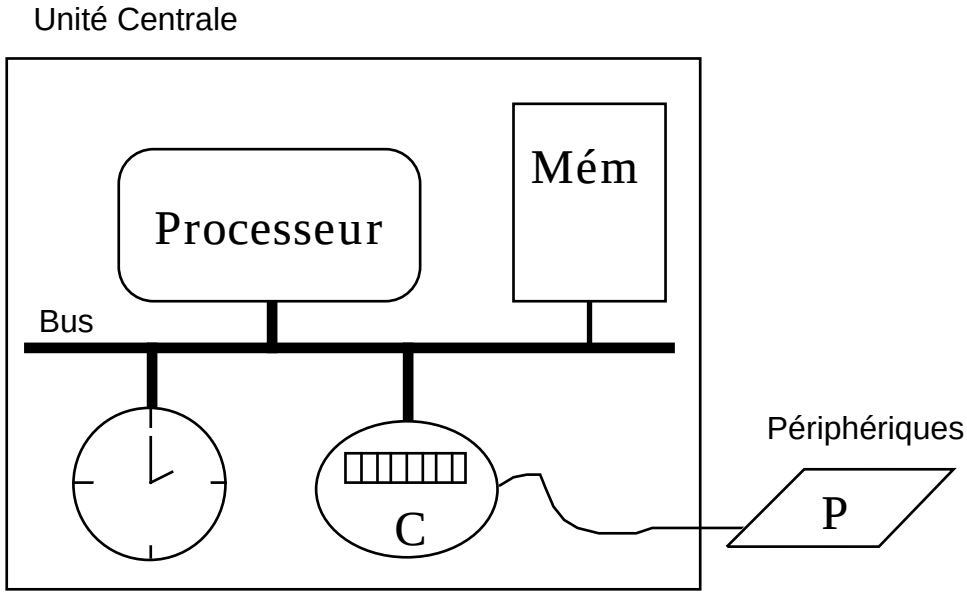
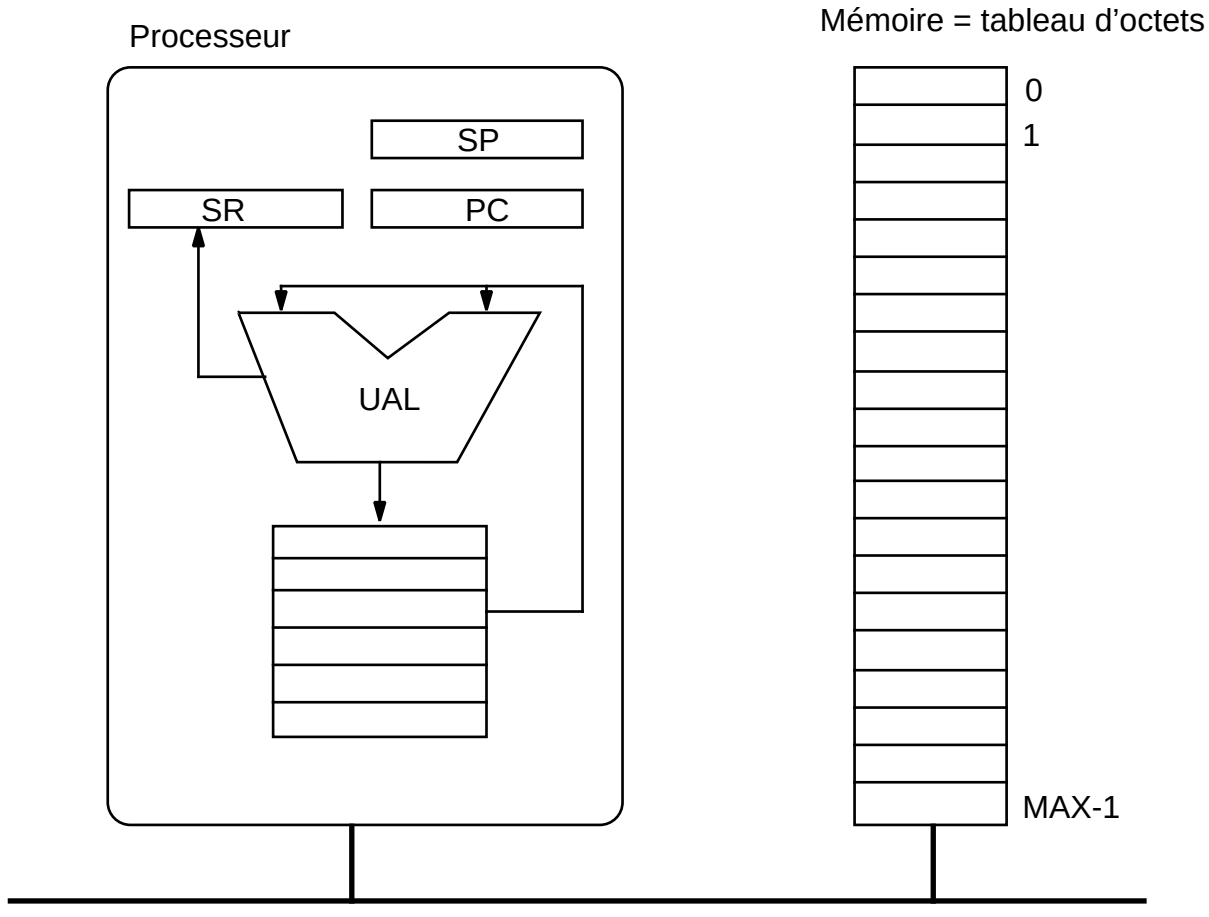
Modèle d'architecture

