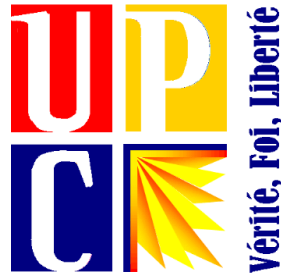


**UNIVERSITE PROTESTANTE AU CONGO**

*Faculté des sciences informatiques*



**B.P 4745**

**KINSHASA/LINGWALA**

**TP5**  
**PROGRAMMATION SYSTÈME**  
**L1 FASI**

Présenté par :



**NGOMA MAKWAMA ALAIN**

**PROF. DR. Patrick MUKALA**

**ANNEE – ACADEMIQUE 2020 - 2021**

## TP 5

« Après traduction dans la langue française »

**LES QUESTIONS**

Nous utiliserons le projet «ScheduleCPUTest», qui simule le comportement d'un simple planificateur du processeur. Dans ce projet, nous avons les classes suivantes:

CPU : Cette classe représente un processeur physique.

Scheduler (Planificateur) : Cette classe planifie les processus sur la CPU.

CircularProcesList : Cette classe implémente une liste circulaire. Il est utilisé par le planificateur comme file d'attente pour suspendre les processus.

ITimeTickReceiver : Interface pour les classes qui souhaitent recevoir les tics de minuterie du HardWareTimer.

HardwareTimer

Cette classe génère des tics de minuterie à intervalles réguliers et fournit le temps écoulé.

Les classes qui souhaitent utiliser ces ticks (dans notre cas CPU et Scheduler) doivent implémenter l'interface ITimeTickReceiver.

IProces : L'interface qui doit être implémentée par les processus qui s'exécutent sur le simulateur.

TestProcess : La classe des processus qui s'exécutent sur la CPU simulée. Cette classe implémente l'interface IProcess.

SchedulerMain : Cette classe contient la méthode principale pour instancier les classes ci-dessus et tester le Planificateur (scheduler).

Dans le projet donné, un simple ordonnanceur CPU est implémenté, mais nous voulons étendre cela afin de faire quelques mesures.

Effectuez les tâches suivantes (vous pouvez afficher la liste des tâches dans Visual Studio pour voir rapidement où vous devez apporter des modifications).

**a. Etudiez le code donné.**

Découvrez quel algorithme de planification est implémenté.

b. Développez le code de TestProces, de sorte que chaque processus garde une trace de son délai d'exécution (propriété TurnAroundTime), son temps d'attente (propriété WaitingTime) et combien de temps processeur le processus a besoin au total (propriété InitialCPUTimeNeeded).

c. Développez le code de TestScheduler, de sorte que les informations suivantes soient affichées dans la console, une fois tous les processus terminés:

- . Temps de traitement initial de chaque processus
- . Délai d'exécution de chaque processus
- . Temps d'attente de chaque processus
- . Délai d'exécution moyen de tous les processus
- . Temps d'attente moyen de tous les processus
- . Débit

d. Changez l'algorithme de planification en FCFS et montrez les différences de ci-dessus, par rapport à l'algorithme de planification d'origine.

