Introduction au Parallélisme
Notion de Thread
Méthodes d'Ordonnancement

PLAN DU COURS

- **Concept de Thread**
- **États d'un Thread**
- **Ordonnancement**
- **API Thread POSIX**
- **API Thread Windows**

Processus ≠ Programmation Parallèle

- Processus pas adapté
  - Objet coûteux à créer et à détruire
  - Changements de contexte pas efficaces
  - Contexte mémoire lourd
- Partage de données "pas naturel"
  - Pas intégré avec langage de programmation
  - Offert par services OS complexes
- Programmation difficile, non contrôlable

Concept de Thread

- Notion combinant avantages
  - Exécution parallèle
  - Partage code et données application (ou noyau)
- Facile à mettre en œuvre
  - Programmable
  - Contrôlable
  - Configurable
- Efficace et "scalable"

Thread – Objet Programmable

- Dérivé de notion de processus
  - Unité d'exécution concurrente dans un même programme applicatif
  - Threads partagent espace d'adressage du processus "englobant"
  - Exécutent même fonction ou fonctions dédiées
- Partagent tout ou partie ressources système du processus
- Contexte de travail privé alloué par application

Exemple : Serveur Multi-Threads

```
static endpoint_t srv_endpoint;

service_thread() {
    struct request_t rqst;
    struct reply_t reply;
    for (;;) {
        wait_client_request(&srv_endpoint, &rqst);
        process_request(&rqst, &reply);
        send_reply_to_client(&srv_endpoint, &reply);
    }
}

main(argc, argv) {
    create_endpoint(SERVICE_ID, &srv_endpoint);
    for (i = 0; i < nb_serv_th; i++)
        thread_create(service_thread, THREAD_PRIO);
}
```

