Mini projet: Allocateur

1 Objectif

Le but du mini-projet est de simuler un allocateur de ressources pour un système centralisé multi-processus. Cette simulation nécessite diffèrents outils de modélisation, issus notamment de la théoèrie des graphes.

Votre travail se déroulera en deux phases successives : spécification puis réalisation. La première phase devra aboutir à un dossier de spécifications, décrivant en détail la modélisation de l'allocateur ainsi que son fonctionnement suivant ce modèle. Cette phase a pour seul support le papier (et la réflexion), et ne donnera lieu à aucune programmation ; celle-ci aura lieu pendant la deuxième phase du projet, qui ne pourra démarrer qu'après validation de vos spécifications par l'enseignant responsable de votre groupe de TP.

La deuxième phase consiste à réaliser le simulateur sous la forme d'un programme en `Python`, en utilisant les ressources logicielles disponibles (bibliothèque de classes de graphes). Vous rendrez ce programme, dûment commenté, accompagné de votre dossier de spécifications dans sa version finale.

1.1 Objet de la simulation : l'allocateur et son environnement

L'allocateur gère un nombre fixe de ressources. Le nombre de processus est variable. L'allocateur doit être capable d'attribuer une ou plusieurs ressources à un processus qui en fait la demande. Il doit veiller à la détection et à la résolution d'interblocages. Fonctionnellement parlant, l'interblocage est défini comme une situation comportant un groupe de processus bloqués,

ceux-ci ne pouvant redevenir actifs qu'au prix de la destruction de l'un de ces processus.

Il vous sera demandé lors de ce projet d'une part de mettre en évidence (entre autres) ces interblocages à l'aide d'un modèle utilisant la théorie des graphes, et d'autre part de décrire les différentes situations pouvant conduire à des interblocages, notamment à l'aide de scénarios.

On considèrera comme impératives les règles suivantes :

R1 chaque ressource est attribuée à au plus un processus ;

R2 si le processus demandeur n'est pas connu de l'allocateur, il est créé ;

R3 une ressource demandée disponible est immédiatement attribuée au processus demandeur ;

R4 si la ressource est déjà attribuée, le processus demandeur est mis en attente de la ressource ;

R5 si une ressource est libérée, elle est attribuée au plus ancien processus demandeur, ou bien elle devient disponible si elle n'est plus demandée ;

R6 un processus est actif si toutes ses demandes de ressources ont été satisfaites ;

R7 un processus est bloqué s'il est en attente d'au moins une des ressources demandées ;

R8 un processus actif peut demander plusieurs ressources à la fois, il devient bloqué si au moins l'une des ressources demandées est indisponible, mais ce blocage n'interviendra qu'à la fin du traitement de la demande ;

R9 un processus actif peut libérer une ou plusieurs ressources qu'il détient ;

R10 un processus bloqué ne peut demander ni libérer une ressource tant qu'il n'est pas redevenu actif ;

R11 un processus détruit libére toutes les ressources qu'il détient.

1.2 Mise en oeuvre de la simulation

La simulation à réaliser concerne l'allocateur, car le système environnant (qui inclut les processus et les ressources) n'a aucune existence réelle. Toutes

les demandes de création, de destruction, d'allocation, etc. qui dans la réalité proviendraient du système sont ici simulées par des ordres de l'utilisateur, saisis au clavier. Chaque ordre est soumis à l'allocateur qui, tenant compte des règles imposées ainsi que de sa connaissance de l'état du système, accepte ou refuse cet ordre, en modifiant au passage les informations décrivant le système, si besoin.

Voici la liste des ordres :

O1 création d'un processus

O2 destruction d'un processus

O3 demande de ressource(s) par un processus

O4 libération d'une ressource par un processus

O5 affichage des files d'attente par ressource

O6 affichage des processus actifs

O7 affichage des attentes entre processus

O8 affichage des processus concernés par un interblocage

L'interface utilisateur permettant à l'utilisateur de soumettre ses ordres devrait être assurée par une fonction appelée `ReadUserInstructions`, qui vous devrez programmer.

2 Consignes

2.1 Le dossier de spécifications

Votre dossier comportera les parties suivantes :

1. *Règles et définitions supplémentaires*
   C'est dans cette partie que vous pourrez exprimer les règles spécifiques découlant de votre analyse du problème, en complément de celles données plus haut. A la différence des règles de modélisation décrites ci-après, ces règles décrivent les fonctionnalités du système, visibles par l'utilisateur, et non la manière dont ce système sera représenté par le logiciel. Quelques exemples :

R12 en cas d'interblocage, les ordres acceptés sont limités à ... (liste à préciser) ;

R13 un processus actif peut/ne peut pas demander ...

2. *Règles de modélisation*
   Ces régles décrivent le modèle permettant de représenter l'état du simulateur, basé sur un graphe ainsi que des ressources. Les règles de modélisation se subdivisent à leur tour en invariants et variants.

