Semestre : Automne 2013

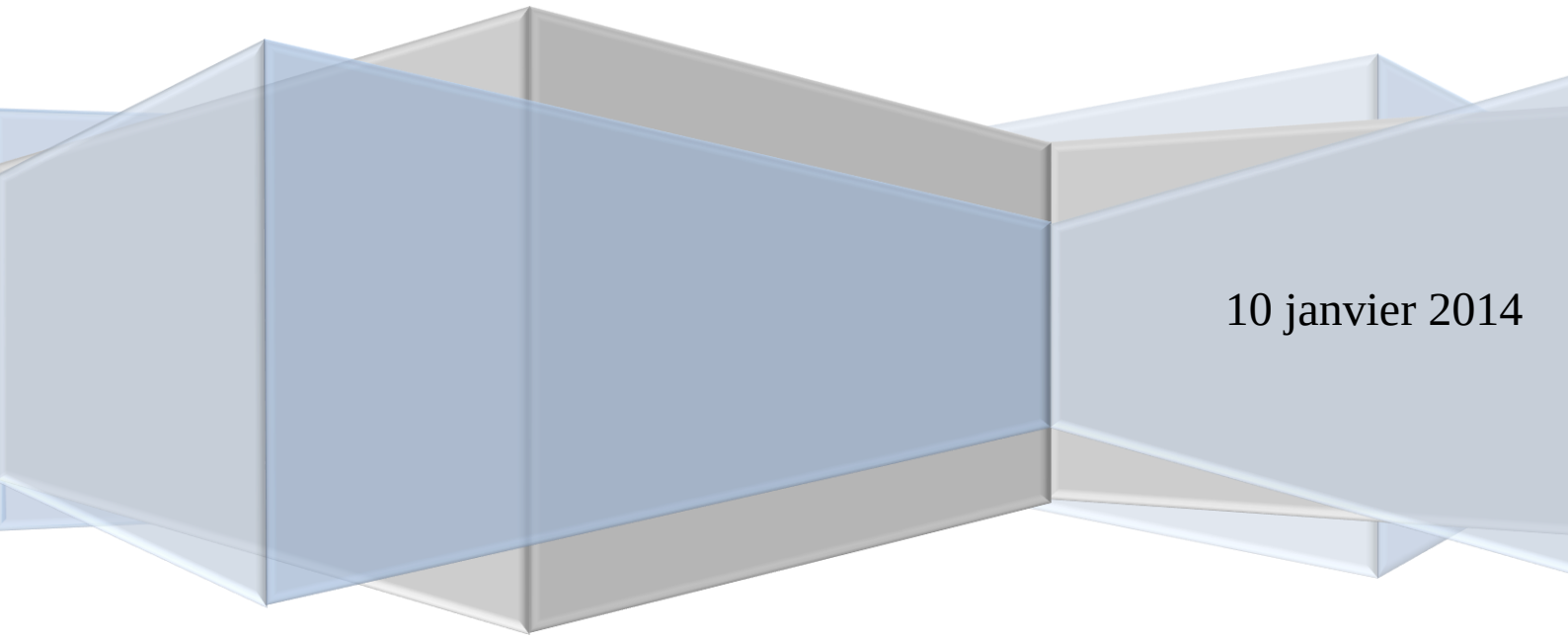
LO41

Rapport de projet

Sujet : Chaîne de montage en anneau

Professeur : *Philippe Descamps*

Etudiant : *Olivier Schweitzer*



10 janvier 2014

Sommaire

I. Introduction

Dans le cadre de l'UV LO41, il nous a été demandé de réaliser un projet de simulation d'une chaîne de montage de fabrication d'une gamme de produits. Ce projet a pour objectif de mettre en œuvre les concepts étudiés en cours.

1. Objectifs

Le but de ce projet est de réaliser un programme capable de simuler une chaîne de montage en anneau autour duquel se situe des robots qui vont récupérer des composants ou des produits en cours de fabrication afin de, soit, dans le cas des composants, les stocker jusqu'à en avoir assez pour créer un produit, soit, s'il récupère des produits, de leur appliquer une opération propre au robot.