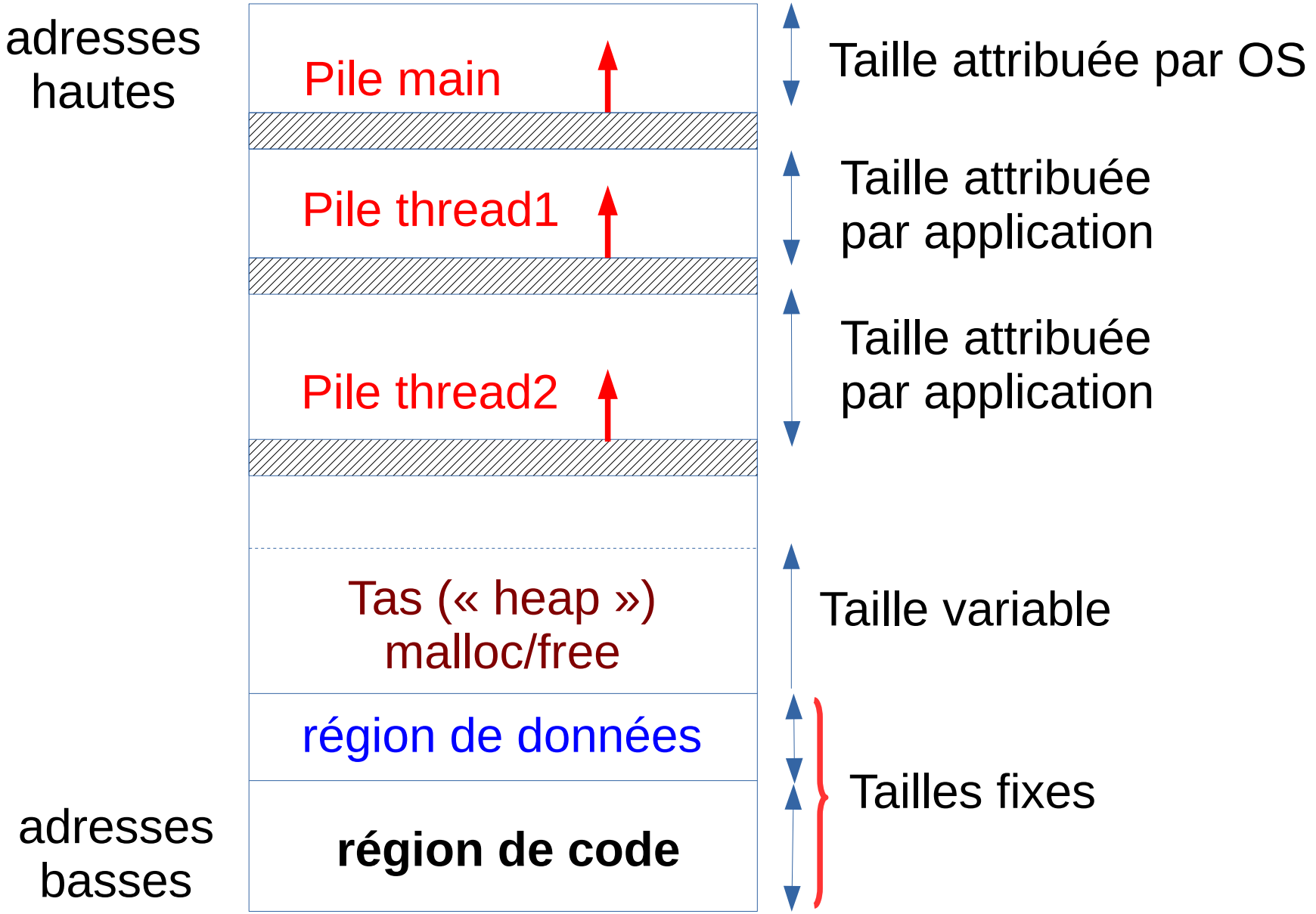
Thread – Objet Noyau de l'OS

- Unité d'Exécution Indépendante
  - Pile applicative
    - Utilisée pour exécution du code applicatif
  - Pile système
    - Utilisée pour exécution du code noyau (appels système, exceptions)
  - Contexte CPU (registres)
  - État courant et attributs d'ordonnancement
  - Identificateur
- Entité d'attribution d'un CPU par ordonnanceur du noyau

Piles d'Exécution d'un Thread

- Pile Système
  - Utilisée pour exécution des appels système
  - Allouée/libérée par noyau du système
- Pile Utilisateur
  - Utilisée pour exécution du code applicatif
  - Allouée/libérée par
    - Fonction de bibliothèque (défaut)
      - Avec cache de piles (POSIX)
    - Application elle-même
- Libération Pile Utilisateur
  - Effectuée par Thread avant appel système pour être détruite
  - Doit éviter d’accéder à la pile après l’avoir libérée...

Espace Adressage Processus Multi-Threads



Threads et OS

- Famille Unix
  - POSIX Threads – BSD, Linux
  - Lightweight Process (LWP) – Solaris
- Windows
- Mach/Mac-OS (Apple)
- Systèmes temps-réels
  - ChorusOS
  - VxWorks (WindRiver)
  - QNX, OSEK, Nucleus, etc.

Lightweight Threads

- Threads uniquement niveau applicatif
  - Implémentées par fonctions de bibliothèque
  - Pas de contexte dans le noyau de l'OS
- Appel système bloquant
  - Bloque capacité d'exécution de toutes les LWT
- Pas capable d'exploiter parallélisme des architectures multi-processeurs