   Les invariants ont un numéro de la forme MIn ; ils décrivent de manière aussi complète que possible l'état du graphe et des ressources en fonction de l'état du système, indépendamment de toute modification. Exemples :

   MI1 les sommets du graphe représentent ... (à compléter)

   MI2 les arcs du graphe représentent ... (à compléter)

   MI3 un interblocage intervient dans ... (état(s) particulier(s) du modèle, exprimé(s) de manière précise par certaines propriétés)

   MI4 l'affichage décrit dans (ordre Ox) est obtenu par .. (parcours du modèle à définir)

   Les variants ont un numéro de la forme MVn ; ils décrivent en détail toutes les transformations qu'il y a lieu d'appliquer au modèle suite aux changements d'états du système. Ces transformations doivent bien entendu respecter les invariants. Exemples :

   MV1 La création d'un processus entraîne ... (modifications à préciser)

   MV2 La destruction d'un processus entraîne ...

   MV3 La demande de ressources par un processus entraîne ...

   MV4 la libération d'une ressource par un processus entraîne ...

3. *Scénarios*
   Dans cette partie vous représenterez de manière exhaustive une ou plusieurs séquences de dialogue entre l'utilisateur et l'allocateur, afin d'illustrer la mise en application d'un maximum de règles, les vôtres aussi bien que celles qui vous sont imposées. L'enjeu est de couvrir un maximum de situations et de règles en un minimum d'étapes, afin d'éviter un trop grand nombre de celles-ci.

   Chaque scénario est une suite cohérente d'étapes. Pour chaque étape, vous préciserez l'ordre donné, la réaction apparente de l'allocateur, l'état interne de l'allocateur suivant l'ordre, et enfin les règles qui ont été appliquées, ainsi que d'éventuels commentaires.

   L'intérêt des scénarios est double : d'une part ils permettent de valider dans la mesure du possible vos choix de modélisation, et d'autre part ils serviront de jeux de test pour votre programme lors de la phase de réalisation.

   Voici un bref exemple :

   Scén1 Etape1
   ordre = création du processus P1 (O1)
   graphe = P1
   règles = R2, MV1

   Scén1 Etape2
   ordre = demande des ressources R1 et R2 par le processus P1 (O3)
   graphe = P1
   file R1 = (P1)
   file R2 = (P1)
   règles = R3, R8, MV3

   Scén1 Etape3
   ordre = création du processus P2 (O1)
   graphe = P1, P2
   files inchangées
   règles = R2, MV1

   Scén1 Etape4
   ordre = demande de la ressource R1 par le processus P2 (O3)

graphe = (à préciser)
file R1 = (P1, P2)
file R2 = (P1)
règles = R4, R8, MV3

Scén1 Etape5
ordre = affichage des processus actifs (O7)
affichage = Processus actif(s) : P1
graphe et files inchangés
règles = R6, R7, MI4

2.2 Phase de réalisation

La réalisation du mini-projet a pour objet un programme `Python` incluant les différentes parties à implémenter. Il vous sera demandé de réaliser un code respectant les spécifications ci-dessus et n'utilisant pas les fonctions de parcours existantes en dans `NetworkX`.

La mise au point et la validation de votre programme se feront à l'aide de jeux de test. Un jeu de test est une séquence d'ordres accompagnée d'un ensemble de résultats "nominaux" prévus à l'avance. La séquence d'ordres est pré-enregistrée sous la forme d'un fichier texte, qui sera lu dans votre environnement. Les résultats effectifs obtenus à l'exécution doivent évidemment se conformer aux résultats nominaux. Ils devront être sauvegarder dans un fichier texte et afficher sur l'écran.

Au final, il vous sera demandé de rendre un dossier, au format .pdf, comportant :

— des explications sur les choix de mise en oeuvre : algorithmes, fonctions, etc. ;
— les modifications éventuelles apportées à votre dossier de spécifications ;
— description de vos scénarios de tests.

Dans ce dernier fichier vous pouvez reprendre certaines parties de vos spécifications.

Dans une archive vous donnerez

— un fichier python contenant fonctions spécifiques commentées ;

— un ou plusieur fichiers python contenant les jeux de test correspondant à vos scénarios ;
— un ou plusieurs fichier contenant les données correspondant à vos jeux de test.

3 Consignes pour les rendus

3.1 Instruction pour rendus

Ces consignes sont valables à la fois pour le rendu du dossier de spécification ainsi que pour le dossier final.

— Tous les fichiers dans une seule archive.
— Pour le dossier de spécification, l'archive sera nommée *spec_nom1_nom2* et pour le dossier final elle sera nommée *final_nom1_nom2*.
— Chaque dossier sera envoyé par mail à votre encadrant de TP, à l'adresse qu'il vous indiquera. Le titre du mail sera *AlgG_spec_nom1_nom2* ou *AlgG_final_nom1_nom2* selon le dossier.

3.2 Conseils

Organisez et structurez bien vos rapports. Soignez le français. Évitez toute ambiguïté dans vos définitions.