```c
/*  Simulação de um Sistema de Call Center
    (para disciplina de Ferramentas de Análise de Desempenho de Sistemas,
    Mestrado em Redes de Computadores, Unifacs 2004)

    2004 by Marcos Portnoi
    Modificações para uso do kernel SM, 2006 Marcos Portnoi
*/

#define MAIN_MODULE //define as variáveis e estruturas, e não apenas declara como extern

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

#define N_sources 3 //número de fontes no modelo
#define N_facilities 5 //número de facilities no modelo
#define Name_facilities {"Voz1", "Voz2", "Voz3", "Espera", "Humano"}
#define Tau {40, 40, 40} //tempos de interchegada
#define S {25, 25, 25, 19, 10} //tempos de serviço atendimento eletrônico de voz, espera forçada e atendente humano
#define Max_time 2600000 //tempo máximo de simulação
#define Max_jobs 1750000 //número máximo de jobs ou clientes a gerar
#define SEED 5 //especifica o stream do gerador de números aleatórios do SMPL a usar (1 a 15)

int main()
{
    /*job é o número do token ou cliente; não há diferenciação entre eles
      num_cheg_fila[] é um contador para o número de chegadas por fila/facility
      tam_max_fila[] é um acumulador para armazenar o tamanho máximo medido por fila/facility
      num_job_serv[] é um contador para o número de jobs servidos por facility
      num_jobs_gerados[] é um contador de jobs gerados por fonte de geração
      name_facility contém o nome de cada facility
      tot_jobs_gerados é o acumulador global de jobs gerados por todas as fontes
      tx_cheg_fila[] representa o cálculo da taxa de chegada por fila
    */
    int job=1, event, server[N_facilities];
    int i, num_cheg_fila[N_facilities], tam_max_fila[N_facilities], num_job_serv[N_facilities];
    int num_jobs_gerados[N_sources], tot_jobs_gerados=0;
    char name_facility[][10]=Name_facilities, file_out[35]; //file_out -> nome de saída do arquivo de resultados
    float tx_cheg_fila[N_facilities], tau_source[]=Tau, s_server[]=S;
    float t_espera_linha1, t_espera_linha2, t_espera_linha3;
    double t;
    FILE *fp;

    //inicialização de todos os contadores/acumuladores
    for (i=0;i<N_facilities;i++) {
      num_cheg_fila[i]=tam_max_fila[i]=num_job_serv[i]=tx_cheg_fila[i]=0;
    }
    for (i=0;i<N_sources;i++) {
      num_jobs_gerados[i]=0;
    }

    //colocar aqui o nome do arquivo de saida de resultados.  o valor do SEED será acrescentado
    sprintf(file_out, "callcenter.out.seed%d.txt", SEED);
    fp = fopen(file_out,"w");

    sm(0,"Simulacao de um Call Center");
    //inicialização das facilities
    for (i=0; i<N_facilities; i++) {
      server[i]=facility(name_facility[i],1); //Voz1=0; Voz2=1; Voz3=2; Espera=3; Humano=4
    }
    //geração dos eventos iniciais para todas as fontes
    for (i=0; i<N_sources; i++) {
      schedulep(i+1,expntl(tau_source[i]),job, NULL);
    }
    stream(SEED); //alimenta (seed) o gerador de números aleatórios do SMPL

    while (simtime()<Max_time && tot_jobs_gerados<Max_jobs) //pára a simulação no primeiro estouro de tempo ou geração
```

```
    {
      causep(&event,&job);
      switch(event)
        {
          case 1:  //chegada no Voz1 (vindo da Linha1)
            schedulep(4,0.0,job, NULL); //escalona atendimento no Voz1
            num_cheg_fila[0]++; //incrementa acumulador de jobs que chegaram para a fila Voz1
            num_jobs_gerados[0]++; //incrementa número de jobs gerados pela fonte
            tot_jobs_gerados++;  //incrementa acumulador de jobs gerados total
            schedulep(1,expntl(tau_source[0]),job, NULL); //escalona nova chegada na Linha1
