Processus / Threads
Ordonnancement

Plan

- Concept de thread
- Etats d'une thread
- Ordonnancement
- Inversion de priorité

Rappel Notion Processus

- Entité dynamique
  - Contexte d'exécution d'un programme séquentiel
  - Entité active opérant sur son environnement en exécutant un programme
- Etat courant de la machine virtuelle exécutant le programme associé au processus
  - Inclut état [d'une partie] de la machine physique
- Entité d'attribution du CPU

Concept de Thread

- Notion combinant avantages :
  - Exécution parallèle
  - Partage code et données applicatifs communs
- Facile à mettre en oeuvre
  - Programmable
  - Contrôlable
  - Configurable
- Efficace et "scalable"

Processus ≠ Thread

- Processus ne convient pas
  - Contexte lourd, coûteux à créer et à détruire
  - Pas efficace (changement de contexte lent)
  - Contexte mémoire important
- Partage de données "pas naturel"
  - Pas intégré dans langage de programmation
  - Offerts par services OS complexes, pas souples
- Programmation difficile, non contrôlable

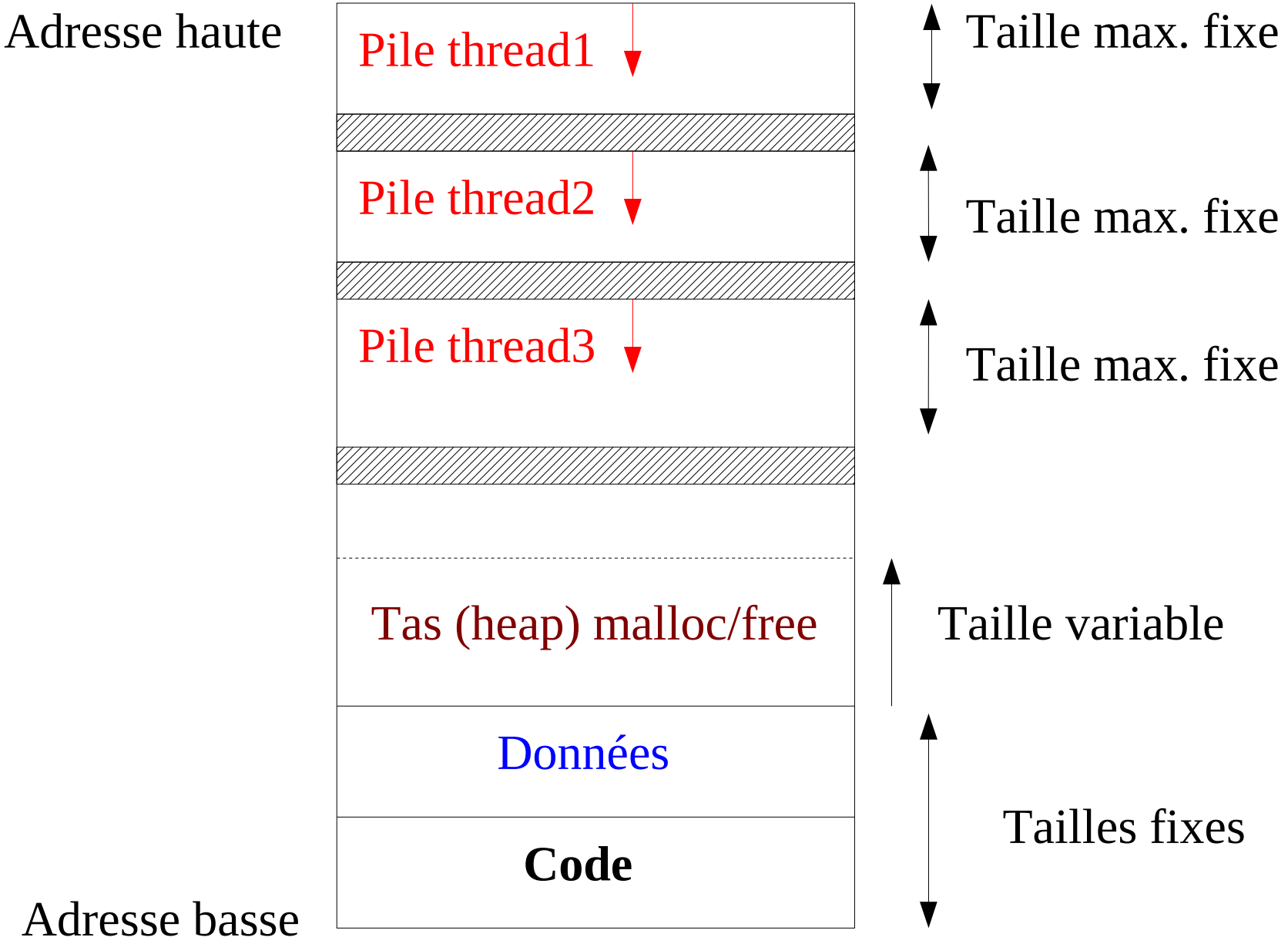
Thread – Objet Programmable

- Dérivée de notion de processus
  - Unité d'exécution concurrente dans un même programme applicatif ("Light-weight process")
  - Introduit notion de programme dit "multi-threads"
- Threads partagent espace d'adressage mémoire du processus "englobant"
  - Code, données globales, tas
  - Exécutent même code ou fonctions dédiées
- Partagent tout ou partie ressources OS
- Contexte de travail privé alloué par application

Thread – Objet OS

- Unité d'exécution indépendante
