Ferramentas de Simulação SMPL e TARVOS

SMPL e TARVOS

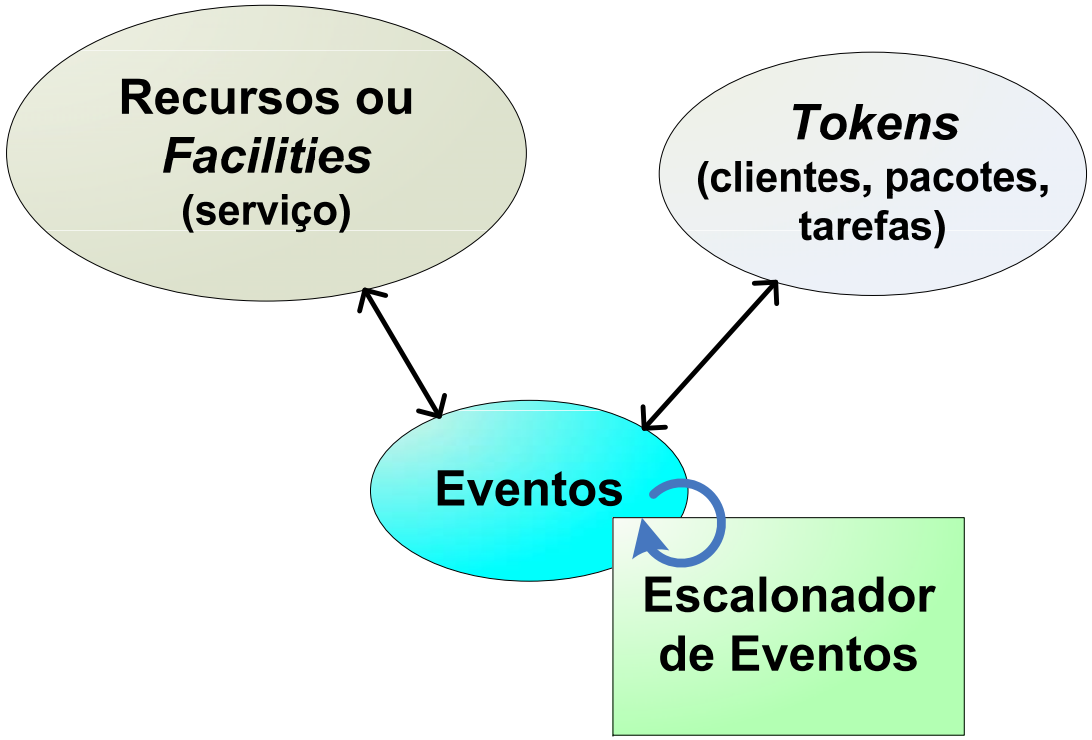
- São extensões funcionais de uma linguagem de programação de propósito geral (no caso, C).
  - Biblioteca de funções
- Estas extensões tomam a forma de um conjunto de funções de biblioteca: um subsistema.
- Este subsistema e a linguagem hospedeira compõem uma linguagem de simulação orientada a eventos.
- O código do SMPL está inteiramente descrito em "M.H. MacDougall, Simulating Computer Systems Techniques and Tools, The MIT Press, 1987"

SMPL e TARVOS

- 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.
- SMPL: adequado para modelos de simulação pequenos ou médios.
  - Arrays (alocação estática)
  - Gerador de números aleatórios com ciclo pequeno
- TARVOS: mais escalável
  - Structs
  - Alocação dinâmica

A visão dos sistemas do SMPL e TARVOS

- **Entidades:**
  - Recursos,
  - *tokens* 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) dos *tokens* entre os recursos.

Tokens (Fichas)

- Representam as entidades ativas do sistema; o comportamento dinâmico do sistema é modelado pelo movimento dos *tokens* entre um conjunto de recursos.
- Um *token* pode representar, por exemplo:
  - uma tarefa
  - um pacote
  - um acesso à memória
  - um cliente

Tokens

- Um *token* pode reservar recursos e pode programar atividades com diversas durações. Se ele tentar reservar um recurso que já se encontra ocupado, ficará enfileirado até que o recurso fique disponível para ele.
- O que o *token* representa depende do modelador; para o **SMPL**, ele é apenas um inteiro. Para o **TARVOS,** pode ser ainda uma estrutura de dados. **O** único atributo imediatamente importante do *token*, para o subsistema, é a sua prioridade.
- Nenhuma decisão é tomada pelos simuladores apenas pelo número do *token*.

