

# Κατανεμημένα Συστήματα I

## Μάθημα Βασικής Επιλογής, Χειμερινού Εξαμήνου Τομέας Εφαρμογών και Θεμελιώσεων

Χρήστος Κονίνης

Τρίτη, 3 Νοεμβρίου, 2009  
Υπολογιστικό



## Επισκόπηση

Controlling the simulation  
Simulation Commands  
save/load world Example

Processors in shown  
The Processor  
The Message Class

Ασκήσεις  
Ασκηση Μερους Α  
Ασκηση Μερους Β



## Simulation Commands

1. Η εντολή `prepare_world` δημιουργεί ένα νέο περιβάλλον εξομοίωσης:
  - ▶ `prepare_world edge_model=simple comm_model=disk_graph transm_model=stats_chain range=1`
2. Η εντολή `rect_world` γεμίζει το περιβάλλον με κόμβους:
  - ▶ `rect_world width=25 height=25 count=800 processors=helloworld`



## Simulation Commands

1. Η εντολή `simulation` εκκινεί την εξομοίωση:
  - ▶ `simulation max_iterations=50`
2. Η εντολή `rect_world` γεμίζει το περιβάλλον με κόμβους:
  - ▶ `prepare_world width=25 height=25 count=800 processors=helloworld`



## Simulation Commands

1. Η εντολή `simulation` εκκινεί την εξομίωση:
  - ▶ `simulation max_iterations=50`
2. Η εντολή `save_world` αποθηκεύει την τρέχουσα τοπολογία:
  - ▶ `save_world file=topology.xml`
3. Η εντολή `load_world` φορτώνει μια αποθηκευμένη τοπολογία:
  - ▶ `load_world file=topology.xml processors=helloworld`



## save/load world Example

1. Φτιάχνουμε ένα αρχείο `hello.conf` στον φάκελο `shawn/buildfiles` και γράφουμε τις παρακάτω εντολές:

```
prepare_world edge_model=simple comm_model=disk_graph range=1
rect_world width=5 height=5 count=30 processors=helloworld
save_world file=topology.xml
simulation max_iterations=10
```

2. Ανοίγουμε μια κονσόλα στον φάκελο `shawn/buildfiles`
3. Εκτελούμε την εντολή:  
`./shawn -f hello.conf`
4. Επαναλαμβάνουμε τα βήματα 1 ως 3 αλλάζοντας το `hello.conf` σε :

```
prepare_world edge_model=simple comm_model=disk_graph range=1
load_world file=topology.xml processors=helloworld
simulation max_iterations=10
```



## Επισκόπηση

### Controlling the simulation

Simulation Commands

save/load world Example

### Processors in shawn

The Processor

The Message Class

### Ασκήσεις

Ασκηση Μερους Α

Ασκηση Μερους Β



## The Processors

- ▶ Μας επιτρέπουν να υλοποιούμε την λογική των κατανεμημένων αλγορίθμων μας
- ▶ Μπορούμε να έχουμε πολλούς `processors` σε ένα κόμβο
  - ▶ Αλγόριθμο Flooding
  - ▶ Αλγόριθμο Leader election
- ▶ Έχουν τρεις καταστάσεις
  - ▶ Active: Πλήρη λειτουργία
  - ▶ Sleep: Δεν δέχονται μηνύματα
  - ▶ Inactive: Καμία λειτουργία
- ▶ Αν όλοι οι `processors` σε ένα κόμβο είναι Inactive τότε ο κόμβος γίνεται Inactive
- ▶ Αν όλοι οι κόμβοι είναι Inactive η εξομίωση τερματίζει



## The Processor API (1 of 3)

- ▶ `void boot()` :  
Εκτελείται μια φορά για κάθε processors στην αρχή της εξομοίωσης. Χρησιμοποιείται συνήθως για αρχικοποίηση του αλγορίθμου μας.
- ▶ `void special_boot()` :  
Εκτελείται μια φορά για κάθε processors που βρίσκεται σε ένα ειδικό (special) κόμβο. Καλείται πριν την `boot()`
- ▶ `void work()` :  
Εκτελείται μια φορά για κάθε γύρο (βήμα) της εξομοίωσης. Χρησιμοποιείται για την εκτέλεση περιοδικών tasks.