            break;
          case 2:  //chegada no Voz2 (vindo da Linha2)
            schedulep(5,0.0,job, NULL); //escalona atendimento no Voz2
            num_cheg_fila[1]++; //incrementa acumulador de jobs que chegaram para a fila Voz2
            num_jobs_gerados[1]++; //incrementa número de jobs gerados pela fonte
            tot_jobs_gerados++;  //incrementa acumulador de jobs gerados total
            schedulep(2,expntl(tau_source[1]),job, NULL); //escalona nova chegada na Linha2
            break;
          case 3:  //chegada no Voz3 (vindo da Linha3)
            schedulep(6,0.0,job, NULL); //escalona atendimento no Voz3
            num_cheg_fila[2]++; //incrementa acumulador de jobs que chegaram para a fila Voz3
            num_jobs_gerados[2]++; //incrementa número de jobs gerados pela fonte
            tot_jobs_gerados++;  //incrementa acumulador de jobs gerados total
            schedulep(3,expntl(tau_source[2]),job, NULL); //escalona nova chegada na Linha3
            break;
          case 4:  //reserva servidor Voz1
            t=expntl(s_server[0]);
            if (requestp(server[0],job,0, 7, t, NULL)==0)
               schedulep(7,t,job, NULL); //se estiver livre, escalona final de serviço
            else
            {
              if (inq(server[0])>tam_max_fila[0]) tam_max_fila[0] = inq(server[0]); // atualiza tamanho máximo de fila
            }
            break;
          case 5:  //reserva servidor Voz2
            t=expntl(s_server[1]);
            if (requestp(server[1],job,0, 8, t, NULL)==0)
               schedulep(8,t,job, NULL); //se estiver livre, escalona final de serviço
            else
            {
              if (inq(server[1])>tam_max_fila[1]) tam_max_fila[1] = inq(server[1]); // atualiza tamanho máximo de fila
            }
            break;
          case 6:  //reserva servidor Voz3
            t=expntl(s_server[2]);
            if (requestp(server[2],job,0, 9, t, NULL)==0)
               schedulep(9,t,job, NULL); //se estiver livre, escalona final de serviço
            else
            {
              if (inq(server[2])>tam_max_fila[2]) tam_max_fila[2] = inq(server[2]); // atualiza tamanho máximo de fila
            }
            break;
          case 7:  //chegada no Espera Forçada (vindo do Voz1)
            num_job_serv[0]++; //incrementa acumulador de jobs servidos
            releasep(server[0],job);
            schedulep(10,0.0,job, NULL); //escalona atendimento no Espera
            num_cheg_fila[3]++; //incrementa acumulador de jobs que chegaram para a fila Espera
            break;
          case 8:  //chegada no Espera Forçada (vindo do Voz2)
            num_job_serv[1]++; //incrementa acumulador de jobs servidos
            releasep(server[1],job);
            schedulep(10,0.0,job, NULL); //escalona atendimento no Espera
            num_cheg_fila[3]++; //incrementa acumulador de jobs que chegaram para a fila Espera
```

```
            break;
          case 9:  //chegada no Atendimento Humano (vindo do Voz3)
            num_job_serv[2]++; //incrementa acumulador de jobs servidos
            releasep(server[2],job);
            schedulep(12,0.0,job, NULL); //escalona atendimento no Humano
            num_cheg_fila[4]++; //incrementa acumulador de jobs que chegaram para a fila Humano
            break;
          case 10:  //reserva servidor Espera
            t=expntl(s_server[3]);
            if (requestp(server[3],job,0, 11, t, NULL)==0)
               schedulep(11,t,job, NULL); //se estiver livre, escalona final de serviço
            else
            {
