Systèmes embarqués temps réel

Systèmes temps réel : définition

- Système dont le fonctionnement est soumis à des contraintes de temps
  - STR strict (temps réel dur) : fonctionnement avec des contraintes qui ne tolèrent aucun dépassement de temps:
    - exemples les contrôles de processus industriels sensibles comme la régulation des centrales nucléaires ou les systèmes embarqués utilisés dans l'aéronautique.
  - STR souple (temps réel mou) : les contraintes peuvent être dépassées dans certains cas particuliers:
    - exemple des systèmes multimédia : si quelques images ne sont pas affichées, cela ne met pas en péril le fonctionnement correct de l'ensemble du système

Temps-réel : définition

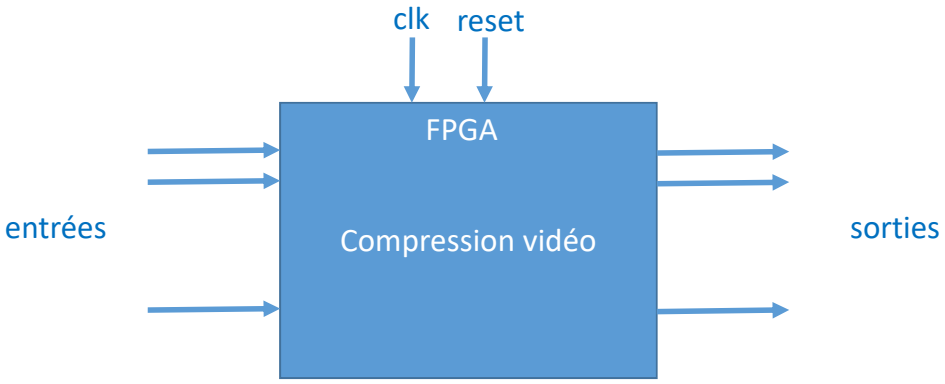
- Temps réel en flux
  - Contraintes sur le délai entre entrées et sorties d'un système
    - Lié principalement à la vitesse de traitement du système
    - **Systèmes orientés calcul**
    - Exemple : filtrage temps réel d'un signal, décodage d'un signal de TV TNT

Temps-réel : définition

- Temps réel en réponse à un événement
  - Contraintes de délai entre un ou des événement(s) et la réponse donnée par le système
    - Lié à la capacité du système à réagir à l'évènement et à la mise en œuvre du système de commande surtout en présence de plusieurs événements possibles
    - **Systèmes orientés contrôle**
    - Exemple :
      - arrêt d'un robot à proximité d'un obstacle (détection par capteur ultrason et commande moteur) → temps réel mou
      - airbag d'une voiture (détection de choc et explosion de l'airbag) → temps réel dur

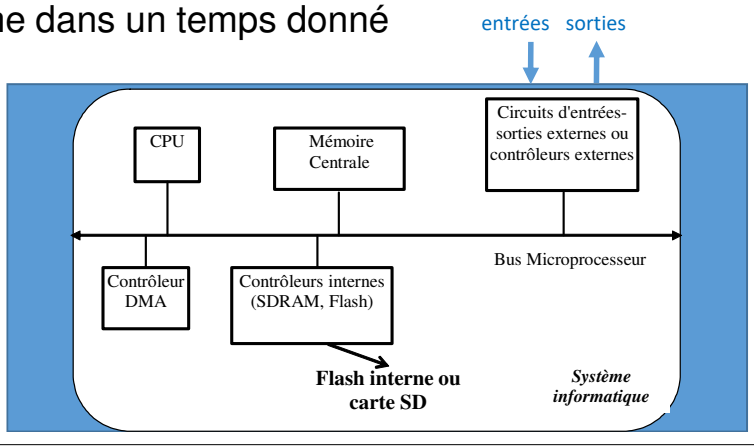
Catégories de systèmes temps réel

- Système temps réel basé sur un composant dédié (type FPGA) pour le traitement de l'information :
  - Le traitement de l'information est assuré par des blocs en logique câblés (décrits en VHDL ou Verilog et synthétisés)
  - Ces blocs peuvent être indépendants et fonctionner en parallèle
  - La vitesse de traitement de l'information est définie par la fréquence max de l'horloge de fonctionnement (qui dépend elle même du chemin critique → chemin combinatoire le plus long)