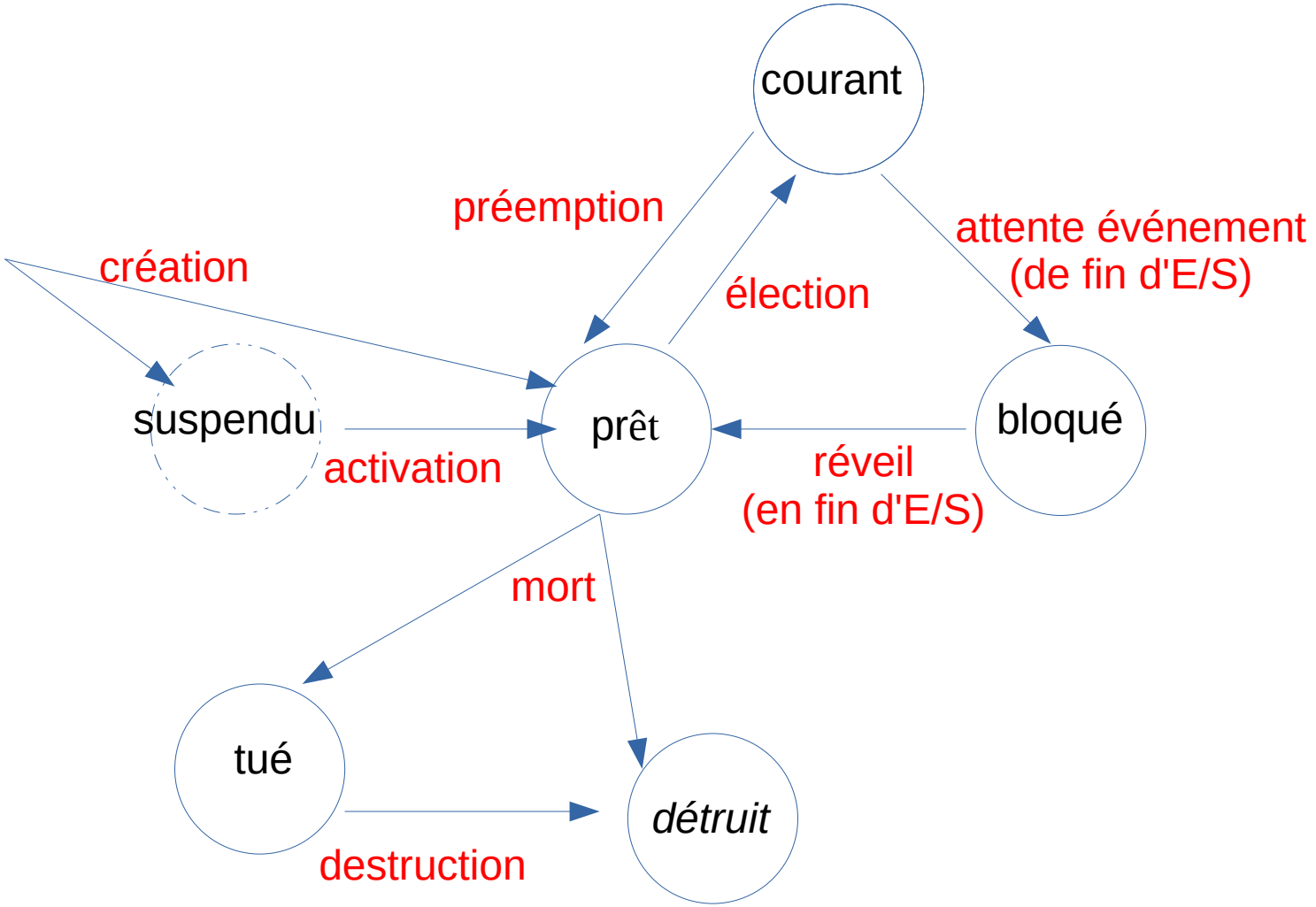
Threads et Langages de Programmation

- Objet Thread intégré dans langage
  - ADA : type Task
  - Java : classe Thread
  - C++ : classe « thread »
  - Implique support mécanisme de synchronisation
- Support concepts OS dans un langage
  - +facile, +maîtrisé => +fiable (soit-disant…)
  - -souple, -clair (mélange niveaux applicatif et OS)
  - Comportement dépendant OS => problèmes de portabilité
  - Création de Thread déclarative traduit par appel système de création de Thread
    => récupération d'erreur confuse

PLAN DU COURS

- Concept de Thread
- **États d’un Thread**
- Ordonnancement
- API Thread POSIX
- API Thread Windows

États d'un Thread



Transitions États d'un Thread (1)

- **Courant -> Bloqué** (mise en attente)
  - Thread doit arrêter son exécution pour attendre l'arrivée d'un événement (fin d'E/S)
  - Libère volontairement [ressource] CPU
  - Noyau alloue CPU à thread prêt le + prioritaire
- **Bloqué -> Prêt** (réveil)
  - Thread réveillé par événement attendu
    - Une [fonction d'] interruption (fin d'E/S)
    - Le Thread courant (client/serveur : pipe)
  - Rejoint groupe des Threads demandeurs de la ressource CPU

Transitions États d'un Thread (2)

- **Prêt -> Élu** (élection)
  - CPU attribué à Thread prêt le + prioritaire
- **Élu -> Prêt** (préemption)
  - CPU retiré à Thread courant pour être attribué à Thread + prioritaire
- Point(s) de préemption du noyau de l'OS
- Ordonnancement des Threads

