

---

# Le processus d'une machine d'un reseau Token Ring

---

*Projet d'atelier*

**Réalisé par :**

- Majda Bendifi
- Meryem El Mansouri
- Nohaila Bentaleb

**Encadrant :**

- Mr. Bensaid Hicham

**Année universitaire :**

- 2024/2025

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Algorithme</b>	<b>2</b>
<b>3</b>	<b>Conception du simulateur</b>	<b>2</b>
3.1	Fonctionnalités principales . . . . .	2
<b>4</b>	<b>Code source</b>	<b>2</b>
<b>5</b>	<b>Exemple d'exécution</b>	<b>4</b>
5.1	Commandes d'exécution . . . . .	4
5.2	Exemple de sortie . . . . .	4
<b>6</b>	<b>Utilisation de Git et GitHub</b>	<b>4</b>
6.1	Initialisation du projet . . . . .	4
6.2	Collaboration . . . . .	5
6.3	Gestion des branches et intégration . . . . .	5
6.4	Suivi des contributions . . . . .	5
<b>7</b>	<b>Conclusion</b>	<b>6</b>

# 1 Introduction

Le but de ce projet est de créer un simulateur d'un réseau Token Ring. Ce type de réseau est une topologie à anneau où un jeton circulant dans une seule direction permet la transmission des données. Chaque station transmet uniquement lorsqu'elle détient le jeton.

Ce rapport explique le fonctionnement de l'algorithme, la conception du simulateur, les commandes Git utilisées, ainsi que le code source développé.

## 2 Algorithme

Voici le fonctionnement logique d'un noeud dans un réseau Token Ring :

```
1 while (true) {
2     attendre_le_jeton();
3     if (a_des_choses_a_transmettre) {
4         transmettre_la_trame_a_transmettre();
5         passer_le_jeton_a_la_station_suivante();
6     } else {
7         passer_le_jeton_a_la_station_suivante();
8     }
9 }
```

Listing 1 – Algorithme 1 : Fonctionnement d'un noeud Token Ring

## 3 Conception du simulateur

Le simulateur est implémenté en langage C. Il permet de paramétrer le nombre de noeuds et de suivre l'état du réseau au fil des itérations.

### 3.1 Fonctionnalités principales

- Paramétrage dynamique du nombre de noeuds.
- Gestion du jeton et transmission de trames.
- Simulations avec gestion des pannes.

## 4 Code source

Voici le code source du simulateur :

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h> // For sleep() on Unix-based systems
4 #include <stdbool.h> // To use the bool type
5 #include <time.h> // For generating random numbers
6
7
8 bool has_data_to_transmit() {
9     return rand() % 2; // Returns true (1) or false (0) randomly
10 }
11
12 // Token Ring simulation function
```

```

13 void token_ring_simulation(int num_nodes) {
14     int current_token = 0;
15
16     int cycle_count = 0; // Limit the number of cycles for the
        simulation
17
18     while (cycle_count < 10) { // Limit to 10 cycles
19         printf("\nNode %d has the token.\n", current_token);
20
21         // Check if the node has data to transmit
22         if (has_data_to_transmit()) {
23             printf("Node %d is transmitting data...\n", current_token)
24             ;
25             sleep(1); // Simulate transmission time
26             printf("Node %d has finished transmitting.\n",
27                 current_token);
28         } else {
29             printf("Node %d has no data to transmit. Passing the token
30                 ...\n", current_token);
31         }
32
33         // Pass the token to the next node
34         current_token = (current_token + 1) % num_nodes;
35         sleep(1); // Pause to observe the simulation
36
37         cycle_count++;
38     }
39 }
40
41 int main() {
42     int num_nodes;
43
44     // Initialize the random number generator
45     srand(time(NULL));
46
47     // Ask the user for the number of nodes
48     printf("Enter the number of nodes in the Token Ring: ");
49     scanf("%d", &num_nodes);
50
51     if (num_nodes <= 0) {
52         printf("The number of nodes must be greater than 0.\n");
53         return 1;
54     }
55
56     // Start the Token Ring simulation
57     printf("\nStarting the Token Ring simulation with %d nodes...\n",
58         num_nodes);
59     token_ring_simulation(num_nodes);
60
61     return 0;
62 }

```

Listing 2 – Code C du simulateur

## 5 Exemple d'exécution

Pour illustrer le fonctionnement du simulateur, voici un exemple d'exécution du programme avec 3 noeuds. Les messages affichés permettent de suivre la simulation du réseau Token Ring.

### 5.1 Commandes d'exécution

Le programme peut être compilé et exécuté avec les commandes suivantes :