Catégories de systèmes temps réel

- Système temps réel basé sur un système informatique
  - Le traitement de l'information est assurée par l'exécution d'instructions par un processeur de façon séquentielle
  - Pour un processeur mono cœur, le processeur exécute une instruction à la fois à un instant donné. Cette instruction peut prendre un ou plusieurs cycles d'horloge du processeur
  - La vitesse de traitement de l'information est définie par la capacité du système à exécuter toutes les instructions nécessaires au fonctionnement du système dans un temps donné



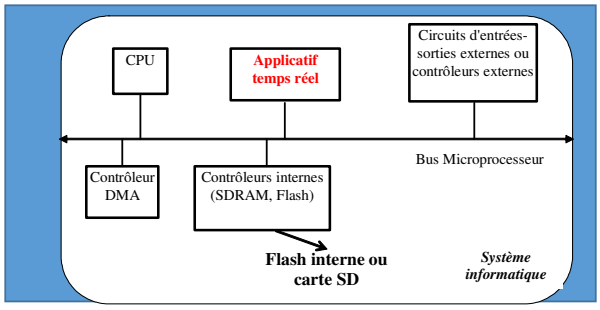
Objet du cours

Différents systèmes informatiques

- Système baremetal :
  - Système informatique **sans OS** (système d'exploitation)
- Système avec un OS mono-tâche
  - **OS** mono-tâche est un programme qui permet
    - de gérer les ressources matérielles (entrées/sorties, mémoire)
    - de fournir un interface utilisateur
  - Exemples: Uboot, DOS
  - Un moniteur (monitor) est un système mono-tâche simplifié
- Système avec un OS multitâche
  - Même fonction qu'un système mono-tâche
  - En plus ordonnancement des tâches
  - Exemples: Linux, Windows

Différents systèmes temps réel basés sur un système informatique

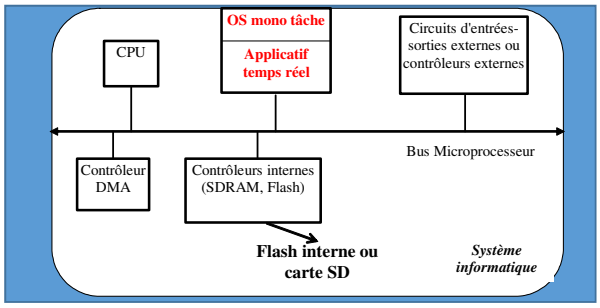
- Système temps réel baremetal :
  - Au démarrage, **un seul** programme (applicatif temps réel) est exécuté par le processeur et est dédié à **une tâche précise**
    - Gestion des entrées /sorties par l'applicatif temps réel lui même
    - Fonctionnalité temps réel en flux à condition que les capacités d'acquisition des données en entrée, le traitement et la restitution des données en sortie soient adaptées aux besoins
    - Fonctionnalité temps réel en réponse à un évènement à condition que le temps de traitement de l'évènement soit adapté au processus que l'on veut commander



Pas de problème de temps réel

Différents systèmes temps réel basés sur un système informatique

- Système temps réel avec OS mono-tâche :
  - OS (système d'exploitation) mono-tâche permet :
    - de faire des initialisations du système (gestion de la pile , des interruptions, …)
    - de lancer l'applicatif temps réel à exécuter (flexibilité par rapport à un système baremetal)
  - Une fois lancé, l'applicatif temps réel exécuté par le processeur est dédié à **une tâche précise et l'OS mono-tâche n'intervient plus** sauf évènement particulier (interruption clavier, …)
    - Gestion des entrées /sorties est faite par du code appartenant à l'l'OS mono-tâche
    - Fonctionnalités temps réel en flux et en réponse à un évènement sont les mêmes qu'en baremetal



