АНОТАЦІЯ

CIM-CI відноситься до класу систем імітаційного моделювання загально-цільового призначення. Типові області застосування: моделювання систем масового обслуговування, подійне моделювання дискретних систем. Призначена для рішення завдань дослідження поводження технічних, економічних, виробничих й управлінських систем.

Система CIM-CI придатна для використання на всіх ЕОМ, що мають транслятор з мови C++. Моделі оформляються у вигляді функцій мовою C++, що забезпечує систему моделювання засобами сполучення із програмами оптимізації, чисельних розрахунків, статистичної обробки експерименту та іншими.

Автоматичні збір і первинна обробка статистики, наявність засобів налагодження, діагностики помилок й убудований діалог для спостереження за динамікою поводження моделі й керування моделюванням значно полегшують розробку й дослідження моделей.

За своїми можливостями повністю перекриває спеціалізовану мову моделювання GPSS-V, маючи більш високу швидкодію й гнучкість.

ЗМІСТ:

ВСТУП

Система імітаційного моделювання на CI (CIM-CI) по реалізації являє собою методологію побудови дискретних подійних моделей і набір підтримуючих дану методологію функцій.

Методологія побудови моделей передбачає роздільне визначення модельного середовища й виконуємих у ній дій. Фактично дискретна подійна модель складається із трьох частин:

1. Частина опису середовища моделювання;

2. Частина ініціалізації модельного середовища;

3. Подійна частина.

Частина 1 передбачає явне визначення усіх змінних, використовуємих програмами.

Частина 2 використовується для створення об'єктів системи моделювання й ініціалізації генераторів, використовуємих у моделюванні.

Частина 3 передбачає визначення дій необхідних при виникненні в моделі подій. У цій частині дуже широко використовуються функції підтримки, застосування яких значно спрощує складання моделей й автоматизує процес моделювання, а також збір й обробку експериментальних даних.

Нижче буде показано, що для подання подійної частини існує багато різних способів.

У системі моделювання CIM-CI обрана форма запису моделей, яка по своєму виду нагадує форми, яві використовуються у мові моделювання GPSS.

Таким чином, у системі CIM-CI сполучаються ідея подійного опису моделей з процесно-орієнтованою формою запису GPSS-моделей, що підвищує наочність подання моделей, зі збереженням можливості опису складних подій.

Основні поняття й визначення

ТРАНЗАКТ - абстрактний динамічний об'єкт, що характеризується рядом властивостей, що задають значеннями параметрів. У ході моделювання транзакти можуть створюватися, переміщатися по моделі й знищуватися.

АКТИВНИЙ ТРАНЗАКТ - транзакт, що просувається по моделі в цей момент виконання програми.

АНСАМБЛЬ - зв'язана по груповій ознаці сукупність транзактів. Для членів одного ансамблю можливо одночасна зміна одного або декількох параметрів.

З кожним переміщенням активного транзакта зв'язана ПОДІЯ. **ПОДІЯ** - послідовність дій, пов'язаних зі зміною стану модельної й/або системного середовища в результаті переміщення транзакта.

Стан моделі в обраний момент часу характеризується значеннями змінних, статичних і динамічних об'єктів моделі, що утворять **МОДЕЛЬНЕ СЕРЕДОВИЩЕ**.

СИСТЕМНЕ СЕРЕДОВИЩЕ - сукупність констант, типів і змінних системи CIM-CI. Містить у собі, зокрема, СИСТЕМНІ СПИСКИ.

СИСТЕМНІ СПИСКИ - визначені в CIM-CI списки: приладів, черг, накопичувачів, гістограм; список списків користувача; списки транзактів (*current*, *future*, *delist*, *histlist* й інші).

ЧЕРГА - динамічний об'єкт, призначений для реєстрації просування транзактів на певних ділянках моделі, збору й обробки відповідної статистики.

ПРИЛАД - динамічний об'єкт, призначений для моделювання обслуговуючого апарата. Транзакт може очікувати входу в прилад, займати або захоплювати прилад, звільняти прилад. По приладу збирається відповідна статистика.

НАКОПИЧУВАЧ (багатоканальний пристрій) - динамічний об'єкт, призначений для моделювання одного або декількох обслуговуючих пристроїв. Транзакт може очікувати входу в накопичувач, займати в накопичувачі одну або кілька осередків, залишати накопичувач.

СИСТЕМНИЙ ЧАС - чисельна величина, щодо якої синхронізуються всі події в моделі. При здійсненні подій системний час може або залишатися незмінним, або збільшуватися.

ПОТОЧНА ПОДІЯ - подія, пов'язана з переміщенням активного транзакта.

СПИСОК КОРИСТУВАЧА - список транзактів, який створюється, керується і знищується користувачем за допомогою призначених для цього функцій. Застосовується, як правило, для організації нестандартних дисциплін обслуговування.

СИГНАЛИ використовуються для організації взаємодії транзактів, що перебувають у різних точках моделі. СИГНАЛ виникає в результаті проходження активним транзактом заданої точки моделі. Поява СИГНАЛУ приводить до активізації всіх очикуєчих його транзактів.

1 ОБ'ЄКТИ СИСТЕМИ МОДЕЛЮВАННЯ CIM-CI

1.1 Загальна класифікація

Для побудови моделей потрібні об’єкти, що володіють всілякими властивостями. У різних прикладних областях властивості, властиві моделюємим системам, різні, різний спосіб формалізації моделей і склад об'єктів, які входять у них. У цьому змісті ступінь універсальності систем моделювання можна характеризувати доступними в них об'єктами, а також властивостями об'єктів, реалізованих через визначені набори операцій.

У системі моделювання CIM-CI використовуються абстрактні набори об'єктів, конкретне семантичне призначення яких установлюється програмістом, програміст же реалізує логіку їхньої взаємодії, використовуючи визначені в системі й (якщо це необхідно) складені самостійно об’єктно-орієнтовані функції. Перш ніж перейти до розгляду окремих об'єктів, дамо їхню загальну класифікацію.

Об'єкти системи моделювання CIM-CI за часом існування в моделі й способу створення діляться на статичні й динамічні.

Статичні об'єкти представляються системними змінними постійно розміщеними в оперативній пам'яті. Вони необхідні увесь час роботи моделі. Прикладом такого об'єкта є *systime* - змінна дійсного типу, значення якої визначає модельний час.

Динамічні об'єкти створюються в міру необхідності. На час їхнього існування для них виділяється місце у вільній області оперативної пам'яті. Знищення такого об'єкта приводить до звільнення займаної пам'яті. Отже, знищення й створення динамічних об'єктів дають принципову можливість моделювання систем із числом об'єктів більшим, ніж при статичному розміщенні. Прикладом динамічних об'єктів можуть служити гістограми, існування яких необхідно тільки на час виміру деяких випадкових значень.

Об'єкти бувають переміщуваними й нерухомими. Переміщення об'єктів викликає здійснення подій у системі та зміну системного часу. Найважливішим типом переміщуваних об'єктів є транзакти, всі інші об'єкти можуть переміщуватися тільки будучи закріпленими за транзактами. Транзакт, як абстрактний об'єкт системи, може

являти собою деталь, яка оброблюється на потокової лінії, літак що прибуває до аеропорту або електричний імпульс, що включає виконавчий пристрій.

Іншими словами транзакт може собою представляти будь-який елемент однорідного потоку об'єктів у реальній системі. Як переміщувані об'єкти транзакти можуть затримуватися, групуватися, змінювати свої властивості або властивості їхнього навколишнього модельного середовища.

У системі моделювання об'єкти можна розглядати як скалярні й множинні.

Скалярні об'єкти характеризуються одним єдиним значенням, приміром, системний час (*systime*) і системна подія (*sysevent*), відповідно, дійсне й позитивне ціле число.

Множинні об'єкти характеризуються набором значень, приміром, транзакт має цілі дійсні й бульові параметри,крім того він несе в собі інформацію про чергову (для даного транзакта) подію що виконується й час його виконання, а також іншу інформацію. Гістограма також являє приклад множинного об'єкта, вона містить інформацію про число спостережень випробовуваної величини, про кількість показань спостережуваного значення і різні інтервали, а також іншу інформацію, яка використовується для підрахунку математичного очікування й дисперсії, а також для виводу гістограми в графічному виді.

Об'єкти бувають одиночні й групові.

Групові об'єкти поєднуються в сукупність по деякій ознаці. Основна форма існування групових об'єктів - списки.

Приміром, транзакти зв'язуються в списки залежно від того, якими груповими властивостями вони володіють.

Транзакти можуть заноситися у список:

delist - список незатребуваних моделлю транзактів;

current - список транзактів, готових до просування в теперішній момент часу;

userlist - списки, створювані користувачем для організації різних дисциплін обробки.

Крім зазначених є й інші списки, обговорення яких буде проводитися нижче. Помітимо, що в списки зв'язуються й інші об'єкти. Найчастіше дії над списками здійснюються автоматично.

Іншим типом групових об'єктів у системі CIM-CI є ансамблі. Ансамблі як сукупності транзактів можуть мати цілий набір групових властивостей, що допускають одночасну зміну для всіх членів ансамблю. Приміром, група транзактів може рухатися по декількох можливих маршрутах у моделі. Окремі транзакти можуть перебувати в різних крапках моделі й бути приписаними до різних списків. Зміна номера маршруту у всіх членів ансамблю приведе до зміни шляху проходження у весь, утворюючий ансамбль транзактів.

Точне визначення об'єктів як типів даних у системі моделювання CIM-CI приводиться нижче. Дії над об'єктами розглядаються під час обговорення функцій.

1.2 Системні константи, типи й змінні

Безліч об'єктів визначена в системі СИМ-СИ шляхом введення абстрактних типів даних. Створення або визначення необхідної кількості об'єктів покладено на програміста, він також зобов'язаний передбачити дії над об'єктами при виникненні в системі певних умов.

Системні константи використовуються для настроювання системи моделювання на доступну при реалізації на конкретній ЕОМ пам'ять. Число системних констант дуже мале, але вони впливають на можливості системи.

Нижче визначаються наступні константи:

evemax - обмежує в системі максимальну кількість подій; кожна подія, як уже вказувалося, визначає послідовність дій (фактично алгоритм поводження транзакта при його переміщенні у відповідну події точку моделі). Чим більше число подій визначене в моделі, тим більш детальним є опис реальної системи. У цьому змісті evemax обмежує можливості CIM-CI по поданню реальних систем.

signmax - константа обмежує кількість сигналів у системі. Сигнали використаються для взаємодії між транзактами, що перебувають у різних частинах моделі. Сигнали можна передавати й приймати (якщо буде потреба очікуючи їхньої появи).

hint - константа, що задає максимальне число інтервалів у гістограмі. Властивість hint задає число точок на числовій осі, що задає інтервали для табулювання значень.

Наступні константи мають відношення до транзактів:

prtymax - максимальна абсолютна величина пріоритету транзакта. Пріоритети транзакта використовуються в багатьох випадках. Зокрема, якщо в деякий фіксований момент часу на переміщення по моделі претендують одночасно декілька транзактів, то, як правило, рухатися буде той, котрий має вищий пріоритет, інші будуть чекати його зупинки.

Константи, що встановлюють число параметрів транзакта:

mptb число бульових параметрів:

mptr число дійсних параметрів:

mpti число цілочисельних параметрів:

mptf число покажчиків на прилади:

mptq число покажчиків на черги:

mpts число покажчиків на накопичувачі.

