EXAM : NORMAL SESSION

Programmation : simulation d'un ordonnanceur, utilisation de la STL (12 points)

On veut simuler le fonctionnement d'un ordinateur, et pour cela, on vous demande d'implémenter une classe *'Processus'* et une classe *'Ordonnanceur'*. Les 2 classes dérivent d'une classe *'Objet'* avec une méthode **virtuelle pure afficher() (justifier pourquoi ce choix)**

Un processus est défini par son identifiant (id), sa durée de traitement , et sa priorité (priotity) de 1 à 10.

Un ordonnanceur est une file de processus (utilisez la file de priorité : **priority_queue** qui insère l'élément à sa bonne position par rapport à sa priorité.

Le fonctionnement de l'ordonnanceur : Il sort le processus à traiter, le traite (décrémenter sa durée de traitement du quantum temps (fixé en millisecondes par le développeur dans le programme), puis le remet dans la file.

Pour utiliser la priority_queue, on utilise la classe foncteur *'Comparaison'* avec un opérateur de comparaison qui retourne un booléen selon les priorités des processus.

```
// Foncteur de comparaison selon les priorités est donnée dan la classe  ci-dessous :
class Comparaison {
public:
   bool operator()(const Processus &p1, const Processus  &p2)    {  return p1.priority < p2.priority ;    }
};
// Construit la classe Ordonnaceur avec une file de priorité:
 priority_queue<Processus, vector<Processus>, Comparaison>  pq;
```

Votre programme sera constitué de 4 classes : Objet, Processus, Ordonnanceur et la classe fournie(Comparaison).

Simulation (la méthode main)

1. Le programme doit générer un nombre de processus <u>choisi en ligne de commande par l'utilisateur</u> au lancement du programme. Pour chaque processus, l'utilisateur fixe sa priorité et sa durée. La saisie de ces paramètres doit être sécurisé(exceptions). Ensuite chaque processus est ajouté dans l'ordonnanceur.
2. L'ordonnanceur traite les processus par ordre de priorité (**priority_queue**). Il prend le processus le plus prioritaire dans la liste, et lui accorde un <u>quantum temps</u> de traitement.
3. Après chaque traitement d'un processus, l'ordonnanceur réduit la durée de traitement du processus traité du quantum temps, et lui décrémente sa priorité d'une unité jusqu'à 1 puis elle reste constante.
   Lorsque la durée de traitement d'un processus est épuisée (terminé), l'ordonnanceur le supprime de la liste.
4. Afficher cette simulation sur l'écran. Afin de bien suivre l'ordonnanceur, utiliser la primitive **sleep(unsigned int ns)** définie dans <unistd.*h*> pour ajouter une petite temporisation à chaque tour de traitement

Annexe : compilation

Compilation : *votreRepertoire> g++ nomFile1.c nomFile2.c -Wall -o nomExecutable*
Execution sous Windows : *votreRepertoire>nomExecutitable param1 param2, ...*
Execution sous Linux : *votreRepertoire>./nomExecutitable param1 param2, ...*

Annexe : méthodes de la classe template priority_queue<T>

<u>**empty**</u> : Test whether container is empty (public member function)
<u>**size**</u> : Return size (public member function)
<u>**top**</u> : Access top element (public member function)
<u>**push**</u> : Insert element (public member function)
<u>**emplace**</u> : Construct and insert element (public member function)
<u>**pop**</u> : Remove top element (public member function)
<u>**swap**</u> : Swap contents (public member function)