

Ordonnancement - implémentation

Christophe Viroulaud

Terminale - NSI

Archi 05

Le processeur peut adopter plusieurs stratégies pour exécuter l'enchaînement des processus. Selon l'algorithme utilisé la structure adoptée pour stocker la liste des tâches a une importance fondamentale.

Quelles structures de données adopter pour implémenter
les algorithmes d'ordonnancement ?

Des structures
héritées de la liste
chaînée

Pile

File

Ordonnancement

1. Des structures héritées de la liste chaînée

1.1 Pile

1.2 File

2. Ordonnancement

Des structures
héritées de la liste
chaînée

Pile

File

Ordonnement

À retenir

Les piles (*stack*) sont fondées sur le principe du *dernier arrivé premier sorti* : **Last In First Out**.

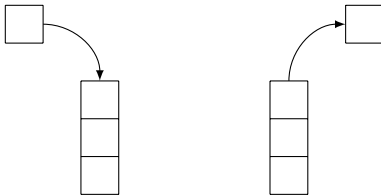


FIGURE 1 – Empiler - dépiler

Une pile stocke des éléments de type **T** quelconque.

- ▶ **creer_pile()** → **Pile()** : crée une pile vide
- ▶ **est_vide(p: Pile)** → **bool** : renvoie **True** si la pile est vide, **False** sinon.
- ▶ **empiler(p: Pile, e: T)** → **None** : ajoute un élément **e** au sommet de la pile.
- ▶ **depiler(p: Pile)** → **T** : retire et renvoie l'élément du sommet de la pile.

Implémentation

- ▶ `creer_pile()` → `Pile()`
- ▶ `est_vide(p: Pile)` → `bool`
- ▶ `empiler(p: Pile, e: T)` → `None`
- ▶ `depiler(p: Pile)` → `T`

Des structures
héritées de la liste
chaînée

Pile

File

Ordonnancement

Activité 1 : La programmation orientée objet est un paradigme adapté pour implémenter une pile.

1. Créer une classe **Element**. Son constructeur initialisera deux attributs :
 - ▶ `donnees: int`
 - ▶ `successeur: Element`
2. Adapter l'interface présentée pour créer une classe **Pile**.
3. **Pour les plus avancés :** Implémenter la méthode `__str__` qui affiche le contenu de la pile.
4. Quelle fonctionnalité du navigateur web utilise une pile ?

Des structures
héritées de la liste
chaînée

Pile

File

Ordonnancement

```
1 class Element:
2     def __init__(self, d: int, s: object):
3         self.donnees = d
4         self.successeur = s
```

```
1 class Pile:
2     def __init__(self):
3         self.sommet = None
4
5     def est_vide(self) -> bool:
6         return self.sommet is None
```

```
1 def empiler(self, e: int) -> None:  
2     self.sommet = Element(e, self.sommet)
```

```
1 def depiler(self) -> int:
2     # gestion d'erreur
3     if not self.est_vide():
4         # récupère la valeur du haut de la pile
5         res = self.sommet.donnees
6         # retire le sommet
7         self.sommet = self.sommet.successeur
8         return res
```

```
1 def __str__(self):
2     affiche = ""
3     last = self.sommet
4     while last is not None:
5         affiche += str(last.donnees) + "\n"
6         last = last.successeur
7     return affiche
```

La fonction **retour** du navigateur web est un exemple de pile.
La fonction **annuler** du traitement de texte également.

1. Des structures héritées de la liste chaînée

1.1 Pile

1.2 File

2. Ordonnancement

Des structures
héritées de la liste
chaînée

Pile

File

Ordonnancement

À retenir

Les files (*queue*) sont fondées sur le principe du *premier arrivé premier sorti* : **F**irst **I**n **F**irst **O**ut.

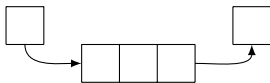


FIGURE 2 – Enfiler - défiler

- ▶ **creer_file()** → **File()** : crée une file vide.
- ▶ **est_vide()** → **bool** : renvoie **True** si la file est vide, **False** sinon.
- ▶ **enfiler(e: T)** → **None** : ajoute un élément *e* à l'arrière de la file.
- ▶ **defiler()** → **T** : retire et renvoie l'élément de l'avant de la file.

- ▶ `creer_file()` → `File()`
- ▶ `est_vide()` → `bool`
- ▶ `enfiler(e: T)` → `None`
- ▶ `defiler()` → `T`

Des structures
héritées de la liste
chaînée

Pile

File

Ordonnancement

Activité 2 :

1. Adapter l'interface présentée pour créer une classe `File`. Il est nécessaire de maintenir deux attributs : **`premier`** et **`dernier`**. Il faudra également réutiliser la classe **`Element`**.
2. **Pour les plus avancés** : Implémenter la méthode **`__str__`** qui affiche le contenu de la file.

```
1 class File():
2     def __init__(self):
3         self.premier = None
4         self.dernier = None
5
6     def est_vide(self) -> bool:
7         return self.premier == None
```

```
1 def enfiler(self, e: int) -> None:
2     nouveau = Element(e, None)
3
4     if self.est_vide():
5         # 1 seul élément: le premier est le
dernier
6         self.premier = nouveau
7     else:
8         # le dernier devient avant-dernier
9         self.dernier.successeur = nouveau
10
11     # le nouveau devient dernier
12     self.dernier = nouveau
```

```
1 def defiler(self) -> int:
2     if not self.est_vide():
3         res = self.premier.donnees
4         self.premier = self.premier.successeur
5         return res
```

Des structures
héritées de la liste
chaînée

Pile

File

Ordonnancement

```
1 def __str__(self):
2     c = self.premier
3     s = ""
4     while not c is None:
5         s = s + str(c.donnees)+"|"
6         c = c.successeur
7     return "\u2BA4|" + s[:] + "\u2BA0"
```

```
1  from random import randint
2
3  a = File()
4  for i in range(6):
5      a.enfiler(randint(1, 20))
6      print(a)
7
8  for i in range(6):
9      a.defiler()
10     print(a)
```

Code 1 – Affichage de la file

Des structures
héritées de la liste
chaînée

Pile

File

Ordonnancement

1. Des structures héritées de la liste chaînée

2. Ordonnancement

À retenir

Plusieurs algorithmes d'ordonnancement utilisent une file.

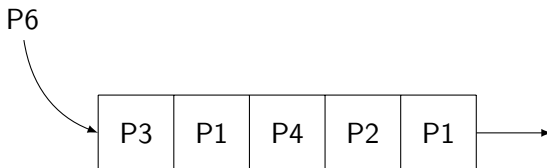


FIGURE 3 – First Come First Served

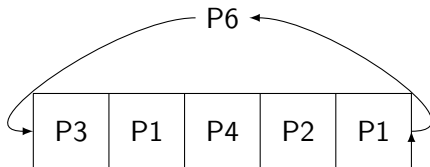


FIGURE 4 – Round Robin

Une *quantum* de temps est alloué à chaque processus. Un processus qui n'est pas terminé retourne en fin de file.