Докладне призначення констант дається у визначенні типів.

Системні типи

Введення абстрактних типів даних, що представлені нижче, покладено в основу формалізації процесу моделювання.

Пропоновані типи служать для створення системного середовища й використовуються користувачем для визначення необхідних йому об'єктів.

Помітимо, що в системі моделювання визначені як типи даних:

1) структури абстрактних об'єктів;

2) покажчики на ці структури:

Покажчики використовуються для визначення положення об'єкта, а сам об'єкт містить необхідні поля даних, значення яких використаються при моделюванні.

Скалярні типи даних

Для визначення змінних, що мають зміст часу, використовується тип ***double***.

При моделюванні час змінюється від нуля до деякої позитивної величини. Одиниця виміру часу в системі не визначається: отже інтерпретація часу в моделі задається тими одиницями часу, які визначив користувач.

Тип *prtyrange=-prtymax..prtymax* визначає діапазон зміни пріоритетів транзактів, надалі використовується для визначення нових типів даних.

Простор зміни системних подій визначено типом *event=1..evemax*.

Нумерація подій у моделі не повинна виходити за межі, задані типом *event*. Моделі, процеси й підмоделі займають зв'язну область простору подій.

Тип *enum parmtype {parmb,parmi,parmr,parmf,parmq,parms,last_parmtype}* специфікую типи параметрів транзакта:

parmb задає параметр бульового типу;

parmi задає параметр цілого типу;

parmr задає параметр дійсного типу;

parmf задає параметр типу покажчик на прилад;

parmq задає параметр типу покажчик на чергу;

parms задає параметр типу покажчик на накопичувач.

Тип *parmtype* використовується допоміжними функціями.

До числа скалярних типів також відносяться покажчики на деякі типи системних змінних. Їхній опис будемо приводити при визначенні відповідного типу.

Множинні типи даних

Для завдання імен у системі визначений тип даних *typedef array<1,8,**char**> alfa*.

Відзначимо, що в деяких функціях у якості параметрів можлива підстановка констант типу *alfa*. **Текстовий рядок-параметр типу *alfa* повинен містити число символів у точності рівне восьми!**

Перераховуємі нижче множинні типи даних будемо визначати за наступною схемою:

- призначення типу;
- специфікація типу в системі моделювання;
- покажчик на розглянутий тип (якщо такий є).

При описі полів об'єктів у розділі "специфікація" призначення поля дається коментарем. У коментарях символом *"U"* відзначені ті з них, які можуть змінюватися користувачем за його розсудом; символом *"S"* відзначені поля, використовуємі для системних цілей. Їхній уміст можна тільки читати. Зміна значень цих полів може привести до непередбачених наслідків.

ТРАНЗАКТ - абстрактний динамічний об'єкт, що характеризується рядом властивостей, що задаються значеннями параметрів. У ході моделювання транзакти можуть створюватися, переміщатися по моделі й знищуватися.

```
struct transact { //      транзакт           Параметри:
     //   Установлюється користувачем при  створенні власних функцій для
роботи зі списками користувача. Див. пр.

     array<1,mptb,bool> pb;              // бульові                    U
     array<1,mpti,int> pi;               // цілі                       U
     array<1,mptr,double> pr;            // дійсні                     U
     array<1,mptf,pfacility> pf;         // посилання на прилади       U
     array<1,mptq,pqueue> pq;            // посилання на черзі         U
     array<1,mpts,pstorage> ps;          // посилання на накопичувачі  U
     prtyrange prty;                     //   пріоритет                U
     bool testprty;                      //   ключ пріоритету          U

     // Всі інші поля використовуються тільки системою !

     int ans,                            //  номер ансамблю
         nans,                           //  кількість членів ансамблю
         nom;                            //  номер транзакта
     ptransact  predans,                 //  посилання на попер., наст.

         sledans,                        // транзакт в ансамблі
         pred,                           // посилання на попер., наст.

         sled;                           // транзакт у списку
     event eve;                          //  подія ініціалізації
     double nexttime;                    //  час активізації
     plistt translist;                   //  посилання на займаємий список
     };
typedef struct transact* ptransact;      //  посилання на транзакт
```

ПРИЛАД - динамічний об'єкт, призначений для моделювання обслуговуючого апарата. Транзакт може очікувати входу в прилад, займати або захоплювати прилад, звільняти прилад. По приладу збирається відповідна статистика.

```
struct facility {        //   прилад
```

```
     bool test;          //   ознака включення        U
     alfa name           //   символьне ім'я          U

     // Всі інші поля використовуються тільки системою!

     enum {free,seized,preempted}
          status;        //   стан приладу
     pfacility sled,     //   посилання на наст. і попер.
          pred;          //   прилади в списку
     ptransact transpoint;//  посилання на обр. транзакт
     int p,              //   число захватів
          ci;            //   число входів
     double timef,       //   часи: зайнятості,
          pretime,       //   попереднього звернення
          mtime,         //   службове поле
          pro;           //   завантаження приладу
     plistt fl,          //   посилання на списки очікуючих
          inter;         //   та перерваних транзактів
};
typedef struct facility* pfacility;     //  посилання на прилад
```

ЧЕРГА - динамічний об'єкт, призначений для реєстрації просування транзактів на певних ділянках моделі, збору й обробки відповідної статистики.

```
struct queue { //     черга
     bool test;                  //   ознака включення     U
     alfa name;                  //   символьне ім'я           U

     // Всі інші поля використовуються тільки системою!

     enum {empty,full} status;   //   статус(порожня/непуста)
     int lq,                     //   довжина черги: поточна,
          mq,                    //   максимальна,
          size;                  //   гранична
     pqueue sled,                //   посилання на наст., попередню
          pred;                  //   черги в списку черг
     int ci,                     //   число входів: загальне,
          co;                    //   у чергу нульової довжини
     double timeq,               //   загальний час зайнятості
          pretime,               //   час попер. звернення
          lm,                    //   середня довжина
          mtime;                 //   службове поле
};
typedef struct queue* pqueue;    //   посилання на чергу
```

НАКОПИЧУВАЧ (багатоканальний пристрій) - динамічний об'єкт, призначений для моделювання одного або декількох обслуговуючих пристроїв. Транзакт може очікувати входу в накопичувач, займати в накопичувачі одну або кілька осередків, залишати накопичувач.

```
struct storage {    //   накопичувач
     bool test;              // ознака включення     U
     alfa name;              // символьне ім'я        U

     // Всі інші поля використовуються тільки системою!

     int s,                  // ємність накопичувача
          ss,                // поточний уміст
          sf,                // ємність, що залишається
          sm,                // макс. уміст
          ci;                // число входів
     double ut,              // завантаження накопичувача
          smean,             // середній уміст
          mtime,             // службове поле
          times,             // час: зайнятості,

          pretime;           // попер. звернення
     pstorage pred,          // посилання на попередній,
          sled;              // наст. накопичувачі
     plistt slt;             // посилання на список очікуючих транзактів
};
typedef struct storage* pstorage; //     посилання на накопичувач
```

Гістограми в системі імітаційного моделювання використовуються для збору й нагромадження статистичної інформації про будь-які скалярні, дійсні або цілі змінні. Всі необхідні гістограми задаються користувачем у частині визначень, створюються в розділі створення модельного середовища. Вимірювані значення табулюються в гістограмі в подійній частини моделі.

```
struct histogram{   // гістограма

     // Значення полів задаються користувачем при створенні гістограми

     bool graf;              //   ключ  друку графіка
     alfa name;              //   символьне ім'я
     hint2 ihint;            //   число точок табулювання
     double maxx,
          minx;

     // Всі інші поля використовуються тільки системою!

     unsigned total,         //   загальна кількість входів
          sum,
          sumsqr,
     phistogram sled,
          pred;
     harr x;
};
typedef struct histogram* phistogram;  //  посилання на гістограму
```

Таблиці в системі моделювання є зручним засобом для визначення табличних розподілів імовірностей.

```
struct table { // таблиця
     array<1,hint,double> x;   //  масив значень
     array<1,hint,double> p;   //  масив імовірностей
     hint2 ihint;              //  число інтервалів
};
```

Спискові структури або списки використовуються для зв'язування об'єктів системи моделювання й створення динамічної (змінюваної в часі) модельного середовища. Нижче визначаються списки різних об'єктів. Загальними для них є наступні параметри:

```
struct parml { //ПАРАМЕТРИ СПИСКУ
     double timel,       //   часи зайнятості,
          pretime;       //   попереднього звернення
     int ci,             //   число входів: загальне,
          co,            //   у порожній список
          ll,            //   довжина поточна,
          lm;            //   максимальна
     alfa name;          //   ім'я списку
};
```

Найважливішим списком є список транзактів - об'єктів, що переміщаються в моделі. У системі моделювання є кілька списків транзактів. Визначення списку наступне:

```
struct listt {   // СПИСОК ТРАНЗАКТІВ
     ptransact first;    //   посилання на перший транзакт
     parml p;            //   поле параметрів списку
     plistt sled,        //   посилання на наступний,
          pred;          //   попередній список
};

typedef struct listt* plistt;   // посилання на список
транзактів
```

Деякі списки транзактів поєднуються в загальні. Для цього використовуються наступні типи даних:

```
struct listl { // СПИСОК СПИСКІВ ТРАНЗАКТІВ
     plistt first;  //   посилання на перший список тр.
```

```
     int ll;        //   кількість списків у списку
};
```

Обумовлені користувачем об'єкти: прилади, черги, накопичувачі й гістограми зв'язуються в списки, типи яких визначені нижче.

```
struct listf { //     СПИСОК ПРИЛАДІВ
     pfacility first;         //   посилання на перший прилад
     parml p;                 //   поле параметрів списку
     plistf sled,             //   посилання на наступний,
          pred;               //   попередній список
};
typedef struct listf* plistf; //   посилання на список приладів

struct listq { //     СПИСОК ЧЕРГ
     pqueue first;       //   посилання на першу чергу
     parml p;            //   поле параметрів списку
     plistq sled,        //   посилання на наступний,
          pred;          //   попередній список
};
typedef struct listq* plistq;  посилання на список черг

struct lists { //     СПИСОК НАКОПИЧУВАЧІВ
     pstorage first;     //   посилання на перший накопичувач
     parml p;            //   поле параметрів списку
     plists sled,        //   посилання на наступний,
          pred;          //   попередній список
};
typedef struct lists* plists; //   посилання на список накопичувачів

struct listh { //     СПИСОК ГИСТОГРАММ
     phistogram first;        посилання на першу гістограму
     parml p;            //   поле параметрів списку
     plists sled,        //   посилання на наступний,
          pred;          //   попередній список
};
typedef struct listh* plisth;      //   посилання на список гістограм
```

Списки приладів, черг, накопичувачів і гістограм необхідні для організації автоматичної видачі результатів моделювання. Покажчики (посилання) на пойменовані списки використовуються системою моделювання.

Системні змінні

У системі моделювання визначена невелика кількість загальносистемних змінних, за допомогою яких здійснюється керування моделюванням і відслідковується стан моделі.

Більшість системних змінних змінюється автоматично, довільне присвоювання їм значень може непередбаченим чином вплинути на хід моделювання.