  - Pile(s) d'exécution(s)
    - Mode utilisateur du CPU (pile applicative)
    - Mode superviseur du CPU (pile système)
  - Contexte CPU (registres)
  - Etat courant et attributs d'ordonnancement
  - Identificateur
- Entité d'attribution d'un CPU

Processus Multi-Threads



Thread et OS

- Unix
  - POSIX threads – Unix, Linux
  - Light-Weight Process (LWP) - Solaris
  - Mach/Mac-OS - Apple
- Systèmes temps-réels
  - ChorusOS
  - VxWorks (WindRiver)
  - QNX, OSEK, Nucleus, etc...

Pile(s) d'exécution des threads

- Pile système
  - Pile utilisée pour exécution des appels système
  - Allouée/libérée par système
- Pile applicative
  - Pile utilisée pour exécution du code applicatif
  - Allouée/libérée par
    - noyau (cas par défaut)
    - application

Libération de la pile applicative...

- Thread toujours créée par une autre thread
  - Alloue mémoire pour pile applicative stack de la nouvelle thread
  - Appelle OS pour créer la thread
- Threads se détruisent en général elles-mêmes, mais :
  - Pas de retour de l'appel système de destruction !
  - La thread qui se suicide ne peut pas libérer sa pile applicative avant d'appeller l'OS pour être détruite
  - les 2 operations doivent aussi être exécutées par une autre thread

Light-Weight Threads

- Threads niveau applicatif
  - Implémentées par fonctions de bibliothèque
    - Run-time langage programmation
  - Pas de contexte OS
  - Exécutées dans contexte thread(s) OS
- Appels système bloquants
  - Bloque capacité d'exécution de toutes les LWT
  - Sauf si pool de threads géré par OS (Solaris)

