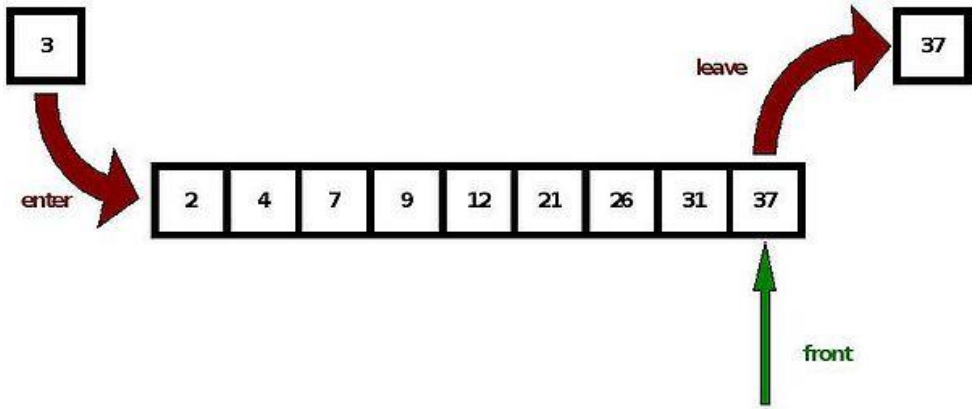
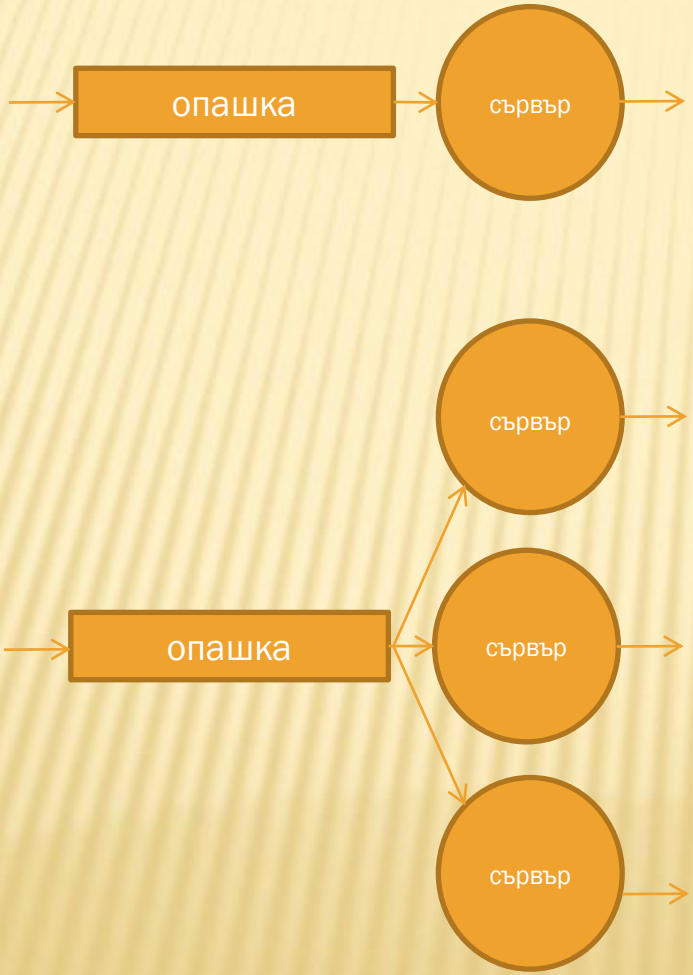
ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ
Катедра “Компютърни Системи”

“Високопроизводителни компютърни системи”



МОДЕЛИРАНЕ И СИМУЛАЦИИ С OMNET++

ТЕОРИЯ НА МАСОВОТО ОБСЛУЖВАНЕ (ТЕОРИЯ НА ОПАШКИТЕ) ТЕОРЕМА НА ЛИТЪЛ



- $E = \lambda T$, където

E - брой заявки, чакащи на опашката

λ - интензивност на входящите заявки (бр./единица време)

T – средно време на престой на заявките в системата

КЛАСИФИКАЦИЯ НА КЕНДАЛ

- A/B/C/K/N/D
- кратката версия A/B/C при която:

  K = ∞, N = ∞ и D = FIFO

  - A – разпределение на входящия поток от заявки
  - C – брой на сървърите
  - K – капацитет на системата. Ако липсва – неограничен брой
  - N – входна популация (възможен брой входни заявки). Ако липсва – неограничен брой
  - B – разпределение на обслужванията от сървъра
  - D – дисциплина на опашките

Означения

- За A и B имаме следните основни означения:
  - M (Markovian) – процес с експоненциално разпределение
  - D (Degenerative) – процес с детерминистично разпределение (или на фиксирани интервали)
  - E (Erlang) – процес с Ерлангово разпределение
  - G (General) – процес с общо разпределение
- За D (дисциплина на опашките) са валидни следните означения:
  - FIFO (First In First Out) – първи влязъл първи излязъл
  - LIFO (Last In First Out) – последен влязъл първи излязъл
  - SIRO (Service In Random Order) – обслужване без да има значене от реда на постъпване в опашката
  - PNPN (Priority service) – обслужване с приоритети

Примери:

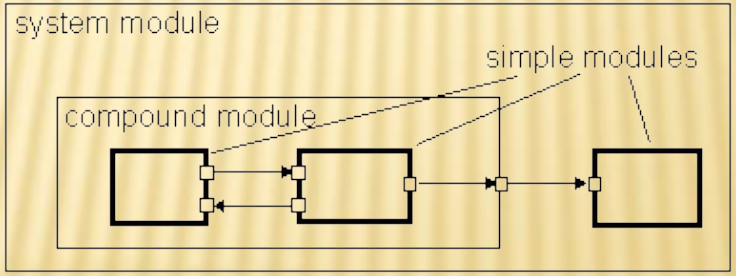
- M/D/1 – вендинг машина; D/M/1 – зъболекар

СИМУЛАЦИЯ НА ДИСКРЕТНИ СЪБИТИЯ. OMNET++

- Симулацията е метод, който цели имитирането на реални обекти и системи, което позволява тяхното безопасно/евтино/гъвкаво изследване. Omnet++ представлява симулатор на дискретни събития, предимно (но не единствено) насочен към симулация на мрежи.

- Основни елементи в Omnet++ са: прости модули (simple module), сложни модули (compound module) и мрежа (network).

- Простите модули се свързват помежду си с връзки, описани на езика NED, давайки съставни модули, които от своя страна могат да се свързват с други прости и съставни модули формирайки мрежа.



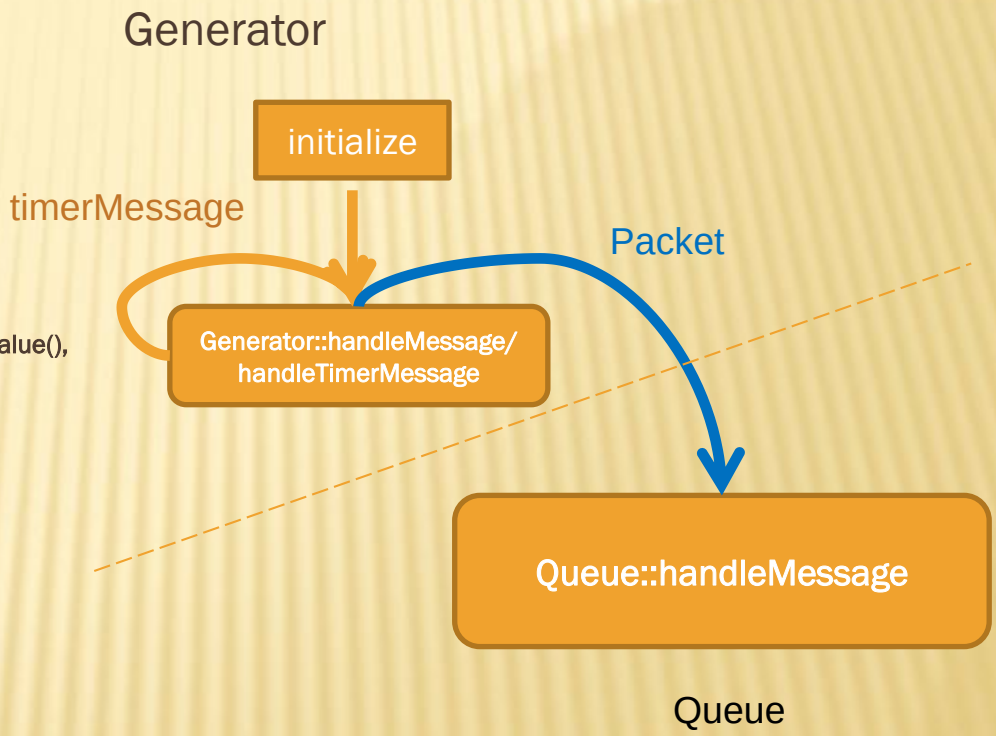
- *За повече информация относно* Omnet++:

  http://www.omnetpp.org/doc/omnetpp/manual/usman.html

РЕАЛИЗАЦИЯ НА СИСТЕМАТА “ГЕНЕРАТОР НА ЗАДАЧИ – ОПАШКА – СЪРВЪР”

Генератор на пакети (входящ процес)

```
void Generator::initialize()
{
        timerMessage = new cMessage("timer");
        scheduleAt(simTime(), timerMessage);
}
.
.
void Generator::handleTimerMessage(cMessage *msg)
{
        Packet * pPacket = new Packet ("Packet");

        scheduleAt(simTime() +  par("inter_arrival_time").doubleValue(),
timerMessage);
        sendMessage(pPacket, 0, "out");
}
.
.

void Generator::handleMessage(cMessage *msg)
{
        if (msg->isSelfMessage()) {
                handleTimerMessage(msg);
        } else {
                ASSERT(0);
        }
}
```

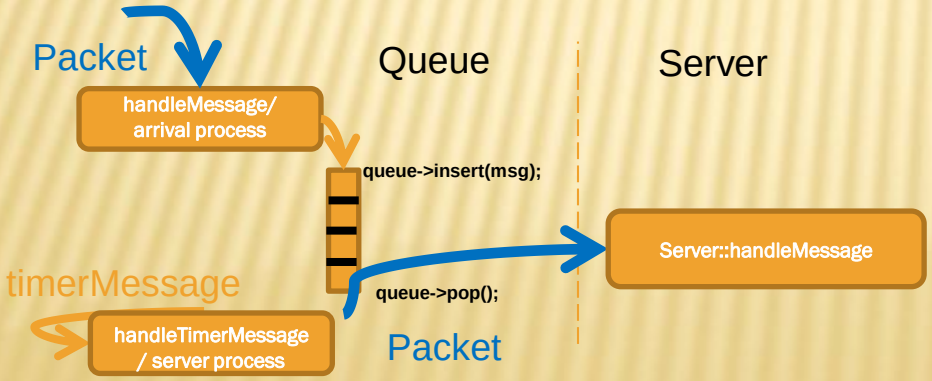


ОПАШКА

Функцията handleMessage получава съобщения от два източника – от генератора (входящия процес) и от таймера, който генерира първото съобщение от конструктора на класа *опашка*: Queue::initialize и в последствие от самият хендлър на таймера handleTimerMessage, чрез scheduleAt(simTime() + par("inter_arrival_time").doubleValue(), timerMessage);

както се вижда, следващото таймер съобщение се планира да бъде изпратено след *inter_arrival_time, което се взема всеки път от инициализационият файл omnetpp.ini :*