Виключення становлять наступні змінні, керуючі вихідним друком по ходу моделювання:

bool *trace*, ключ включення трасування

errtest; ключ розширеної (аварійної) видачі

Далі будемо давати визначення змінної й приводити її опис.

Для відліку відносного часу на черговому кроці модельного експерименту використовується змінна, яка встановлюється в момент скидання статистики:

double *resettime*; час останнього скидання статистики

Змінна **"СИСТЕМНИЙ ЧАС"** - чисельна величина, щодо якої синхронізуються всі події в моделі. При здійсненні подій системний час може або залишатися незмінним, або збільшуватися.

double *systime*; системний час

Службова змінна, що використовується в підсумковій видачі:

event ievemax; остання виконувана подія

"**ПОТОЧНА ПОДІЯ**" - подія, пов'язана з переміщенням активного транзакта.

event sysevent; поточна подія

Службова змінна, що встановлює число транзактів доступних у моделі:

int *itransmax*; кількість транзактів в *delist*

"**АКТИВНИЙ ТРАНЗАКТ**" - транзакт, що просувається по моделі в поточний момент виконання програми. Він визначається змінної:

ptransact trans; посилання на транзакт, що просувається

Наступні змінні задають основні системні списки транзактів:

plistt delist, посилання на пасивний буфер

current, посилання на ланцюг поточних подій

future; посилання на ланцюг майбутніх подій

Об'єкти системи моделювання заносяться в списки, що задаються змінними:

plistq quelist; посилання на сист. список черг

plistf faclist; посилання на сист. список приладів

plists stlist; посилання на сист. список накопичувачів

plisth histlist; посилання на сист. список гістограм

Транзакти, що очікують появи деякого сигналу, події, що збираються в ансамблях, заносяться в списки, визначені змінними:

array<min_signal,max_signal,plistt> signlist: посилання на списки транзактів, очікуючих сигналів

array<1,evemax,plistt> ass; посилання на списки ансамблів, що збираються

array<1,evemax,plistt> waitl; посилання на списки транзактів, очікуючих подій

Списки, що задаються користувачем, заносяться в загальний список або список списків:

listl userlist; список списків користувача

Наступні змінні є службовими:

*array<1,evemax,**bool**> waitevent*; ознаки здійснення подій

*array<1,evemax,**int**> maxevent*; лічильники виконання подій

***long** V0*, джерела генераторів випадкових чисел

V1,V2,V3,V4,V5,V6,V7,V8,V9,V10,V11,V12,V13,V14,V15;

***double** eventall*; загальне число подій, що відбулися

ofstream outfile; основний вихідний файл

ofstream f; допоміжний вихідний файл

2 ПРИНЦИПИ Й МЕТОДИ ПОБУДОВИ МОДЕЛЕЙ

2.1 Створення транзактів

Здійснення подій у моделі логічно пов'язане з переміщенням окремих транзактів. У момент здійснення події готовими до переміщення можуть виявитися один або декілька транзактів. Вони переміщуються по черзі. Порядок пересування готових транзактів визначається їхнім пріоритетом. Модельний час змінюється в тому випадку, коли список готових до переміщення в поточний момент часу транзактів вичерпаний. Переміщуємий транзакт назвемо активним.

Перш ніж транзакти стануть доступними до переміщення в моделі, вони повинні бути створені.

У моделі з єдиним просуваємим (активним) транзактом зв'язується посилання *trans*. Якщо її значення дорівнює *nil*, то в моделі транзактів немає взагалі, або вони перебувають в одному зі списків (по посиланнях: *current*, *future*, *waitl* (масив списків), списки приладу, накопичувача).

Створювані транзакти надходять у модель зі списку *delist*, а після завершення свого шляху в моделі повертаються в нього назад.

Уведення транзакта в модель здійснюється функцією ***void*** *create(**double** r)*, її єдиний параметр задає інтервал часу, через який у систему ввійде наступний транзакт. Величина "*r*" може бути константою або змінної, задаватися функцією. Залежно від способу визначення "*r*", функція *create* може виконувати роль генератора транзактів з різними (довільними) законами розподілу інтервалів між моментами їхнього надходження в модель. Всі події, що містять функцію *create*, повинні бути ініціалізовані при створенні модельного середовища. Для цього використовується функція ***void*** *initcreate(event e,**double** r)*, де *e* - номер події, що містить функцію *create(r)*, *r* - час надходження першого транзакта, що створюється функцією *create(r)*. Якщо ж необхідно забезпечити одночасний вихід транзактів у подію *e*, то функція *initcreate(e,r)* виконується відповідну кількість разів.

Приклад 1.1. Уведення транзактів у модель здійснюється кожні 5 одиниць модельного часу

```
#include "simc.h"
void main() {
     ...;
     initcreate(4,0);     Ініціалізація події 4
     ...;
     while(systime<1000) {
          plan();
          switch(sysevent) {   Вибір системної події
               case 1: ...; break;
               ...: ...;
               case 4: create(5); break;      Створення транзактів
               ...: ...
          }
     }
     printall();
}
```

Приклад 1.2. Транзакти надходять у модель за законом нормального розподілу із середнім 4.0 і дисперсією 1.5.

```
#include "simc.h"

void main() {
     ...;
     initcreate(3,0);     Ініціалізація події 3
     ...;
     while(systime<1000) {
          plan();
          switch(sysevent){
               ...: ...;
               case 3: create(randnorm(4.0,1.5,v1)); break;
               ...: ...    v1 - джерело генератора випадкових чисел
          }
     }
     printall();
}
```

2.2 Знищення транзактів

Транзакти, що завершили своє просування по моделі, залишають її, проходячи через функцію ***void*** *destroy()*. Фактично ця функція поміщає транзакт, на якій указує посилання *trans*, у список *delist*.

Приклад 2.1. Знищення транзакта в події номер 12.

```
#include "simc.h"
```

```
void main() {
     ...;
     while(systime<1000) {
          plan();
          switch(sysevent) {
               ...: ...;
               case 12: destroy(); break;     Знищення транзактів
               ...: ...
               }
          }
     }
     printall();
}
```

2.3 Просування транзактів

Транзакти можуть просуватися в моделі тільки в моменти здійснення подій. Кожен транзакт у моделі рухається від функції *create* до функції *destroy*. Звернень і до тих, і до інших функцій у системі може бути кілька. Будь-які паралельні процеси в моделі здійснюються послідовно (їх можна синхронізувати), тому в будь-який момент виконання програми може просуватися тільки один транзакт.

Якщо в деякий момент модельного часу почалося просування транзакта, то він переміщається по моделі (без збільшення модельного часу) від події до події доти, поки не зустріне одну з функцій: *destroy*, *delayt*, *wait*, *seize*, *infac*, *enter* або функції, що поміщають транзакт у список користувача.

Перші дві безумовно припиняють просування транзакта, інші - залежно від настання зазначеної у функції події (*wait*) або залежно від стану зайнятості зазначених у функціях пристроїв (*seize, infac, enter*).

Послідовність дій при виконанні події довільна, однак варто пам'ятати, що зазначені функції можуть установлювати *trans=nil* (*delayt*, *destroy* - завжди встановлюють) і тимчасово виводити транзакт із числа активних (здатних просуватися в моделі в цей момент модельного часу при даному стані системи). Єдина функція - *delayt* - точно встановлює значення модельного часу нової активації транзакта. Момент активації транзактів, що пройшли через функції *destroy*, *wait*, *seize*, *infac*, *enter* залежить від наступного поводження моделі.

2.4 Затримка транзактів. Функція *delayt*

Якщо значення модельного часу (*systime==c)*, а час затримки у функції ***void*** *delayt(**double**)* дорівнює *r*, то транзакт заноситься в список *future*, а час виходу транзакта зі списку дорівнює *r*+*c*. Розміщення транзактів у списку *future* здійснюється в порядку зростання часів виходу. Якщо в списку декілька транзактів мають однаковий час виходу, то ці транзакти розміщуються в порядку убування їхніх пріоритетів. Транзакти з однаковими пріоритетами й часом виходу розміщуються в порядку їхнього надходження в список.

Приклад 4.1. Модель із найпростішим випадком затримки.

```
#include "simc.h"

void main() {
     ...;
     initcreate(1,0);
     ...;
     while(systime<1000) {
          plan();
          switch(sysevent) {
               case 1: create(40); break;
               ...: ...;
               case 10: delayt(50); break;  // затримати транзакт на 50
одиниць модельного часу
               ...: ...;
               case 12: destroy(); break;
               }
          }
     printall();
}
```

Приклад 4.2. Затримка транзакта на час, експоненціально розподілене з μ=4.0

```
#include "simc.h"

void main() {
     ...;
     while(systime<2500) {
          plan();
          switch(sysevent) {
               ...: ...;
               case 42: delayt(randexp(4.0,v1)); break;
               ...: ...
               }
          }
```

```
     printall();
}
```

2.5 Блокування транзактів

2.5. 1 Функція *wait*

Просування транзакта в моделі може бути припинене до настання заданої події. Для цієї мети призначена функція ***void*** *wait(event e)*, де ***e*** - номер очікуваної події. Якщо подія ***e*** уже відбулася, то транзакт продовжує просуватися по моделі, інакше транзакт, на який указувало посилання *trans*, заноситься в список по посиланню *waitl*[*e*] останнім у своєму класі пріоритетів. При цьому значення посилання *trans* установлюється рівним *nil*. Після виконання події *e* транзакт, перший у списку *waitl*[e], переводиться в список *current*.

Приклад 5.1. Використання функції *wait* для організації черги перед вузлом затримки. Надходження транзактов експонентне, затримка - постійна, дисципліна обслуговування - FIFO. У цьому прикладі транзакти, що надходять на обробку, накопичуються в списку транзактів, що очікують подію 4, яке пов'язане зі знищенням транзакта, що переходить після затримки до події 4. Помітимо, що всі транзакти, що перебувають у списку *waitl[4]* у момент виникнення четвертої події будуть переправлені до події 3. Моделювання буде завершено, якщо число очікуючих транзактів буде більше або дорівнює 10.

```
void main(){
     //variables
     double j,lambda;
     ...;
     while(waitl[4]->p->ll<10) {
          plan();
          switch(sysevent) {
               case 1: create(randexp(lambda,v1)); break;
               case 2: if(future->p->ll>1) wait(4); break;
               case 3: delayt(j); break;
               case 4: destroy(); break; }
               }
     }
     ...
}
```

2.5. 2 Функції *accept* й *send*

У системі СИМ-СИ визначені:

- тип "сигнал": *signal=1..signmax*;

- масив списків сигналів: *array<min_signal,max_signal,plistt> signlist*;

- дві функції *accept(signal sg), send(signal sg)*;

Функція void *accept(signal sg)* переводить активний транзакт у список сигналу *sg* зі списку *current*. Транзакт перестає бути активним і залишається в цьому списку, очікуючи виконання функції ***void*** *send(signal sg)*. При виконанні функції **accept,** сигнали, які раніше надходили від функції *send,* не враховуються.

У відмінності від блокування просування транзактів чекаючих події, коли при здійсненні події активізується тільки один транзакт із очікуючих його, при формуванні сигналу функцією *send* всі транзакти зі списку сигналу sg переводяться в список *current*.

Фактично, *accept* - очікування сигналу *sg*, а *send* - посилка цього сигналу.

signmax - системна константа, визначається при генерації CIM-CI.