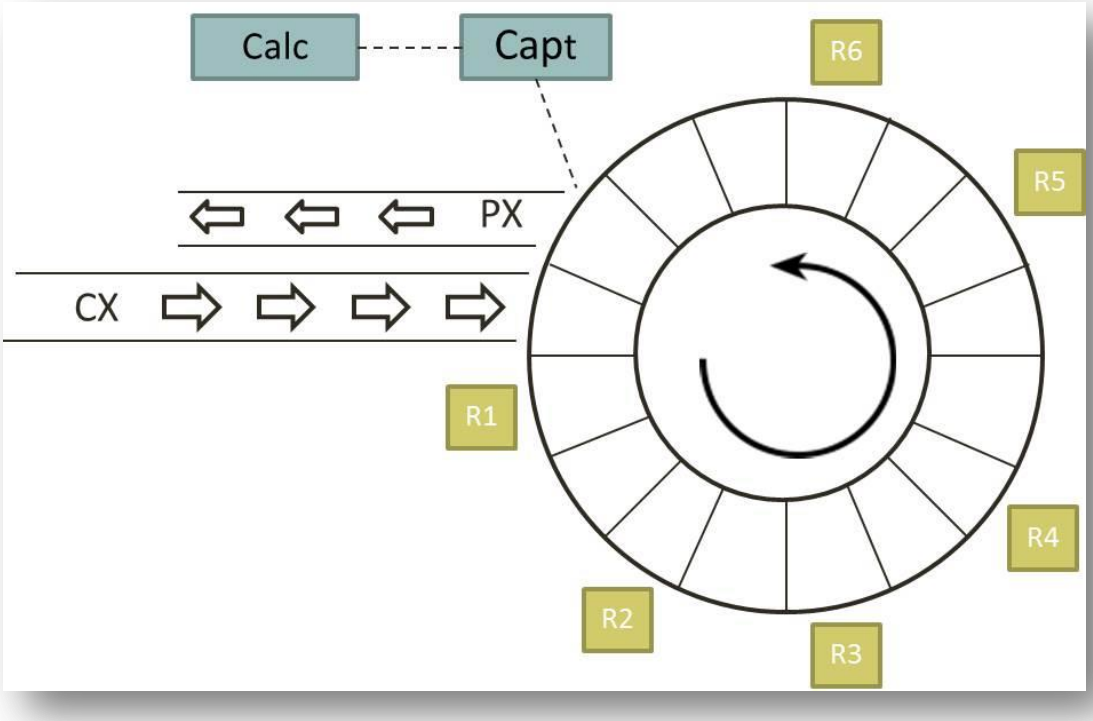


Figure 1 : Schéma de la chaîne de montage

Les robots une fois leur opération ou création terminée, reposeront leur produit dans la première section libre de l'anneau.

Cet anneau est approvisionné en composant par une file d'attente qui dispose de composant de chaque type, mais placé aléatoirement dans la file.

Ce système disposera de deux modes, le mode normal et le mode dégradé. Dans le premier mode chaque robot ne sera associé qu'à une seule opération tandis que dans le second mode certain robot pourront se substituer ou compléter les opérations d'autres robots ainsi si un des robots ne fonctionne plus ou si la cadence de production augmente, le système fonctionnera toujours.

Une fois les produits terminés un capteur les dirige vers la sortie et prévient un calculateur qui compte les produits sortis.

2.Contraintes

Les notions travaillées en cours ne s'appliquant pas à tous les systèmes, certaines contraintes délimitent naturellement le sujet :

- Une contrainte forte est imposée sur le langage à utiliser, en effet, le C est un langage dédié à la programmation système. C'est donc le candidat idéal pour mettre en œuvre les concepts vu en cours.
- L'implémentation des différents éléments du système doit se faire soit avec des processus, soit avec des threads. De plus la synchronisation est fixée selon le choix précédent. Avec des processus, la synchronisation se fera à l'aide des sémaphores tandis qu'avec des threads elle se fera avec des moniteurs et des mutex.