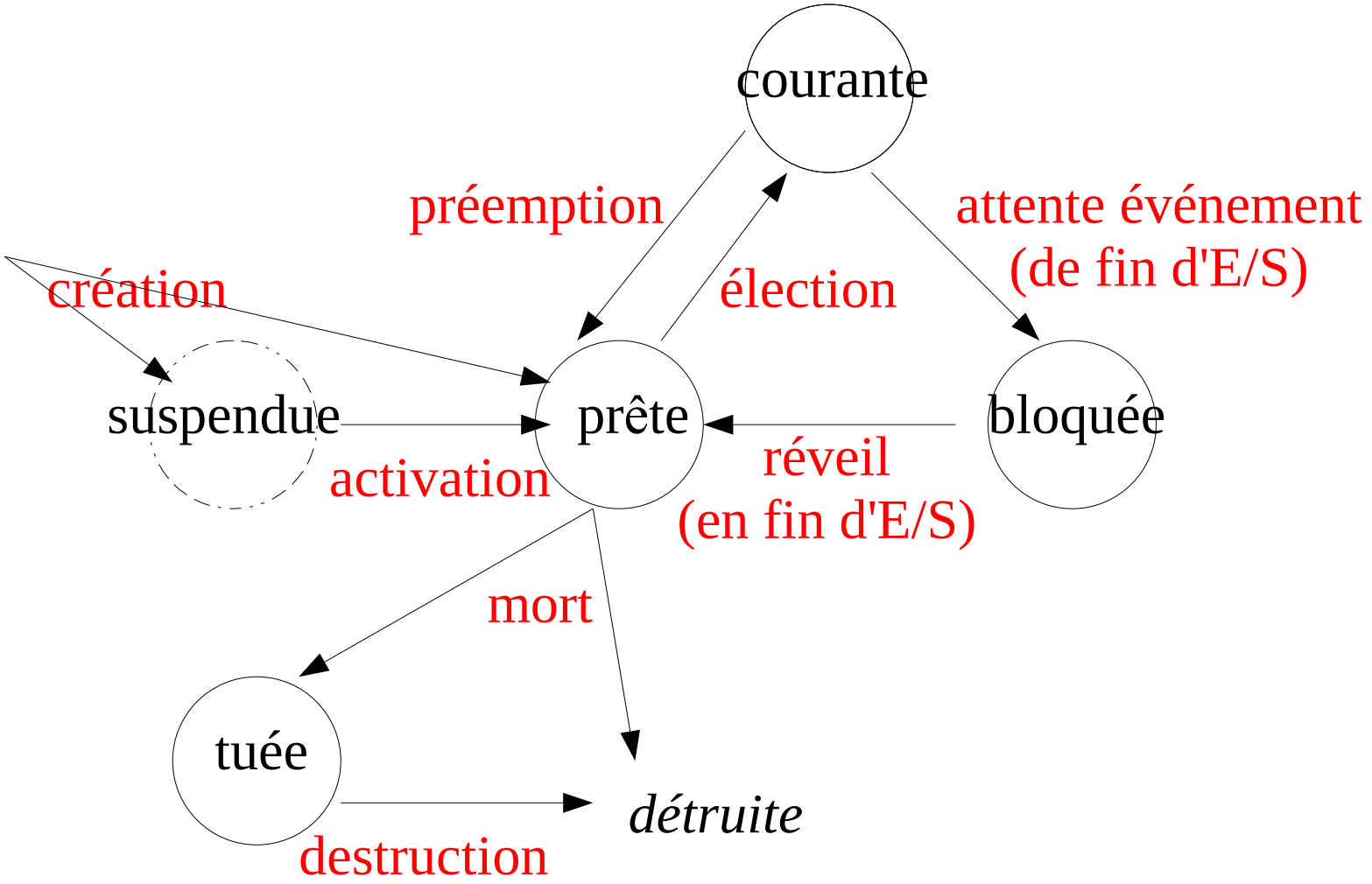
Threads et Langages de programmation

- Objet thread intégré dans langage
  - ADA : objets de type `Task`
  - Java : objets de la Classe `Thread`
  - Implique support autres notions (synchro, etc.)
- Support concepts OS dans un langage
  - +facile, +maîtrisé => +fiable (soit-disant...)
  - -souple, - clair (mélange niveaux appli et OS)
  - Comportement dépendant OS => Pbs portabilité

Plan

- Concept de thread
- Etats d'une thread
- Ordonnancement
- Inversion de priorité

Etats d'une Thread



Transitions états d'une thread (1)

- Courante -> bloquée (**mise en attente**)
  - Thread doit arrêter son exécution pour attendre l'arrivée d'un événement (fin d'E/S)
  - => libère volontairement [ressource] CPU
    => OS alloue CPU à thread prête la + prioritaire
- Bloquée -> prête (**réveil**)
  - Thread réveillée [par événement posté] par
    - une [fonction d'] interruption (fin d'E/S)
    - la thread courante (client/serveur)
  - Rejoint groupe des demandeurs ressource CPU

Transitions états d'une thread (2)

- Prête -> élue (**élection**)
  - CPU attribué à thread prête la + prioritaire
  - pour donner le CPU à thread + prioritaire
    => thread **préemptée** par OS
- Élue -> prête (**préemption**)
  - CPU retiré à la thread courante pour être attribué à thread + prioritaire
- Point(s) de préemption d'un OS
- Ordonnancement des threads

Points de Préemption du Système

- Après réveil thread + prioritaire thread courante
  - Quand/où OS préempte thread courante
- Coarse-Grain system
  - Pas de préemption dans le noyau
  - Préemption au retour d'un appel système
  - Dépend durée exécution appel système
- Fine-Grain system
  - Préemption dans le noyau
  - Immédiate