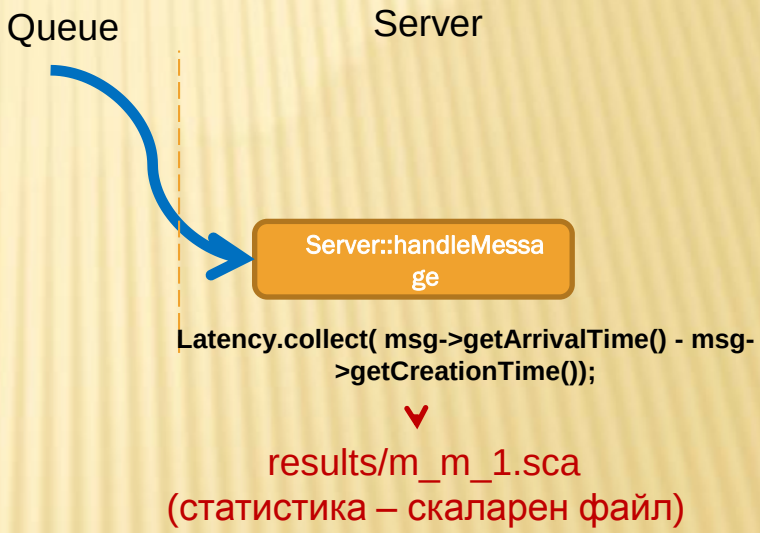
**.queue.inter_arrival_time = exponential(0.1), като exponential(0.1) представлява функция, генерираща експоненциялно разпределени случайни стойности отговарящи на Марковски процес (M). Съобщенията които не са isSelfMessage идват от модула Generator, през канала generator.out –> queue.in описан в QueueNet.ned




```
void Queue::initialize()
{
        queue_length = par("queue_length").longValue();
        queue = new cQueue;

        Lenght.setName("queque lenght");

        timerMessage = new cMessage("timer");
        scheduleAt(simTime(), timerMessage);
}

void Queue::handleTimerMessage(cMessage *msg)
{
        Packet *pPacket;

        if(!queue->empty()) {
                pPacket = (Packet *)queue->pop();
                UpdateDisplay(queue->getLength());
                sendDelayed(pPacket, 0, "out");
        }
        Lenght.collect(queue->getLength());

        scheduleAt(simTime() +
par("inter_arrival_time").doubleValue(), timerMessage);
}
void Queue::handleMessage(cMessage *msg)
{
        if (msg->isSelfMessage()) { // server process
                handleTimerMessage(msg);
        } else { // arrival process
                msg->setTimestamp();
                queue->insert(msg);
                UpdateDisplay(queue->getLength());
                        Lenght.collect(queue-
>getLength());
        }
}
```

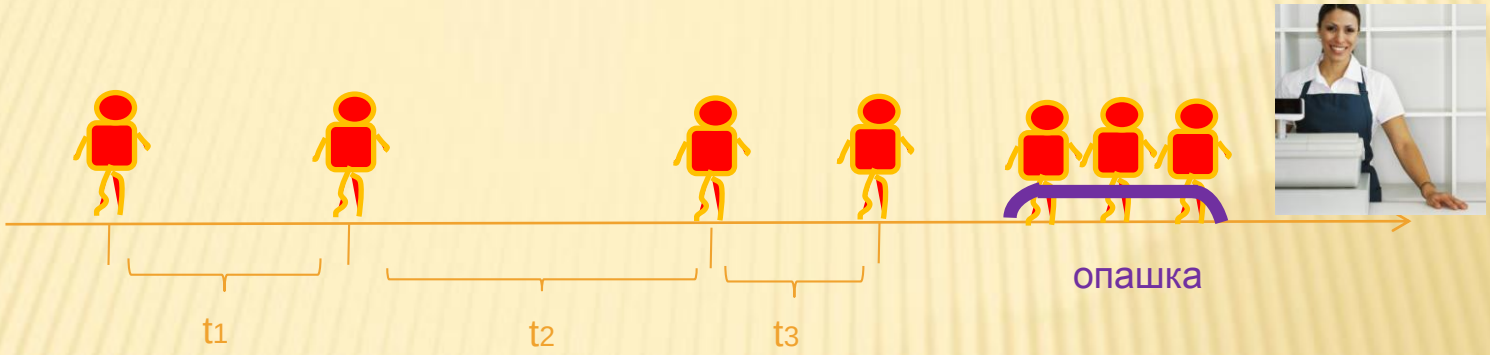
СЪРВЪР

```
void Server::handleMessage(cMessage *msg)
{
        if (msg->isSelfMessage()) {
                ASSERT(0);
        } else {
                Latency.collect( msg->getArrivalTime() - msg-
>getCreationTime());
                delete msg;
        }
}
.
.
.


void Server::finish()
{
        Latency.record();
}
```