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 do *token* envolvido com o evento.

Funções: SMPL

- Inicialização do modelo:

```
void smpl(int m, char *s);
```

  Na implementação disponível, `m` deve ser sempre `0`.
  `s` é um ponteiro para o nome do modelo.

- Usos:
  - Fazer múltiplas simulações dentro de um mesmo programa, mudando os parâmetros de entrada
  - Fazer múltiplas simulações dentro de um mesmo programa, mudando a semente do gerador de números aleatórios. A cada vez que a função é chamada, a semente do gerador de números aleatórios é incrementada.
- `reset();`
  - Inicializa todas as medidas acumuladas pelo **SMPL**, mas não o relógio de simulação
- Usos:
  - Evitar o período de "aquecimento" (*warm-up*) ou transitório no início da simulação, quando as filas estão ainda vazias

Funções: TARVOS

- Inicialização do modelo:

```
void tarvos(int m,char *s);
Ou
void simm(int m,char *s);
```

  Na implementação disponível, `m` deve ser sempre `0`.
  `s` é um ponteiro para o nome do modelo.

- Usos:
  - Fazer múltiplas simulações dentro de um mesmo programa, mudando os parâmetros de entrada
  - Fazer múltiplas simulações dentro de um mesmo programa, mudando a semente do gerador de números aleatórios. A cada vez que a função é chamada, a semente do gerador de números aleatórios é incrementada
- `reset();`
  - Inicializa todas as medidas acumuladas pelo **TARVOS**, mas não o relógio de simulação
- Usos:
  - Evitar o período de "aquecimento" (*warm-up*) ou transitório no início da simulação, quando as filas estão ainda vazias

Funções: SMPL

- Criação de um recurso:

```
int facility(char *s, int n);
```

`s` é o nome do recurso

O retorno da função é um descritor (inteiro) do recurso utilizado em outras operações.

`n` é o número de servidores do recurso.

Funções: TARVOS

- Criação de um recurso:

```
int facility(char *s, int n);
```

`s` é o nome do recurso

O retorno da função é um descritor (inteiro) do recurso utilizado em outras operações.

`n` é o número de servidores do recurso.

Funções: SMPL

- Pedido de reserva de um recurso:

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

f:  número da facility
tkn:  número do token
pri:  prioridade
```

  - Quanto maior o valor de `pri`, maior a prioridade.
  - Se houver um servidor livre, esta função retorna o valor `0;` o token foi colocado em serviço. Se todos os servidores estiverem ocupados, retorna 1; token enfileirado.
  - 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, o *token* é enfileirado; ao liberar o recurso, o **SMPL** gera um novo evento "request" para este *token*.
  - 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: TARVOS

- Pedido de reserva de um recurso:

```
int requestp (int f, int tkn, int pri, int ev, double
  te, TOKEN *tkp);

f:  número da facility
tkn:  número do token
pri:  prioridade
ev:  próximo evento chamado após serviço completado
te:  tempo de serviço
*tkp:  ponteiro para a estrutura do token
```

  - Quanto maior o valor de `pri`, maior a prioridade.
  - Se houver um servidor livre, esta função retorna o valor `0;` o token foi colocado em serviço. Se todos os servidores estiverem ocupados, retorna 1; token enfileirado. Se o recurso estiver *down* (não operacional), a função retorna 2; nada foi feito com o token.
  - 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, o *token* é enfileirado.
  - Ao liberar o recurso, o **TARVOS** coleta o primeiro *token* em fila e coloca-o imediatamente em serviço, escalonando o evento "releasep" para o *token*. A filosofia é diferente da do SMPL. Por isso, o **TARVOS** precisa do tempo de serviço e número do evento tipo "release" a ser chamado para liberação como parâmetros para a função *requestp*.

Funções: SMPL

- Pedido de reserva de um recurso com preempção:

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

f:  número da facility
tkn:  número do token
pri:  prioridade
```