```
1 gcc -o token_ring token_ring.c
2 ./token_ring
```

Listing 3 – Compilation et exécution du programme

### 5.2 Exemple de sortie

Voici un exemple des messages affichés lors de l'exécution avec 3 noeuds :

```
1 Enter the number of nodes in the Token Ring: 3
2
3 Starting the Token Ring simulation with 3 nodes...
4
5 Node 0 has the token.
6 Node 0 is transmitting data...
7 Node 0 has finished transmitting.
8
9 Node 1 has the token.
10 Node 1 has no data to transmit. Passing the token...
11
12 Node 2 has the token.
13 Node 2 is transmitting data...
14 Node 2 has finished transmitting.
15
16 Node 0 has the token.
17 Node 0 has no data to transmit. Passing the token...
18 ...
```

Listing 4 – Exemple de sortie d'exécution

Observez les messages affichés pour suivre la simulation du réseau Token Ring. Chaque noeud vérifie s'il a des données à transmettre et, si c'est le cas, effectue une transmission avant de passer le jeton au noeud suivant.

## 6 Utilisation de Git et GitHub

Pour collaborer efficacement, nous avons utilisé Git et GitHub. Le développement du projet a été organisé de manière à permettre une gestion fluide et structurée des contributions.

### 6.1 Initialisation du projet

Le projet a été initialisé avec les commandes suivantes :

```

1 git init
2 git add .
3 git commit -m "Initialisation du projet"
4 git branch -M main
5 git remote add origin <url_du_d\'ep\'ot>
6 git push -u origin main

```

Listing 5 – Initialisation du dépôt

Cela a permis de créer le dépôt principal sur GitHub pour centraliser le projet.

## 6.2 Collaboration

Les membres ont utilisé Git pour synchroniser leurs modifications et travailler sur des fonctionnalités. Voici les commandes principales utilisées :

```

1 git pull # R\'ecup\'erer les modifications depuis le d\'ep\'ot distant
2
3 git add fichier_modifi\'e.c
4 git commit -m "Ajout de la g\'en\'eration de donn\'ees al\'eatoires"
5 git push

```

Listing 6 – Commandes de collaboration

## 6.3 Gestion des branches et intégration

Les branches ont été utilisées pour isoler les modifications. Voici un exemple des étapes suivies pour créer, modifier et fusionner une branche :

```

1 # Cr\'eer une branche pour la documentation
2 git branch doc
3 git checkout doc
4
5 # R\'edaction de la documentation
6 git add rapport.tex
7 git commit -m "Ajout de la section sur l\'utilisation de Git et GitHub"
8 git push -u origin doc
9
10 # Fusion dans la branche principale
11 git checkout main
12 git merge doc
13 git push

```

Listing 7 – Créer et fusionner des branches

## 6.4 Suivi des contributions

L'historique des contributions sur GitHub montre l'implication de chaque membre. Les pull requests ont été utilisées pour discuter des modifications importantes et les valider avant leur intégration.

## 7 Conclusion

Ce projet nous a permis d'apprendre à utiliser Git et GitHub pour collaborer efficacement. Les branches ont aidé à organiser les contributions de manière structurée. Ce travail pourrait être amélioré en automatisant les tests via GitHub Actions pour un développement encore plus fluide.