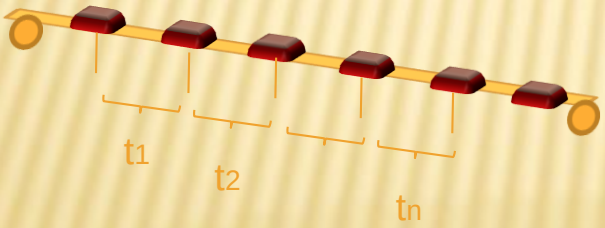
РАЗПРЕДЕЛЕНИЯ НА ТРАФИКА

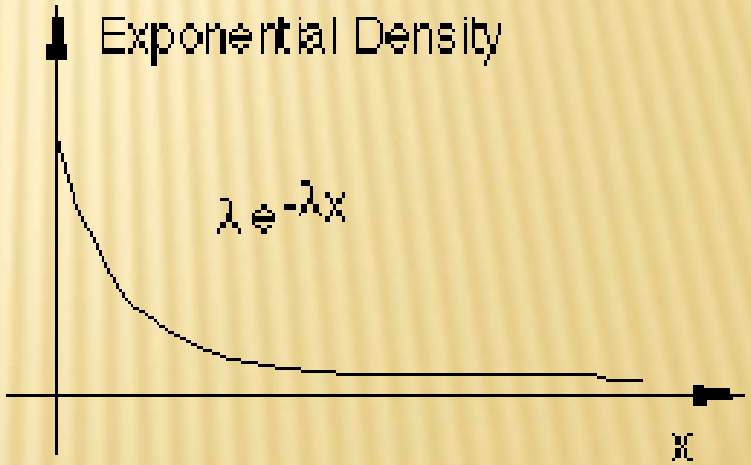
РАЗПРЕДЕЛЕНИЯ



При M разпределение, средният интервал м/у пристигащите заявки е tavg=1/λ, с функция на разпределение $f(t)=\lambda e^{-\lambda t}$

Повечето случайни процеси от заобикалящият ни свят са от този тип – като заявките които пристигат към даден сървър, студентите редящи се на опашката в стола и много други ☺





При D разпределението имаме еднакви интервали $t_1 = t_2 = tn = 1/\lambda$. Типичен пример – индустриален конвейр

M/M/1

- $N = \rho / (1- \rho)$, където $\rho = \lambda / \mu$;

  $\lambda < \mu$ (в противен случай – безкрайно нарастване на опашката)

  λ – интензивност на входните заявки (входящ трафик)

  μ – интензивност на обслужваните заявки (изходящ трафик)

  N – среден брой заявки, чакащи в опашката

- $T = 1 / (\mu - \lambda)$; $\lambda < \mu$

  T – време на престой заявките в опашката

M/D/1

$N = \rho + \rho^2 / 2 (1 - \rho)$, където

$\rho = \lambda / \mu$; $\lambda < \mu$ (в противен случай – безкрайно нарастване на опашката)

λ – интензивност на входните заявки (входящ трафик)

μ – интензивност на обслужваните заявки (изходящ трафик)

N – среден брой заявки, чакащи в опашката

$T = \rho / 2 \mu (1 - \rho)$

T – време на престой заявките в опашката

M/M/1/K

$$N = \begin{cases} \dfrac{k\rho^{k+2} - (k+1)\,\rho^{k+1} + \rho}{(\rho^{k+1} - 1)(\rho - 1)} & , \rho \neq 1 \\ \dfrac{k}{2} & , \rho = 1 \end{cases}$$