2.6 Зміна порядку здійснення подій. Функція *next*

Якщо при обробці події *n* зустрілася функція ***void*** *next(event e)*, де *e* - номер події, що повинне здійснитися після *n*, то значення *trans->eve* (а отже й *current->first->eve*) установлюється рівним *e*, а *trans=nil*. Далі автоматично вибирається активний транзакт: *trans=current->first*. Системна змінна *sysevent* одержить значення *sysevent=trans->eve*, тобто стане дорівнювати *e*.

2.7 Використання приладів

Використання приладів дозволяє зробити процес обробки більш наочним; крім того, по приладах збирається статистика, що може бути використана для оцінки їхнього функціонування. Прилади відносяться до об'єктів, названими у системі пристроями, тому функції звертання до них діляться на дві групи: функції уведення й функції виводу транзактів. Порядок запису функцій у моделі довільний, важливо, щоб логіка моделювання не була порушена спробою виводу транзакта з вільного приладу.

2.7.1 Функції уведення транзактів у прилад

2.7. 1.1 Захват приладу. Функція *infac*

Функцією ***void*** *infac(pfacility&)* здійснюється захват приладу. Якщо прилад був вільний, то її дія аналогічна функції *seize*. Якщо ж у приладі перебував транзакт, то його обробка переривається й він надходить у список перерваних транзактів даного приладу (*f->inter*), а полю *status* приладу привласнюється значення *preempted*. Після звільнення приладу перерваний транзакт повертається в нього на дообробку. Список перерваних транзактів будується в порядку, зворотньому надходженню. Якщо мають місце кілька рівнів захвату, то перервані транзакти обробляються з дисципліною LIFO. Число рівнів захвату не обмежено.

Якщо транзакт, що займає прилад перебуває в списку *current*, тобто захват повинен відбутися в той же момент модельного часу, що й звільнення приладу, то активний транзакт не захоплює прилад, а заноситься у відповідний список *waitl* і чекає звільнення приладу.

Приклад 7.1. Обробка транзактів з абсолютними пріоритетами

```
...;
if(f->transpoint!=nil)
     if(trans->prty>f->transpoint->prty)
          infac(f);
     else seize(f);
else seize(f);
```

2.7. 1.2 Заняття приладу. Функція *seize*

Для реалізації дисципліни з відносними пріоритетами використовується функція ***void*** *seize(pfacility&)*. Якщо статус приладу дорівнює *free*, то його займає транзакт вищого пріоритету із числа, що претендують на його використання. Якщо прилад зайнятий (його статус дорівнює *seized*), то транзакт заноситься в список приладу останнім у своєму класі пріоритетів.

2.7. 2 Функція *outfac*. Вивід транзакта з приладу

Всі події, здійснювані при занятті приладу, розміщуються після події, що містить функцію ***void*** *infac(pfacility&)*. Після того, як вони виконані, прилад необхідно

звільнити, щоб забезпечити подальше просування транзакта, що займає прилад, і зробити прилад доступним для інших транзактів. Для цього використовується функція ***void*** *outfac(pfacility&)*.

Прилад звільнюється поза залежністю від яких-небудь умов, однак функція перевіряє, який транзакт звільняє прилад. Якщо прилад звільняється не тим транзактом, що його займав, то друкується повідомлення про помилку.

Якщо прилад був захоплений, то повертається на дообробку транзакт, що раніше його займав.

Якщо список приладу, що звільняє, містить транзакти, то **функція** outfac робить заняття приладу першим транзактом із цього списку.

Функція *outfac* звільняє прилад, зайнятий як функцією *seize*, так і функцією *infac*.

2.8 Реєстрація черг. Функції *inqueue*, *outqueue*

Черги можуть утворюватися в будь-яких частинах моделі, де виникає необхідність затримки або блокування транзактів. Спеціального обліку їхньої динаміки в загальному випадку не ведеться. Засобами для збору статистики й оцінки динаміки черг служать об'єкти типу *queue*.

Для входу в чергу використовують функцію ***void*** *inqueue(pqueue&),* для виходу з неї - ***void*** *outqueue(pqueue&)*.

Функції дозволяють безумовно вставати в чергу й виходити з неї.

2.9 Накопичувачі (багатоканальні пристрої). Функції *enter* й *leave*

Для заняття транзактом багатоканального пристрою використовується функція ***void*** *enter(pstorage s,**int** c),* де c – кількість займаних осередків

Для звільнення багатоканального пристрою використовується функція ***void*** *leave(pstorage s,**int** c),* де c – кількість осередків, що звільняють

В відмінність від приладу накопичувач може бути звільнений не тим транзактом, яким був зайнятий.

2.10 Побудова гістограм. Функції *tabulate*, *newhist*, *prnhist*

Побудова гістограм у системі можлива для будь-якого типу скалярних даних, які можливо привести до типу ***double***.

З поняттям "гістограма" зв'язується об'єкт типу *histogram*. У моделі він визначається посиланням на нього:

phistogram h;

Перед використанням змінної *h* для табулювання значень деякої випадкової величини необхідно створити гістограму. Для цього служить функція ***void*** *newhist(phistogram&,**double**,**double**,hint2,**bool**,alfa)*:

newhist(<посилання на гістограму>,

<нижня межа зміни табулюємої величини>,

<верхня межа зміни табулюємої величини>,

<число інтервалів табулювання>,

<"ключ" печатки графіка>,

<символьне ім'я гістограми>);

Максимальне число точок табулювання задається системною константою *hint=33*.

Функція **tabulate** має формат запису:

void *tabulate(phistogram hist,**double** r),* де *hist* - посилання на гістограму, *r* - табулюєма величина. Функція записується в тій частині моделі, де вимірюється величина, що *цікавить* програміста, r.

Функція ***void*** *prnhist(phistogram)* служить для друку гістограми.

2.11. Створення, використання й обробка списків

У системі моделювання визначені поняття списків транзактів, приладів, черг, накопичувачів, гістограм і списку списків транзактів.

Для роботи з цими типами списків (крім списку списків транзактів) визначені функції створення списків, включення в них елементів й їхнє виключення. На списках задані функції проходження: вибір наступного й попереднього елементів зі списку. Всі списки в системі моделювання мають кільцеву структуру: перший елемент списку вказує на другий й останній, другий - на перший і третій... останній - на передостанній

і перший. Елементи списку не нумеруються. Щоб виділити в списку перший елемент, на нього вказує посилання *first* змінної типу список: *[посилання на список]->first*. Кільцева структура забезпечує пошук потрібного елемента в кожному із двох обраних програмістом напрямків.

Поняття списків визначається в CI, тому нижче дається лише короткий огляд функцій, список їхніх параметрів і спосіб звертання до них.

2.11.1 Створення списків

Для створення списку необхідно визначити відповідну змінну типу посилання на нього:

plistt lt	посилання на список	транзактів
plistq lq	посилання на список	черг
plistf lf	посилання на список	приладів
plists ls	посилання на список	накопичувачів
plisth lh	посилання на список	гістограм

Списки створюються за допомогою наступних функцій:

void *newtlist(plistt&)*	створення списку транзактів;
void *newqlist(plistq&)*	створення списку черг;
void *newflist(plistf&)*	створення списку приладів;
void *newslist(plists&)*	створення списку накопичувачів;
void *newhlist(plisth&)*	створення списку гістограм.

Списки створюються порожніми без яких-небудь об'єктів, що їх наповнюють. Для того, щоб помістити об'єкт у список, необхідно попередньо його згенерувати.

2.11.2 Включення об'єктів у списки

Для включення об'єкта в список використовуються наступні функції:

void *inlt(plistt&,ptransact)*	для транзактів
void *inlf(plistf&,pfacility)*	для приладів
void *inlq(plistq&,pqueue)*	для черг
void *inls(plists&,pstorage)*	для накопичувачів
void *inlh(plisth&,phistogram)*	для гістограм

2.11.3 Видалення об'єктів зі списків

На объект, що видаляється, завжди вказує голова списку. наприклад: *l->first* - посилання на перший елемент списку l.

Таке визначення дозволяє до видалення елемента зі списку запам'ятати його присвоюванням *pl=l->first;* де *pl* - змінна посилального типу (*ptransact, pfacility, pqueue, pstorage, phistogram*).

Виключити елемент зі списку можна за допомогою наступних функцій:

void *outtlist(plistt&)*

void *outflist(plistf&)*

void *outqlist(plistq&)*

void *outslist(plists)*

void *outhlist(plisth)*

Покажчик списку *first* після видалення вказує на наступний один по одному об'єкт у списку. Якщо объект, що видаляє, єдиний у списку *l*, то встановлюється *l->first=nil*.

2.11.4 Перегляд елементів списку. Сканування

Доступ до елемента списку визначається посиланням на нього й записом після точки поля, значення якого необхідно програмістові, наприклад: *quelist->first->lq* - поточна довжина першої черги в списку черг *quelist*, *current->first->pr[1]* - перший дійсний (double) параметр транзакта, що стоїть першим у списку *current*.

Посилання first указує на голову списку. Для її просування вправо необхідно привласнити: *l->first=l->first->sled*

Для просування посилання first уліво привласнюється: *l->first=f->first->pred*

2.11.5 Списки користувача. Організація різних дисциплін обслуговування за допомогою списків користувача

Список користувача створюється функцією ***void*** *newuserlt(plistt& list,alfa name)*.

При створенні список користувача заноситься в системний список списків транзактів *userlist*.

Функції ***void*** *inlfifo(plistt)* і ***void*** *inllifo(plistt)* виводять активний транзакт зі списку *current* і поміщають його в список користувача відповідно останнім або першим у списку.

Користувач може написати свої функції, що впорядковують список по якій-небудь ознаці, наприклад, у порядку зростання першого цілочисельного параметра:

```
void inlpi1(plistt lt) {
     ptransact t;
     outtlist(current);
     trans->testprty=false;    // !!!
     if(lt->first==nil)
          inlt(lt,trans);
     else if(lt->first->pi[1] > trans->pi[1])
               inlt(lt,trans);
          else {
               t=lt->first; scanlt(lt);
               while(lt->first->pi[1] >= trans->pi[1])
                    scanlt(lt);
               inlt(lt,trans);
               lt->first=t;
               }
     trans=nil;
}
```

Функція ***void*** *outuserlt(plistt)* поміщає транзакт, що стоїть першим у списку користувача в список *current*.

Транзакт має поле *testprty*, що може приймати значення *true* або *false*. При виконанні функції *priority* (див. п. 2.13. 3) транзакти, значення поля *testprty* у яких *true*, переміщуються в списках відповідно до нового значення пріоритету. Якщо список упорядковується не по пріоритетах, а по якій-небудь іншій ознаці, то перед включенням транзакта в список його полю *testprty* необхідно привласнити значення *false*.

2.12 Модельне середовище

2.12.1 Створення модельного середовища

Сукупність приладів, черг, накопичувачів й інших об'єктів моделі являє собою модельне середовище. Для створення модельного середовища написані спеціальні функції.

Для створення черги використовується функція *newqueue(<посилання на чергу>,<ім'я>)*. Її параметрами є змінна типу посилання на чергу й ім'я черги з восьми символів. Посилальна змінна визначається в такий спосіб:

pqueue q1;

...

newqueue(q1,'q1 ');

Значення q1 установлюється на створену чергу.