              if (inq(server[3])>tam_max_fila[3]) tam_max_fila[3] = inq(server[3]); // atualiza tamanho máximo de fila
            }
            break;
          case 11:  //chegada no Atendimento Humano (vindo do Espera)
            num_job_serv[3]++; //incrementa acumulador de jobs servidos
            releasep(server[3],job);
            schedulep(12,0.0,job, NULL); //escalona atendimento no Humano
            num_cheg_fila[4]++; //incrementa acumulador de jobs que chegaram para a fila Humano
            break;
          case 12:  //reserva servidor Humano
            t=expntl(s_server[4]);
            if (requestp(server[4],job,0, 13, t, NULL)==0)
               schedulep(13,t,job, NULL); //se estiver livre, escalona final de serviço
            else
            {
              if (inq(server[4])>tam_max_fila[4]) tam_max_fila[4] = inq(server[4]); // atualiza tamanho máximo de fila
            }
            break;
          case 13:  //saída servidor Humano
            num_job_serv[4]++; //incrementa acumulador de jobs servidos
            releasep(server[4],job);
            break;

        }
    }
  //report();

  fprintf(fp,"Simulacao de um Call Center com SPML\n2004 por Marcos Portnoi");
  fprintf(fp,"\n\n\nTempo Simulado        :  %.2f\nNumero de Jobs Gerados:  %d", simtime(), tot_jobs_gerados);
  fprintf(fp, "\nSeed Utilizado        :  %d", stream(0));
  fprintf(fp,"\n\n *** FONTES ***\n");
  fprintf(fp,"\nFONTE - NUM.JOBS.GER. - TAXA DE GERACAO");
  fprintf(fp,"\n          [jobs]        [uts/jobs]");
  fprintf(fp,"\n------------------------------------------------------------");
  for(i=0;i<N_sources;i++) {
    fprintf(fp,"\n    %d %13d      %8.4f", i, num_jobs_gerados[i], tau_source[i]);
  }

  fprintf(fp,"\n\n\n *** SERVIDORES ***\n");
  fprintf(fp,"\nSERV. -   NOME  - UTILIZ. - JOB.SERV. - TX SERVICO - TEMPO SERVICO - OCUP.FINAL");
  fprintf(fp,"\n                              [jobs]     [jobs/uts]    [uts/jobs]  [0-livre;1-ocup.]");
  fprintf(fp,"\n-------------------------------------------------------------------------------");
  for(i=0;i<N_facilities;i++) {
    fprintf(fp,"\n  %d    %-8.8s %10.4f %10d %12.4f     %8.4f          %d",
      i, fname(server[i]), U(server[i]), num_job_serv[i], num_job_serv[i]/(simtime()*U(server[i])), s_server[i], status(server[i]));
  }

  fprintf(fp,"\n\n\n *** FILAS ***\n");
  fprintf(fp,"\nFILA -   NAME   - TX.CHG. - TAM.MED.FIL - TEM.MED.FIL - TAM.MAX - CLI. FINAL");
  fprintf(fp,"\n               [jobs/uts]    [jobs]        [uts]       [jobs]     [jobs]
  ");
```

```
    fprintf(fp,"\n-------------------------------------------------------------------------
    ---");
    for(i=0;i<N_facilities;i++) {
      tx_cheg_fila[i]=num_cheg_fila[i]/simtime();
      fprintf(fp,"\n  %d    %-8.8s %10.4f %10.4f   %10.4f %10d   %6d",
        i, fname(server[i]), tx_cheg_fila[i], Lq(server[i]), Lq(server[i])/tx_cheg_fila[i],
     tam_max_fila[i], inq(server[i]));
    }

    fprintf(fp, "\n\n\nOUTRAS ESTATISTICAS:");
    fprintf(fp, "\n\nLinha1:");
    t_espera_linha1=Lq(server[0])/tx_cheg_fila[0]+Lq(server[3])/tx_cheg_fila[3]+Lq(server
    [4])/tx_cheg_fila[4]+
      s_server[0]+s_server[3];
    fprintf(fp, "\n\nTempo de Espera ate ser atendido pelo Atendente Humano [uts]:  %.4f",
    t_espera_linha1);
    fprintf(fp, "\nTempo de Resposta Total ate final atendimento [uts]         :  %.4f",
    t_espera_linha1+s_server[4]);
    fprintf(fp, "\n\nLinha2:");
    t_espera_linha2=Lq(server[1])/tx_cheg_fila[1]+Lq(server[3])/tx_cheg_fila[3]+Lq(server
    [4])/tx_cheg_fila[4]+
      s_server[1]+s_server[3];
