Université Badji Mokhtar Annaba
Faculté de Technologie
Département d'Informatique

Haiahem Rahim
Évaluation des performances des systèmes
RSI M2

TP N°3
Paramètres, génération aléatoire, signal

I. Paramètres

Créer un système composé de 2 modules (nœuds), un **Générateur** et **un Queue**, connectés via une communication unidirectionnelle comme il est décrit sur le schéma 1.

Schéma 1 : Disposition des nœuds

Scénario 1 :

Dans ce scénario, nous voulons que la source (**Générateur**) fournit un nombre de tâches k (job) au second module d'attente (**Queue)** qui va les stocker (enfiler) pour un future traitement. Le nombre de jobs k doit être choisi dynamiquement lors du lancement de la simulation.

II. Les Self-messages ou Timers

Dans les scénarios précédents (TP1, scénario 1), nous avons vu qu'un nœud fait un traitement lors de l'initialisation et lors de la réception d'un message. Si nous voulons qu'un nœud attend un certain temps avant de faire un certain traitement, il faut utiliser les self-messages (ou Timers) qui sont des messages programmés au niveau du nœud et envoyés à lui-même pour l'alerter qu'il faut faire le traitement en question. Pour programmer le self-message :

scheduleAt (simTime()+5.0, *Msg);

Scénario 2 :

Dans ce scénario, on veut garder le même principe utilisé dans le scénario précédent sauf que nous volons que la génération des jobs par le module **Générateur** s'effectue chaque T seconds. Le temps T doit être choisi dynamiquement lors du lancement de la simulation.

III. Génération des nombres aléatoire

Un générateur de nombres aléatoires, Random Number Generator (RNG) en anglais, est un dispositif capable de produire une séquence de nombres indépendants appelée distribution de nombres aléatoires.

Les RNG sont rarement utilisés directement, car ils produisent des nombres aléatoires uniformément distribués. Lorsque des nombres aléatoires non uniformes sont nécessaires, des transformations mathématiques sont utilisées pour produire des nombres aléatoires à partir d'une entrée RNG qui correspondent à des distributions spécifiques.
Le tableau suivant décrit comment obtenir des flux de nombres aléatoires non uniformément distribués à partir de diverses distributions.

Distribution	**Description**
uniform(a, b)	distribution uniforme dans l'intervalle [a,b)
exponential(mean)	distribution exponentielle de moyenne ***mean***
normal(mean, stddev)	distribution normale d'une moyenne et l'écart type ***mean et stddev, respc.***
poisson(lambda)	distribution de Poisson avec paramètre lambda

Pour générer des nombres aléatoires il suffit de faire appeler la méthode (voir tableau au dessus) permettant de produire des nombres aléatoires qui correspondent à une distributions spécifiques.
Example :

double exponential(double mean, int rng=0) const;
double normal(double mean, double stddev, int rng=0) const;

ou directement :
scheduleAt(simTime() + exponential(mean), msg);

Scénario 3 :

On garde toujours le même principe utilisé dans les scénarios précédents sauf que dans ce scénario le temps inter-arrivée des jobs suit une loi exponentielle de paramètre **mean.** Ce dernier paramètre est introduit par l'utilisateur lors du lancement de la simulation.

IV. Collecte des statistiques en se basant sur les signaux

Une utilisation des signaux consiste à exposer des variables pour la collecte des résultats sans dire où, comment et s'il faut les enregistrer. Avec cette approche, les modules ne publient que les variables et l'enregistrement réel du résultat aura lieu dans les objets listeners.

Déclarer des statistiques

Afin d'enregistrer des résultats de simulation basés sur des signaux, il faut ajouter des propriétés @**statistic** à la définition NED du module (ou du canal) simple.

Une propriété @**statistic** définit le nom de la statistique, quels signaux sont utilisés comme entrée, quelles étapes de traitement doivent leur être appliquées (par exemple, filtrage, sommation, quotient différentiel) et quelles propriétés doivent être enregistrées (minimum, maximum, moyenne, etc.) et sous quelle forme (vecteur, scalaire, histogramme).

```
simple Queue
{
    parameters:
        @statistic[queueLength](record=max,timeavg,vector?);
    gates:
        input in;
        output out;
}
```

Scénario 4 :

En se basant sur l'approche des signaux, effectuez les changements nécessaires au code du scénarios 3 afin de collecter et sauvegarder les résultats de simulation qui s'intéressent au :

- Nombre moyen de tâches stockées dans le module d'attente (Queue).

Les résultats doivent être sauvegarder sous forme vectorielle et scalaire.

Questions :

Q1. Quelle est la différence entre un résultat scalaire et un résultat vectoriel ?

Q2. Qu'est-ce qu'un «Signal» ?

Q3. Qu'est-ce qu'un « seed » ?