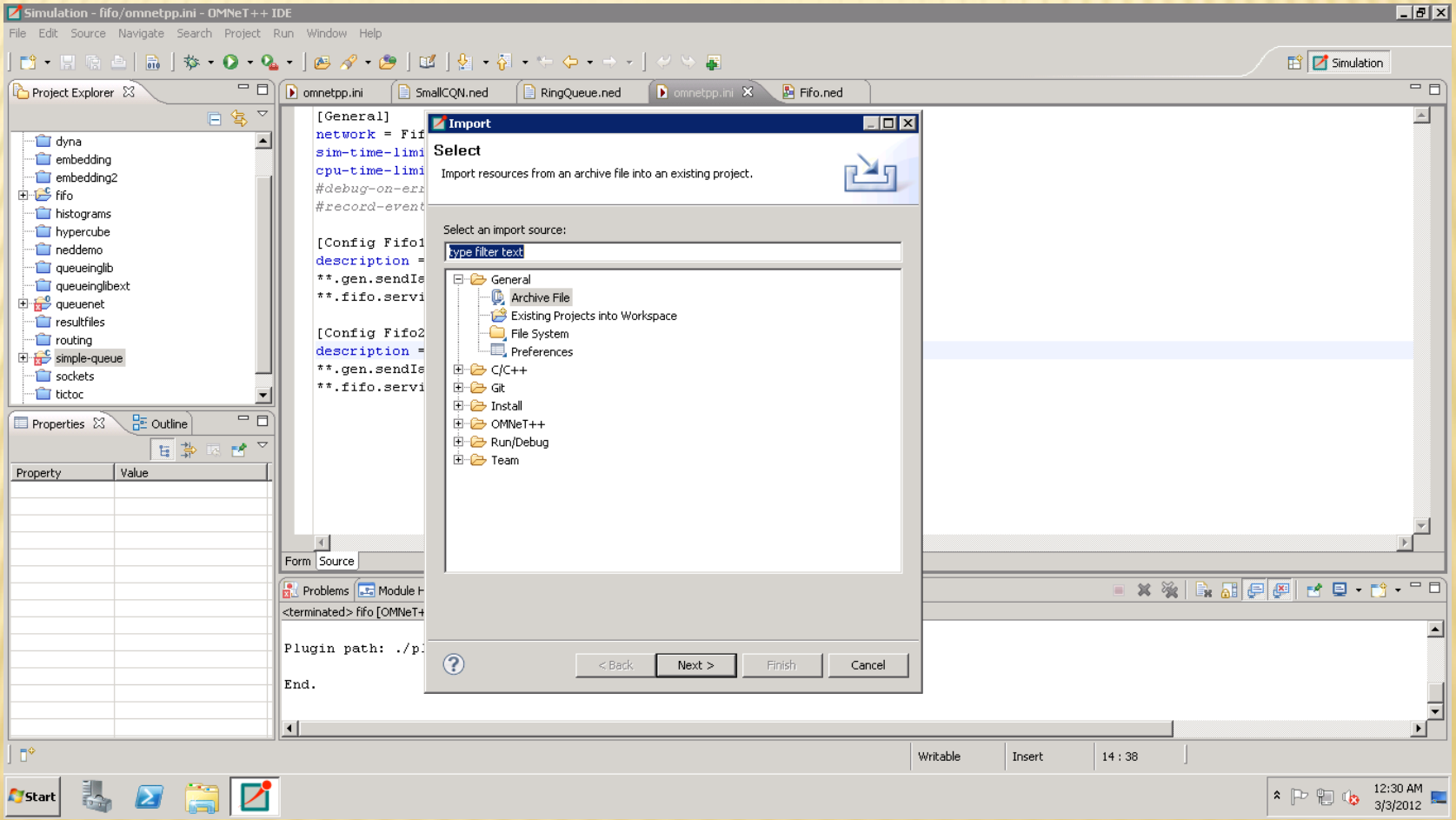
$$T = \begin{cases} \dfrac{k\,\rho^{k+1} - (k+1)\,\rho^{k} + 1}{\mu\,(\rho^{k} - 1)(\rho - 1)} & , \rho \neq 1 \\ \dfrac{k+1}{2\mu} & , \rho = 1 \end{cases}$$

Където ρ отново е λ / μ

k – капацитет на опашката

OMNET++ ПРОЕКТ

Стартирайте средата **OMNeT++.** От падащите менюта изберете**: File->Import**
От диалоговия прозорец който се появява изберете за сорс **General->Archive File**

