Processus et temps partagé

Christophe Viroulaud

Terminale - NSI

Archi 08

Dans les années 60, **Multics** est un des premiers systèmes d'exploitation avec :

- système de fichier hiérarchique,
- temps partagé,
- multitâche préemptif,
- multi-utilisateur.

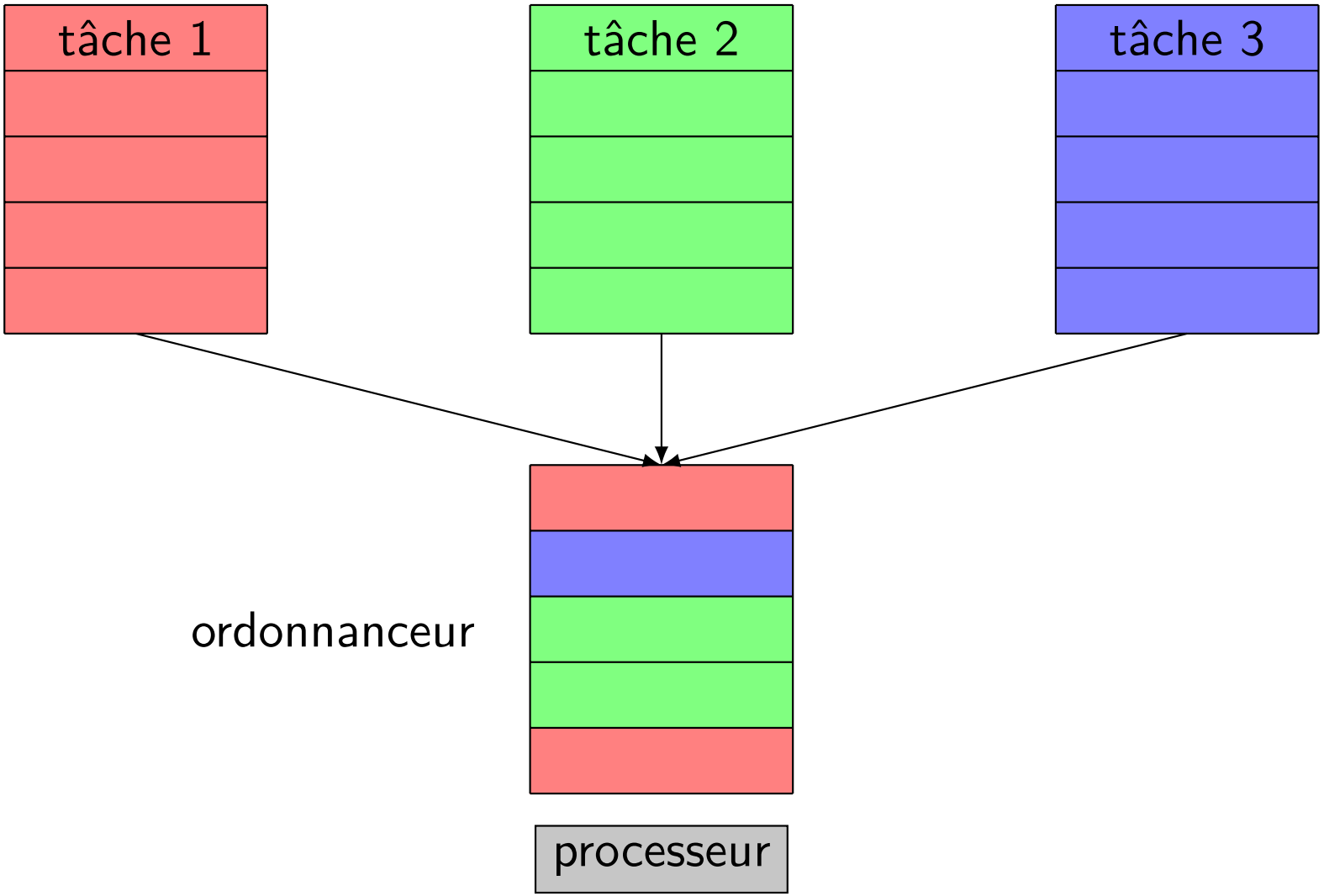


FIGURE 1 – Système mono-processeur

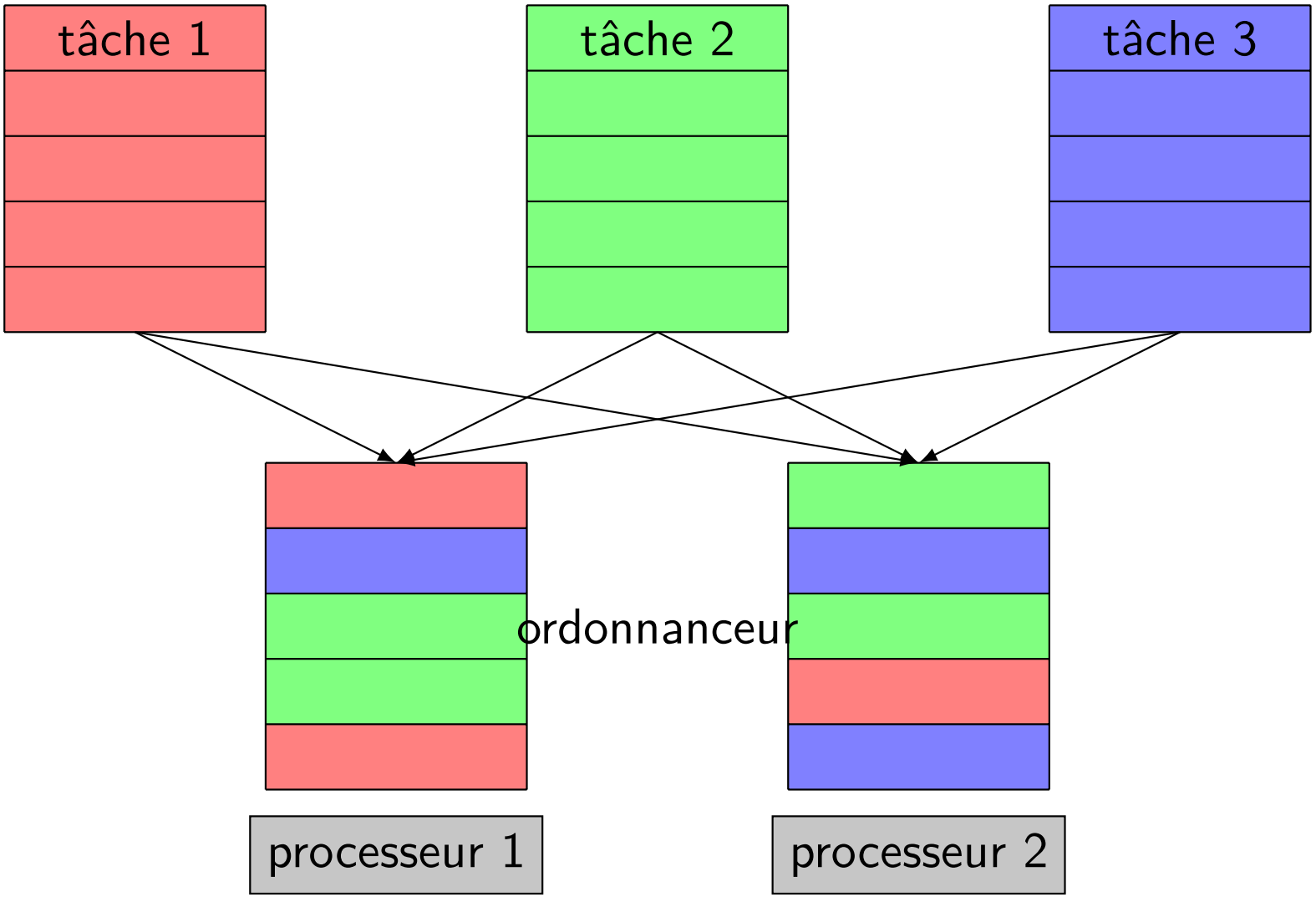


FIGURE 2 – Système multi-processeurs

Activité 1 : Identifier les problèmes éventuels dans les deux types d'architectures.

Correction

- Deux tâches ont besoin d'utiliser la même ressource.
- Une tâche a besoin de réaliser certaines actions dans un ordre précis.

Comment ordonner les processus pour minimiser les erreurs d'exécution ?

Sommaire

Programmation séquentielle

```
1  def f1():
2      for _ in range(5):
3          print("appel 1")
4
5  def f2():
6      for _ in range(5):
7          print("appel 2")
8
9  f1()
10 f2()
```

Code 1

Activité 2 : Exécuter le code 1.

```
1  appel 1
2  appel 1
3  appel 1
4  appel 1
5  appel 1
6  appel 2
7  appel 2
8  appel 2
9  appel 2
10 appel 2
```

Code 2 – L'interpréteur exécute les lignes de code séquentiellement.

