

Compte Rendu – CSE –  
Colm RYAN et Emmanuel STONE  
(Excusez nous les erreurs de Français svp)

Un Fast Food possède 4 types d'employés.

1. Les prevoirs d'ordre, qui enregistrent les commandes des clients.
2. Les cuisiniers qui preparent les repas.
3. Les conditionneurs, qui placent les aliments dans des sachets.
4. Les caisseurs qui remettent les sachets aux clients et encaissent l'argent.

Chacun des employés peut être considéré comme étant un processus séquentiel communiquant.

Quelle forme de communication interprocessus veut-ils utiliser? Reliez ce modèle en comportement des processus sous unix. (Tannenbaun p.165 no. 29)

Comment aborder le problème?

Objectif:

Comprendre la situation de manière à repondre, par exemple à la question-  
“Combien d'employés de chaque type pour un débit maximal?”

Idées de la cours:

- Ça fait une chaîne
- Y'en qui sont indépendants/ ou dépendants
- Preveurs d'ordres aussi caissiers
- Il n'a pas de ressource critique
- Si 1 client commande 1 seul repas doit être préparé
- Connaitre le temps d'execution de chaque tâche
- Voir chaque type d'employé comme un producteur/consommateur

Une solution de la problème utilise une système des objets (commandes sur papier dans le Fast Food) pour représenter les tâches différents.

Chaque cuisiner (et aussi les autres employés) est un Producteur/Consommateur individuel et la production et la gestion de tous les repas est contrôlé par les commandes et des moniteurs.

Nous avons dessiner le diagram de control dessus.

Nous avons écrire une modèle de la système en Java pour cuisiniers et les preneurs aussi

```

public class Resto
{
    public static void main(String[] args)
    {
        Monitor monitor=new Monitor();
        Monitor frites=new Monitor();
        Monitor boissons=new Monitor();
        Monitor steak=new Monitor();
        Preneur p=new Preneur("Preneur1",monitor);
        Cuisineur c=new Cuisineur
("Cuisineur1",monitor,frites,boissons,steak);
        //new Cuisineur("Cuisineur2",ordres).start();
        p.start();
        c.start();
    }
}

import java.*;
public class Cuisineur extends Thread
{
    Monitor monitor,boisson, frites, steak;
    public Cuisineur(String str,Monitor m,Monitor f, Monitor b,
Monitor s)
    {
        super(str);
        monitor=m;
        frites=f;
        boisson=b;
        steak=s;
    }

    public void run()
    {
        while(true)
        {
            Repas r=monitor.remove();
            prepare(r);
            System.out.println("Order " + r.getID()+"
Prepared");
        }
    }

    public void prepare(Repas repas)
    {
        try{
            sleep(repas.getNum() * 1000);
        }catch(InterruptedException e){}
    }
}

```

```

        switch (repas.getNum())
        {
            case 1:
                frites.insert(repas);
                break;
            case 2:
                steak.insert(repas);
                break;
            case 3:
                boisson.insert(repas);
                break;
        }
    }
}

import java.*;
public class Preneur extends Thread

{
    Monitor monitor;
    int i;
    public Preneur(String str,Monitor m)
    {
        super(str);
        monitor=m;
        i=0;
    }

    public void run()
    {
        while(true)
        {
            i=i+1;
            try
            {
                sleep((long)Math.random()*3000);
            }
            catch(InterruptedException e){}
            System.out.println("Order "+i+" Taken");
            monitor.insert(new Repas(2,i));
        }
    }
}

public class Repas
{
    int num, id;
    public Repas(int n, int i)
    {
        n=num;
        id=i;
    }
}

```

```

    }
    public int getNum()
    {
        return num;
    }
    public int getID()
    {
        return id;
    }
}

```

```

import java.util.*;
class Monitor {
    private final int N = 5;
    private int count = 0;
    private Vector theData;

    synchronized public void insert(Repas o) {
        while (count == N) {
            try{
                wait(5000); // Full, wait to add
            } catch (InterruptedException ex) {};
        }

        insert_item(o);
        count++;
        if(count == 1) {
            notify(); // Not empty, notify a
                    // waiting consumer
        }
    }

    synchronized public Repas remove() {
        Repas data;
        while(count == 0)
            try{
                wait(5000); // Empty, wait to consume
            } catch (InterruptedException ex) {};

        data = remove_item();
        count--;
        if(count == N-1){
            notify(); // Not full, notify a
                    // waiting producer
        }
        return data;
    }

    private void insert_item(Repas data){

```

```

        theData.addElement(data);
    }

    private Repas remove_item(){

        Repas data = (Repas) theData.firstElement();
        theData.removeElementAt(0);
        return data;
    }

    public Monitor(){
        theData = new Vector();
    }
}

```

Exemple:

```

Order 1 Taken
Order 2 Taken
Order 1 Prepared
Order 2 Prepared
Order 3 Taken
Order 3 Prepared
Order 4 Taken
Order 5 Taken
Order 4 Prepared
Order 5 Prepared
Order 6 Taken
Order 7 Taken
Order 6 Prepared
Order 8 Taken
Order 7 Prepared
Order 8 Prepared
Order 9 Taken
Order 10 Taken
Order 9 Prepared
Order 11 Taken
Order 10 Prepared
Order 11 Prepared
Order 12 Taken
Order 13 Taken
Order 12 Prepared
Order 14 Taken
.....

```

### Commandes sur papier