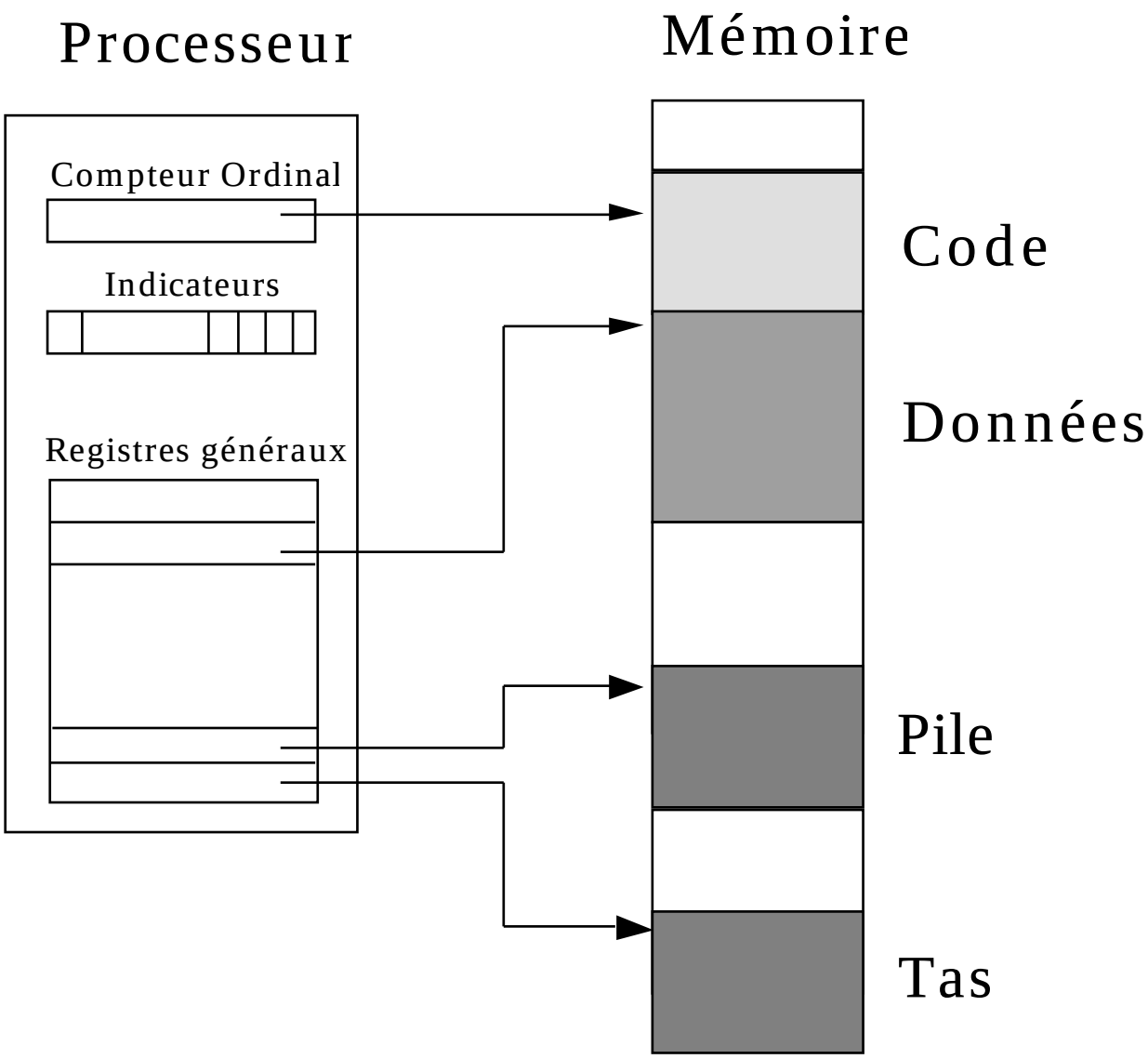
Modèle de Von Neumann



Mémoire et processeur

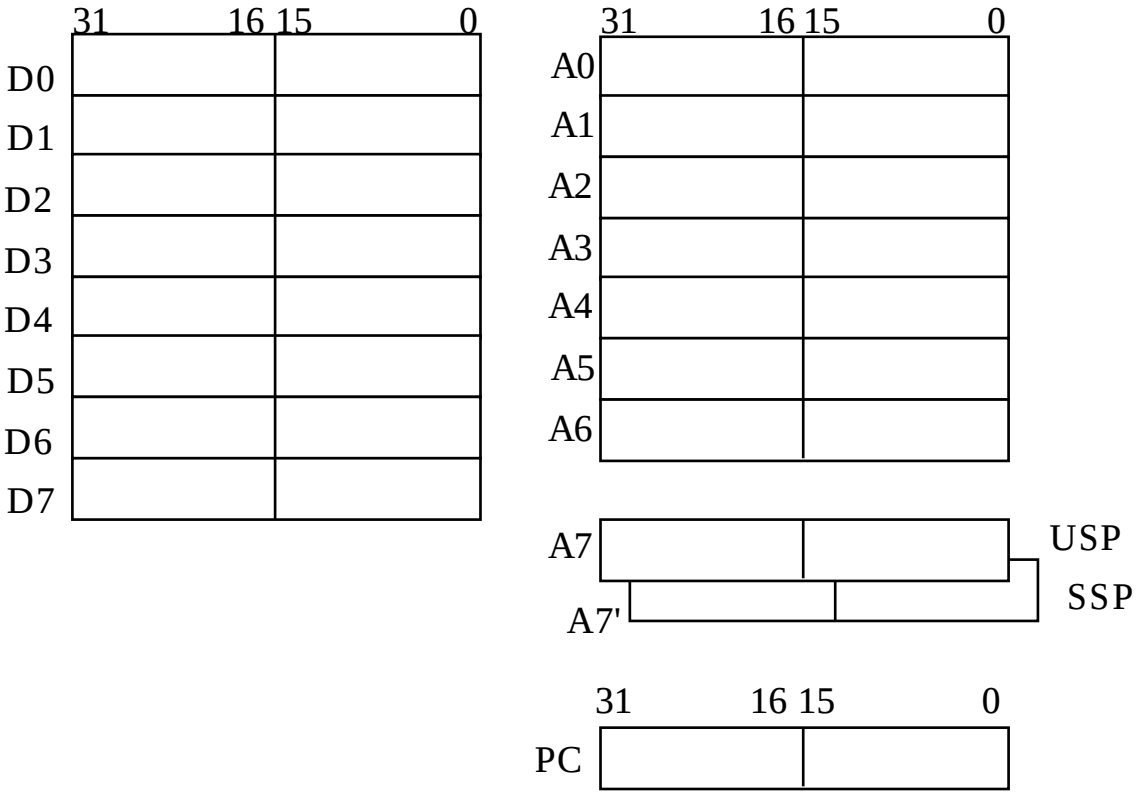


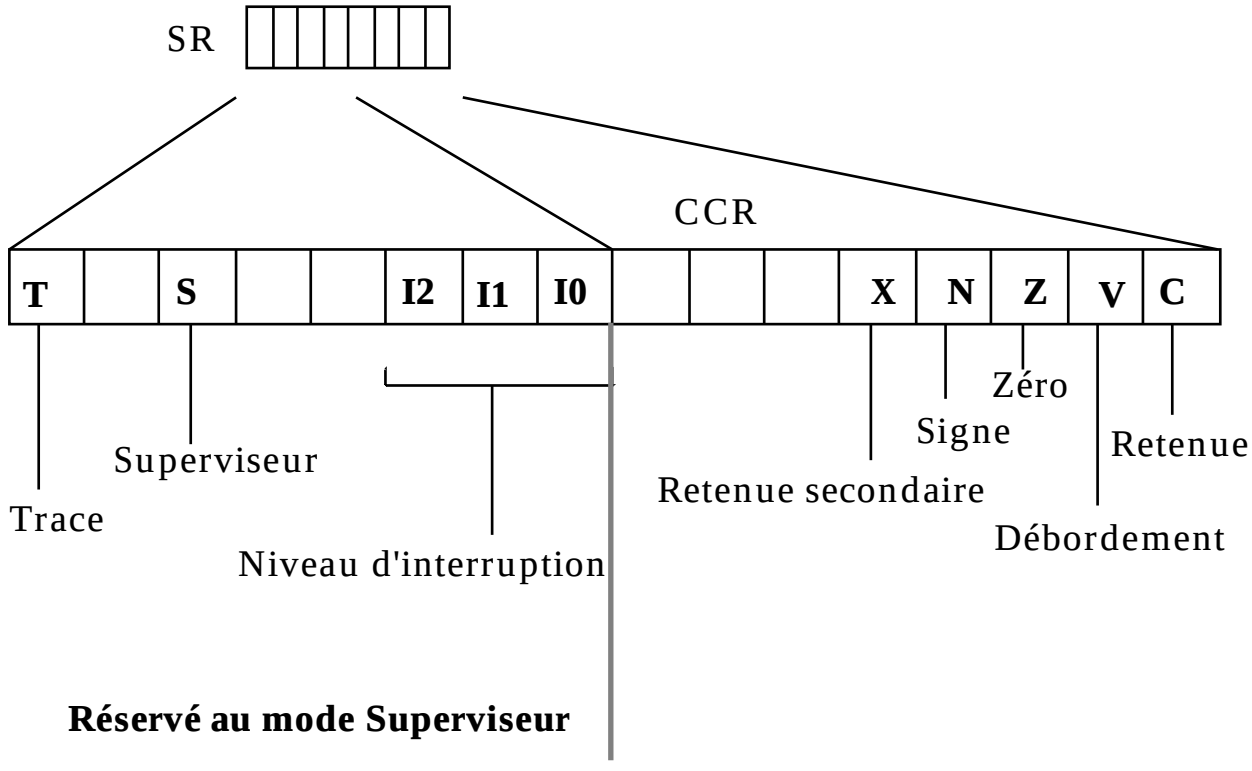
Modèle d'exécution



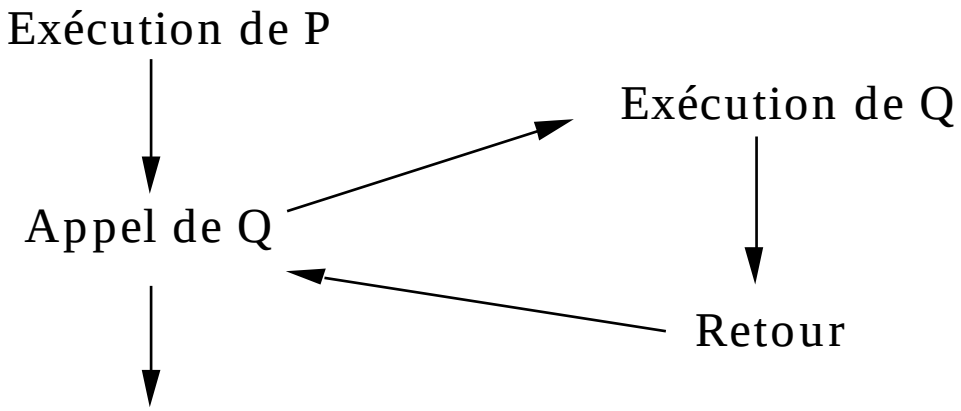
Compteur Ordinal + Indicateur
=
Mot d'état du processeur

