

RAPPORT DE PROBLÈME DINING HALL

Nom: Nguyễn Sang

Code d'étudiant: 18126029

1. Le rôle de l'utilisation du sémaphore dans la résolution de problèmes:

- Le nombre de sémaphore utilisé pour résoudre ce problème est 2 sémaphore avec 2 variables globales. Les deux sémaphores que j'utilise dans le code sont **mutex** et **okToLeave**. Les deux variables globales **eating** et **readyToLeave** représentent les états.

+) mutex:

- mutex est utilisé pour protéger deux variables **readyToLeave** et **eating**. Sinon, la vérification de la condition entraîne des problèmes inattendus.

```

63  getFood(arg);
64  sem_wait(&mutex);
65  eating++;
66  if( eating == 2 && readyToLeave == 1){
67      sem_post(&okToLeave);
68      readyToLeave--;
69  }
70  sem_post(&mutex);
71  dine(arg);
72  sem_wait(&mutex);
73

```

```

78  if( eating == 1 && readyToLeave == 1 ){
79      sem_post(&mutex);
80      waiting1(arg);
81      sem_wait(&okToLeave);
82  }

```

Supposons que c'est deux processus légers: étudiant A et étudiant B.

Cas 1: Utilisation du sémaphore mutex:

- D'après l'image, le mutex sémaphore de la ligne 64 à 70. Cela signifie que pendant que l'étudiant A met à jour les deux variables eating et readyToLeave, l'étudiant B ainsi que l'autre processus ne peuvent rien faire tant que l'étudiant A n'a pas fini de mettre à jour ces deux variables. → synchronisation

Cas 2: ne pas utiliser de mutex sémaphore.

- Pendant que l'étudiant B vérifie les conditions et voit que la variable eating est égale à 1 à la ligne 78 et que la condition renvoie Vrai. Cependant, l'étudiant A met soudainement à jour la variable eating de 1 à 2, de sorte que la condition que l'étudiant B vérifie n'est plus Vrai → le résultat peut ne pas être correct. - pas de synchronisation.

+) okToLeave:

- okToLeave est utilisé pour vérifier s'il faut autoriser ou non les étudiants à quitter la salle à manger en vérifiant les différentes conditions décidées par les deux variables **eating** et **readyToLeave**.

```

77
78 □ if( eating == 1 && readyToLeave == 1 ){
79     sem_post(&mutex);
80     waiting1(arg);
81     sem_wait(&okToLeave);
82 }
83 □ else if ( eating == 0 && readyToLeave == 2){
84     sem_post(&okToLeave);
85     readyToLeave -= 2;
86     waiting2(arg);
87     sem_post(&mutex);
88 }
```

- Supposons que c'est deux processus légers: étudiant **A** et étudiant **B**.
- L'étudiant **A** est prêt à partir et un autre étudiant **B** mange à table dans la salle de manger (dining hall). Comme vous pouvez le voir, la ligne 78 représente qu'il y reste un étudiant **B** en train de manger à table (**eating == 1**) et que l'étudiant **A** est prêt à partir (**readyToLeave == 1**). La ligne 81 (**sem_post(&okToLeave)**) indique que l'étudiant A est bloqué à partir.
- L'étudiant **A** peut partir si et seulement si l'étudiant **B** a fini manger. Si l'étudiant B a fini de manger, alors le variable "eating" sera soustraite de 1 (**eating--**). Par conséquent, la ligne 83 indique que personne n'est en train de manger dans la salle à manger (**eating == 0**) et que l'étudiant B est maintenant prêt à partir avec l'étudiant A (**readyToLeave == 2**).
- La commande dans la ligne 84 (**sem_post(&okToLeave)**) sera utilisée pour annoncer que l'étudiant B a fini manger et qu'il est prêt à partir et puis l'étudiant A reçoit un signal dans la ligne 81 (**sem_wait(&okToLeave)**) et l'étudiant A et B peuvent partir.

2. Implémentation:

- Langage de programmation: C
- Part1 pour déclarer les variables globales, les sémaphores et certaines fonctions utilisées pour représenter l'étape à laquelle se trouve un processus léger (comme **getFood()**, **dine()**, **finishDining()**, **leave()**, **waiting()...**).
- Part 2 est fonction pour ordonnancer les processus légers dans le problème Dining Hall void ***diningHall(void *arg)**. Et le paramètre **arg** est utilisé pour différencier les processus légers
- Part 3 est **main** fonction, utilisé pour déclarer et gérer le processus légers...