## The Processor API (2 of 3)

- ▶ `bool process_message(MessageHandle& )` :  
Εκτελείται όταν ο κόμβος λάβει ένα μήνυμα. Το μήνυμα παραδίδεται στους processors του κόμβου. Μπορεί να λάβει διαφορετικά μηνύματα οπότε πρέπει να εξετάζουμε τον τύπο του μηνύματος.

```
bool
MyProcessor::
process_message( const shawn::ConstMessageHandle& mh ){
    const MyMessage* mymsg =
        dynamic_cast<const MyMessage*>( mh.get() );

    if ( MyMessage!= NULL ) {
        // handle my message
        return true;
    }

    return shawn::Processor::process_message( mh );
}
```



## The Processor API (3 of 3)

- ▶ `void send(MessageHandle& )` :  
Στέλνει ένα μήνυμα στους γείτονες.  
`send( new MyMessage );`
- ▶ `void set_state(MessageHandle& )` :  
Θέτει την κατάσταση του processor.  
`set_state( shawn::Processor::Active )`  
`set_state( shawn::Processor::Sleep )`  
`set_state( shawn::Processor::Inactive )`
- ▶ `const Node& owner_w()` :  
Επιστρέφει τον κόμβο στον οποίο βρίσκεται ο processor.
  - ▶ `owner_w().Id()`
  - ▶ `owner_w().label()`
  - ▶ `owner_w().world_w().simulation_round()`



## Τα μηνύματα στο Shawn (1 of 2)

- ▶ Όλα τα μηνύματα που στέλνουμε είναι κλάσεις. Και κληρονομούν από την κλάση `shawn Message`

```
class MyMessage
    : public shawn::Message
{
public:
    MyMessage();
    virtual ~MyMessage();
};
```

- ▶ Κάθε πρωτόκολλο υλοποιεί τα δικά του μηνύματα, δηλαδή διαφορετικές κλάσεις
- ▶ Κάθε processor κάνει cast το γενικό μήνυμα που λαμβάνει στην `process_message(MessageHandle& )` για να πάρει το δικό του μήνυμα

```
const MyMessage* mymsg =
    dynamic_cast<const MyMessage*>( mh.get() );
```



- ▶ Οι πιο σημαντικές συναρτήσεις ενός μηνύματος είναι:
  - ▶ `const Node& source()` :  
Ο κόμβος που έστειλε το μήνυμα.
  - ▶ `set_size(Int)` :  
Θέτουμε το μέγεθος του μηνύματος, δεν πρόκειται για το πραγματικό μέγεθος των δεδομένων που στέλνουμε αλλά για εικονικό μέγεθος
  - ▶ `Int size()` :  
Παίρνουμε μέγεθος του μηνύματος που έχουμε θέσει
  - ▶ `double timestamp_time()` :  
Η χρονική στιγμή που στάλθηκε το μήνυμα, το θέτει αυτόματα ο κόμβος-αποστολέας



## Επισκόπηση

Controlling the simulation  
Simulation Commands  
save/load world Example

Processors in shawn  
The Processor  
The Message Class

Ασκήσεις  
Ασκηση Μερους Α  
Ασκηση Μερους Β



## Περιγραφή του προβλήματος

- ▶ Υλοποίηση ενός απλού κατανεμημένου προγράμματος,
- ▶ Tree Routing Algorithm
  1. Ένας κόμβος το (Gateway) κάνει flood στο δίκτυο ένα μήνυμα.
  2. Το flood μήνυμα περιέχει το id του κόμβου που το έστειλε και το hop-count απο το Gateway
  3. Κάθε κόμβος που λαμβάνει το flood μήνυμα:
    - 3.1 αποθηκεύει το id του κόμβου από τον οποίο πήρε το μήνυμα (parent node) και το hop-count, μόνο αν το hop-count είναι μικρότερο από αυτό που ήδη έχει
    - 3.2 Μπαίνει σε κατάσταση connected
  4. Μετά στέλνουν ένα νέο flood μήνυμα που έχει αποστολέα τον ίδιο και hop-count= hop-count +1