  - O atendimento de um usuário com menor prioridade pode ser interrompido pelo pedido de reserva de um usuário com maior prioridade.
  - O usuário *(token)* interrompido entra na fila com uma indicação do tempo restante de atendimento necessário. Este *token* é chamado de *suspenso (suspended or preempted)*.
  - Se houver um servidor livre ou a preempção for bem sucedida, esta função retorna o valor `0;` o token foi colocado em serviço. Se todos os servidores estiverem ocupados com *tokens* de maior prioridade que o *token* em pedido, retorna 1; token enfileirado.

Funções: TARVOS

- Pedido de reserva de um recurso com preempção:

```
int preemptp (int f, int tkn, int pri, int ev, double
  te, TOKEN *tkp)

f:  número da facility
tkn:  número do token
pri:  prioridade
ev:  próximo evento a ser chamado após serviço completado
te:  tempo de serviço
*tkp:  ponteiro para estrutura do token
```

  - O atendimento de um usuário com menor prioridade pode ser interrompido pelo pedido de reserva de um usuário com maior prioridade.
  - O usuário *(token)* interrompido entra na fila com uma indicação do tempo restante de atendimento necessário. Este *token* é chamado de *suspenso (suspended or preempted)*.
  - Se houver um servidor livre ou a preempção for bem sucedida, esta função retorna o valor `0;` o token foi colocado em serviço. Se todos os servidores estiverem ocupados com *tokens* de maior prioridade que o *token* em pedido, retorna 1; token enfileirado. Se o recurso estiver *down* (não operacional), a função retorna 2; nada foi feito com o token.

Funções: SMPL

- Liberação do servidor de um recurso:

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

f:  número da facility
tkn:  número do token
```

- 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: TARVOS

- Liberação do servidor de um recurso:

```
void releasep(int f,int tkn);

f:  número da facility
tkn:  número do token
```

- Servidor dentro do recurso agora está livre.
- Se houver fila para o recurso, o primeiro elemento desta fila é desenfileirado e imediatamente colocado em serviço, com as informações de tempo armazenadas.
- um evento "releasep" é escalonado para o *token* agora colocado em serviço, usando o número de evento armazenado (quando da chamada da função *requestp*).

Funções: SMPL e TARVOS

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

  `int inq(int f)`

  - retorna o número de *tokens* que se encontram na fila de um recurso (sem incluir os que se encontram em serviço).

  `int status(int f)`

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

- Somente no TARVOS:

  `int getFacUpStatus(int f)`

  - Retorna 1 se o recurso estiver operacional (*up*) e 0 se estiver não-operacional (*down*).

Funções: SMPL e TARVOS

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

```
double U(int f)
double B(int f)
double Lq(int f)
f:  número da facility
```

- retornam, respectivamente, a utilização média, o período médio ocupado e o comprimento médio da fila.
- Somente no TARVOS:

```
int getFacMaxQueueSize(int f)
```

  - Retorna o tamanho máximo de fila da *facility* indicada, medido durante a simulação.
  - No SMPL, isso precisa ser obtido através de algoritmo do usuário.

Funções: SMPL

- Escalonamento ou agendamento de eventos:

  ```
  void schedule(int ev, double te, int tkn);
  ```

  - `ev` é o número do evento a escalonar.
  - `te` é o intervalo de tempo a partir do instante atual, para o qual deve ser programado o evento.
  - `tkn` é o número do *token*.
- Recuperação do próximo evento da lista:

  ```
  void cause(int *ev, int *tkn);
  ```

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

Funções: TARVOS

- Escalonamento ou agendamento de eventos:

```
void schedulep(int ev, double te, int tkn, TOKEN
  *tkp);
```

  - `ev` é o número do evento a escalonar.
  - `te` é o intervalo de tempo a partir do instante atual, para o qual deve ser programado o evento.
  - `tkn` é o número do *token*
  - `*tkp` é o ponteiro para a estrutura do *token*, definida pelo usuário, ou NULL se não houver.
- Recuperação do próximo evento da lista:

```
TOKEN *causep(int *ev, int *tkn);
```

  - Retira o próximo evento da cadeia de eventos; coloca o número (tipo) do evento em ev e o número do *token* em tkn; retorna a estrutura de dados correspondente ao *token*, se houver, e NULL, se não houver.
  - O tempo de simulação é avançado automaticamente para o instante de ocorrência do evento recuperado.

Funções: SMPL

- Cancelamento de eventos:

  ```
  int cancel(int ev);
  ```

  - Esta função retorna o valor do *token* 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:

  ```
  double stime();
  ```

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

Funções: TARVOS

- Cancelamento de eventos:

  `struct evchain *cancelp_ev(int ev);`

  - Esta função elimina da cadeia de eventos o primeiro evento identificado por `ev` e retorna um ponteiro para o elemento da cadeia de eventos que foi retirado ou NULL, se nenhum evento foi encontrado. Se houver múltiplas ocorrências do mesmo evento, apenas o próximo será cancelado.

  `struct evchain *cancelp_tkn(int tkn);`

  - Esta função elimina da cadeia de eventos o primeiro evento relacionado ao *token* passado como parâmetro, e retorna um ponteiro para o elemento da cadeia de eventos que foi retirado ou NULL, se nenhum evento foi encontrado. Se houver múltiplas ocorrências do mesmo *token*, apenas o próximo será cancelado.

- Tempo atual de simulação:

  `double simtime();`

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

Funções do SMPL e TARVOS

- Geração de valores aleatórios

```
double ranf();
```

  - Retorna um valor pseudo-aleatório distribuído uniformemente entre 0 e 1.
  - O **SMPL** usa um gerador próprio de números aleatórios. O **TARVOS** usa o gerador nativo do compilador C.

```
int stream(int n);
```

  - Para o SMPL: seleção de seqüências aleatórias:
    - O *n* é o número da sequência do gerador. O **SMPL** fornece 15 seqüências (de 1 a 15) aleatórias, que na implementação do livro corresponde a um ciclo de 100.000 amostras. Se *n* for 0, a função apenas retorna o número da seqüência atual.
  - Para o TARVOS:
    - O *n* é o número da semente para o gerador nativo de números aleatórios (srand). A função retorna sempre o valor da semente atual.

Funções do SMPL e TARVOS

- Geração de valores aleatórios:

```
double expntl(double x);
double erlang(double x, double s);
double hyperx(double x, double s);
double normal(double x, double s);
```

- Somente TARVOS:

```
double pareto(double a, double k);
```

- Distribuições uniformes:

```
double uniform(double a, double b);
int irandom(int i, int j);
```

Recomendações

- 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.

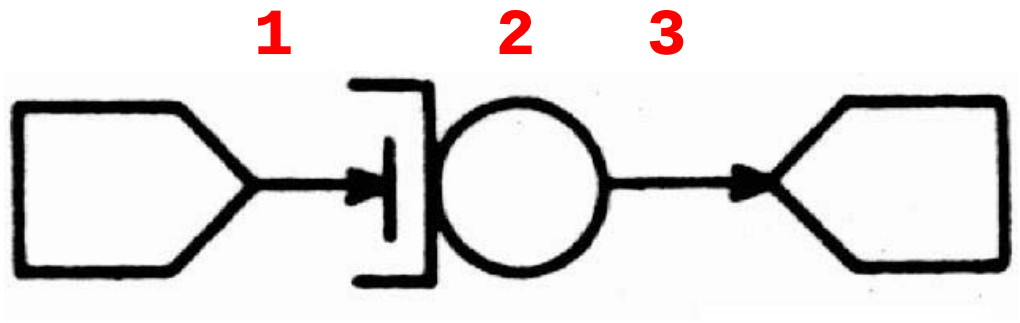
Recomendações

- 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()/schedulep()`.
- *Tokens* que utilizam recursos que podem ser "preemptados" devem possuir apenas um evento agendado por vez para elas.

Recomendações

- 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 dos *tokens* e, em particular, no gerenciamento de índices e ponteiros.

Simulação de uma fila M/M/1 com SMPL



Eventos:

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

Simulação de uma fila M/M/1 com SMPL

- 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

mm1.pdf

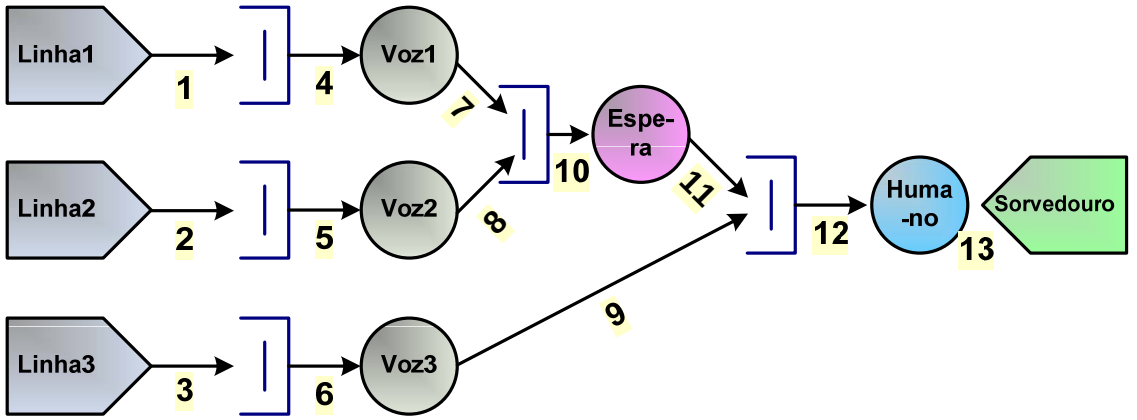
```
/*  Simulação de uma fila M/M/1