Points de Préemption du Système

- Thread réveillé + prioritaire que Thread courant
  - Quand/où préempte Thread courant ?
- Coarse Grain System
  - Pas de préemption durant exécution code du noyau
  - Préemption au retour d'un appel système
  - Dépend durée exécution appel système
- Fine Grain System
  - Préemption durant exécution code du noyau
  - Immédiate

PLAN DU COURS

- Concept de Thread
- États d’un Thread
- **Ordonnancement**
- API Thread POSIX
- API Thread Windows

Principes de l'Ordonnancement

- Choisir prochain Thread à exécuter
- Systèmes Interactifs / Serveurs
  - Nombre de Threads variable, événements non prévisibles
  - Partager ressource CPU de façon équitable
- Systèmes Temps Réel
  - Temps-Réel "dur"
    - Garantir échéances de toutes les activités
    - Implique connaissance durées d'exécution
- Temps-Réel "mou"
  - "Best Effort" : essaie de garantir échéances

Politiques d'Ordonnancement

- Critères choix prochain Thread à exécuter
  - Réduire temps moyen d'attente
  - Garantir temps de réponse des Threads + prioritaires
  - Éviter phénomènes de famine
    - Garantir un minimum [de temps CPU] à tous les Threads
- Compromis avec contraintes
  - Optimiser taux utilisation [autres] ressources
  - Limiter fréquence changements de contexte
  - Mise en œuvre simple, maîtrisable, adaptable

Classes d'Ordonnancement

- A l'ancienneté
  - Premier arrivé, Premier servi ("FIFO")
- Par priorités
  - Fixes, imposées par applications
  - Variables, modifiées par noyau de l'OS :
    - Durée d'attente écoulée
    - « Prix » des ressources allouées à chaque Thread
- Par quantum de temps (durée maximale)
- Par échéances

Méthodes d'Ordonnancement

- Méthode du tourniquet ("Round-Robin")
  - Une file d'attente par priorité
  - FIFO + [re]mise en attente en fin de file
- Priorité Thread diminue après utilisation quantum de temps
  - Appliquée uniquement quand Thread exécute code applicatif
- Distinction entre plusieurs niveaux de priorités
  - Threads temps-réels
  - Threads ordinaires

PLAN DU COURS

- Concept de Thread
- États d’un Thread
- Ordonnancement
- **API Thread POSIX**
- API Thread Windows

Threads POSIX

C

- API POSIX pour programmes multi-threads
- Assure portabilité entre systèmes différents
- Restrictions concernant ressources OS partagées par Thread d'un même processus
  - fichiers ouverts, répertoire courant, etc.
- Inclut mécanismes de synchronisation
- Inclut support architectures multi-processeurs
  - Sélection CPUs d'exécution

Création Thread POSIX

```
#include <pthread.h>
int
pthread_create (pthread_t* thread,
                pthread_attr_t* attr,
                void* (*start_func)(void*),
                void* star_func_arg);
```

- Thread créé dans l'état actif
  - Pas d'état optionnel « suspendu »
- API simple : un seul appel système pour créer un Thread

Paramètres pthread_create()

- `pthread_t* thread`
  - si OK, contient identificateur du Thread créé
- `pthread_attr_t* attr`
  - Si NULL, Thread créé avec attributs par défaut
- `void* (*start_func)(void*)`
  - Adresse fonction exécutée par le Thread
- `void* start_func_arg`
  - Argument de la fonction exécutée par le Thread

Destruction Thread POSIX

- `void pthread_exit(void* exit_status)`
  - **exit_status** : status de fin de Thread courant
  - Détruit Thread courant – fonction sans retour
- `int pthread_join(pthread_t target_thread_id, void** exit_status_ptr)`
  - **target_thread_id** : identificateur du Thread cible
  - **exit_status_ptr** : pointeur où copier valeur de « exit_status » du Thread cible après qu'il se soit terminé
  - Bloque Thread courant jusqu'à terminaison Thread cible

Attributs d'un Thread POSIX

- **JOINABLE / DETACHED**
  - Terminaison « joignable » par autre Thread ou pas
- Taille de la pile
- Taille de la zone de protection de la pile
- Adresse de la pile
- Politique d'ordonnancement
  - **SCHED_FIFO** : temps-réel
  - **SCHED_RR** : préemptable
  - **SCHED_OTHER** : ordinaire (non temps-réel)
- Priorité