**Montrez votre programme final à l'enseignant. !!!**

## LES RESOLUTIONS

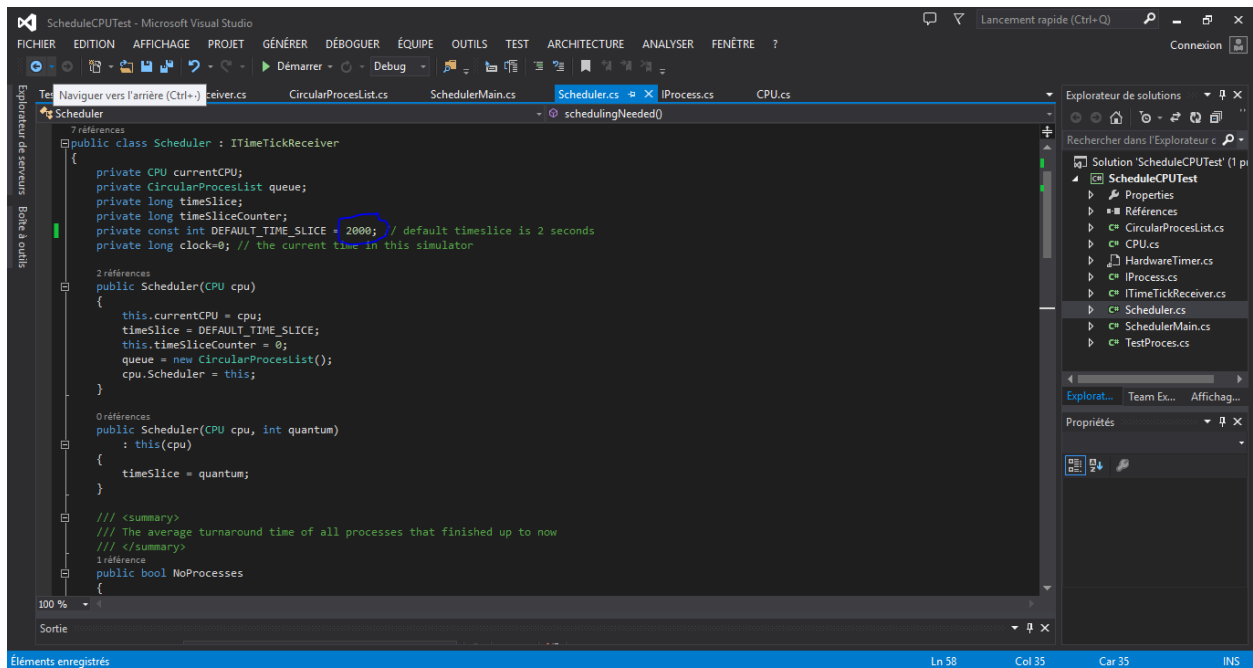
Après avoir murement réfléchi et travaillé d'arrache-pied dans un commun accord, en voici les différentes résolutions du présent T.P

Résolution assertion A

Etudiez le code donné.

Découvrez quel algorithme de planification est implémenté.

```
file:///C:/Users/hp/Documents/Visual Studio 2013/Projects/ScheduleCPUTest/ScheduleCPUTest/bin/Debug/ScheduleCPUTest.EXE
8000    * * * Context Switch * * *
8000    removing from CPU process 1
8000    adding to queue process 1
8000    putting on CPU process 2
10000   * * * Context Switch * * *
10000   removing from CPU process 2
10000   adding to queue process 2
10000   putting on CPU process 1
12000   * * * Context Switch * * *
12000   removing from CPU process 1
12000   adding to queue process 1
12000   putting on CPU process 2
14000   * * * Context Switch * * *
14000   removing from CPU process 2
14000   adding to queue process 2
14000   putting on CPU process 1
16000   * * * Context Switch * * *
16000   removing from CPU process 1
16000   adding to queue process 1
16000   putting on CPU process 2
18000   * * * Context Switch * * *
18000   removing from CPU process 2
18000   FINISHED: process 2
18000   putting on CPU process 1
20000   * * * Context Switch * * *
20000   removing from CPU process 1
20000   FINISHED: process 1
20000   queue empty
All processes finished
```



## Conclusion :

L'algorithme utilisé ici est le tourniquet ou round-robin parce qu'il y a un quantum qui est défini ici impactant le processus

Cet algorithme permet le temps partagé entre les 3 processus utilisé

### Résolution assertion B

Développez le code de TestProces, de sorte que chaque processus garde une trace de son délai d'exécution (propriété TurnAroundTime), son temps d'attente (propriété WaitingTime) et combien de temps processeur le processus a besoin au total (propriété InitialCPUTimeNeeded).

```

106     public long TurnAroundTime
107     {
108     get
109     {
110         //return cpuTimeNeeded + waitingTime;
111
112         return clock - startTime;
113     }
114 }
115 /// <summary>
116 /// Returns turnaround time of proces in timerticks between begin and end of the process
117 /// Note: If the process did not finish yet, return 0
118 /// </summary>
119 4 references
120 public long WaitingTime
121 {
122     get
123     {
124         // TODO waiting time
125         return waitingTime;
126     }
127     set
128     {
129         waitingTime += value;
130     }
131 }

```

## Résolution assertion C

Développez le code de TestScheduler, de sorte que les informations suivantes soient affichées dans la console, une fois tous les processus terminés:

- Temps de traitement initial de chaque processus
- Délai d'exécution de chaque processus
- Temps d'attente de chaque processus
- Délai d'exécution moyen de tous les processus
- Temps d'attente moyen de tous les processus
- Débit

```

153     1 reference
154     public void Afficher(int commencement, int execution, int attente,
155         int executionMoyen, int attenteMoyen, int debit)
156     {
157         System.Console.Out.WriteLine("\t \n\n");
158         System.Console.Out.WriteLine("\t A commencé à : " + commencement);
159         System.Console.Out.WriteLine("\t Temps d'exécution : " + execution);
160         System.Console.Out.WriteLine("\t Temps d'attente : " + attente);
161         System.Console.Out.WriteLine("\t Temps d'exécution moyen : " + executionMoyen);
162         System.Console.Out.WriteLine("\t Temps d'attente moyen : " + attenteMoyen);
163         System.Console.Out.WriteLine("\t Debit : " + debit);
164         System.Console.Out.WriteLine("\t \n\n");
165     }