    2004 by Marcos Portnoi
 */

 #include <stdio.h>
 #include "smpl.h"
 #include <stdlib.h>

 void main()
 {
    double 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;
```

```
smpl(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* com SMPL



call center.pdf

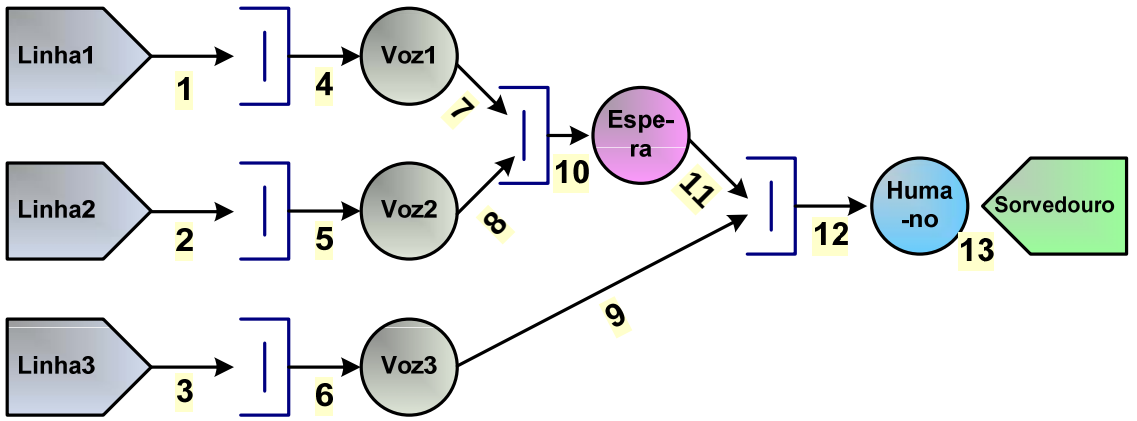
Depuração de Programas de Simulação

- 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 um token, ligue o depurador, e acompanhe a execução deste token. Reveja os dados do trace e verifique se o programa executou corretamente.

Depuração de Programas de Simulação

- Modifique o programa para que execute dois tokens seqüencialmente (de modo que a execução do segundo token só inicie após o término da execução do primeiro) e acompanhe a execução deles. O objetivo é o de detectar problemas deixados pelo primeiro token, 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 dois tokens concorrentemente, acompanhe a execução dos tokens, e observe problemas na interação dos dois tokens. Pode ser necessário iniciar ao mesmo tempo a execução dos dois tokens 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 os tokens 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 tokens. 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).