Modèle de programmation du 68000





Appel et retour de procédure



Séquence d'appel

Préparation des paramètres transmis à Q

Sauvegarde du contexte de P

Remplacement du contexte de P par le contexte de Q

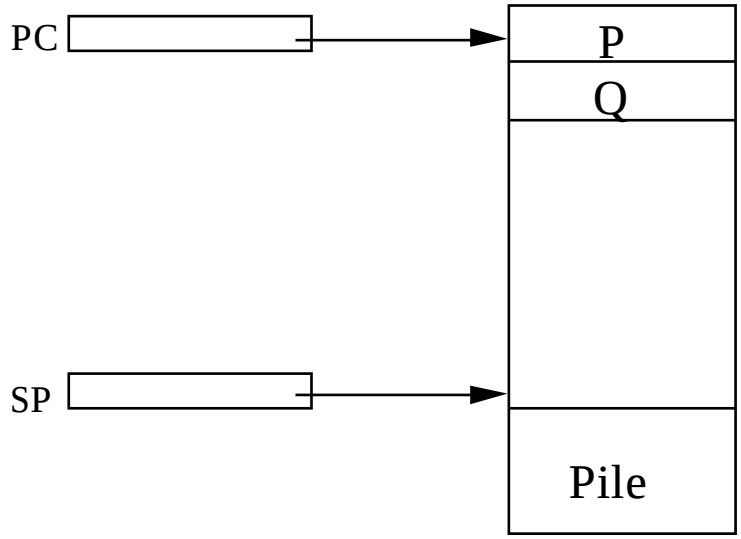
Retour

Préparation des résultats transmis par Q

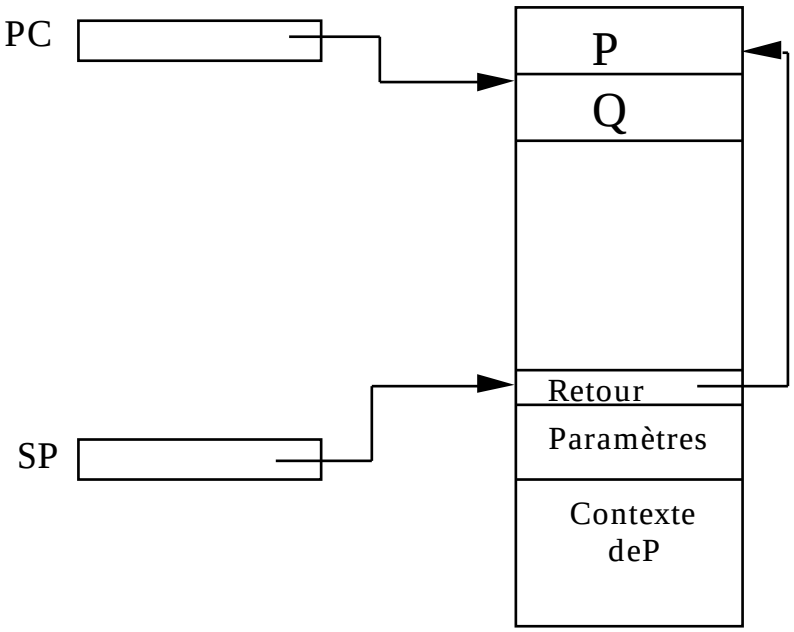
Restauration du contexte de P avant l'appel

Réalisation avec une pile (1)

État avant l'appel

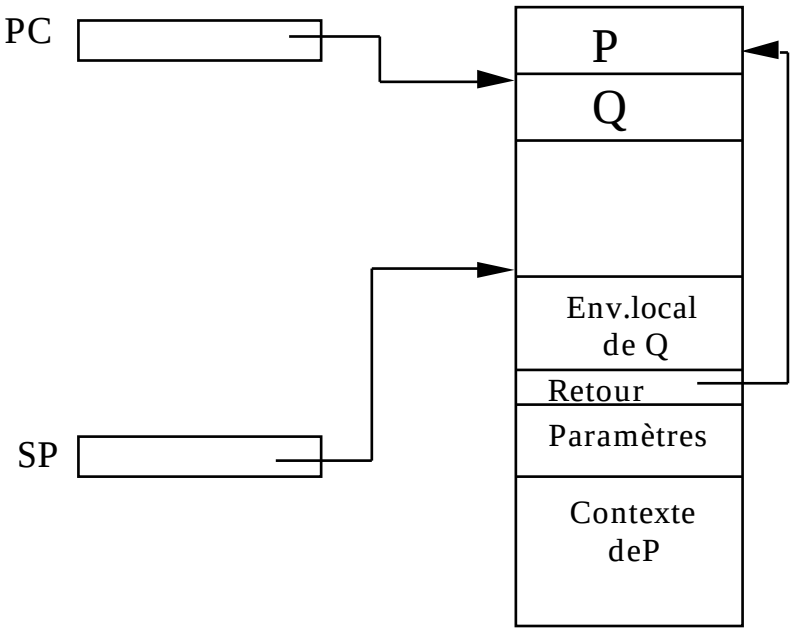


P : Préparation des paramètres et commutation

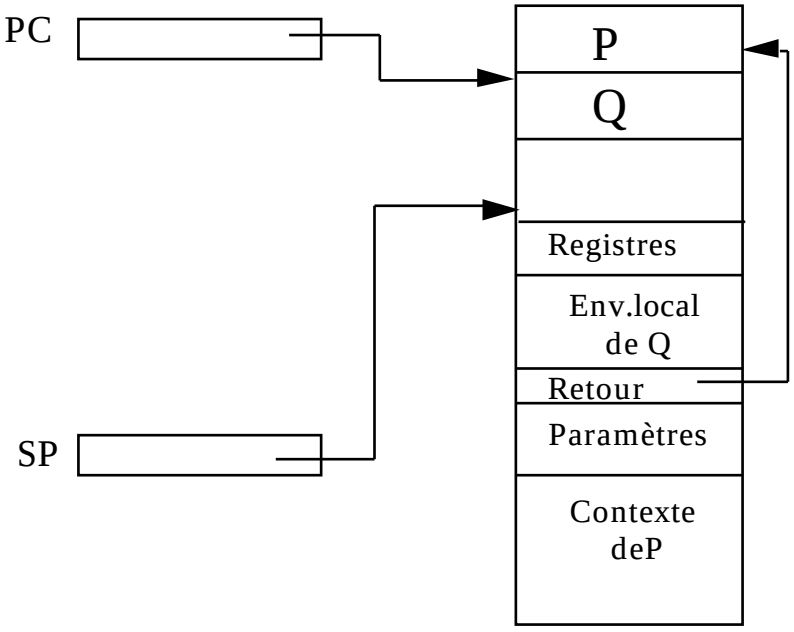


Réalisation avec une pile (2)