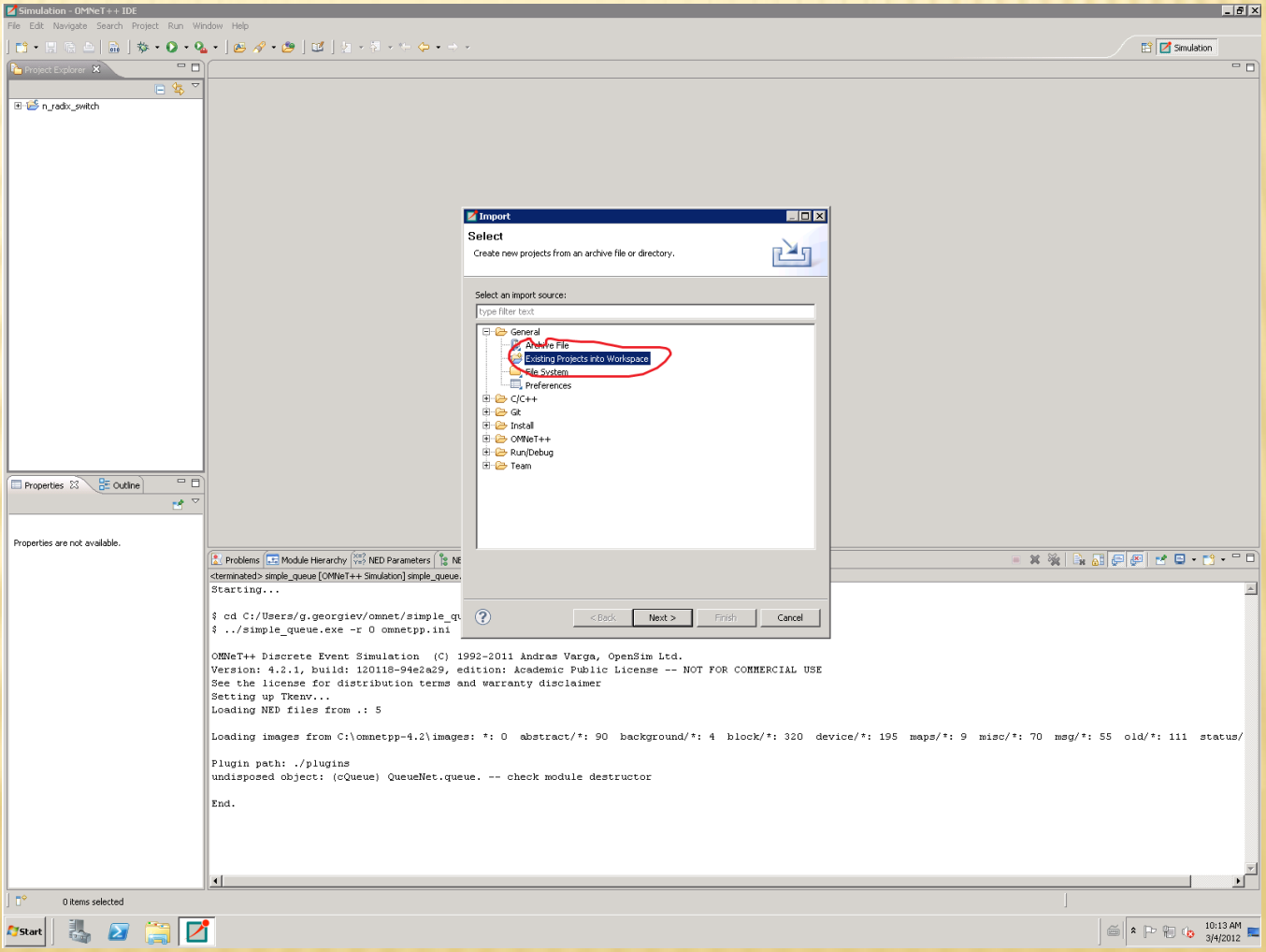


След натискане на бутона **NEXT,** в следващия прозорец се указва физическия път до архивния файл в който се намира проекта. Трябва да навигирате до файла **«simple-queue»**

OMNET++ ПРОЕКТ I

Omnet++ Project:

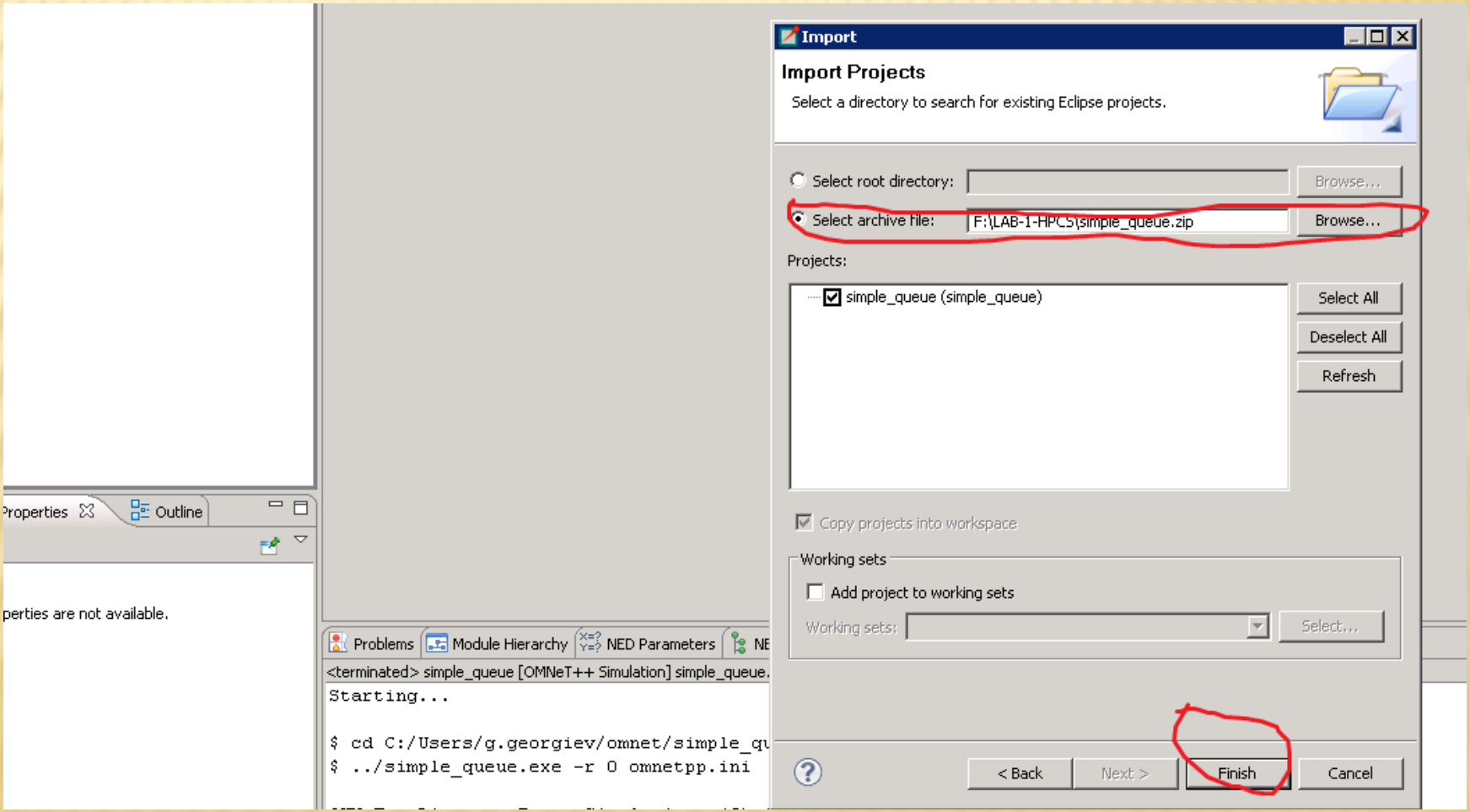
1. Import -> Existing Projects into Workspace
2. Select archive file -> \\192.168.70.5\Information\LAB-1-HPCS\simple_queue.zip
3. Finish



OMNET++ ПРОЕКТ II

Omnet++ Project:

1. Import -> Existing Projects into Workspace
2. Select archive file -> \\192.168.70.5\Information\LAB-1-HPCS\simple_queue.zip
3. Finish



OMNET++ ПРОЕКТ III

- Разгледайте сорс файловете и конфигурационните файлове на проекта.
- Компилирайте проекта, след което стартирайте модела – от менюто: **Run->Run As->OMNeT++ Simulation**

ЗАДАЧИ ЗА ДОМАШНА РАБОТА

- Да се свалят данните от симулацията за средното закъснение и броя чакащи заявки при вариращ интензитет на входните заявки и да се построят графики:
  - При M (експоненциално) разпределение (M/M/1)
  - При D (детерминистично) разпределение (M/D/1)
- Да се модифицира симулационната постановка така, че да представя система M/M/1/c и да се свалят симулационните данни за средното закъснение и брой чакащи заявки при вариращ интензитет на входните заявки и да се построят графиките:
  - При M (експоненциално) разпределение (M/M/1/c), като параметъра c = {1,2,4,8,16} *(според вашата подгрупа)*
- Да се сравнят аналитичните данни (чрез формулите) и тези получени от симулациите

ЗА ПОВЕЧЕ ТЕОРЕТИЧНА ИНФОРМАЦИЯ

- http://en.wikipedia.org/wiki/Kendall%27s_notation
- http://en.wikipedia.org/wiki/M/M/1_queue
- http://en.wikipedia.org/wiki/M/M/c_queue
- http://www.cs.rit.edu/~ark/docs/QueuingTheory.pdf