Plan

- Concept de thread
- Etats d'une thread
- Ordonnancement
- Inversion de priorité

Ordonnancement

- Choisir prochaine thread à exécuter
- Systèmes interactifs / Serveurs
  - Threads variables, événements non prévisibles
  - Partager ressource CPU dynamiquement
- Systèmes temps réel (embarqués, multi-médias)
  - Temps-réel "dur"
    - Garanti échéances de toutes les tâches
    - Implique connaisance de leurs [pires] durées d'exécution
  - Temps-réel "mou"
    - Essai de garantir les échéances

Politiques d'ordonnancement

- Critères pour choisir prochaine thread à exécuter
  - Réduire temps moyen d'attente
  - Garantir temps de réponse de threads dites + prioritaires
  - Eviter phénomènes de famine
    - Garantir un minimum [de temps CPU] à toutes
- Compromis avec contraintes
  - Optimiser taux utilisation [autres] ressources
  - Limiter fréquence changements de contexte (coût)
  - Mise en oeuvre simple, maîtrisable, adaptable...

Classes d'ordonnancement

- A l'ancienneté
  - premier arrivé, premier servi ("FIFO")
- Par priorités
  - Fixes, imposées par applications
  - Variables, modifiées par OS :
    - Durée d'attente écoulée
    - [Prix des] ressources allouées à la thread
- Par quantum de temps (durée maximale)
- Par échéances

Méthodes d'ordonnancement

- Gestion de l'ensemble des threads prêtes
- Méthode du tourniquet ("Round-Robin")
  - Une file d'attente par priorité
  - FIFO + [re]mise en attente en fin de la file
- Priorité thread diminue après utilisation quantum
  - Appliquée en mode applicatif
- Distinction entre plusieurs niveaux de priorités
  - Threads temps-réelles
  - Threads ordinaires

Posix Threads

- POSIX API pour programmes multi-threads
  - Portable entre OS
  - Restrictions concernant ressources OS partagées (fichiers ouverts, répertoire courant, etc)
- 3 classes d'ordonnancement (*scheduling*)
  - SCHED_FIFO : temps-réel
  - SCHED_RR : préemptible
  - SCHED_OTHER : ordinaire (non temps-réel)

Plan

- Concept de thread
- Etats d'une thread
- Ordonnancement
- Inversion de priorité

Inversion de Priorité

- Phénomène faisant qu'une thread de priorité inférieure est exécutée avant une autre thread plus prioritaire
- Conditions :
  - Threads partagent dynamiquement une autre ressource exclusive [que le CPU]
  - Allocation ressource exclusive découplée de celle du CPU
  - Threads de priorités fixes

Inversion de Priorité (2)

- Prio(T1) < Prio(T2) < Prio(T3)
- Séquence d'opérations
  - T1 acquiert accès exclusif à ressource R
  - T1 préemptée par T3
  - T3 demande accès exclusif à ressource R
    - T3 bloquée car R acquise par T1
    - T1 élue
  - T1 préemptée par T2
- => inversion de priorité entre T2 et T3

Héritage de Priorité

- Enregistre thread propriétaire de la ressource
- Quand thread + prioritaire tente acquérir ressource :
  - Attribue à thread propriétaire priorité thread demandeuse
  - Met en attente thread demandeuse
    - En FIFO (pas de famine)
    - Par priorité
- Thread propriétaire libère ressource :
  - Transmet ressource à prochaine thread en attente
  - Restaure priorité initiale de l'ex thread propriétaire

Héritage de Priorité (2)

- T1 acquiert ressource R
- T1 préemptée par T3
- T3 demande ressource R
  - T3 bloquée car R acquise par T1
  - Attribue prio(T3) à T1
  - T1 élue avec priorité T3
- T1 libère ressource R
  - Ressource R transmise à T3
  - T1 reprend sa priorité initiale
  - T3 élue

Priorité Plafond

- Augmente temporairement priorité de la thread propriétaire de la ressource à un maximum fixe (le "plafond")
- Évite préemptions inutiles
- Valeur du plafond déterminée par :
  - Connaisance des threads demandeuses
  - Criticité de la ressource acquise
  - etc...