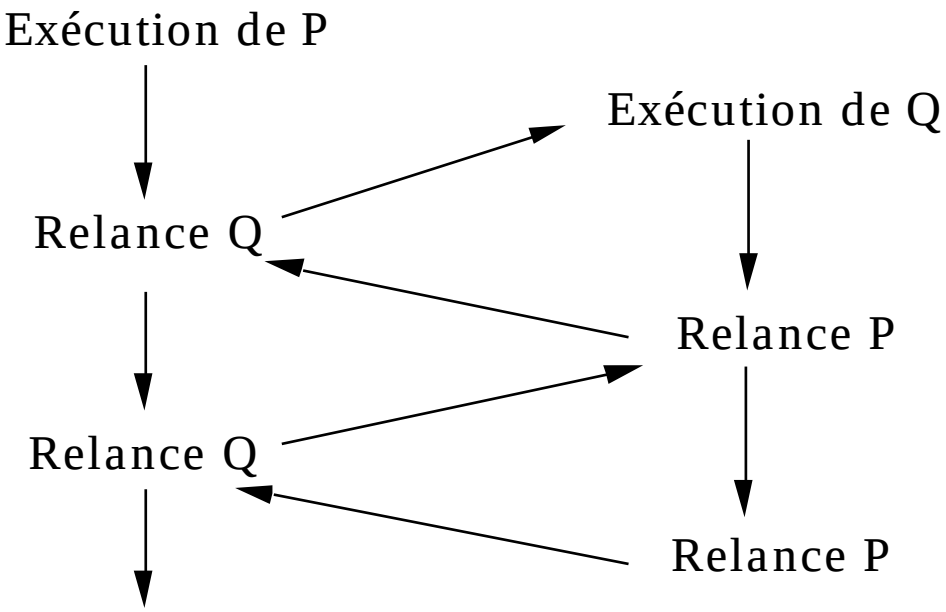
Q : préparation de l'environnement local



Q : sauvegarde (partielle) du contexte



Fonctionnement en co-routines



Relance (Resume) = Séquence d'appel

Préparation des paramètres transmis

Sauvegarde du contexte courant

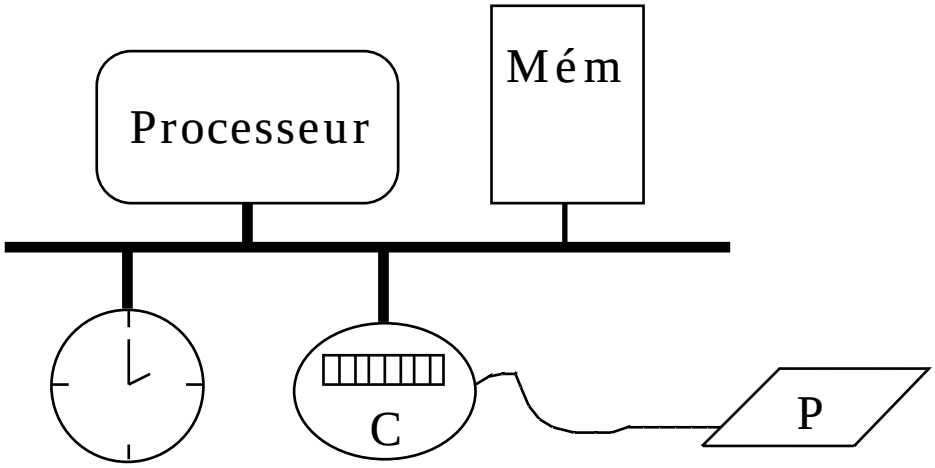
Remplacement de l'ancien contexte par le nouveau

Réalisations

Avec une pile ?

Avec contexte global

Activités asynchrones



Entrées/Sorties par canal, Horloge,

Intervention externe, Traitement d'erreur

→ **Nécessité de pouvoir interrompre le processeur**

Mécanismes d'exception

Mécanisme	**Cause**	**Utilisation**
Interruption	Extérieure à l'instruction en cours	Réaction à un évenement externe (E/S, horloge, sécurité)
Déroutement	Déclenché par l'instruction en cours	Traitement d'erreurs (débordement, division par 0, instruction illégale)
Appel au superviseur	Exception voulue, programmée	Appel d'une fonction du système d'exploitation (passage en mode privilégié)

Commutation de contexte

Principe

Sauvegarde du mot d'état du processeur

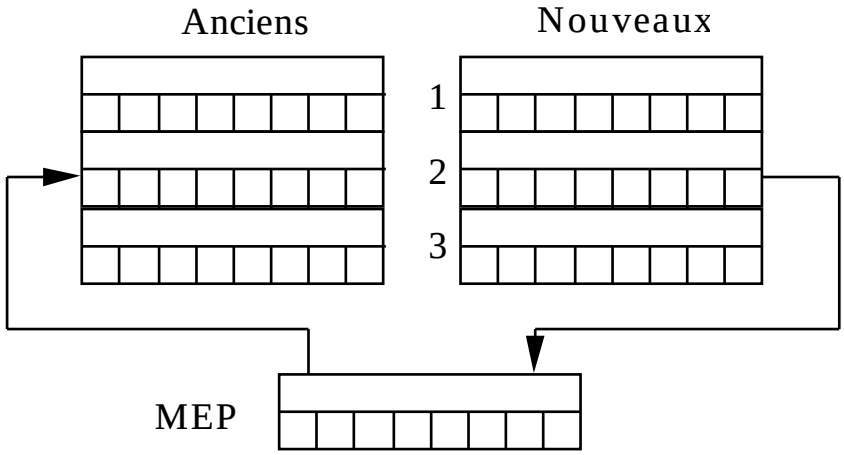
(Compteur ordinal et indicateurs)