II. Réalisation technique

1. Analyse du problème

Le sujet proposé est une réalisation technique d'une chaîne de montage en anneau, en vue d'exploiter les notions vues en cours et travaillées durant les TP. L'analyse du sujet fait ressortir plusieurs points à mettre en œuvre :

- En mode normal, chaque robot doit effectuer plus ou moins les mêmes opérations à savoir :
    - Prendre le composant si celui-ci lui est utile et s'il passe devant sa section
    - Prendre le produit s'il doit effectuer une opération sur celui-ci et vérifier si les opérations précédentes nécessaires ont été effectuées, en effet les opérations sur un produit sont séquentielles et doivent donc être effectuées dans le bon ordre.
    - Déposer le produit stocké (que ce soit un nouveau produit (si le robot a le bon nombre de composants, il crée un nouveau produit), ou un produit

en cours de fabrication) devant lui si la section est vide sinon il attend qu'elle le soit.

- En mode dégradé, que l'utilisateur peut choisir d'activer, les robots doivent communiquer entre eux afin que ce ne soit pas toujours le même robot qui prenne le même composant, le robot qui en a le plus besoin doit le prendre.
- L'anneau doit tourner à une certaine cadence et se synchroniser avec la file d'attente (le tapis des composants).
- La file d'attente doit créer des composants aléatoirement les faire avancer jusque dans l'entrée de l'anneau ou s'arrêter si l'anneau est plein.
- Le capteur doit repérer les produits finis et les faire sortir de l'anneau puis en informer le calculateur qui fait les comptes.

2. Modélisation du système

Ce système a été modélisé à l'aide d'un réseau de Pétri (voir en annexe). Le principe est le suivant :

- La file crée des composants aléatoirement (C1, C2, C3 ou C4)
- Ces composants sont envoyés dans l'anneau jusqu'au premier robot (R1)
  - Si le composant est utile au robot, il le prend et incrémente son compteur de composant
  - Si le robot ne peut pas utiliser le composant devant lui, il le laisse passer et le robot suivant effectuera la même opération.
  - Si aucun robot n'a pris le composant (par exemple si le robot fait une opération sur un produit il ne peut pas prendre de composant), ce dernier revient au robot 1.
- Une fois qu'un robot a le bon nombre de composants, il peut créer un produit et faire la première opération de ce dernier
- Si la section devant le robot est occupée, il attend qu'elle soit libre
- Si la section est libre, le robot pose le produit qui va se diriger vers un autre robot. Ce dernier va regarder s'il peut effectuer une opération sur le produit.

- Si le produit doit subir dans son processus de fabrication une opération de ce robot et que les opérations précédentes ont été effectuées alors le robot prend le produit.
- Sinon le produit continuera sa route

- Une fois que toutes les opérations sont effectuées sur un produit, il est terminé et il sort du système. Un capteur va alors prévenir un calculateur afin de compter le nombre de produits fabriqués par le système. Une fois les quotas atteints, le système s'arrête.

3. Choix d'implémentation

Concernant la réalisation du projet en elle-même, la plupart des notions de cours ont été utilisées.

L'anneau, la file, les robots, le capteur et le calculateur ont été gérés par des threads. Ce choix se justifie par un meilleur contrôle pour ce qui est de la création, du traitement et de la synchronisation entre les différents éléments du système.

La synchronisation a été effectuée à l'aide de mutex et de moniteur afin de protéger les ressources critiques.