Створення приладів і накопичувачів здійснюється аналогічним чином:

pfacility f1;

pstorage st1;

...

newfac(f1,'f1 ');

newstorage(sm1,'sm1 ');

Створені об'єкти включаються в системні списки *quelist*, *faclist* й *stlist*. Якщо користувачеві треба помістити об'єкт у який-небудь інший список черг, приладів або накопичувачів, необхідно попередньо видалити його з відповідного системного списку.

Створення гістограм було розглянуто вище.

2.12.2 Знищення черг, приладів, накопичувачів і гістограм

Об'єкти, виключені зі списків можуть бути знищені за допомогою функцій

void *destrs(pstorage&)*

void *destrf(pfacility&)*

void *destrq(pqueue&)*

void *destrh(phistogram&)*

при цьому виділена під об'єкт пам'ять звільняється.

Користуватися цими функціями треба обережно. При знищенні приладу або накопичувача можуть втратитися транзакти, що перебувають у їхніх списках (буде видане відповідне повідомлення про помилку). Якщо в ході моделювання буде звертання до знищенного об'єкту, то виконання програми буде перервано.

2.13 Ансамблі

2.13. 1 Створення ансамблів. Функція *split*

Транзакти можуть уводитися в модель як функціями *create* й *initcreate*, так функцією ***void*** *split(**int** n,event e)*, де *n* - число створюємих транзактів, *e* - номер події, у яке направляються знову створені транзакти. Активний транзакт, на який указує посилання *trans*, називається транзактом-батьком, а створені тразакти - нащадками.

Всі нащадки мають такі ж значення параметрів і пріоритет як у батька. Транзакти, уведені у модель функцією *split* разом із транзактом-батьком є членами одного ансамблю. Якщо який-небудь із нащадків буде оброблений функцією *split* і сам стане батьком, то всі його нащадки будуть належати до того ж ансамблю.

2.13. 2 Збір членів ансамблю. Функція *assemble*

Функцією ***void*** *assemble(**int** n)* здійснюється збір *n* членів ансамблю. При вході першого члена ансамблю в подію, що містить функцію *assemble*, він заноситься в системний список *ass[sysevent]*.

Наступні члени цього ансамблю виводяться функцією *assemble* у список *delist*.

При вході n-го члена ансамблю в подію, що містить функцію *assemble* перший член ансамблю, що збирається, видаляється зі *списку ass[sysevent*] і заноситься в *список* delist, а n-ий член ансамблю продовжує просуватися по моделі далі.

2.13.3 Зміна пріоритету всіх членів ансамблю. Функція *priority*

Для зміни пріоритету активного транзакта й всіх транзактів - членів того ж ансамблю служить функція ***void*** *priority(prtyrange p)*, де p - нове значення пріоритету. При цьому інші члени ансамблю переміщуються в списках відповідно до нового значення пріоритету.

Призначення пріоритету окремому транзакту виробляється простим присвоюванням:

trans->prty=<новий пріоритет>;

2.13. 4 Зміна значення параметра всіх членів ансамблю. Функція *parmans*

Для зміни значення параметра всіх членів ансамблю служить функція ***void*** *parmans(parmtype,**int**)*. Попередньо необхідно привласнити нове значення даному параметру активного транзакта. Після виконання функції *parmans* призначений параметр всіх членів ансамблю приймає те ж значення, що й відповідний параметр активного транзакта.

2.14 Генератори випадкових чисел

У процесі моделювання майже завжди необхідна генерація випадкових величин. Для цього в системі існують функції:

double *rand01(**long**& v)* - генератор випадкових чисел, рівномірно розподілених в інтервалі [0;1). v - число-"джерело".

double *randab(**double** a, **double** b, **long**& v)* - генератор випадкових чисел, рівномірно розподілених в інтервалі [a;b). v - число-"джерело".

double *randexp(**double** lambda, **long**& v)* - генератор експоненціально розподілених випадкових чисел з інтенсивністю lambda. v - число-"джерело".

double *randnorm(**double** xmean, **double** disp, **long**& v)* - генератор випадкових чисел, розподілених за нормальним законом із середнім xmean і дисперсією disp. v - число-"джерело".

Приклад 14.1. Присвоєння змінної *r* значення випадкової величини, рівномірно розподіленої в інтервалі [4;7)

```
r=randab(4.0,7.0,v)
```

Закон розподілу випадкових чисел може бути заданий таблицею. Для цього в системі визначений тип "таблиця" у такий спосіб:

```
struct table {
     array<1,hint,double> x,p;
     hint2 ihint;
};
```

Для роботи з генераторами випадкових чисел, розподілених відповідно до таблиці, необхідно задати таблицю.

Для дискретного розподілу *ihint* відповідає кількості значень, які може приймати випадкова величина. *h* - масив значень, *p* - масив імовірностей, причому *p[1]* - імовірність присвоєння випадковій величині значення *x[1]*, *p[2]* - імовірність присвоєння значення *x[2]* за умови, що їй не привласнене значення *x[1]*, і т.д., так що *p[ihint]=1.0* (імовірність присвоєння випадковій величині значення *x[ihint]* за умови, що вона вона не прийняла жодне з попередніх значень.

Реалізує дискретний табличний розподіл функція ***double*** *randdtable(table t,* ***long****& v)*, де *t* - таблиця, *v* - число-"джерело".

Для безперервного табличного розподілу генератор - функція ***double*** *randtable(table t,* ***long****& v)* (*t* - таблиця, *v* - "джерело") - дає випадкові числа, з імовірностями *p[i]* які потрапляють в інтервали від *x[i-1]* до *x[i]*. При цьому в таблиці повинне бути *p[1]=0.0.*

2.15 Процес моделювання. Функція *plan*

Процес моделювання полягає в багаторазовому повторенні функції ***void*** *plan****()*** і блоку вибору події.

Усередині блоку вибору події оператор ***switch****(sysevent)* вибирає подію, номер якого збігається зі значенням *sysevent*.

Функція *plan*:

1) переводить перший транзакт зі списку *waitl[sysevent]*, якщо цей список непустий, у список *current*; при цьому встановлюється *waitevent[sysevent]=****false****;* якщо список *waitl[sysevent]* був порожній, установлюється *waitevent[sysevent]=****true****;*

2) перевіряє *trans==nil*:

якщо *trans==nil*, то функція *plan* виконує (у порядку запису) одне з наступних дій:

- намагається вибрати активний транзакт зі списку *current*;

- якщо список *current* порожній, то вибирає активний транзакт зі списку *future* і встановлює нове значення модельного часу, потім вибирає зі списку *future* всі транзакти, значення поля *nexttime* яких збігається з новим значенням модельного часу;

- призначає *sysevent=trans->eve*.

Якщо *trans!=nil*, те призначається *sysevent=succ(sysevent)*.

2.16 Створення системного середовища. Функція *initlist*

Перед початком моделювання необхідно створити системні списки, створити необхідну кількість транзактів, помістивши їх у пасивний буфер і привласнити початкові значення системним змінним.

Це здійснюється функцією ***void*** *initlist(**int** n)*. Параметр цієї функції *n* - число транзактів, необхідне для моделювання.

2.17 Структура моделі

Модель являє собою функцію на C++ й оформлюється відповідним чином. Всі моделі мають загальні риси побудови. Типова структура моделі приводиться нижче:

```
void model() {
     //variables
     pfacility  <посилання на прилади>;
     pqueue     <посилання на черзі>;
     pstorage   <посилання на накопичувачі>;
     plistt     <посилання на списки транзактів>;
     phistogram <посилання на гістограми>;
     ...
     initlist(<у транзактів>);
     newfac(...);              ...;
     newqueue(...);            ...;
     newstorage(..., ...);     ...;
     newuserlt(...);           ...;
     newhist(..., ...);        ...;
     initcreate(<номер події>,<час>); ...;
     обмеження числа повторень, наприклад, за часом:
     while(systime<...) {
          plan();
          switch(sysevent){
               case 1: <дія для події 1>; break;
               case 2: <дія для події 2>; break;
               case 3: <дія для події 3>; break;
               ...
               case n: <дія для події N>; break;
               }
          }
     printall();
}
```

2.18 Скидання статистики й очищення системного й модельного середовища. Функції *resetall* й *clear*

Функція ***void*** *resetall()* служить для скидання статистики, накопиченої в процесі моделювання. Для всіх списків, черг, приладів і накопичувачів обнуляются число входів, час зайнятості; час попереднього обігу стає рівним *systime*, і т.д.

Функція ***void*** *clear()* повертає усі транзакті в список *delist*, скидає всю накопичену статистику, тобто повертає систему в стан, що передує початку моделювання.

3 ВИВІД РЕЗУЛЬТАТІВ МОДЕЛЮВАННЯ

Для друку статистики, зібраної в результаті моделювання, служить функція ***void*** *printall*(). Статистика, що роздруковується цією функцією при *errtest==false*, містить:

- значення абсолютного системного часу у вигляді

systime=<абсолютний системний час>;

- номер поточної події у вигляді

sysevent=<номер поточної події>;

- для всіх виконуваних подій: кількість виконань кожної події у вигляді

event <номер події>

total <число виконань>

- для всіх черг у моделі: ім'я черги, число входів, число входів з нульовим часом очікування, максимальна довжина, середній час очікування, відсоток входів у порожню чергу, поточна довжина, середній час очікування без обліку нульових входів, середня довжина черги;

- для всіх приладів у моделі: ім'я приладу, число входів, середній час обробки транзакта в приладі, завантаження, число захватів;

- для всіх накопичувачів у моделі: ім'я накопичувача, ємність, завантаження, середній час перебування транзакта в накопичувачі, поточний уміст, максимальний уміст, середній уміст, число входів;

- для всіх списків користувача: ім'я списку, поточна довжина, максимальна довжина, транзакти, що перебувають у списку з полями: номер, *prty* (пріоритет), *eve* (подія ініціалізації), *nexttime* (час потрапляння в список), *ans* (номер ансамблю), параметри транзакта.

- для видачі на друк розширеної (аварійної) статистики необхідно виконати присвоювання *errtest=true*; при цьому додатково друкуються системні списки по посиланнях *current*, *future*, *waitl[i]*, де *i=1..iEveMax* (номер останньої виконуваної події) і інші з розміщеними в них транзактами. Друк системних списків виконується по аналогії з друком списків користувача, який описано вище.

У системі існують функції для виводу на друк деякої частини статистики:

void *prnt1(ptransact t)* - друк полів транзакта. t - посилання на транзакт

void *prnlt(plistt& lt)*; - друк полів списку транзактів і транзактів, що перебувають у ньому . *lt* - посилання на список

void *prwaitl()* - друк списків, що очікують транзактів (списків *waitl[i]*)

void *prnq(pqueue q)* - друк полів черги. *q* - посилання на чергу

void *prnlq(plistq lq)* - друк списку черг. *lq* - посилання на список

void *prns(pstorage st)* - друк полів накопичувача. *st* - посилання на накопичувач

void *prnls(plists ls)* - друк списку накопичувачів. *ls* - посилання на список

void *prnf(pfacility f)* - друк полів приладу. *f* - посилання на прилад

void *prnlf(plistf ls)* - друк списку приладів. *lf* - посилання на список

void *prnuserlt()* - друк списків користувачів

void *lfprint(plistf lf)* - друк полів транзактів, що входять у списки приладів. *lf* - посилання на список приладів

void *lsprint(plists ls)*; - друк полів транзактів, що входять у списки накопичувачів. *ls* - посилання на список накопичувачів.