Sommaire

Programmation en parallèle - notion de **thread**

Les ordinateurs possèdent maintenant plusieurs processeurs. Il semble alors pertinent de partager les tâches pour accélérer l'exécution des processus.

Un thread est un processus qui va s'exécuter simultanément avec d'autres thread en partageant l'espace des données.

```python
from threading import Thread
from time import sleep

def f1():
    for _ in range(5):
        print("appel 1")
        sleep(0.01)

def f2():
    for _ in range(5):
        print("appel 2")
        sleep(0.01)

p1 = Thread(target=f1)
p2 = Thread(target=f2)
p1.start()
p2.start()
```

Code 3 – Deux thread en parallèle

```
1  appel 1
2  appel 2
3  appel 1
4  appel 1
5  appel 2
6  appel 1
7  appel 2
8  appel 1
9  appel 2
10 appel 2
```

Code 4 – L'interpréteur exécute les lignes de code séquentiellement.

Sommaire

Répartition des tâches : solution à risques

```
1  from time import sleep
2  INTERMEDIAIRE = 0
3
4  def calcul():
5      global INTERMEDIAIRE
6      print("section non critique 1")
7      for c in range(400):
8          temp = INTERMEDIAIRE
9          # simule un traitement long
10         sleep(0.000000001)
11         INTERMEDIAIRE = temp + 1
12     print("section non critique 2")
13
14 calcul()
15 print(INTERMEDIAIRE)
```

Code 5 – Simuler un calcul (séquentiel) long

Activité 4 :

1. Télécharger et extraire le dossier compressé `processus-prog-concurrente-annexe`.
2. Ouvrir le fichier `compteur_sequentielle.py` qui simule un calcul long.

Commentaire

- La variable globale `INTERMEDIAIRE` simule un résultat qui évolue quand le traitement long est terminé.
- La boucle est la **section critique** du calcul.

La variable globale `INTERMEDIAIRE` contient 400 à la fin de l'exécution.

Partager les traitements longs

```
1 tab_threads = []
2 # Lance en parallèle 4 exécutions de calcul
3 for i in range(4):
4     p = Thread(target=calcul)
5     p.start()
6     tab_threads.append(p)
```

Code 6 – Exécution en parallèle

Le code 6 lance quatre processus en parallèle pour réaliser les calculs longs.

Activité 5 :

1. Ouvrir le fichier `compteur_thread.py`.
2. Dans la boucle de la fonction, remarquer que le nombre d'itérations a été divisé par 4.
3. Exécuter le programme.
4. Qu'observe-t-on ? Comment expliquer ce résultat ?

Correction

- La variable `temp` est locale : elle dépend de chaque appel de la fonction.
- La variable `INTERMEDIAIRE` est globale : elle est *partagée* entre les appels.

Un scénario possible :

- **`thread 1:`** $temp_1 = 10$
- **`thread 2:`** $temp_2 = 10$
- **`thread 3:`** $temp_3 = 10$
- **`fin thread 1:`** $INTERMEDIAIRE = temp_1 + 1 = 11$
- **`fin thread 2:`** $INTERMEDIAIRE = temp_2 + 1 = 11$
- **`thread 4:`** $temp_4 = 11$
- **`fin thread 3:`** $temp_3 = 11$
- **`fin thread 4:`** $INTERMEDIAIRE = temp_4 + 1 = 12$

`INTERMEDIAIRE` vaut 12 alors qu'il y a eu 4 appels à la fonction **`calcul`**.

Observation

On ne maîtrise pas l'ordre d'exécution des threads.

Contrôler la *section critique*

Il est possible de bloquer l'exécution d'un thread pour l'obliger à attendre la fin d'une **section critique**.

Activité 6 :

1. Ouvrir le fichier `compteur_verrou.py`.
2. Interpréter le rôle du verrou.

- `acquire` prend le verrou ou attend qu'il se libère.
- `release` libère le verrou.

`INTERMEDIAIRE` vaut 100 à la fin de l'exécution du programme.

Sommaire

Interblocage - exemple de situation

En 1997, la mission Mars Pathfinder rencontre un problème alors que le robot est déjà sur Mars. Après un certain temps, des données sont systématiquement perdues. Les ingénieurs découvrent alors un bug lié à la synchronisation de plusieurs tâches.