Chargement du mot d'état à partir

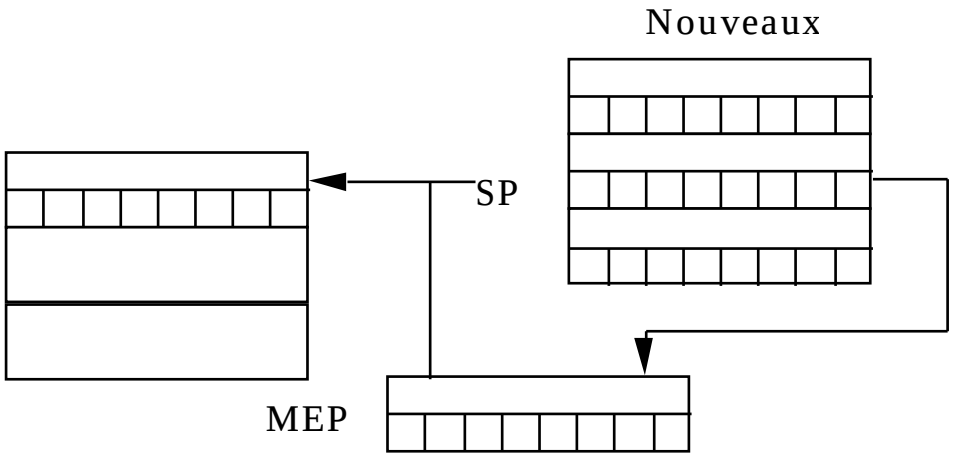
d'un emplacement mémoire spécifié

2 méthodes pour la sauvegarde

Emplacements fixes



Pile

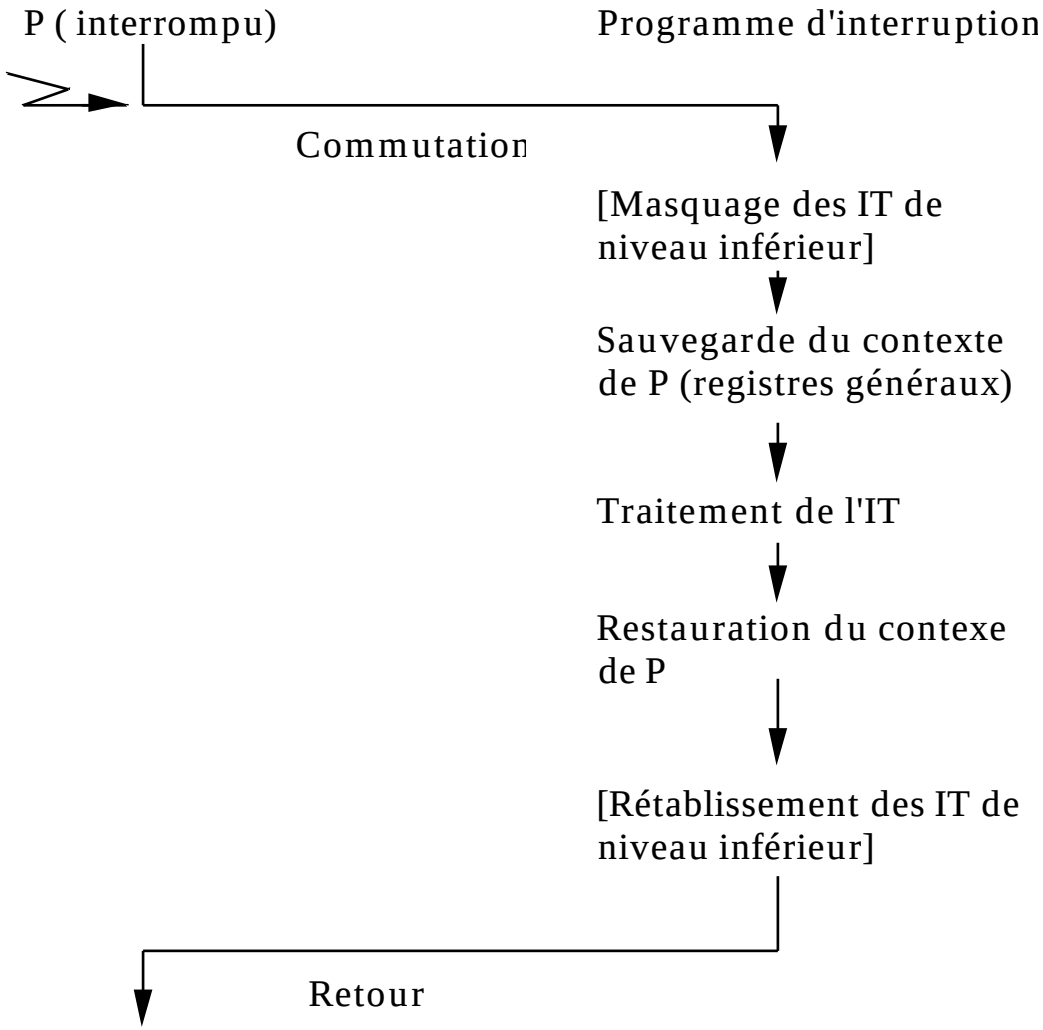


Interruptions

En général plusieurs niveaux (priorités)

Masquables (armement, désarmement)

Schéma d'un programme d'interruption



Déroutements et appels au superviseur

Déroutements

Données incorrectes (débordement, division par 0)

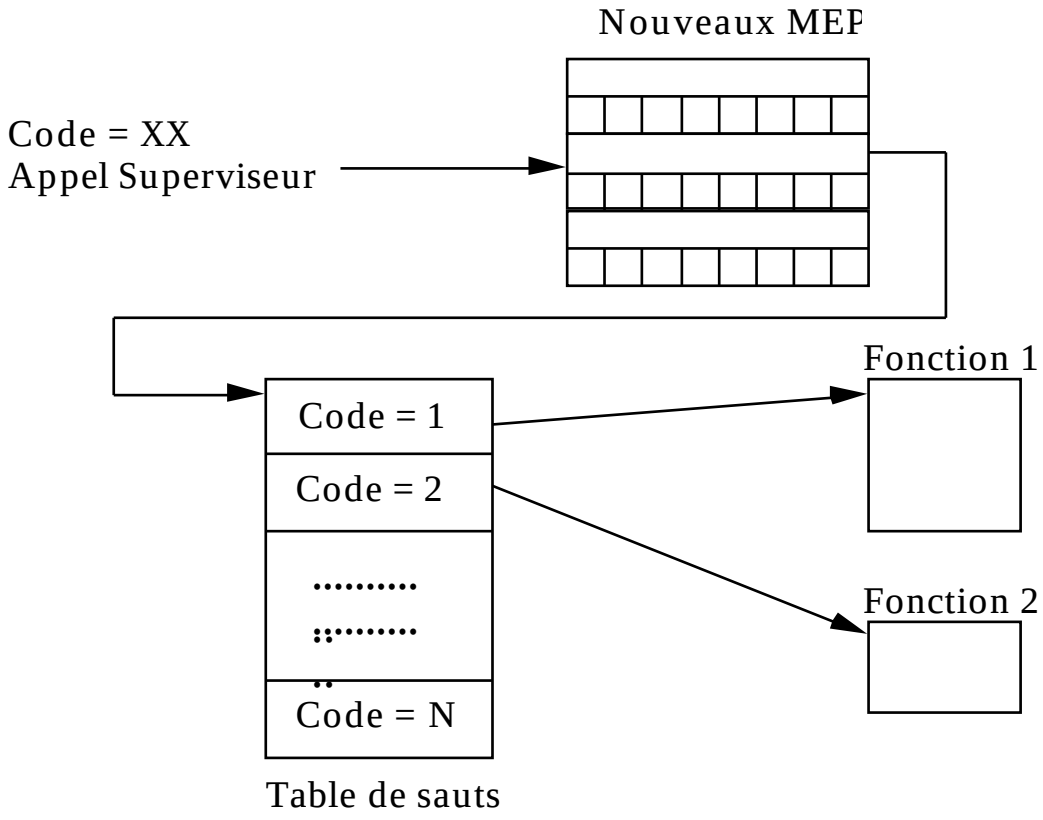
Violation de protection (de la mémoire, du mode privilégié)

Instruction non-exécutable (code inconnu, erreur d'adresse,...)

Appels au superviseur

Comme un appel de procédure mais avec des droits étendus (mode privilégié, IT masquées, droits d'accès)

Appel aux fonctions du système



Gestion des exceptions par l'utilisateur

1. Préparation

Écriture du programme de traitement d'exception :

Sauvegarde contexte
Traitement
Restauration contexte

Déterminer où est rangée l'adresse du programme dans la mémoire du système (indice I dans la table des nouveaux MEP)

2. Installation du programme

Sauvegarde de l'ancienne valeur (adresse de l'ancien programme)

Écriture de l'adresse du nouveau programme dans l'emplacement prévu (nouveaux MEP [I])

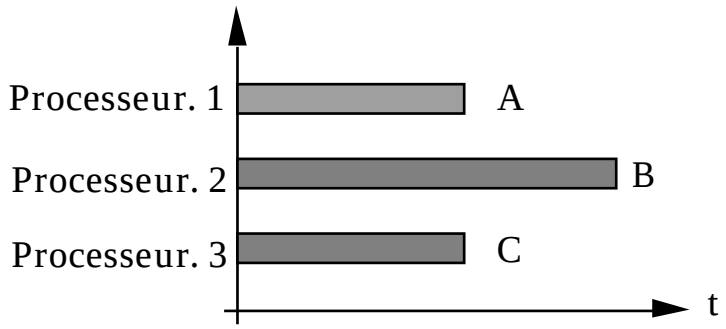
3. Utilisation

4. Restauration de l'ancienne valeur

Réécriture de l'adresse de l'ancien programme dans l'emplacement prévu (nouveaux MEP [I])

