Processus :

I. Définitions :
   1. Processus ou tâche : instance d'un programme

- Code :
- Mémoire :
- IP :
- Pile :
- numéro de processeur :
- numéro de processus.

Tous ces éléments définissent le contexte d'exécution du processus.

Un système peut exécuter plusieurs processus en même temps :
- en confiant les différents processus à des processeurs différents.
- En partageant le temps d'un processeur et en attribuant ces fractions de temps aux différents processus : chaque processus s'exécutera pendant une des fractions de temps processeur.

Pour passer d'un processus à un autre sur un même processeur, il suffit de sauvegarder le contexte et de charger un autre contexte.

2. Thread (ou processus léger) dans un processus :

- Code
- Utilise la mémoire du processus : partage avec les autres Thread.
- IP
- Pile
- Les thread d'un même processus partagent les mêmes ressources.
- Numéro de thread.

Un thread ne peut faire qu'une seul chose à la fois : il exécute la liste des instructions sur laquelle pointe le compteur de programme IP.

3. Partage des ressources : imaginer qu'il y ait plusieurs professeurs dans la salle, comment cela va-t-il se passer ? (très mal ?)

De même quand nous utilisons le tableau, je ne doit pas effacer le tableau tant vous y accédez (lisez ou recopiez).

Solutions :

a) Concurrence sans précaution : tous le monde accède à tout, c'est l'anarchie ! Mais parfois ça marche !
b) Exclusion mutuelle : quand un processus utilise une ressource commune, il interdit à tous les autres processus d'y accéder. C'est le concept d'**exclusion mutuelle ou mutex**.
La zone dans laquelle le programme utilise la ressource partager s'appelle une **section critique**. On appelle une fonction pour entrer dans la section critique et une autre fonction pour en sortir.

Technique de mutex:

- Masquage d'interruption : interdire tout changement de contexte.
- Alternance stricte : une variable interne indique qui a le droit sur la ressource. (utilisation d'une instruction TEST AND SET des microprocesseur.)
algo :

Thread A	Thread B

// drapeau initialement à 0
Tant que (vrai)
Début
…
tant que (drapeau <> 0) // attendre
rien;
// entrer dans la section critique
…
Drapeau<-1 // sortir de la section critique
…
fin
	Tant que (vrai)
	Début
	…
	tant que (drapeau <> 1) // attendre
	rien;
	//entrer dans la section critique
	…
	Drapeau<- 0 //sortir de la section critique
	…
	fin

Problème : le processus qui attend occupe le processeur inutilement.

- Sémaphore :
  - Entrée : down : si valeur>0 alors valeur<-valeur – 1
    sinon dormir
  - Sortie : Up : incrément valeur.
    Si des processus sont en attente alors
    début
    en réveiller un au hasard
    valeur <-0.
    fin

- Dead-lock : étreinte mortelle ou interblocage.

Problème du diner des philosophes à la table ronde : 5 philosophes sont assis autour d'une table ronde, chacun est devant un plat de spaghettis. Le philosophe a besoin de 2 fourchettes pour manger, entre chaque plat il y a une fourchette.
La vie d'un philosophe se résume à manger et à penser alternativement. Quand un philosophe a faim, il essaie de prendre sa fourchette à sa droite, ensuite sa fourchette à sa gauche ou inversement. S'il peut les prendre, il mange un moment, remet les fourchettes sur la table et se remet à penser.

Un exemple d'algorithme :

4. Communication :

- Tube : analogie avec un fichier ou une file.
- Message : Windows, socket, port,
- Mémoire : partage une mémoire commune, il doit y avoir un mécanisme de partage de ressources.
- Appel de procédure : procédure « callback ». Quand on crée un thread, on lui passe une adresse de fonction contenu dans le processus père.

5. le Programme = processus.

Au lancement d'un processus un seul thread est crée : le « primary thread ».

II. Exemple d'application : serveur simple

1. Introduction :

Pour répondre aux requêtes des clients (messages passant par un port), un serveur doit écouter en permanence se qui se passe sur le port. Pour cela, il se sert d'une chaussette ou plutôt d'une socket.
Mais pendant qu'il écoute, il ne peut plus s'occuper des messages Windows !

Il faut donc utiliser deux processus légers ou thread dans le programme :

- Le premier s'occupera des messages Windows.
- Le deuxième s'occupera des messages reçus via la socket.

2. Cahier des charges : le programme devra afficher dans une fenêtre les messages reçus sur le port 40000 au moyen du protocole TCP.
3. On créera un formulaire avec une TextBox pour afficher les messages reçus. On utilisera le composant BackgroundWorker pour créer le thread d'écoute du port.
4. Proposition d'implémentation du thread d'écoute du port : voir page suivante.

```
class Ecoute40000
   {   int Port = 40000;  // port 40000
       TcpListener Serveur;   // Socket TCP
       String s;
       bool bLu;   // vrai si , on a reçu un message
       public Ecoute40000()
       {
          s = "";
          bLu = true;
       }
       public void Run() {
          try
          {
             Serveur = new TcpListener( Port);
             Serveur.Start();
             byte[] buffer = new byte[256];
             byte[] bufferReponse = new byte[256];
             while (true)  // Boucle d'écoute attente d'un client
             {        // appel bloquant attendant la réception d'une demande
                TcpClient client = Serveur.AcceptTcpClient();   // bloque le thread
                NetworkStream   Flux = client.GetStream();   // flux de donnée reçues
                int i;
                     // lecture des octets envoyés tant que l'on reçoit des messages
                while ((i = Flux.Read(buffer, 0, buffer.Length)) != 0)
                {
                   bLu = false;
                   // convertir en chaîne de caractères
                   s = System.Text.Encoding.ASCII.GetString(buffer, 0, i);
                   // réponse au client
                   bufferReponse = System.Text.Encoding.ASCII.GetBytes("OK");
                   Flux.Write(bufferReponse, 0, bufferReponse.Length);
                   break;  // une seule poignée de main
                } // comment fermer la connexion ?
                Flux.Close();
                client.Close(); //fermer la connexion
             }
          }
          catch (SocketException e)
          {
             MessageBox.Show("Erreur de liaison avec le client : "
+Environment.NewLine + e.Message);
          }
          finally { Serveur.Stop(); }
       }
}
```

5. Ajouter à cette classe un mécanisme afin de passer le texte reçu au thread primaire afin de l'afficher dans l'EditBox.
6. Implémenter le tout et tester avec le programme Client fournit par le professeur.

    - D'abord en locale.
    - Puis à partir d'une autre machine du réseau « Pédagogie ».
    - Expliciter les modifications que vous avez effectuées sur la configuration des postes (notamment au niveau du parefeu ou de l'exécution préventive).