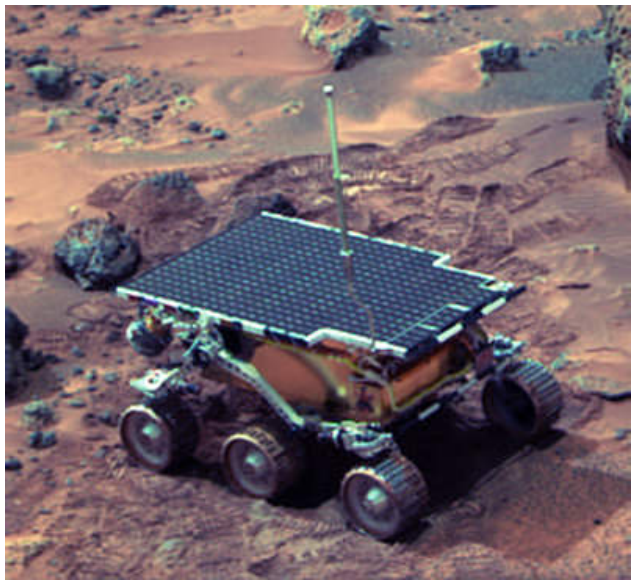


FIGURE 3 – Robot Pathfinder

Considérons un robot qui possède 3 ressources :

- des **moteurs** qui lui permettent de se déplacer,
- une liaison **wifi** qui lui permet de communiquer,
- une **caméra** qui filme son environnement.

Il peut réaliser 3 tâches :

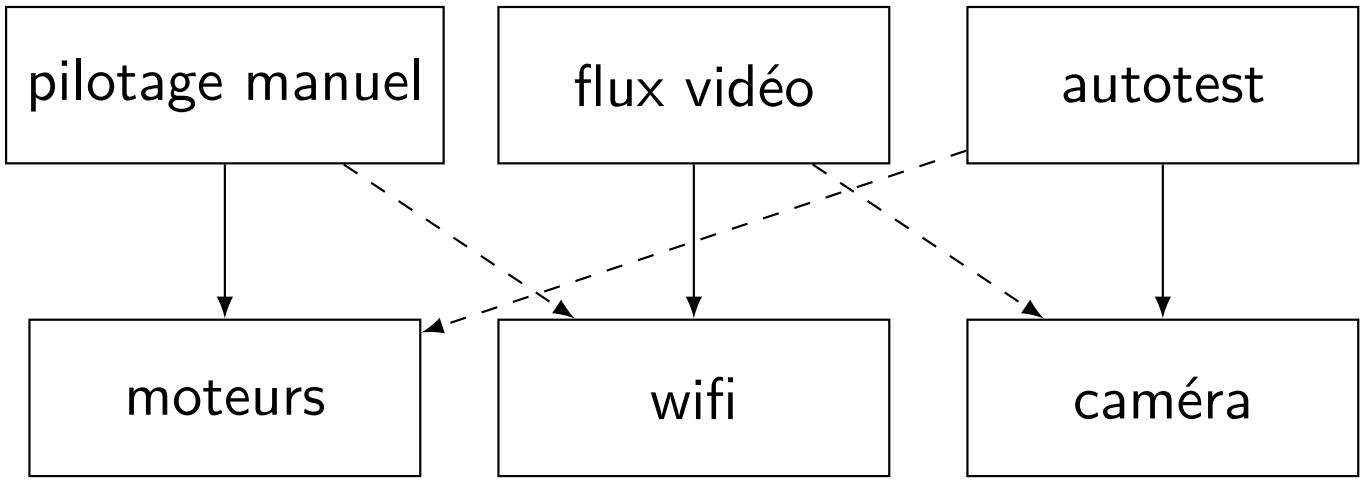
- le **pilotage manuel** qui reçoit les ordres par le wifi et actionne les moteurs,
- envoie le **flux vidéo** via la liaison wifi,
- fait un **autotest** matériel, hors liaison wifi.

pilotage manuel	**flux vidéo**	**autotest**
Demande moteurs	Demande wifi	Demande caméra
Demande wifi	Demande caméra	Demande moteurs
Libère moteurs	Libère wifi	Libère caméra
Libère wifi	Libère caméra	Libère moteurs

Tableau 1 – Détails des tâches

Imaginons le scénario :

- Le pilotage manuel demande les moteurs et les obtient.
- Le flux vidéo demande le wifi et l'obtient.
- L'autotest demande la caméra et l'obtient.
- Le pilotage manuel demande le wifi mais doit attendre que le flux vidéo le libère.
- Le flux vidéo demande la caméra mais doit attendre que l'autotest la libère.
- L'autotest demande les moteurs mais doit attendre que le pilotage manuel les libère.