Pas de problème de temps réel

Différents systèmes temps réel basés sur un système informatique

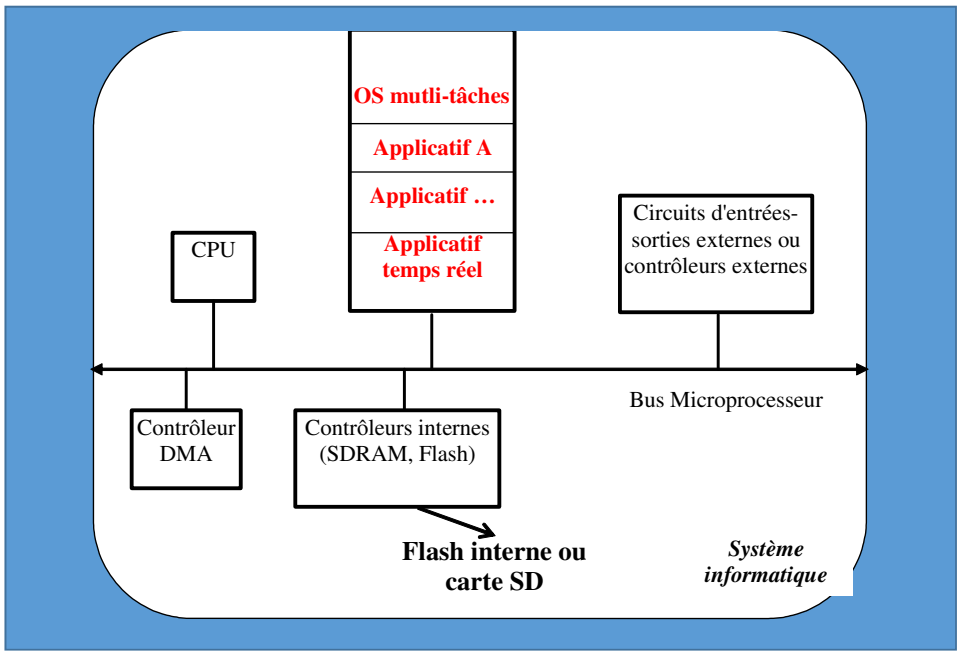
- Système temps réel avec OS multi-tâche :
  - Dans un système multitâche, il y a concurrence entre les différentes tâches. L'ordonnanceur qui gère les tâches est lui même une tâche qui a la priorité la plus élevée
  - Pour un traitement en flux en temps réel, l'applicatif est lancé par l'ordonnanceur et peut être interrompu.
  - Pour une réponse à un évènement en temps réel, l'applicatif doit être exécuté immédiatement sur l'évènement (interruption) mais cette exécution peut être différé à cause de l'ordonnancement

Problème de temps réel

→ **nécessité d'un OS multi-tâche temps réel (RTOS)**

Différents systèmes temps réel basés sur un système informatique

- Système temps réel avec OS multi-tâche (suite) :



Ordonnancement sur un système multi-tâches

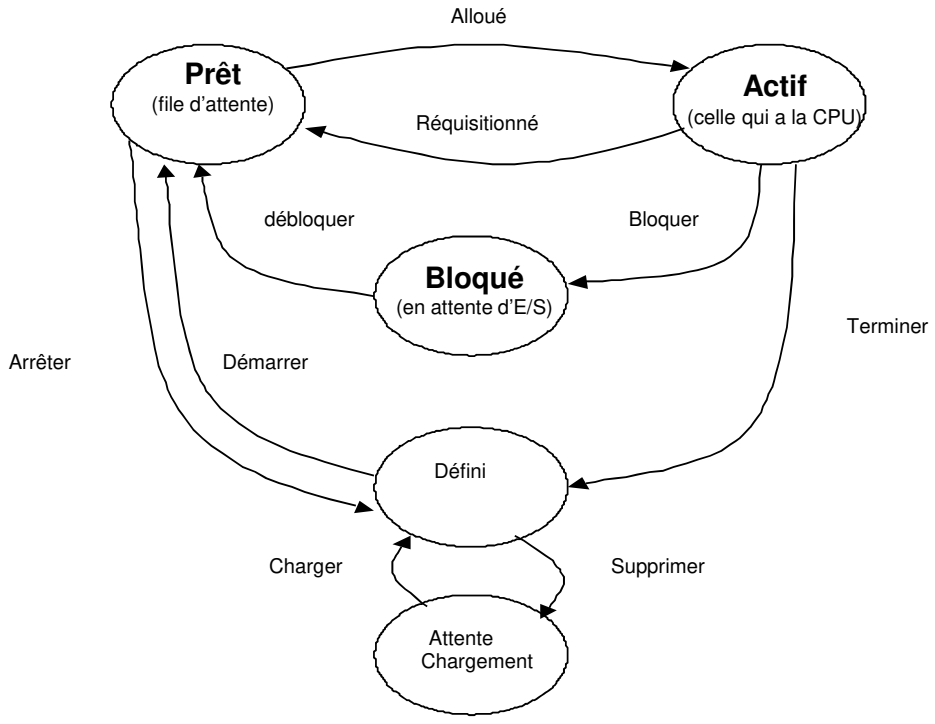
- Notions de programme et tâches
  - Programme:
    - Ensemble d'instructions constitué d'une ou plusieurs tâches s'exécutant de manière séquentielle (processeur mono-coeur) ou concurrentielle (processeur multi-coeur).
  - Tâches ou processus
    - partie du code à exécuter par le processeur
    - une tâche est constituée :
      - d'un ensemble de ligne de code
      - de données sur lesquelles agit le code
      - d'une zone de mémoire donnant les caractéristiques de la tâche (priorité, état…)
    - Toutes les tâches sont stockées en mémoire centrale. Un descripteur de tâche va permettre de les adresser.