Les produits et les composants sont construits à l'aide d'une même structure (voir ci-dessous), ainsi un composant est un produit sans *etat*, sans *operations* et sans *type_produit*, à l'inverse un produit n'a pas de *type_composant*. Cette solution a été choisie, car l'anneau, tout comme la file, est implémenté sous forme de tableau de produits.

```
//Structure d'un produit ou d'un composant
typedef struct
{
    int etat;
    int type_produit;
    int operations[5];

    int type_composant;
```

```
}Produit;

Produit anneau[TAILLE_ANNEAU]; // Anneau de 16 sections
Produit file_attente[TAILLE_FILE]; // File d'attente des composants
```

Chaque robot est implémenté à l'aide d'une structure contenant le thread qui le gère, un compteur de composant et un produit permettant de stocker un produit qui doit subir une opération. Les 6 robots sont stockés dans un tableau.

```
typedef struct
{
    pthread_t operation;
    int compteur_composant;
    Produit p;

}Robot; //Objet robot
```

Le capteur est implémenté via un thread et le calculateur est défini par une structure contenant le thread pour le gérer ainsi qu'un tableau d'entier qui fait office de compteur de produits, en effet chaque produit est associé à un compteur.

```
typedef struct
{
    pthread_t analyse;

}Capteur;

typedef struct
{
    int compteurProduit[4];
    pthread_t arret_produit;

}Calculateur;

Calculateur calc;
Capteur capteur;
```

III. Résultats

1. Mode normal

Tout d'abord le programme rempli la file avec des composants aléatoires. Ensuite il crée les différents threads, d'abord celui de gestion de l'anneau, puis celui de la file, ceux des robots et ceux du capteur et du calculateur.

L'anneau et la file sont synchronisés à l'aide d'un moniteur, l'anneau attend que la file ait mis un composant dans la section d'entrée. Même si la section est déjà occupée la file envoie quoi qu'il en soit un signal pour réveiller l'anneau. Ce dernier se réveille et tourne puis, il se remet à attendre le signal de la file.

Pendant ce temps chaque robot regarde dans la section devant eux s'il y a un composant ou produit à prendre, ou si la section est vide, ils regardent s'ils ont un produit à déposer. Les 6 robots, utilisant les mêmes procédures, sont synchronisés à l'aide d'un mutex, bloquant ainsi l'accès aux ressources critiques.

L'anneau et les robots sont également synchronisés à l'aide d'un mutex, empêchant ainsi les robots d'accéder à leur section d'anneau lorsque celui-ci est en train de tourner.

Le capteur, qui regarde si le produit devant lui est terminé, utilise le même mutex que l'anneau et les robots, ainsi lors de sa lecture du produit il s'assure que l'anneau ne tourne pas.

Le capteur et le calculateur se synchronisent eux aussi grâce à un moniteur, en effet le calculateur attend que le capteur lui signale qu'un produit terminé est sorti. A ce moment-là, le calculateur se réveille, il incrémente le compteur du produit terminé si celui-ci n'a pas atteint son maximum, et, si tous les compteurs de produits ont atteint leur maximum, le calculateur appelle la fonction de l'arrêt du système.

Si l'anneau est plein, il commence à incrémenter un compteur permettant de compter le nombre de rotations effectuées, et lorsque l'anneau a fait un tour et s'il est toujours plein, on supprime les composants présents sur l'anneau ainsi que les produits n'ayant qu'une seule opération d'effectuée. Si le problème n'est toujours pas résolu et au bout d'un certain nombre de rotations, on supprime tous les produits stockés sur les robots.

Pour que le programme se termine, il faut que le calculateur ait le bon nombre de produits P1, P2, P3 et P4. Si c'est le cas, il va appeler une fonction qui va mettre fin à tous les threads et va détruire les mutex et les moniteurs évitant ainsi les fuites de mémoire.

Le programme gère aussi les arrêts forcés, en effet lorsque l'utilisateur procède à un arrêt forcé (Ctrl+C), le signal est intercepté, le programme arrête les threads et supprime les mutex et les conditions des moniteurs.

2. Mode dégradé

En raison d'un manque de temps et afin de fournir un projet stable et fonctionnel, ce mode n'a pas été implémenté, en revanche une réflexion a tout de même été menée quant aux problématiques de ce mode.