Le code d'implémentation:

```
//Part 1
#include <pthread.h>
#include <stdio.h>
#include <semaphore.h>
#include <unistd.h>
#include <stdlib.h>

sem_t mutex;
sem_t okToLeave;
//global variable
int eating = 0;
int readyToLeave = 0;

void getFood(void *arg){
    char *message;
    message = (char *) arg;
    printf("%s ",message);
    printf("got food\n");
}

void dine(void *arg){
    char *message;
    message = (char *) arg;
    printf("%s ",message);
    printf("is dining\n");
    if(arg == "Student 3") //to make the student 3 stay in the dining hall longer.
        usleep(5000);
    if(arg == "Student 4") //to make the student 4 stay in the dining hall longer.
        usleep(3000);
}

void finishDining(void *arg){
    char *message;
    message = (char *) arg;
    printf("%s ",message);
    printf("has just done dining\n");
}

void leave(void *arg){
    char *message;
    message = (char *) arg;
```

```

        printf("%s ",message);
        printf("left\n");
    }

void waiting1(void *arg){
    char *message;
    message = (char *) arg;
    printf("%s ",message);
    printf("is waiting for other friends to finish dining\n");
}

void waiting2(void *arg){
    char *message;
    message = (char *) arg;
    printf("%s ",message);
    printf("finds that there is nobody in the dining hall, so he is ready to leave\n");
}
//part 2
void *diningHall(void *arg){
    getFood(arg);
    sem_wait(&mutex);
    eating++;
    if( eating == 2 && readyToLeave == 1){
        sem_post(&okToLeave);
        readyToLeave--;
    }
    sem_post(&mutex);
    dine(arg);
    sem_wait(&mutex);
    eating--;
    finishDining(arg);
    readyToLeave++;
    if( eating == 1 && readyToLeave == 1 ){
        sem_post(&mutex);
        waiting1(arg);
        sem_wait(&okToLeave);
    }
    else if ( eating == 0 && readyToLeave == 2){
        sem_post(&okToLeave);
        readyToLeave -= 2;
        waiting2(arg);
        sem_post(&mutex);
    }
}

```

```

    }
    else{
        readyToLeave--;
        sem_post(&mutex);
    }
    leave(arg);
    return NULL;
}

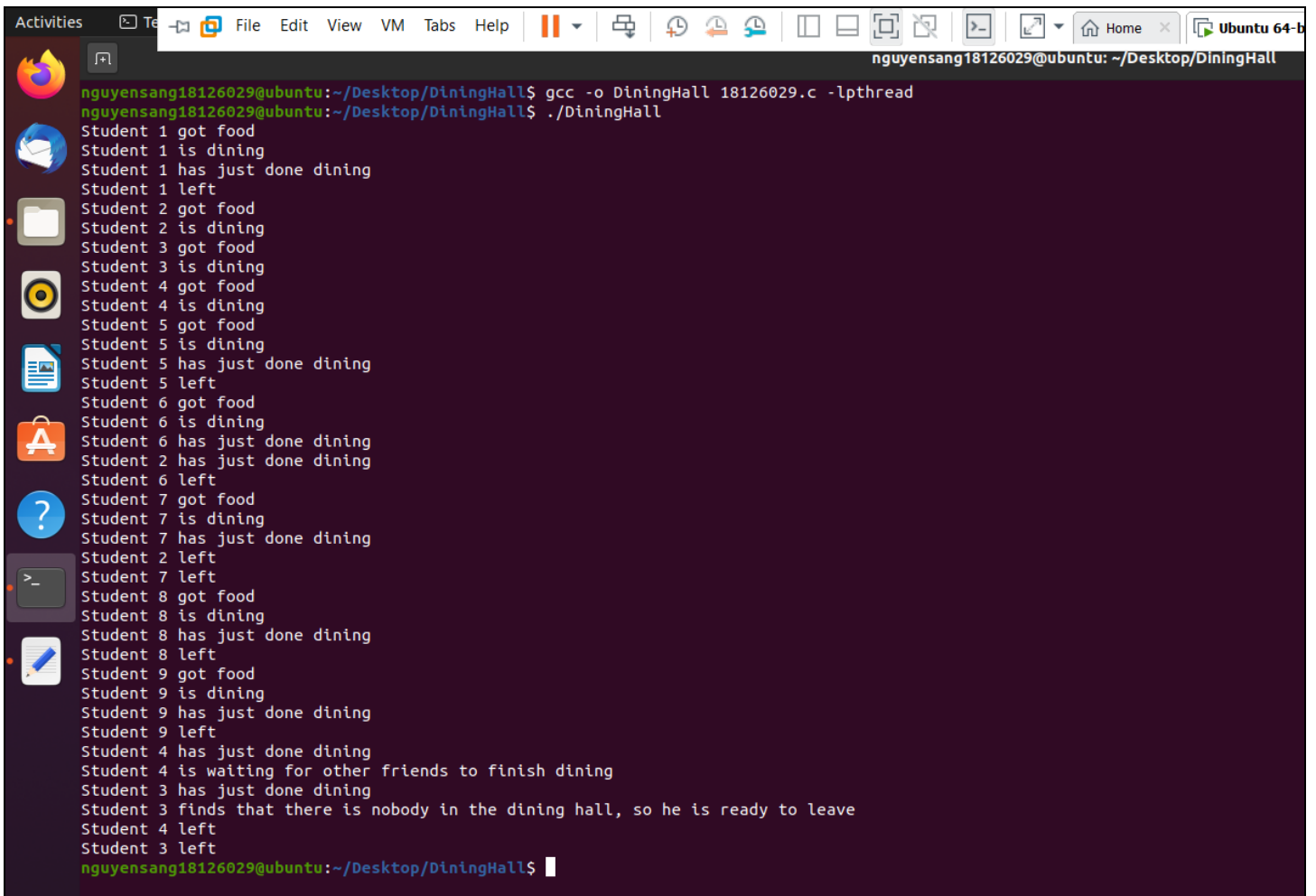
//part 3
int main(void){
    sem_init(&mutex, 0, 1);
    sem_init(&okToLeave, 0, 0);
    pthread_t pth1, pth2, pth3, pth4, pth5, pth6, pth7, pth8, pth9;
    char *signal1 = "Student 1";
    char *signal2 = "Student 2";
    char *signal3 = "Student 3";
    char *signal4 = "Student 4";
    char *signal5 = "Student 5";
    char *signal6 = "Student 6";
    char *signal7 = "Student 7";
    char *signal8 = "Student 8";
    char *signal9 = "Student 9";
    pthread_create(&pth1, NULL, diningHall, (void*)signal1);
    pthread_create(&pth2, NULL, diningHall, (void*)signal2);
    pthread_create(&pth3, NULL, diningHall, (void*)signal3);
    pthread_create(&pth4, NULL, diningHall, (void*)signal4);
    pthread_create(&pth5, NULL, diningHall, (void*)signal5);
    pthread_create(&pth6, NULL, diningHall, (void*)signal6);
    pthread_create(&pth7, NULL, diningHall, (void*)signal7);
    pthread_create(&pth8, NULL, diningHall, (void*)signal8);
    pthread_create(&pth9, NULL, diningHall, (void*)signal9);
    pthread_join(pth1, NULL);
    pthread_join(pth2, NULL);
    pthread_join(pth3, NULL);
    pthread_join(pth4, NULL);
    pthread_join(pth5, NULL);
    pthread_join(pth6, NULL);
    pthread_join(pth7, NULL);
    pthread_join(pth8, NULL);
    pthread_join(pth9, NULL);
    sem_destroy(&mutex);
}