    fprintf(fp, "\n\nTempo de Espera ate ser atendido pelo Atendente Humano [uts]:  %.4f",
    t_espera_linha2);
    fprintf(fp, "\nTempo de Resposta Total ate final atendimento [uts]         :  %.4f",
    t_espera_linha2+s_server[4]);
    fprintf(fp, "\n\nLinha3:");
    t_espera_linha3=Lq(server[2])/tx_cheg_fila[2]+Lq(server[4])/tx_cheg_fila[4]+s_server[2]
    ;
    fprintf(fp, "\n\nTempo de Espera ate ser atendido pelo Atendente Humano [uts]:  %.4f",
    t_espera_linha3);
    fprintf(fp, "\nTempo de Resposta Total ate final atendimento [uts]         :  %.4f",
    t_espera_linha3+s_server[4]);

    fprintf(fp, "\n\n\nLEGENDA:");
    fprintf(fp, "\n\nNUM.JOBS.GER.   = Numero de Jobs Gerados por Fonte");
    fprintf(fp, "\nTAXA DE GERACAO = Taxa de Chegada de Jobs por Fonte");
    fprintf(fp, "\nUTILIZACAO      = Utilizacao por Servidor");
    fprintf(fp, "\nJOB.SERV        = Numero de Jobs Servidos por Servidor");
    fprintf(fp, "\nTX SERVICO      = Taxa de Servico por Servidor");
    fprintf(fp, "\nTEMPO SERVICO   = Tempo Medio de Servico por Servidor");
    fprintf(fp, "\nOCUP.FINAL      = Estado do Servidor ao Final da Simulacao (ocupado ou
    livre)");
    fprintf(fp, "\nTX.CHG.         = Taxa de Chegada de Jobs por Fila");
    fprintf(fp, "\nTAM.MED.FIL     = Tamanho Medio da Fila");
    fprintf(fp, "\nTEM.MED.FIL     = Tempo Medio de Espera em Fila");
    fprintf(fp, "\nTAM.MAX         = Tamanho Maximo Atingido pela Fila durante Simulacao");
    fprintf(fp, "\nCLI.FINAL       = Numero de Jobs em Fila ao termino da Simulacao");
    fclose(fp);
}
```

```
/*
*          Sistema Simulation Machine - Simulacao de Filas
*
*          Versao modificada do smpl usando a filosofia de
*         construcao de modelos de forma reestruturada para
*       utilizar mais adequadamente os recursos computacionais
*
*                  Arquivo <sm_types>
*                                       Sergio Brito - agosto/2002
*/

/* Este arquivo contem a especificacao das structs utilizadas para o simulador
   implementar a sua execucao */

/* Aqui sao declaradas as duas estruturas principais do simulador sm (simulation
   machine):  a estrutura de facilities e a estrutura da cadeia de eventos */

/* Uma modificacao realizada na cadeia de eventos e a possibilidade de se
   introduzir nesta cadeia um campo do tipo apontador que permitira
     alem da designacao de um numero para a token tambem um apontador associado
     a esta token  permitindo assim que cada evento tenha uma estrutura particular
     para o seu processamento */

/* Dentro da cláusula #ifdef USE_TKN, abaixo, deve-se declarar a estrutura do token como
    extern e usar
   uma cláusula typedef para definir a estrutura como TOKEN.  Por exemplo:
   extern struct Packet;
   typedef struct Packet TOKEN;

   A estrutura do token deve então ser definida apropriadamente posteriormente, por exemplo
    ,
   num arquivo .h do modelo.
*/

#define USE_TKN 0   /* Este define indica 1 se se deseja utilizar a estrutura de apontador
                       de token ou 0 se nao se deseja. Caso nao se deseje utilizar a
    estrutura
                       apontador de token, deve-se em todas as chamadas de funcoes que a
    definam,
                       colocar NULL no campo que seria preenchido por esta */


#ifdef USE_TKN

extern struct Packet; //a estrutura deve ser definida posteriormente, no modelo do usuário
typedef struct Packet TOKEN;    //declaracao do typedef associando a estrutura da token

#else //definir então uma estrutura de TOKEN padrão.  (isso é realmente necessário?)