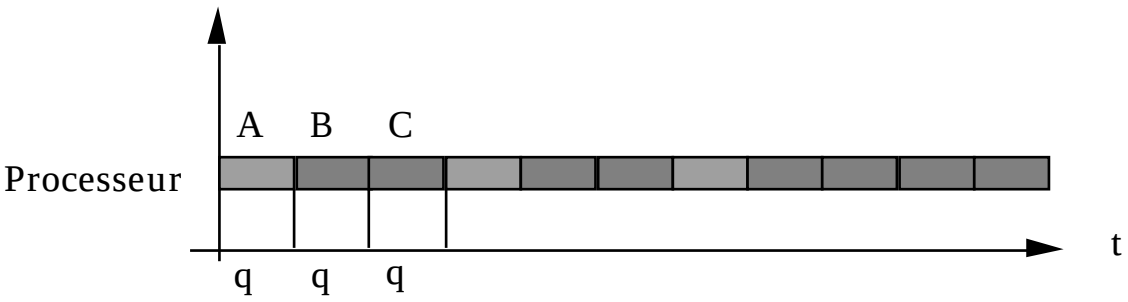
Multi-tâches

Système Multiprocesseur



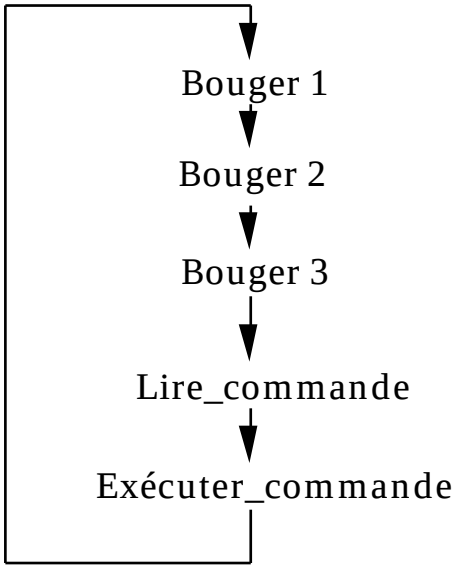
Système Monoprocesseur, Monotâche

Système Monoprocesseur, Multitâches



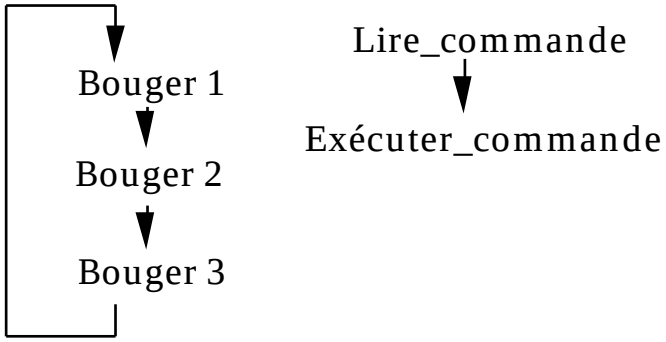
Exemples : jeux vidéos

Monotâche

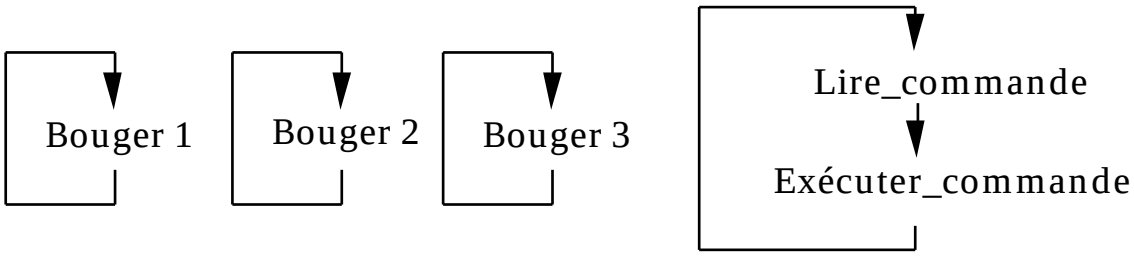


Monotâche avec interruptions

tâche principale/interruptions



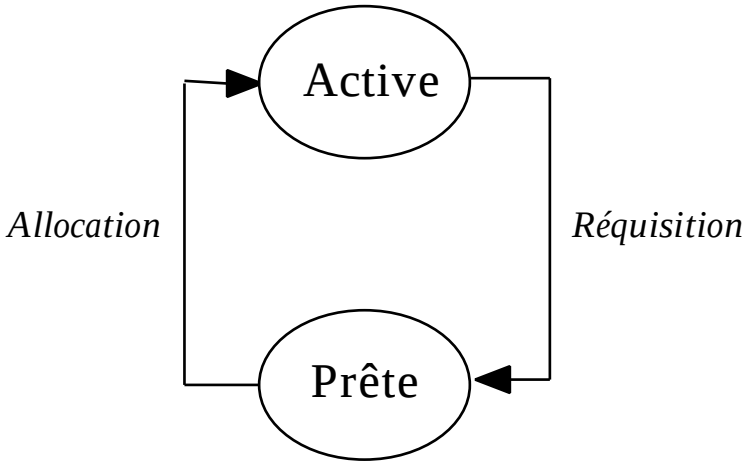
Multi-tâches



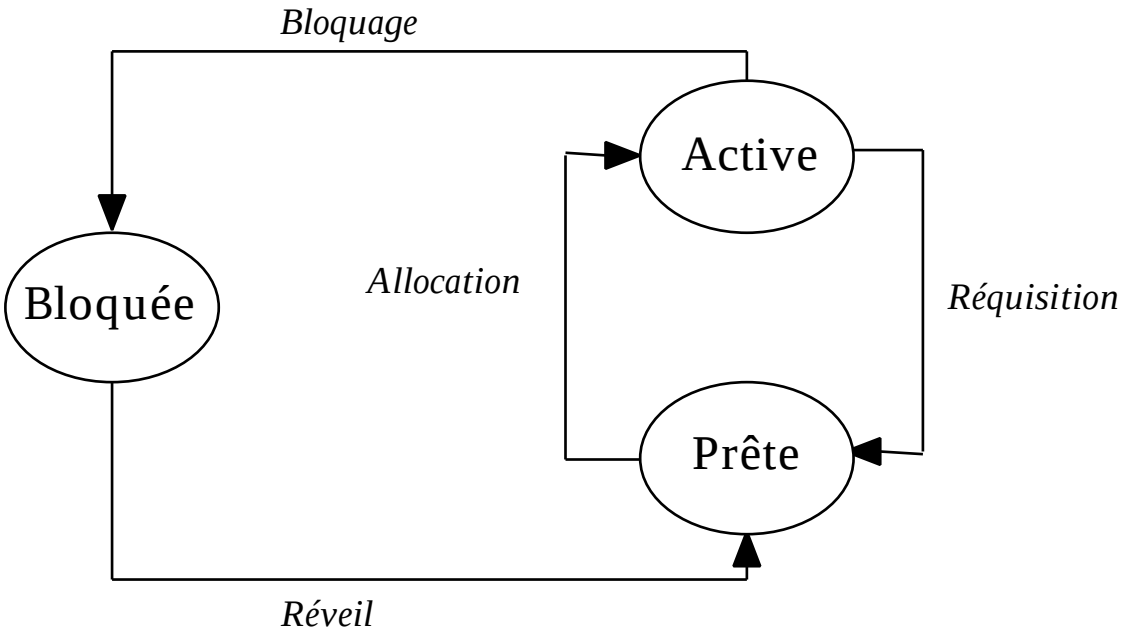
→ **Souplesse**
Partage du code

États d'une tâche

Contexte mono-processeur : une tâche active à la fois



Mais une tâche peut être bloquée en attente d'une E/S



Création de tâches

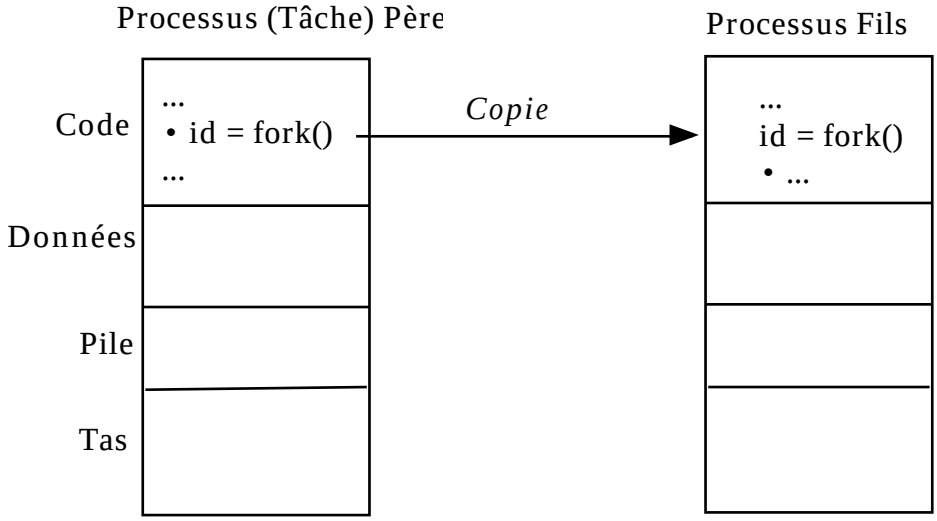
Création dynamique (Unix, OS/2, ADA, ...)

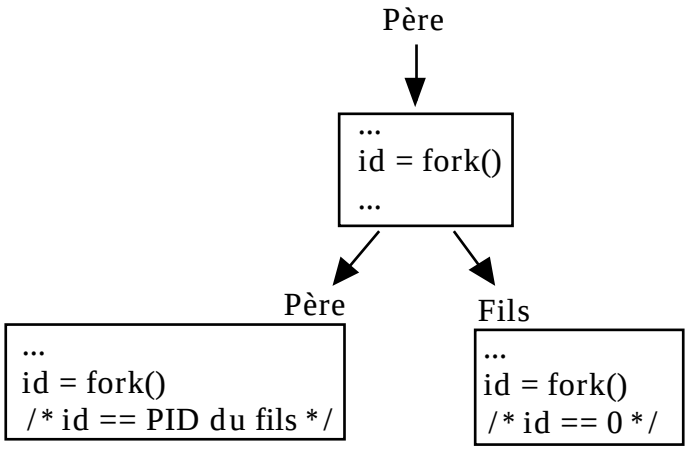
Une tâche est toujours créée par une autre tâche

- **Par copie**
- **Par chargement depuis un support**

Problème de l'initialisation du système

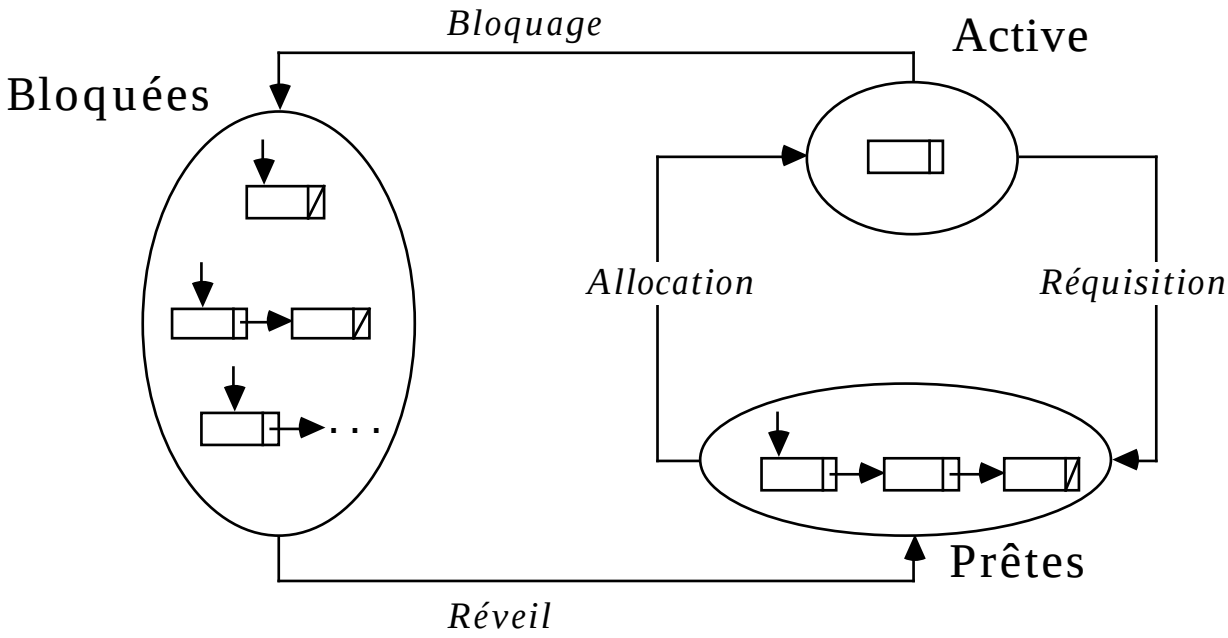
Exemple d'Unix





Gestion du processeur

États d'une tâche, Files d'attente



Ordonnanceur (*scheduler)*

Ensemble des algorithmes utilisés pour faire les transitions entre tâches.

Distributeur (*dispatcher)*

Plus particulièrement chargé de l'allocation.

Politique (ou stratégie) d'ordonnancement

Préemptif ou non préemptif

Lié à la gestion des files d'attente

Stratégies d'ordonnancement (1)

Temps partagé	Temps réel