```

```
sem_destroy(&okToLeave);  
return 0;  
}
```

3. Resultats:

Le code exécuté en terminal dans le systèmes d'exploitation **Linux** (terminal):



```
nguyensang18126029@ubuntu: ~/Desktop/DiningHall  
nguyensang18126029@ubuntu:~/Desktop/DiningHall$ gcc -o DiningHall 18126029.c -lpthread  
nguyensang18126029@ubuntu:~/Desktop/DiningHall$ ./DiningHall  
Student 1 got food  
Student 1 is dining  
Student 1 has just done dining  
Student 1 left  
Student 2 got food  
Student 2 is dining  
Student 3 got food  
Student 3 is dining  
Student 4 got food  
Student 4 is dining  
Student 5 got food  
Student 5 is dining  
Student 5 has just done dining  
Student 5 left  
Student 6 got food  
Student 6 is dining  
Student 6 has just done dining  
Student 2 has just done dining  
Student 6 left  
Student 7 got food  
Student 7 is dining  
Student 7 has just done dining  
Student 2 left  
Student 7 left  
Student 8 got food  
Student 8 is dining  
Student 8 has just done dining  
Student 8 left  
Student 9 got food  
Student 9 is dining  
Student 9 has just done dining  
Student 9 left  
Student 4 has just done dining  
Student 4 is waiting for other friends to finish dining  
Student 3 has just done dining  
Student 3 finds that there is nobody in the dining hall, so he is ready to leave  
Student 4 left  
Student 3 left  
nguyensang18126029@ubuntu:~/Desktop/DiningHall$
```

Le code exécuté en IDLE DevC dans le systèmes d'exploitation **Window** (l'écran console):

```
C:\Users\ASUS-PC\Desktop\Semaphore_DiningHall.exe
Student 1 got food
Student 2 got food
Student 5 got food
Student 4 got food
Student 3 got food
Student 2 is dining
Student 2 has just done dining
Student 2 left
Student 4 is dining
Student 3 is dining
Student 1 is dining
Student 1 has just done dining
Student 1 left
Student 8 got food
Student 8 is dining
Student 8 has just done dining
Student 8 left
Student 7 got food
Student 9 got food
Student 5 is dining
Student 5 has just done dining
Student 5 left
Student 9 is dining
Student 9 has just done dining
Student 9 left
Student 6 got food
Student 7 is dining
Student 7 has just done dining
Student 7 left
Student 6 is dining
Student 6 has just done dining
Student 6 left
Student 4 has just done dining
Student 4 is waiting for other friends to finish dining
Student 3 has just done dining
Student 3 finds that there is nobody in the dining hall, so he is ready to leave
Student 3 left
Student 4 left

-----
Process exited after 0.04741 seconds with return value 0
Press any key to continue . . .
```