```

```

104     if (!removedProcess.Ready)
105     {
106         removedProcess.WaitingTime = 2000 * (processEncours-1);
107
108         System.Console.Out.WriteLine("\n\tWaiting time " + removedProcess.Name + " = " + removedProcess.WaitingTime +
109             System.Console.Out.WriteLine(clock + "\tadding to queue " + removedProcess.Name);
110         this.queue.AddItem(removedProcess);
111     }
112     else
113     {
114         processEncours -=1;
115         Afficher((int)removedProcess.StartTime, (int)removedProcess.TurnAroundTime,
116             (int)removedProcess.WaitingTime, (int)0, (int)0, (int)0);
117         System.Console.Out.WriteLine(clock + "\n\n\tTemps total : " +
118             removedProcess.TurnAroundTime + " pour le processus " + removedProcess.Name + "\n\n");
119
120         System.Console.Out.WriteLine(clock + "\tFINISHED: " + removedProcess.Name);
121         System.Console.Out.WriteLine(clock + "\tTemps de fin d'exécution " + removedProcess.Name + ": " + clock);
122         System.Console.Out.WriteLine(clock + "\tTurnAroundTime " + removedProcess.Name + ": " +
123             removedProcess.TurnAroundTime);
124     }
125 }

```

```

C:\Users\HP\Documents\tp systeme\ScheduleCPUTest\ScheduleCPUTest\bin\Debug\ScheduleCPUTest.exe
0      * * * Context Switch * * *
0      putting on CPU process 1
2000   * * * Context Switch * * *
2000   removing from CPU process 1

      Waiting time process 1 = 4000

2000   adding to queue process 1
2000   putting on CPU process 2
4000   * * * Context Switch * * *
4000   removing from CPU process 2

      Waiting time process 2 = 4000

4000   adding to queue process 2
4000   putting on CPU process 3
6000   * * * Context Switch * * *
6000   removing from CPU process 3

      A commencé à : 4000
      Temps d'exécution : 2000
      Temps d'attente : 0
      Temps d'exécution moyen : 0
      Temps d'attente moyen : 0
      Debit : 0

6000

      Temps total : 2000 pour le processus process 3

6000   FINISHED: process 3
6000   Temps de fin d'exécution process 3: 6000
6000   TurnAroundTime process 3: 2000
6000   putting on CPU process 1
8000   * * * Context Switch * * *
8000   removing from CPU process 1

      Waiting time process 1 = 6000

8000   adding to queue process 1
8000   putting on CPU process 2

```



```
A commencé à : 0
Temps d'exécution : 20000
Temps d'attente : 10000
Temps d'exécution moyen : 0
Temps d'attente moyen : 0
Debit : 0
```

```
20000
```

```
Temps total : 20000 pour le processus process 1
```

```
20000 FINISHED: process 1
20000 Temps de fin d'exécution process 1: 20000
20000 TurnAroundTime process 1: 20000
20000 queue empty
All processes finished
TOTAL TIME PROCESSOR :
```

### Résolution assertion D

Changez l'algorithme de planification en FCFS et montrez les différences de ci-dessus, par rapport à l'algorithme de planification d'origine.