Attributs de Création de Thread

- Réunis dans une structure de données
  - Configurables globalement
  - `pthread_attr_init(pthread_attr_t* attr)`
- Modifiables individuellement par fonctions dédiées
  - `pthread_attr_setdetachstate(pthread_attr_t* attr, int detach_state)`
- Minimise coût de création de threads de même nature
- Priorité d'un Thread peut être changée après sa création
  - `pthread_setschedprio(pthread_t thread_id, int prio)`

PLAN DU COURS

- Concept de Thread
- États d’un Thread
- Ordonnancement
- API Thread POSIX
- **API Thread Windows**

Création Thread Windows

```
HANDLE CreateThread (
   LPSECURITY_ATTRIBUTES    lpThreadAttributes,
   SIZE_t                   dwStackSize,
   LPTHREAD_START_ROUTINE lpStartAddress,
   LPVOID                    lpParameter,
   DVOID                    dwCreationFlags,
   LPDWORD                   lpThreadId
);
```

Paramètres CreateThread (1)

- `lpThreadAtributes`
  - si TRUE, handle Thread créé héritable dans processus fils
- `dwStackSize`
  - Taille pile d'exécution du Thread créé
  - Si zéro, utiliser taille par défaut
  - Dernière page de la zone mémoire allouée pour la pile utilisée comme protection contre débordements de pile
- `lpStartAddress`
  - Adresse fonction exécutée par Thread créé
- `lpParameter`
  - Argument de la fonction

Paramètres CreateThread (2)

- dwCreationFlags
  - CREATE_SUSPENDED
    - Le thread est créé dans un état suspendu jusqu'à ce que la fonction « ResumeThread » soit appelée
  - STACK_SIZE_PARAM_IS_A_RESERVATION
    - Le champs « dwStackSize » spécifie la taille de réserve initiale de la pile
- lpThreadId
  - Pointeur vers une variable qui reçoit l'identificateur du Thread créé. Si ce paramètre a la valeur NULL, l'identificateur du Thread créé n'est pas retourné

Ordonnanceur Windows

- Priorités entre 0 (min) et 31 (max)
- Priorité 0 à Thread système spécial (mise à zéro pages libres)
- Priorité d'un Thread fonction de :
  - Partie fixe appelée priorité de base
  - Partie variable appelée priorité dynamique
- Ordonnancement selon la méthode du tourniquet
  - Threads de même priorité en FIFO

Priorité de Base d'un Thread

- Priorité de Base fonction de la classe de priorité du processus
- IDLE_PRIORITY_CLASS
  - Ex: écran de veille
- NORMAL_PRIORITY_CLASS
  - Priorité par défaut
- HIGH_PRIORITY_CLASS
  - Ex : gestionnaire de tâches
- REALTIME_PRIORITY_CLASS

Priorité Dynamique Threads

- Priorité de tous les Threads d'un processus
  - Augmentée de 2 lorsque le processus passe au premier plan
- Priorité d'un Thread
  - Diminuée de 1 après son quantum de temps jusqu'à sa priorité de base

Changement Priorité Thread (1)

```
BOOL SetThreadPriority(HANDLE hThread,
                       int priority)
```

- THREAD_PRIORITY_LOWEST
  - priorité thread = priorité de base processus – 2
- THREAD_PRIORITY_BELOW_NORMAL
  - priorité thread = priorité de base processus – 1
- THREAD_PRIORITY_NORMAL
  - priorité thread = priorité de base processus

Changement Priorité Thread (2)

- THREAD_PRIORITY_ABOVE_NORMAL
  - priorité thread = priorité de base processus + 1
- THREAD_PRIORITY_HIGHEST
  - priorité thread = priorité de base processus + 2
- THREAD_PRIORITY_IDLE
  - priorité thread = 16 si processus de classe REALTIME
    1 sinon
- THREAD_PRIORITY_TIME_CRITICAL
  - priorité thread = 31 si processus de classe REALTIME
    15 sinon

ANNEXE – Liens Utiles

- POSIX API

  man7.org/linux : PTHREADS manual

  - Création de Thread

    man3.org/linux : pthread_create()

  - Changement de priorité de Thread

    man3.org/linux : pthread_setschedprio()

- Windows API

  - Création de Thread

    learn.microsoft.com : CreateThread()

  - Changement de priorité de Thread

    learn.microsoft.com : SetThreadPriority()