4 ЗАСОБИ ВІДЛАГОДЖЕННЯ

4.1 Діагностика помилок

Під час виконання моделі можуть виникати помилки, які реєструються C++ (приводять до аварійної зупинки), або не реєструються C++, але порушують логіку моделювання. Помилки першого виду можуть бути наслідком помилок другого виду, тому частина дій, які можуть викликати виникнення помилки, реєструється функціями моделювання. Якщо ці помилки серйозні, вони можуть завершитися функцією *exit()*. При цьому роздруковується розширена (аварійна) статистика. Коди всіх помилок з поясненнями виводяться на печатку за допомогою функції ***void*** *error(****int*** *errcode)*, де *errcode* - код помилки. Повний перелік кодів помилок і повідомлень, які регеструються, приводиться в додатку.

Появлення регеструємої помилки, у результаті якої продовження моделювання стає неможливим, приводить до запису аварійної статистики у файл *error.html*.

4.2 Трасування

У системі існує можливість "покрокового трасування". Для цього необхідно виконати функцію ***void*** *starttrace(),* інформація, *що* друкує при цьому, наведена в додатку.

ДОДАТОК 1. КОДИ Й ПОВІДОМЛЕННЯ ПРО ПОМИЛКИ

101: негативне значення часу затримки;

102: у списку *delist* немає транзактів

201: спроба знищити неіснуючий транзакт;

202: спроба знищити неіснуючий прилад;

203: спроба знищити неіснуючу чергу;

204: спроба знищити неіснуючий накопичувач;

205: знищується зайнятий прилад;

206: прилад знищується разом з очікуючими транзактами;

207: прилад знищується разом з перерваними транзактами;

208: накопичувач знищується разом з очікуючими транзактами;

209: знищується непустий накопичувач;

301: спроба зайняти заповнену чергу;

302: спроба звільнити порожню чергу;

402: спроба звільнити порожній прилад;

403: при спробі зайняти прилад транзакт відсутній;

404: при спробі звільнити прилад транзакт відсутній;

405: прилад звільняються не тим транзактом, яким був зайнятий;

501: негативний час затримки;

502: спроба затримати відсутній транзакт;

511: при виконанні функції *accept* транзакт відсутній;

512: спроба вивести транзакт із неіснуючого списку *signlist*;

601: наступна подія виходить за діапазон припустимих;

602: спроба планування при порожніх списках *future, current, waitl[i]*;

701: спроба видалити об'єкт із порожнього списку;

702: спроба помістити в список користувача транзакт відсутній;

711: у функції *split* транзакт відсутній;

712: при виконанні функції *assemble* активний транзакт відсутній;

713: неприпустиме значення параметра *n* функції *split*;

714: неприпустиме значення параметра *n* функції *assemble*;

801: параметр функції *next e<1*;

802: при звертанні до функції *next* активний транзакт відсутній

901: спроба сканування порожнього списку;

902: спроба сканування неіснуючого списку

7301: у табличному розподілі повинно бути *p[1]=0.0*

7302: у табличному розподілі неприпустимо *p[i]=p[i-1]*

ДОДАТОК 2. ПОВІДОМЛЕННЯ В РЕЖИМІ ТРАСУВАННЯ

initcreate - транзакт <номер> поміщений у модель.

eve= <подія ініціалізації>

create - створений транзакт <номер>

eve= <подія ініціалізації>

nexttime= <час ініціалізації>

destroy - знищений транзакт <номер>

delayt - транзакт <номер> затриманий на <час затримки>

nexttime= <час ініціалізації>

wait - транзакт <номер> eve= <подія ініціалізації> поміщений у список *waitl*[<очікувана подія>]

next - транзакт <номер> спрямований на виконання події <номер події>

infac - прилад <ім'я> захоплене транзактом <номер> обробка транзакта <номер> перерваний або транзакт <номер> зайняв прилад <ім'я> або транзакт <номер> намагався захопити прилад <ім'я>

outfac - транзакт <номер> звільнив прилад <ім'я> або транзакт <номер> звільнив прилад <ім'я> транзакт <номер> повернутий у прилад <ім'я> на дообробку

outqueue - транзакт <номер> покинув чергу <ім'я>

inqueue - транзакт <номер> устав у чергу <ім'я>

seize - транзакт <номер> намагався зайняти прилад <ім'я> або транзакт <номер> зайняв прилад <ім'я>

enter - транзакт <номер> зайняв <число> осередків у накопичувачі <ім'я> або транзакт <номер> намагався ввійти в накопичувач <ім'я>

leave - транзакт <номер> звільнив <число> осередків накопичувача <ім'я> або транзакт <номер> звільнив <число> осередків накопичувача <ім'я>

транзакт <номер> переведений зі списку накопичувача в *current*

- *sysevent= <подія> systime= <час>*

split - для транзакта <номер> ансамбль <номер> створений <число> нащадків, які спрямовані за адресою <подія>

assemble - розпочато збір членів ансамблю <номер> у події <подія>

транзакт <номер> ансамбль <номер> виведений з моделі в події <подія> або транзакт <номер> ансамбль <номер> виведений з моделі в події <подія> або закінчена збір членів ансамблю <номер> у події <подія>

parmans - параметру <тип і номер параметра> всіх членів ансамблю <номер> привласнений значення <число або ім'я> <перелік всіх членів ансамблю>

priority - всім членам ансамблю <номер> привласнений пріоритет <число> <перелік всіх членів ансамблю>

accept - транзакт <номер> поміщений у список чекаючих сигналу <номер сигналу>

send - по сигналу <номер сигналу> транзакти переведені в список *current* чи немає транзактів, що чекають сигналу <номер сигналу>

ДОДАТОК 3. СИСТЕМНІ КОНСТАНТИ Й ТИПИ ДАНИХ

```
// Константи й структури
const int evemax = 1024;
const int signmax = 64;
const int hint = 33;
/*Транзактно-орієнтовані Константи*/
const int prtymax = 10;
const int mptb = 2;
const int mptr = 3;
const int mpti = 3;
const int mptf = 2;
const int mptq = 2;
const int mpts = 2;
typedef signed char prtyrange;
const int min_prtyrange = -prtymax;
const int max_prtyrange = prtymax;
typedef array<1,8,char> alfa;
typedef unsigned short event;
const int min_event = 0;
const int max_event = evemax;
typedef unsigned char signal;
const int min_signal = 1;
const int max_signal = signmax;
typedef unsigned char hint2;
const int min_hint2 = 2;
const int max_hint2 = hint;
typedef array<0,hint,int> harr;
enum parmtype {parmb,parmi,parmr,parmf,parmq,parms, last_parmtype};
typedef struct transact* ptransact;
typedef struct facility* pfacility;
typedef struct queue* pqueue;
typedef struct histogram* phistogram;
typedef struct storage* pstorage;
typedef struct listt* plistt;
typedef struct listf* plistf;
typedef struct listq* plistq;
typedef struct lists* plists;
typedef struct listl* plistl;
typedef struct listh* plisth;

struct facility {
    enum {free,seized,preempted} status;
    pfacility sled,pred;
    ptransact transpoint;
    int p,ci;
    alfa name;
    double timef,pretime,mtime,pro;
    plistt fl,inter;
    bool test;
};

struct storage {
```

```
    int s,ss,sf,sm,ci;
    double ut,smean,mtime,times,pretime;
    alfa name;
    pstorage pred,sled;
    plistt slt;
    bool test;
};

struct queue {
    enum {empty,full} status;
    int lq,mq,size,ci,co;
    pqueue sled,pred;
    alfa name;
    double timeq,pretime,lm,mtime;
    bool test;
};

struct transact {
    array<1,mptb,bool> pb;
    array<1,mpti,int> pi;
    array<1,mptr,double> pr;
    array<1,mptf,pfacility> pf;
    array<1,mptq,pqueue> pq;
    array<1,mpts,pstorage> ps;
    int nans,nom,ans;
    ptransact predans,sledans,pred,sled;
    event eve;
    prtyrange prty;
    double nexttime;
    plistt translist;
    bool testprty;
};

struct listl {
  plistt first;
  int ll;
};

struct parml {
  double timel,pretime;
  int ci,co,ll,lm;
  alfa name;
};

struct lists {
  pstorage first;
  parml p;
  plists sled,pred;
};

struct listt {
  ptransact first;
  parml p;
  plistt sled,pred;
```

```
};

struct listq {
  pqueue first;
  parml p;
  plistq sled,pred;
};

struct listh {
  phistogram first;
  parml p;
  plisth sled,pred;
};

struct listf {
  pfacility first;
  parml p;
  plistf sled,pred;
};

struct histogram {
  bool graf;
  alfa name;
  unsigned total;
  hint2 ihint;
  double sum,sumsqr,maxx,minx;
  phistogram sled,pred;
  harr x;
};

struct table {
  array<1,hint,double> x,p;
  hint2 ihint;
};

struct typtrace {
  pfacility f;
  pqueue q;
  ptransact t;
  pstorage s;
  event e;
  double r;
  int n;
};
typedef array<1,10,char> a10;

struct typdata {
  a10 menu;
  int act,x,y;
};
enum typsel {nch,mch,rch, last_typsel};

struct typnmr {
  int i,x,y;
```

```
  bool test;
};

struct typdiarec {
  array<1,10,typdata> data;
  a10 menubase,menubuf;
  bool stoptest,endtest,test,errinput;
  alfa name;
  typnmr n,m,r;
  int k,nom,number,actbase,actbuf,xbase,ybase;
  double endtime,time,stoptime,steptime;
  plistt lt;
  pqueue que;
  pfacility fac;
  pstorage st;
  phistogram hist;
};
```

ДОДАТОК 4. СИСТЕМНІ ЗМІННІ

```
//variables
bool trace,          ключ включення трасування
 errtest;            ключ розширеної (аварійної) видачі
double resettime,    час останнього скидання статистики
 systime;            системний час
event ievemax,       остання виконувана подія
 sysevent;           поточна подія
int itransmax;       кількість транзактів в  delist
array<1,evemax,bool> waitevent;     ознаки здійснення подій
array<1,evemax,int>  maxevent;      кількості виконань подій
listl userlist;      список списків користувача
ptransact trans;     посилання на транзакт, що просувається
plistt delist,       посилання на пасивний буфер
 current,            посилання на ланцюг поточних подій
 future;             посилання на ланцюг майбутніх подій
plistq quelist;      посилання на сист. список черг
plistf faclist;      посилання на сист. список приладів
plists stlist;       посилання на сист. список накопичувачів
plisth histlist;     посилання на сист. список гістограм
array<min_signal,max_signal,plistt> signlist посилання на списки
                     транзактів, очикуючих сигналів
array<1,evemax,plistt> ass,   посилання на списки ансамблів, що
                              збираються
     waitl;          посилання на списки транзактів, очикуючих подій
long v0,             джерела генераторів випадкових чисел
v1,v2,v3,v4,v5,v6,v7,v8,v9,v10,v11,v12,v13,v14,v15;
typtrace tracerec;   керування трасуванням
double eventall;     загальне число подій, що відбулися
ofstream outfile;
ofstream f;
double ctime1,ctime2,realtime;
double msystime,mrealtime,evespeed;
```

ДОДАТОК 5 ЗАГОЛОВКИ СИСТЕМНИХ ФУНКЦІЙ

Функція друку заголовка списку транзактів

void *prnlt1(plistt&);*

Функція друку списку транзактів

void *prnlt(plistt&);*

Функція друку гістограми

void *prnhist(phistogram);*

Функція друку полів черги

void *prnq(pqueue);*

Функція друку полів накопичувача

void *prns(pstorage);*

Функція друку полів приладу

void *prnf(pfacility);*

Функція друку списків транзактів, що очікують подій

void *prwaitl(**void**);*

Функція друку списків користувача

void *prnuserlt(**void**);*

Функція друку списків транзактів, очикуючих звільнення приладів

void *lfprint(plistf);*

Функція друку списків транзактів, очикуючих звільнення накопичувачів

void *lsprint(plists);*

Функція друку списку черг. *lq* - посилання на список

void *prnlq(plistq lq);*

Функція друку списку накопичувачів. *ls* - посилання на список

void *prnls(plists ls);*

Функція друку списку приладів. *lf* - посилання на список

void *prnlf(plistf lf);*

Функція друку списку гістограм. *lh* - посилання на список

void *prnlh(plisth lh);*

Функція друку всієї статистики та стану системи

***void** printall();*

Функція видачі повідомлень у режимі трасування

***void** tracer(**int**);*

Функція видачі повідомлень про помилки. *errcode* - код помилки

***void** error(**int** errcode);*

Функція генератора випадкових чисел,рівномірно розподілених на інтервалі [0..1). *v* - число-джерело

***double** rand01(**long**& v);*

Функція генератора випадкових чисел, розподілених за експонентним законом.