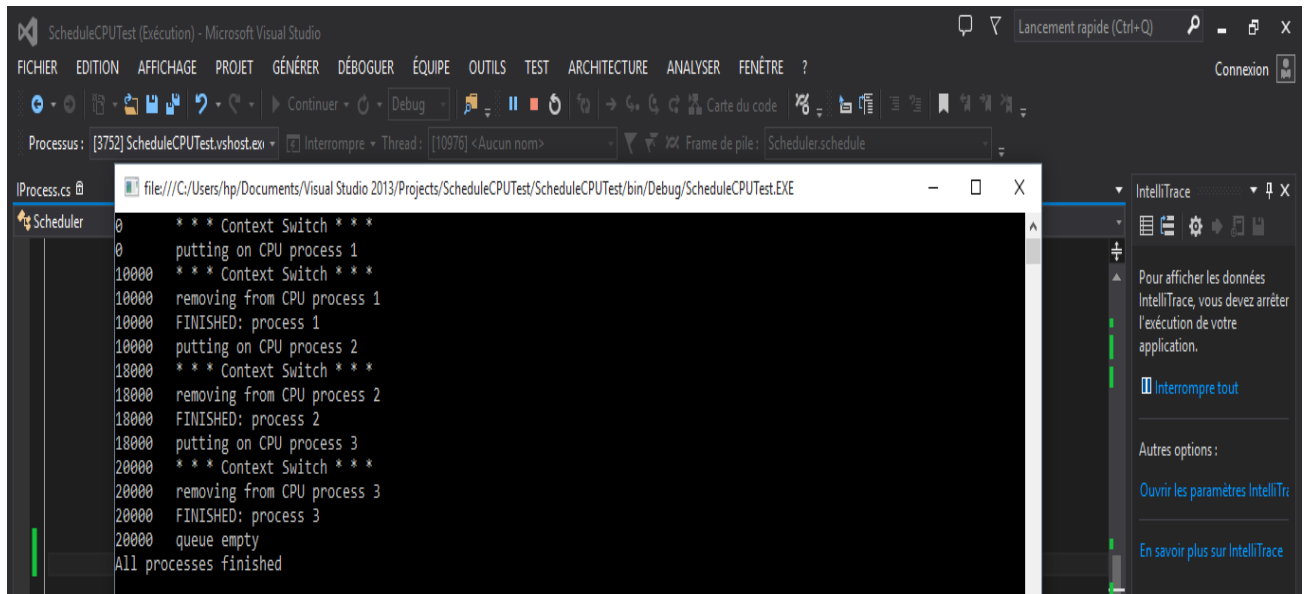
### Méthode Tourniquet

```

file:///C:/Users/hp/Documents/Visual Studio 2013/Projects/ScheduleCPUTest/ScheduleCPUTest/bin/Debug/ScheduleCPUTest.EXE
0      * * * Context Switch * * *
0      putting on CPU process 1
2000   * * * Context Switch * * *
2000   removing from CPU process 1
2000   adding to queue process 1
2000   putting on CPU process 2
4000   * * * Context Switch * * *
4000   removing from CPU process 2
4000   adding to queue process 2
4000   putting on CPU process 3
6000   * * * Context Switch * * *
6000   removing from CPU process 3
6000   FINISHED: process 3
6000   putting on CPU process 1
8000   * * * Context Switch * * *
8000   removing from CPU process 1
8000   adding to queue process 1
8000   putting on CPU process 2
10000  * * * Context Switch * * *
10000  removing from CPU process 2
10000  adding to queue process 2
10000  putting on CPU process 1
12000  * * * Context Switch * * *
12000  removing from CPU process 1
12000  adding to queue process 1
12000  putting on CPU process 2
14000  * * * Context Switch * * *
14000  removing from CPU process 2
14000  adding to queue process 2
14000  putting on CPU process 1
16000  * * * Context Switch * * *
16000  removing from CPU process 1
16000  adding to queue process 1
16000  putting on CPU process 2
18000  * * * Context Switch * * *
18000  removing from CPU process 2
18000  FINISHED: process 2
18000  putting on CPU process 1
20000  * * * Context Switch * * *
20000  removing from CPU process 1
20000  FINISHED: process 1

```

## Méthode FIFO



```

file:///C:/Users/hp/Documents/Visual Studio 2013/Projects/ScheduleCPUtest/ScheduleCPUtest/bin/Debug/ScheduleCPUtest.EXE
Scheduler
0      * * * Context Switch * * *
0      putting on CPU process 1
10000  * * * Context Switch * * *
10000  removing from CPU process 1
10000  FINISHED: process 1
10000  putting on CPU process 2
18000  * * * Context Switch * * *
18000  removing from CPU process 2
18000  FINISHED: process 2
18000  putting on CPU process 3
20000  * * * Context Switch * * *
20000  removing from CPU process 3
20000  FINISHED: process 3
20000  queue empty
All processes finished
  
```

### Conclusion :

Pour une compréhension adéquate, il est important de comprendre qu'ici le Premier algorithme de Tourniquet présente un quantum de 2 secondes. Après deux secondes le processus qui est dans le processus doit laisser place à un autre processus même si il n'a pas fini son traitement ; Le Second algorithme ne tient plus compte du quantum de 2 secondes. Le 1er processus entre et termine son exécution, après c'est le 2e processus qui va s'exécuter puis le 3e processus va finir

**Ainsi fait, est le travail demandé !**