Rendre le système agréable à utiliser : • **tâches interactives** • **algorithmes complexes, avec vieillisement des priorités, régulation de la charge**	**Le système doit être efficace et sûr :** • **hiérarchie de tâches (priorités)** • **souplesse (adaptation à de nombreuses applications, même très contraintes)**

Non préemptif	Préemptif
Exécution de la tâche courante jusqu'à appel du noyau ou interruption externe	**La tâche est de toute façon interrompue en fin de quantum**
Le noyau décide ou non de commuter la tâche active	**Commutation en fonction des priorités**
↗ Facile à réaliser	**↘ Plus difficile à réaliser**
↘ Peu fiable	**↗ Plus fiable**
	↗ Meilleure prise en compte des tâches prioritaires
↗ Très bon rendement	**↘ Rendement affaibli**

Temps de latence aux interruptions

Durée maximum pendant laquelle les interruptions sont masquées. Critère important des applications temps réel à cause des tâches de sécurité.

Unix : temps de commutation ± 1ms, temps de latence non précisé

Noyau temps réel : temps de commutation 40 à 250 µs, temps de latence inférieur à 100µs

Stratégies d'ordonnancement (2)

Critères de qualité de l'ordonnancement

Efficacité/Rendement : le maximum de temps doit être consaré à l'application

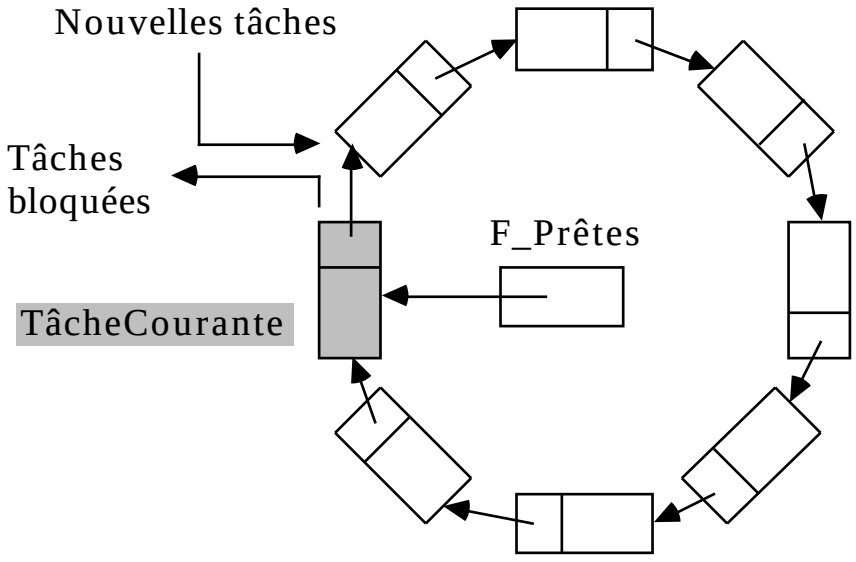
Temps de réponse : le plus faible possible (lié au temps de latence)

Impartialité : partage équitable entre tâches

Débit : le plus de tâches possibles dans un temps donné

Tourniquet : ordonnancement circulaire sans priorité

La file des tâches prêtes est circulaire et gérée en FIFO.