- En mode dégradé, les robots peuvent prendre plusieurs types de composants, et donc créer plusieurs types de produits. Afin d'éviter à ce qu' un seul robot ne prenne tous les composants d'un même type, il serait judicieux de mettre en place une communication entre les robots via une file de messages (entre R1 et R2 et entre R3 et R4).
    - Exemple : lorsqu'un composant C1 se présentera devant R1, avant de le prendre, il demandera à R2 s'il n'en a pas plus besoin que lui (un robot qui a besoin de 3 C1 et qui dispose déjà de 2 C1 sera prioritaire par rapport à un robot qui ne possèdera qu'un seul C1). Dans le cas d'une égalité, un tirage aléatoire pourra être implémenté.

- Dans le cas des opérations, la communication n'est pas nécessaire puisque c'est la cadence de l'anneau et le temps des opérations qui répartiront les produits. En effet, pendant que R1 fait une opération1 sur un produit, un autre produit

pourra lui passer devant, mais ce n'est pas grâce puisque R2 pourra éventuellement le prendre. Ce système permettra d'accélérer le processus de fabrication des produits.

- Les robots R5 et R6 n'ayant plus d'utilité, ils pourront éventuellement être soit mis en pause (si l'on souhaite éventuellement revenir au mode normal), soit complètement arrêtés.

- Le programme devra permettre à l'utilisateur de déclencher le mode dégradé en traitant un signal par exemple (le passage à nouveau en mode normal pourra se faire de la même façon).

IV. Conclusion

1. Compétences acquises

Ce projet m'a permis de mettre en application directe les principes des systèmes d'exploitation, d'en comprendre les problématiques et d'y apporter des solutions. L'utilisation des threads m'a permis de mieux comprendre leur fonctionnement et de réfléchir sur les problématiques de synchronisation et protection des ressources critiques qui sont des questions essentielles au niveau des systèmes d'exploitation et même au-delà.

De plus ce projet m'a permis d'acquérir une plus grande aisance dans l'utilisation du langage C et de l'IDE CodeBlocks ainsi que dans la gestion et la modélisation (réseau de Petri) d'un projet.

2. Difficultés rencontrées

Lors de la réalisation de ce projet, j'ai été confronté à divers problèmes, principalement liés à la synchronisation et au choix des structures des différents éléments. Concernant la synchronisation, la mise en place des mutex et des moniteurs n'a pas été aisée notamment lorsqu'il a fallu définir les ressources critiques.

Pour ce qui est de l'implémentation des éléments, il m'a fallu faire des choix de conception afin de respecter au mieux le sujet tout en offrant une modularité et en conservant une cohérence.

