos pedidos entre os recursos para ver se os valores fazem sentido intuitivamente e se eles são consistentes no sentido de uma análise operacional. Se os resultados estiverem operacionalmente corretos mas forem contra-intuitivos, descubra o porquê : pode haver um erro no cálculo do tempo de serviço ou uma situação de atraso imprevisto (mesmo se válida).