struct Packet
{
    int f;
};
typedef struct Packet TOKEN;

#endif

struct simul {                          /* Armazena informacoes de simulacao do modelo
    atual */
    char name[50];                      /* nome a ser usado para denominar o modelo
    simulado */
    struct evchain *evc_begin;          /* Apontador para o inicio da cadeia de eventos
    (tokens) */
    struct evchain *evc_end;            /* Apontador para o final da cadeia de eventos
    (tokens) */
    struct facilit *fct_begin;          /* Apontador para o inicio da lista de facilities *
    /
    int fct_number;                     /* Especifica o ultimo numero de facility
    especificada */

};                  /* Poderemos ter varias simulacoes para o mesmo modelo */

struct facilit {                /* estrutura da facility tipicamente associada a servidores
     */
```

```
    char f_name[50];            /* nome da facility */
    int f_number;               /* numero da facility */
    int f_n_serv;               /* num de servidores associados a facility */
    int f_n_busy_serv;          /* num de servidores ocupados */
    int f_n_length_q;           /* tamanho fila; cada facility tem só uma fila associada */
    int f_max_queue;            /* tamanho maximo da fila */
    int f_exit_count_q;         /* numero de dequeues durante o período simulado */
    double f_last_ch_time_q;        /* momento da ultima mudanca na fila */
    double f_busy_time;         /* tempo que a facility fica ocupada - somatorio de
    fs_busy_time de todos os seus servidores */
    int f_preempt_count;        /* numero de preempções efetivamente realizadas */
    double length_time_prod_sum;    /* utilizado para obter o tamanho medio da fila */
    int f_release_count;        /* total de tokens que foram servidas pela facility */
    struct fserv *f_serv;       /* apontador para a lista de servidores */
    struct fqueue *f_queue;     /* apontador para a fila de tokens de espera */
    struct facilit *fct_next;   /* apontador para a proxima facility */
    int f_up;                   // status da facility:  1 para operacional (up), 0 para não
    -operacional (down) (25.Dec.2005 Marcos Portnoi)
    int f_tkn_dropped;          // número de tokens descartadas pela facility (ao entrar em
     estado down, a fila é descartada e este contador é atualizado
                                // se uma token for escalonada para uma facility (requestp
    ou preemptp) e esta estiver down, este contador também será incrementado
};

struct fserv {                  /* estrutura de cada servidor associado a uma facility */
    int fs_number;              /* numero do servidor */
    int fs_tkn;                 /* numero da token em processamento neste servidor */
    int fs_p_tkn;               /* prioridade da token */
    int fs_release_count;       /* numero de tokens que foram servidas por este servidor */
    double fs_start;                /* momento de inicio do servico */
    double fs_busy_time;            /* somatorio do tempo que o servidor ficou ocupado na
    simulacao */
    struct fserv *fs_next;      /* apontador para o proximo servidor */
};

struct fqueue {                 /* estrutura da fila que é associada a facility */
    int fq_tkn;                 /* numero da token */
    int fq_ev;                  /* numero do evento ao qual a token esta associada */
    double fq_time;             /* tempo de execucao do processamento da token, podera ser
                                   tambem o restante do tempo para processamento da token
                                   se esta tiver sido retirada de servico por uma token
                                   preempted; para uma token bloqueada, este fq_time será
    zero */
    int  fq_pri;                /* prioridade da token */
    struct fqueue *fq_next;     /* apontador para a proxima token */
    TOKEN *fq_tkp;              /* apontador para struct com informacoes da token */
};

/* A estrutura da cadeia a seguir tem algumas caracteristicas de interesse
   que sao:
     1. Ela e uma lista duplamente encadeada e que estara ordenada no tempo,
        ou seja o tempo mais proximo sera o primeiro da lista. Assim, o
            cabecalho da lista estara apontando sempre para o proximo evento a
            ser processado.
     2. Esta lista tera um apontador inicio da lista que sera usado para obter
            o proximo evento a ser processado e um apontador de final da lista que
            sera utilizado para permitir que os novos eventos a serem incluidos
            comecem pesquisando pelo final da lista. Ele pesquisara a partir do final
            da lista e encontrara o local onde devera ser inserido o novo evento.
            Esta estrategia provavelmente diminuira sensivelmente o tempo de busca
            do local de insercao na cadeia de eventos.
            Os apontadores de inicio e final sao globais:
            ev_begin -> apontador para o inicio da cadeia de eventos
            ev_end -> apontador para o final da cadeia de eventos
     3. A estrutura permitira que seja incluido um campo do tipo apontador que
            pode variar de modelo para modelo para descrever a estrutura da token que
            esta sendo usada para simulacao.
*/

struct evchain {                    //declaracao da estrutura dos elementos da cadeia de
    eventos
    double ev_time;                 //tempo do proximo processamento desta token
    int ev_tkn;                     //numero da token
    int ev_type;                    //tipo do evento associado a esta token para ser
```

```
    processada
    struct evchain *ev_previous;    //apontador para a token anterior (alterado de 'prior', ↙
     (2005) Marcos Portnoi)
    struct evchain *ev_next;        //apontador para a proxima token

                                    //Inicio da declaracao da estrutura da token
    TOKEN *ev_tkn_p;                //Apontador do tipo da struct associada a token -      ↙
    packet
                                    //Final da declaracao da estrutura da token
};
```