3. Améliorations

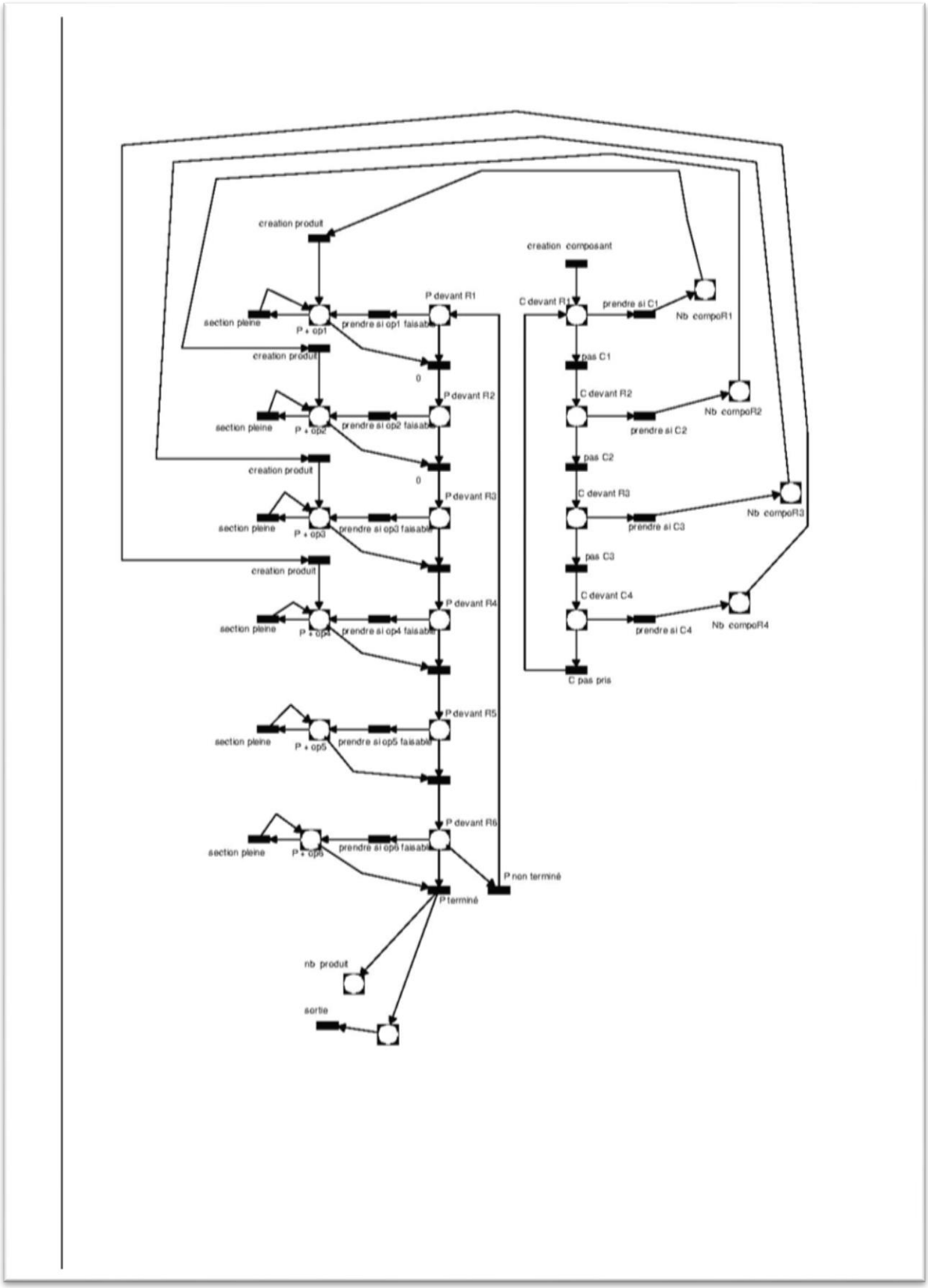
Ce projet n'étant pas entièrement terminé, il est évident qu'il reste des choses à améliorer :

- Implémenter le mode dégradé.

- Adaptation du programme à un nombre plus important de robots avec d'autres opérations.
- Une interface plus évoluée permettant une meilleure interprétation des résultats.
- Offrir la possibilité à l'utilisateur de changer les options sans passer par le code (via un fichier de configuration par exemple).

Pour terminer, ce projet rentre parfaitement dans le cadre de l'UV car il permet d'utiliser un grand nombre de concepts vus en cours et de se confronter aux problématiques des systèmes d'exploitation afin de mieux les appréhender.

Annexe


creation produit
creation composant
P devant R1
C devant R1
prendre si C1
Nb compoR1
section pleine
P + op1
prendre si op1 faisable
creation produit
0
pas C1
C devant R2
Nb compoR2
P devant R2
section pleine
P + op2
prendre si op2 faisable
prendre si C2
creation produit
0
pas C2
C devant R3
Nb compoR3
P devant R3
section pleine
P + op3
prendre si op3 faisable
prendre si C3
creation produit
pas C3
C devant C4
P devant R4
section pleine
P + op4
prendre si op4 faisable
prendre si C4
Nb compoR4
C pas pris
P devant R5
section pleine
P + op5
prendre si op5 faisable
P devant R6
section pleine
P + op6
prendre si op6 faisable
P non terminé
P terminé
nb produit
sortie