***double** randexp(**double**,**long**&);*

Функція генератора випадкових чисел, рівномірно розподілених на інтервалі від *a* до *b*.

***double** randab(**double** a,**double** b,**long**&);*

Функція генератора випадкових чисел, розподілених за нормальним законом

***double** randnorm(**double**,**double**,**long**&);*

Функція розподілу Пуассона:

***int** randpoisson(**double**,**long** &);*

Функція розподілу:

***double** randdtable(table,**long** &);*

Функція розподілу

***double** randtable(table,**long** &);*

Функції включення об'єктів у списки:

***void** inlt(plistt&,ptransact);*	транзактів
***void** inlf(plistf&,pfacility);*	приладів
***void** inlq(plistq&,pqueue);*	черг
***void** inls(plists&,pstorage);*	накопичувачів
***void** inlh(plisth&,phistogram);*	гістограм

Функції вилучення об'єктів зі списків:

***void** outtlist(plistt&);*	транзактів

void *outflist(plistf&);* приладів

void *outqlist(plistq&);* черг

void *outslist(plists);* накопичувачів

void *outhlist(plisth);* гістограм

Функція включення транзакта в список *delist*

void *indelist(ptransact&);*

Функція включення транзакта в список *future*

void *infuture(ptransact&);*

Функція включення транзакта в список *current*

void *incurrent(ptransact&);*

Функція включення активного транзакта в список користувача в режимі FIFO. *lt* - посилання на список користувача

void *inlfifo(plistt lt);*

Функція включення активного транзакта в список користувача в режимі LIFO. *lt* - посилання на список користувача

void *inllifo(plistt lt);*

Функція переводу першого транзакта зі списку користувача в список *current*

void *outuserlt(plistt);*

Функція ініціалізації генератора транзактів. *e* - подія, у якій відбувається генерація, *r* - час генерації першого транзакта

void *initcreate(event e,**double** r);*

Функція створення транзактів. *r* - часовий інтервал між моментами створення транзактів

void *create(**double** r);*

Функція знищення активного транзакта

void *destroy();*

Функція затримки активного транзакта на час *r*

void *delayt(**double** r);*

Функція включення транзакта в список *waitl[e]*

void *inwaitl(ptransact&,event e);*

Функція блокування просування активного транзакта до здійснення події *e*

void *wait(event e);*

Функція призначення наступної події *e*

void *next(event e);*

Функція захвата приладу. *f* - посилання на прилад

void *infac(pfacility& f);*

Функція звільнення приладу. *f* - посилання на прилад

void *outfac(pfacility& f);*

Функція виходу із черги. *q* - посилання на чергу

void *outqueue(pqueue& q);*

Функція входу в чергу. *q* - посилання на чергу

void *inqueue(pqueue& q);*

Функція заняття приладу. *f* - посилання на прилад

void *seize(pfacility& f);*

Функція заняття накопичувача. *st* - посилання на накопичувач

void *enter(pstorage st,**int**);*

Функція звільнення накопичувача. *st* - посилання на накопичувач

void *leave(pstorage st,**int**);*

Функції створення списків по посиланнях :

void *newtlist(plistt&);* транзактів

void *newflist(plistf&);* приладів

void *newqlist(plistq&);* черг

void *newslist(plists&);* накопичувачів

void *newhlist(plisth&);* гістограм

Функції створення об'єктів по посиланнях:

void *newfac(pfacility&,alfa);* приладу

void *newqueue(pqueue&,alfa)* черги

void *newstorage(pstorage&,alfa,int s1);* накопичувача. *s1* - ємність накопичувача

void *newhist(phistogram&,**double**,**double**,hint2,**bool**,alfa);* гістограми

void *newuserlt(plistt&,alfa);*

Функції знищення об'єктів по посиланнях :

void *destrlt(plistt&);*	списків транзактів
void *destrf(pfacility&);*	приладу
void *destrq(pqueue&);*	черги
void *destrs(pstorage&);*	накопичувача
void *destrh(phistogram&);*	гістограми

void *split(**int**,event);*

void *assemble(**int**);*

void *priority(prtyrange);*

void *tabulate(phistogram,**double**);*

void *accept(signal);*

void *send(signal);*

void *resetall();*

void *clear();*

void *initlist(**int**);*

void *plan();*

ДОДАТОК 6. ПРИКЛАДИ МОДЕЛЕЙ

Приклад 1.

Усяка функція-модель починається функцією *initlist* - створенням системного середовища. Її перший параметр задає максимальне число транзактів у системі, а другий параметр - число подій.

функцією *initcreate* ініціалізуються генератори транзактів у подіях 1 й 8. Функцією *newfac* по посиланню *f* створюється прилад.

Оператор циклу ***while****(systime<28800)* обмежує час моделювання до 28800 одиниць модельного часу. Усередині циклу виконується функція *plan*, що призначає активний транзакт (на нього вказує посилання *trans*) і поточну подія (значення змінної *sysevent*), потім виконується одна з подій (та, що була обрана функцію *plan*).

У даному прикладі показана обробка транзактів з різними пріоритетами. пріоритет призначається присвоюванням значення полю транзакта *prty*.

```
#include "simc.h"

void main() {
     //variables
     pqueue q,q1,q2;     // оголошення модельного середовища
     pfacility f;
     initlist(50);       // создание системного середовища
     initcreate(1,0);    // ініціалізація генератора повідомлень
     initcreate(8,0);
     newqueue(q,"Queue  Q"); // створення модельного середовища
     newfac(f,"Fac  F");
     while(systime<28800){    // обмеження часу моделювання
          plan();
          switch(sysevent) {
               case 1: create(randab(60,780,v1)); break;
               case 2: inqueue(q); trans->prty=1; break;
               case 3: seize(f); break;
               case 4: outqueue(q); break;
               case 5: delayt(randab(210,390,v2)); break;
               case 6: outfac(f); break;
               case 7: destroy(); break;
               case 8: create(randab(120,600,v3)); break;
               case 9: inqueue(q); trans->prty=1; break;
               case 10: seize(f); break;
               case 11: outqueue(q); break;
               case 12: delayt(randab(70,130,v4)); break;
               case 13: outfac(f); break;
```

```
        case 14: destroy(); break;
        };
}
printall();
}
```

Загальні параметри середовища:	
Поточний час	2.89e+04
Поточна подія	1
Поточний транзакт	7
Усього подій	1.2e+03
Час моделювання	0 сек.
Середній час виконання події	0 сек/подія

ПОДІЯ	1	2	3	4	5	6	7	8	9	10
УСЬОГО	73	72	138	70	70	70	70	80	80	159

ПОДІЯ	11	12	13	14	15	16	17	18	19	20
УСЬОГО	79	79	78	78						

Черги					
Черга	Число входів	Макс. довжина	Порівн. час. оч.	Середня довжина	% вх. у порожню чер.
	З 0 час. оч.	Текущ.довжина	Без об. 0 вх.		
Queue	152	10	763	4.03	2.63
	4	3	783		

Прилади					
Прилад	Число входів	Порівн. час обробки	Завантаження	Число захватів	Стан
Fac F	149	194	0. 995	0	SEIZED

Приклад 2.

У цьому прикладі демонструється робота з багатоканальними пристроями. Функцією *newstorage* створюється багатоканальний пристрій і визначається його ємність. У функціях *enter* й *leave* другий параметр задає кількість осередків, займаних транзактом при вході в багатоканальний пристрій.

Модель тестується кілька разів при різній ємності багатоканального пристрою *men* і різному числі транзактів, що циркулюють у моделі. Після кожного прогону

функцією *printall* роздруковується зібрана статистика, функцією clear модель скидається у вихідний стан, функцією *destrs* знищується накопичувач *men*. Потім накопичувач *men* створюється знову з новою ємністю, у модель міститься потрібне число транзактів і починається наступний прогін.

```
#include "simc.h"
void main() {
     //variables
     pstorage nowon,men;
     initlist(56);
     newstorage(nowon,”Nowon   ”,50);
     for(int vv=53;vv<=55;vv++)
         for(int ww=3;ww<=4;ww++) {
               for(int k=1;k<=vv;k++)
                    initcreate(1,0);
               newstorage(men,”Men ”,ww);
               while(systime<6240) { // обмеження часу моделювання
                    plan();
                    switch(sysevent) {
                         case 1: enter(nowon,1); break;
                         case 2: delayt(randab(132,182,v1)); break;
                         case 3: leave(nowon,1); break;
                         case 4: enter(men,1); break;
                         case 5: delayt(randab(4,10,v2)); break;
                         case 6: leave(men,1); break;
                         case 7: next(1); break;
                         }
                    }
               printall();
               clear();
               destrs(men);
               }
          }
```

Нижче приводяться результати моделювання.

