


|   |   |            |
|---|---|------------|
|  | TRAVAUX PRATIQUES N°4   | Année : I4 |
|   | Matière : Programmation multitâches<br><br>Remis par : S.Pomportes / Y. Guerroudj<br>J. Olive / C. Sarroche | Temps : 4h |

## OBJECTIFS

Utiliser les sémaphores pour résoudre les paradigmes de la concurrence du producteur consommateur.

Ce TP utilise la librairie graphique OpenGL – Glut.

## REMARQUES

Vous placerez l'ensemble de vos fichiers sources dans le répertoire PMTP4 de votre répertoire personnel (commande mkdir \$HOME/PMTP4).

Vous rédigerez un compte rendu au format openoffice ou txt.

Chaque fichier (compte rendu ou code) doit comporter vos noms au début.

# Thread et OpenGL

## Producteur / Consommateur

### I. Déplacement d'une balle

Le but de cet exercice est d'écrire un programme qui génère une fenêtre OpenGL, dessine une balle et l'anime.

Pour ce faire, le processus principal créera deux threads.

- Le premier (thread\_balle) changera les coordonnées du centre du disque. (x,y sont des variables globales au processus).
- Le deuxième s'occupera de l'interface OpenGL (création et rafraichissement de l'image).

Ci-dessous, vous trouverez le code commenté du Thread\_affichage initialisant la fenêtre OpenGL :

```
Void *Thread_affichage (void *arguments)
```

```
{
```

```
    int nbarg=1;
```

```
    glutInit(&nbarg,(char**) arguments); // fonction d'initialisation
```

```
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB); // utilise le codage RGB en double
```

```
buffer
```

```
    glutInitWindowSize(500,500); // taille de la fenêtre que l'on va créer
```

```
    glutInitWindowPosition(50,10); // position de la fenêtre sur l'écran
```

```
    glutCreateWindow("Bonjour"); // nom la fenêtre
```

```
glClearColor(1.0,1.0,1.0,0.0); // défini la couleur d'effacement par défaut, ici blanc
glOrtho(0,500,0,500,-1.0,1.0) ; // défini la position du repère dans la scène visible
```

```
glutDisplayFunc(display); // défini la fonction appelée pour afficher la scène
glutIdleFunc(affiche); // défini la fonction lancée lorsque le système n'a plus rien à faire
glutMainLoop(); // fonction bloquante permettant la gestion de l'affichage
}
```

La fonction affiche est lancée par OpenGL lorsque le système n'a pas d'autre opération à gérer. Dans notre cas, cette fonction réaffichera la scène toutes les 10ms :

```
Void affiche()
{
    glutPostRedisplay() ;
    usleep(10000) ;
}
```

La fonction display sera ainsi exécutée toutes les 10ms. Dans l'exemple ci-dessous, une ligne est tracée à l'écran.

```
Void display ()
{
    glClear (GL_COLOR_BUFFER_BIT) ; // efface le buffer
    glColor3f(0.0,0.0,1.0); // défini la couleur (RVB)
    glBegin(GL_LINES) ; // trace des lignes entre des couples de points
        glVertex3f(10,10,0.0) ;// point 1
        glVertex3f(100,100,0.0) ;// point 2
    glEnd ;
    glutSwapBuffers() ; // affiche à l'écran le buffer dans lequel nous avons dessiné
}
```

Questions à réaliser dans l'ordre:

- 1- La balle traverse l'écran et le programme s'arrête.
- 2- Les spécificités de la balle sont regroupées dans une structure (vitesse, taille, direction, couleur,...). Lors du lancement du programme, ces spécificités sont définies aléatoirement.
- 3- La balle rebondi un nombre aléatoire de fois contre les bords de l'écran avant de disparaître (paramètre supplémentaire à ajouter à la structure balle).

## II. Déplacement de n balles

En reprenant l'exercice précédent, réalisez un programme lançant n balles de couleurs et tailles différentes, rebondissant un nombre de fois limité avant de disparaître.

Chaque balle est en fait une thread. Le programme principal créera la thread affichage puis les n thread balles (qui seront sous la forme d'un tableau de threads).

Un tableau de structure balle de n cases sera utilisé pour définir les propriétés de chacune des balles qui seront affichée à l'écran quand elles sont vivantes (un champ booléen « vivant » sera alors ajouté à la structure balle).

Vous penserez à utiliser des sémaphores pour protéger les variables si besoin.

## II. Producteur / Consommateur de balles

Nous imaginons maintenant un producteur / consommateur de balles.

La thread principale lancera de nouvelles balles régulièrement (temps aléatoire). Un tableau circulaire sera créé pour informer des cases vides du tableau de structure balles.

Un producteur consommateur sera mis en place pour arrêter la création de balles lorsque toutes les ressources balles sont utilisées et pour recréer des balles lorsqu'une d'entre elles aura disparue.