Activité 7 :

1. Ouvrir le fichier **`interblocage.py`**.
2. Observer le cas d'interblocage. Appuyer deux fois sur **`ctrl+C`** pour tuer les processus.

Sommaire

Conditions d'interblocage

L'interblocage est le grand danger de la programmation concurrente. Il existe quatre conditions nécessaires à la présence d'un interblocage, décrites par Edward Grady Coffman en 1971 :

- **Exclusion mutuelle :** au moins une ressource doit être en accès exclusif.

Conditions d'interblocage

L'interblocage est le grand danger de la programmation concurrente. Il existe quatre conditions nécessaires à la présence d'un interblocage, décrites par Edward Grady Coffman en 1971 :

- **Exclusion mutuelle :** au moins une ressource doit être en accès exclusif.
- **Rétention et attente :** un processus détient une ressource et demande une autre ressource détenue par un autre processus.

Conditions d'interblocage

L'interblocage est le grand danger de la programmation concurrente. Il existe quatre conditions nécessaires à la présence d'un interblocage, décrites par Edward Grady Coffman en 1971 :

- **Exclusion mutuelle :** au moins une ressource doit être en accès exclusif.
- **Rétention et attente :** un processus détient une ressource et demande une autre ressource détenue par un autre processus.
- **Non préemption :** une ressource détenue par un processus ne peut être récupérée de force (*préemptée*) par un autre processus.

Conditions d'interblocage

L'interblocage est le grand danger de la programmation concurrente. Il existe quatre conditions nécessaires à la présence d'un interblocage, décrites par Edward Grady Coffman en 1971 :

- **Exclusion mutuelle :** au moins une ressource doit être en accès exclusif.
- **Rétention et attente :** un processus détient une ressource et demande une autre ressource détenue par un autre processus.
- **Non préemption :** une ressource détenue par un processus ne peut être récupérée de force (*préemptée*) par un autre processus.
- **Attente circulaire :** chaque processus attend une ressource détenue par un des autres.

Sommaire

Traitement de l'interblocage

Sans rentrer dans le détail, nous pouvons citer les politiques mises en place pour traiter l'interblocage :

- **Politique de guérison :** le système maintient un état permanent des demandes de ressources et résout les éventuels interblocages en tuant un processus.

Traitement de l'interblocage

Sans rentrer dans le détail, nous pouvons citer les politiques mises en place pour traiter l'interblocage :

- **Politique de guérison :** le système maintient un état permanent des demandes de ressources et résout les éventuels interblocages en tuant un processus.
- **Politique de prévention :** le système fait en sorte que les quatre conditions ne soient jamais réunis. Par exemple, ne jamais donner une ressource si elle est déjà utilisée par un autre processus.

Traitement de l'interblocage

Sans rentrer dans le détail, nous pouvons citer les politiques mises en place pour traiter l'interblocage :

- **Politique de guérison :** le système maintient un état permanent des demandes de ressources et résout les éventuels interblocages en tuant un processus.
- **Politique de prévention :** le système fait en sorte que les quatre conditions ne soient jamais réunis. Par exemple, ne jamais donner une ressource si elle est déjà utilisée par un autre processus.
- **Politique de l'évitement :** à chaque demande de ressource, le système vérifie si cela peut causer un interblocage. Si tel est le cas, l'allocation est retardée.

Traitement de l'interblocage

Sans rentrer dans le détail, nous pouvons citer les politiques mises en place pour traiter l'interblocage :

- **Politique de guérison :** le système maintient un état permanent des demandes de ressources et résout les éventuels interblocages en tuant un processus.
- **Politique de prévention :** le système fait en sorte que les quatre conditions ne soient jamais réunis. Par exemple, ne jamais donner une ressource si elle est déjà utilisée par un autre processus.
- **Politique de l'évitement :** à chaque demande de ressource, le système vérifie si cela peut causer un interblocage. Si tel est le cas, l'allocation est retardée.
- **Politique de l'autruche :** on ne s'en occupe pas, et on se contente de redémarrer la machine quand trop de processus sont en interblocage. Les trois autres politiques étant très coûteuses cette solution n'est finalement pas si farfelue.

Bibliographie

- http://y.legouzouguec.free.fr
- http://lycee.educinfo.org/index.php?page=introduction&activite=processus