Загальні параметри середовища:	
Поточний час	6.24e+03
Поточна подія	6
Поточний транзакт	30
Усього подій	1.56e+04
Час моделювання	0 сек.
Середній час виконання події	0 сек/подія

ПОДІЯ	1	2	3	4	5	6	7	8	9	10

УСЬОГО	2921	1984	1936	2933	1934	1932	1931			

Накопичувачі							
Накопичувач	Емн.	Заван.	Порівн. час перебування	Уміст			Число входів
				Поточ.	Макс.	Середовищ.	
Nowon	50	0. 984	155	48	50	49	1984
Men	3	0. 728	7.04	2	3	2.2	1934

Загальні параметри середовища:	
Поточний час	6.24e+03
Поточна подія	6
Поточний транзакт	25
Усього подій	3.09e+04
Час моделювання	0. 016 сек.
Середній час виконання події	5. 1852e-07 сек/подія

ПОДІЯ	1	2	3	4	5	6	7	8	9	10
УСЬОГО	3108	1999	1949	2388	1949	1947	1946			

Накопичувачі							
Накопичувач	Емн.	Зав.	Порівн. час перебування	Уміст			Число входів
				Поточ.	Макс.	Середовищ.	
Nowon	50	0.99	155	50	50	49	1999
Men	4	0. 544	6.97	2	4	2.2	1949

Загальні параметри середовища:	
Поточний час	6.24e+03
Поточна подія	3
Поточний транзакт	17
Усього подій	4.69e+04
Час моделювання	0. 016 сек.
Середній час виконання події	3. 4151e-07 сек/подія

ПОДІЯ	1	2	3	4	5	6	7	8	9	10
УСЬОГО	3332	1989	1940	2920	1939	1937	1937			

Накопичувачі							
Накопичувач	Емн.	Зав.	Порівн. час перебування	Уміст			Число входів
				Поточ.	Макс.	Середовищ.	
Nowon	50	0.99	155	49	50	49	1989
Men	3	0. 731	7.06	2	3	2.2	1939

Загальні параметри середовища:	
Поточний час	6.24e+03
Поточна подія	6
Поточний транзакт	43
Усього подій	6.26e+04
Час моделювання	0. 016 сек.
Середній час виконання події	2. 5576e-07 сек/подія

ПОДІЯ	1	2	3	4	5	6	7	8	9	10
УСЬОГО	3503	2001	1951	2402	1951	1950	1949			

Накопичувачі							
Накопичувач	Емн.	Зав.	Порівн. час перебування	Уміст			Число входів
				Поточ.	Макс.	Середовищ.	
Nowon	50	0. 994	155	50	50	50	2001
Men	4	0.55	7.04	1	4	2.2	1951

Загальні параметри середовища:	
Поточний час	6.24e+03
Поточна подія	6
Поточний транзакт	45
Усього подій	7.9e+04
Час моделювання	0. 016 сек.
Середній час виконання події	2. 0246e-07 сек/подія

ПОДІЯ	1	2	3	4	5	6	7	8	9	10
УСЬОГО	3636	2009	1959	2992	1959	1958	1957			

Накопичувачі							
Накопичувач	Емн.	Зав.	Порівн. час перебування	Уміст			Число входів
				Поточ.	Макс.	Середовищ.	
Nowon	50	0. 994	154	50	50	50	2009
Men	3	0. 728	6.95	1	3	2.2	1959

Загальні параметри середовища:	
Поточний час	6.24e+03
Поточна подія	6
Поточний транзакт	36

Усього подій	9.5e+04
Час моделювання	0. 031 сек.
Середній час виконання події	3. 2627e-07 сек/подія

ПОДІЯ	1	2	3	4	5	6	7	8	9	10
УСЬОГО	3709	2008	1959	2444	1957	1954	1953			

Накопичувачі							
Накопичувач	Емн.	Зав.	Порівн. час перебування	Уміст			Число входів
				Поточ.	Макс.	Середовищ.	
Nowon	50	0. 996	155	49	50	50	2008
Men	4	0.55	7.01	3	4	2.2	1957

Приклад 3.

```
#include "simc.h"
void main(){
     //variables
     pqueue q,q1,q2;     // оголошення модельного середовища
     pfacility f;
     initlist(50);       // створення системного середовища
     initcreate(1,0);    // ініціалізація генератора повідомлень
     initcreate(8,0);
     newqueue(q, "Queue Q");  // створення модельного середовища
     newqueue(q1,"Queue Q1");
     newqueue(q2,"Queue Q2");
     newfac(f,   "Fac    F");
     while(systime<4800) {    // обмеження часу моделювання
          plan();
          switch(sysevent){
               case 1: create(randab(12,24,v1)); break;
               case 2: inqueue(q); inqueue(q1); trans->prty=2; break;
               case 3: seize(f); break;
               case 4: outqueue(q); outqueue(q1); break;
               case 5: delayt(randab(12,20,v2)); break;
               case 6: outfac(f); break;
               case 7: destroy(); break;
               case 8: create(randab(40,80,v3)); break;
               case 9: inqueue(q); inqueue(q2); break;
               case 10: seize(f); break;
               case 11: outqueue(q); outqueue(q2); break;
               case 12: delayt(randab(8,12,v4)); break;
               case 13: delayt(randab(12,24,v2)); break;
               case 14: outfac(f); break;
               case 15: destroy(); break;
               }
          }
      printall();
```

```
}
```

У цьому прикладі є події, у яких виконуються дві дії (події 2, 4, 9, 11). Така побудова моделі зменшує час моделювання. В одній події можна поєднувати функції, які не встановлюють *trans=nil*. Допускається також, щоб останньою в події стояла функція, що безумовно встановлює *trans=nil*. Наприклад:

```
case 3: seize(f); break;
case 4: inqueue(q); inqueue(q1); delayt(randab(12,20,v2)); break;
case 5: outfac(f); break;
```

Діагностика помилок

Error code # 102

Загальні параметри середовища:	
Поточний час	3.57e+03
Поточна подія	8
Поточний транзакт	10
Усього подій	1.85e+03
Час моделювання	0 сек.
Середній час виконання події	0 сек/подія

ПОДІЯ	1	2	3	4	5	6	7	8	9	10
УСЬОГО	199	199	395	197	197	196	196	61	60	75

ПОДІЯ	11	12	13	14	15	16	17	18	19	20
УСЬОГО	15	15	15	15	15					

Черги					
Черга	Число входів	Макс. довжина	Порівн. час. оч.	Середня довжина	% вх. у порожню чер.
	З 0 час. оч.	Поточ.довжина	Без об. 0 вх.		
Queue Q	259	47	341	24.7	0. 386
	1	47	342		
Queue Q	199	3	16.8	0. 935	0. 503
	1	2	16.9		
Queue Q	60	45	1.38e+03	23.1	0
	0	45	1.38e+03		

Прилади					
Прилад	Число входів	Порівн. час обробки	Завантаження	Число захватів	Стан
Fac F	212	16.9	1	0	SEIZED

Системні списки

Список CURRENT

Поточна довжина	Максимальная довжина
1	2

NUMB.	PRTY	EVE	NEXTTIME	ANS	NANS	TESTPRTY	TRANSLIST
10	0	8	3.57e+03	10	1	1	CURRENT
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							

Список приладу Fac F

Поточна довжина	Максимальная довжина
47	47

NUMB.	PRTY	EVE	NEXTTIME	ANS	NANS	TESTPRTY	TRANSLIST
41	2	3	3.56e+03	41	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
49	2	3	3.57e+03	49	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
16	0	10	817	16	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0							

NUMB.	PRTY	EVE	NEXTTIME	ANS	NANS	TESTPRTY	TRANSLIST
PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
18	0	10	882	18	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
17	0	10	955	17	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
3	0	10	1.01e+03	3	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
19	0	10	1.07e+03	19	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
15	0	10	1.14e+03	15	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
21	0	10	1.22e+03	21	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL							

NUMB.	PRTY	EVE	NEXTTIME	ANS	NANS	TESTPRTY	TRANSLIST
PF: NIL NIL PS: NIL NIL							
1	0	10	1.26e+03	1	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
22	0	10	1.31e+03	22	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
8	0	10	1.35e+03	8	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
20	0	10	1.42e+03	20	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
5	0	10	1.5e+03	5	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
23	0	10	1.55e+03	23	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							

NUMB.	PRTY	EVE	NEXTTIME	ANS	NANS	TESTPRTY	TRANSLIST
26	0	10	1.6e+03	26	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
2	0	10	1.67e+03	2	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
25	0	10	1.73e+03	25	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
27	0	10	1.78e+03	27	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
29	0	10	1.82e+03	29	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
31	0	10	1.87e+03	31	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
24	0	10	1.91e+03	24	1	1	Fac F

NUMB.	PRTY	EVE	NEXTTIME	ANS	NANS	TESTPRTY	TRANSLIST
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
32	0	10	1.98e+03	32	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
30	0	10	2.02e+03	30	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
28	0	10	2.09e+03	28	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
34	0	10	2.16e+03	34	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
33	0	10	2.2e+03	33	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
7	0	10	2.26e+03	7	1	1	Fac F
Параметри: PI: 0 0 0							

NUMB.	PRTY	EVE	NEXTTIME	ANS	NANS	TESTPRTY	TRANSLIST
PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
36	0	10	2.33e+03	36	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
37	0	10	2.39e+03	37	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
38	0	10	2.46e+03	38	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
4	0	10	2.54e+03	4	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
39	0	10	2.62e+03	39	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
35	0	10	2.7e+03	35	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE							

NUMB.	PRTY	EVE	NEXTTIME	ANS	NANS	TESTPRTY	TRANSLIST
PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
12	0	10	2.74e+03	12	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
11	0	10	2.81e+03	11	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
40	0	10	2.87e+03	40	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
14	0	10	2.93e+03	14	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
42	0	10	2.99e+03	42	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
13	0	10	3.06e+03	13	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL							

NUMB.	PRTY	EVE	NEXTTIME	ANS	NANS	TESTPRTY	TRANSLIST
PF: NIL NIL PS: NIL NIL							
44	0	10	3.14e+03	44	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
43	0	10	3.2e+03	43	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
9	0	10	3.28e+03	9	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
46	0	10	3.35e+03	46	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
45	0	10	3.4e+03	45	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
48	0	10	3.47e+03	48	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							

NUMB.	PRTY	EVE	NEXTTIME	ANS	NANS	TESTPRTY	TRANSLIST
47	0	10	3.52e+03	47	1	1	Fac F
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							

Список FUTURE

Поточна довжина	Максимальная довжина
2	3

NUMB.	PRTY	EVE	NEXTTIME	ANS	NANS	TESTPRTY	TRANSLIST
50	2	6	3.58e+03	50	1	1	FUTURE
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							
6	0	1	3.59e+03	6	1	1	FUTURE
Параметри: PI: 0 0 0 PR: 0 0 0 PB: FALSE FALSE PQ: NIL NIL PF: NIL NIL PS: NIL NIL							

Як видно з тросування, моделювання завершилося аварійно. Аналіз завантаження приладів показує, що прилад *f* - перевантажений. Перед ним збираються практично усі транзакти. Це пояснюється помилкою при визначенні часу генерації транзактів у події 1:

```
case 1: create(randab(12,24,v1)); break;
```

Змінивши інтервал можливих значень часів між створенням транзактів на наступний:

```
case 1: create(randab(120,240,v1)); break;
```

Одержимо:

Загальні параметри середовища:	
Поточний час	4.81e+03
Поточна подія	14
Поточний транзакт	1
Усього подій	850
Час моделювання	0 сек.
Середній час виконання події	0 сек/подія

ПОДІЯ	1	2	3	4	5	6	7	8	9	10
УСЬОГО	26	26	38	26	26	26	26	81	81	90

ПОДІЯ	11	12	13	14	15	16	17	18	19	20
УСЬОГО	81	81	81	81	80					

Черги					
Черга	Число входів	Макс. довжина	Порівн. час. оч.	Середня довжина	% вх. у порожню чер.
	З 0 час. оч.	Поточ.довжина	Без об. 0 вх.		
Queue Q	107	1	2.6	0. 0582	80.4
	86	0	13.3		
Queue Q	26	1	6.7	0. 0373	53.8
	14	0	14.5		
Queue Q	81	1	1.29	0. 0218	88.9
	72	0	11.6		

Прилади					
Прилад	Число входів	Порівн. час обробки	Завантаження	Число захватів	Стан
Fac F	107	24.4	0. 542	0	FREE