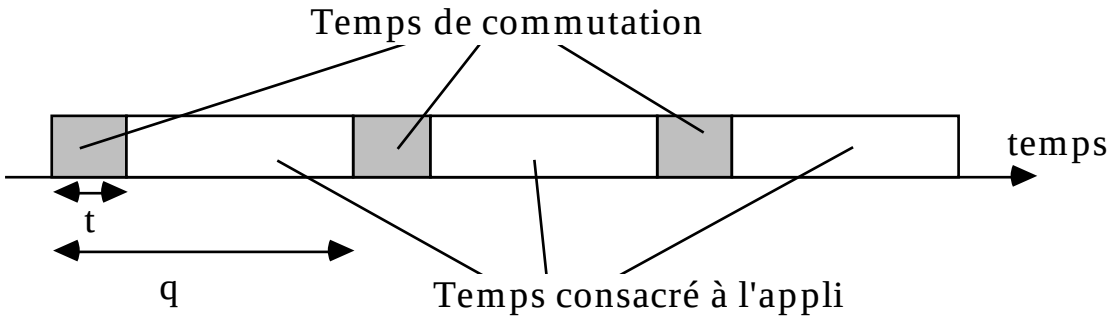
Sans réquisition : **très efficace s'il y a peu de tâches qui se bloquent souvent (E/S)**

Avec réquisition : **c'est une méthode impartiale pour les tâches de même priorité.**

Stratégies d'ordonnancement (3)

Tourniquet : choix du quantum

Compromis entre débit et efficacité



Efficacité $\mathbf{E = \frac{q - t}{q}}$ **= rapport** $\mathbf{\frac{temps\ consacré\ à\ l'appli}{temps\ total}}$

Exemple : **t = 1ms**

E = 0.8 si q=5ms

E = 0.98 si q = 50ms

Débit $\mathbf{D = \frac{1}{q}}$ **= nombres de tâches traitées par seconde**

Exemple :

D = 200 si q=5ms

D = 20 si q = 50ms

Meilleur débit = meilleur temps de réaction

Meilleure efficacité = temps total d'exécution plus court

Bien sûr il faut que t soit le plus petit possible.

Réalisation du tourniquet

Avec des structures chaînées : listes circulaires

Avec un tableau de suivants codant la liste chaînée (comme si on rajoutait un champ suivant au tableau Contextes).

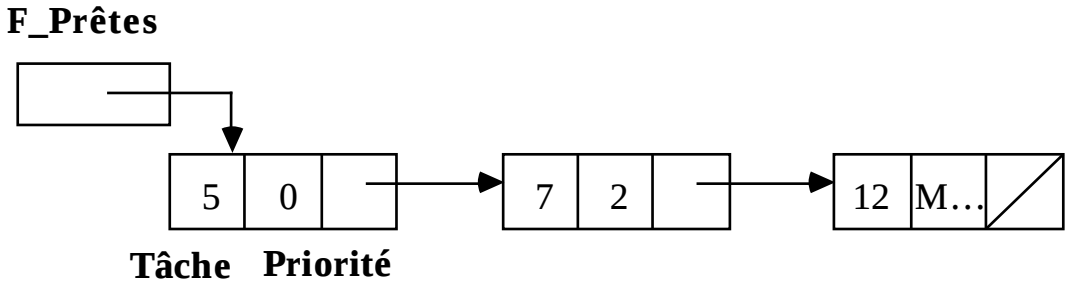
Stratégies d'ordonnancement (4)

Priorité pure

Toutes les tâches ont une priorité et pour une priorité on a au plus une tâche.

La préemption présente peu d'intérêt : la tâche la plus prioritaire s'exécute jusqu'à sa fin ou jusqu'au blocage. L'ordonnancement est donc lié au interruptions externes.

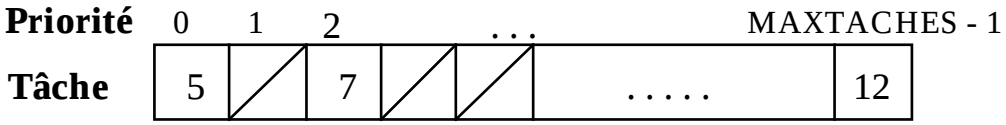
Réalisation avec liste chaînée



Accès immédiat à la première

Recherche de la position d'insertion (insertion en ordre de priorité)

Réalisation avec table



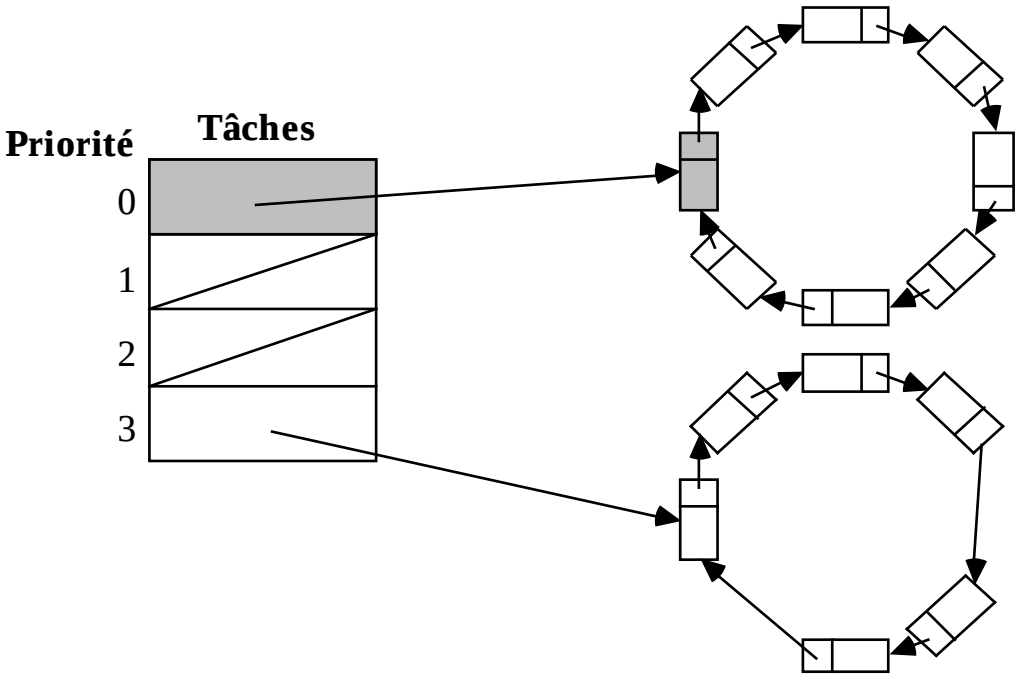
Insertion directe avec le niveau de priorité.

Accès au premier par parcours de la table à la recherche de la première case non vide.

Stratégies d'ordonnancement (5)

Méthode mixte : tourniquet multi-niveaux

On a quelques niveaux de priorité mais on peut avoir plusieurs tâches au même niveau de priorité.



Souplesse et généralité

une tâche par niveau = priorité pure

toutes les tâches au même niveau = tourniquet simple

Autres méthodes

Systèmes à temps partagé de type Unix : l'algorithme d'ordonanncement est plus complexe :

on privilégie les tâches courtes et les tâches interactives

la priorité diminue avec le temps

la durée du quantum alloué peut elle aussi diminuer