## Ασκηση Μερος Α

- ▶ Κατεβάστε το template του πρώτου προβλήματος από εδώ  
`ftp://carrot.cti.gr/fr02/fr02_partA.tar.bz2`
- ▶ Μεταφέρεται τα αρχεία `fr02-config.conf` και `fr02-topology.xml` στο φάκελο `shawn/buildfiles`
- ▶ Δημιουργήστε ένα νέο φάκελο `shawn/src/legascyapps/fr02`
- ▶ Μεταφέρεται υπόλοιπα αρχεία στον φάκελο `shawn/src/legascyapps/fr02`
- ▶ Από την κονσόλα μεταβείτε στον φάκελο `shawn/buildfiles`
- ▶ Δώστε την εντολή `cp make ../src` και μετά:
  - ▶ `c`, για να ενημερωθεί για τον νέο κώδικα που προσθέσατε
  - ▶ Ενεργοποιήστε την επιλογή FR02 (πατώντας `enter`)
  - ▶ `c`, για να ενημερωθεί για τις αλλαγές
  - ▶ `g`, για να αποθηκεύσει τις αλλαγές
- ▶ Τέλος δώστε την εντολή `make` για να γίνει `compile` το `shawn`



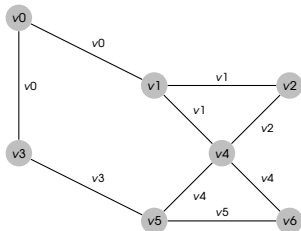
## Ασκηση Μερος Α

- ▶ Προσθέτοντας κώδικα στις συναρτήσεις `boot()` `work()` του `fr02_Processor.cpp`, καλείστε να υλοποιηθεί τον αλγόριθμο `tree routing`
- ▶ Υπάρχουν σχόλια στα σημεία που πρέπει να προσθέσετε κώδικα για να σας καθοδηγήσουν
- ▶ Σε αυτό το μέρος θα χρησιμοποιήσουμε μόνο ένα τύπο μηνύματος για το `flooding`:

```
Fr02FloodingMessage( int , std::string );  
  
int hops( void );  
std::string parent( void );  
private:  
int hops_;  
std::string source_;
```



## Τοπολογία Άσκησης



## Ασκηση Μερος Α

- ▶ Εκτελείται τον αλγόριθμο χρησιμοποιώντας την τοπολογία `fr02-topology.xml` και το `config` αρχείο `fr02-config.conf`:  
`$.shawn -f fr02-config.conf`
- ▶ Στην έξοδο πρέπει να βλέπετε περίπου τις παρακάτω γραμμές:

```
----- BEGIN ITERATION 0  
processors.fr02: v0: Sending flood message  
----- DONE ITERATION 0  
.....  
v1: Joint tree under parent v0 hops to gateway: 1  
v3: Joint tree under parent v0 hops to gateway: 1  
.....  
v2: Joint tree under parent v1 hops to gateway: 2  
v4: Joint tree under parent v1 hops to gateway: 2  
v5: Joint tree under parent v3 hops to gateway: 2  
.....  
v6: Joint tree under parent v4 hops to gateway: 3
```



- ▶ Στο μέρος Β θά επεκτείνουμε την προηγούμενη άσκηση ώστε:
  - ▶ Κάθε κόμβος που μπαίνει στο δέντρο θα στέλνει σε κάθε γύρο ένα μήνυμα προς το Gateway
  - ▶ Το μήνυμα θα περιέχει το hop-count του κόμβου
  - ▶ Κάθε κόμβος θα στέλνει το μήνυμα με προορισμό τον πατέρα του στο δέντρο, μέχρι να φτάσει στην ρίζα
  - ▶ Τα μηνύματα είναι τύπου Fr02HelloMessage
  - ▶ Όταν η ρίζα λάβει ένα μήνυμα Fr02HelloMessage θα τυπώνει τον αποστολέα και το hop-count