Ordonnancement sur un système multi-tâches

- Etats d'une tâche: 5 états possibles
  - tâche active (ou élue)
    - tâche en cours d'exécution
    - puissance CPU dédiée à cette tâche
  - tâche prête (ou éligible)
    - tâche disposant de toutes les ressources sauf le CPU
  - tâche bloquée
    - tâche en attente d'une ressource (E/S par exemple) autre que le CPU
  - tâche définie
    - tâche résidant en mémoire mais absente de la file d'attente des tâches prêtes
    - état intermédiaire avant le lancement d'une tâche et son exécution effective
  - tâche en attente de chargement
  - tâche n'ayant pu être chargée en mémoire

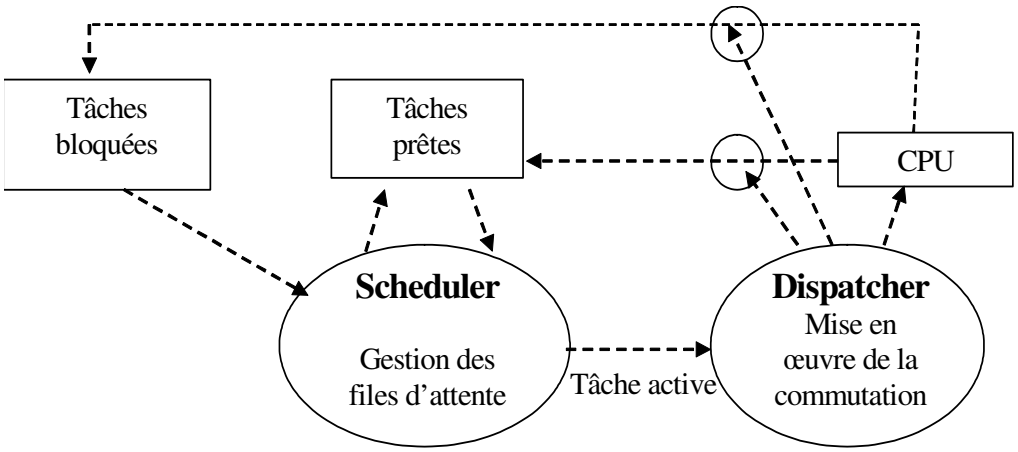
Ordonnancement sur un système multi-tâches

- Etats d'une tâche (suite)



Ordonnancement sur un système multi-tâches

- Ordonnancement de tâches
  - Réalisé par 2 programmes de l'OS
    - Scheduler: définir à chaque instant quelle est la tâche active:
      - Tâche désignée parmi les tâches prêtes (file d'attente)
    - Dispatcher: met en œuvre la commutation de tâches



Ordonnancement sur un système multi-tâches

- 2 stratégies d'ordonnancement possibles
  - Non préemptif
    - la tâche active conserve le CPU jusqu'à
      - ce qu'elle se termine
      - ou ce qu'elle soit bloquée (ressource indisponible)
    - le CPU est alors attribué à la tâche élue par l'Ordonnanceur
    - inconvénient : répartition irrégulière du temps CPU
      - une tâche peut conserver indéfiniment le CPU et bloquer toutes les autres
  - Préemptif
    - La tâche active ne conserve pas la CPU:
      - elle est interrompue par l'ordonnanceur selon une politique (policy) donnée d'ordonnancement
    - Le but est de répartir les tâches dans le temps selon la politique choisie: illusion du multi-tâche

Ordonnancement sous Linux (version 3.16)

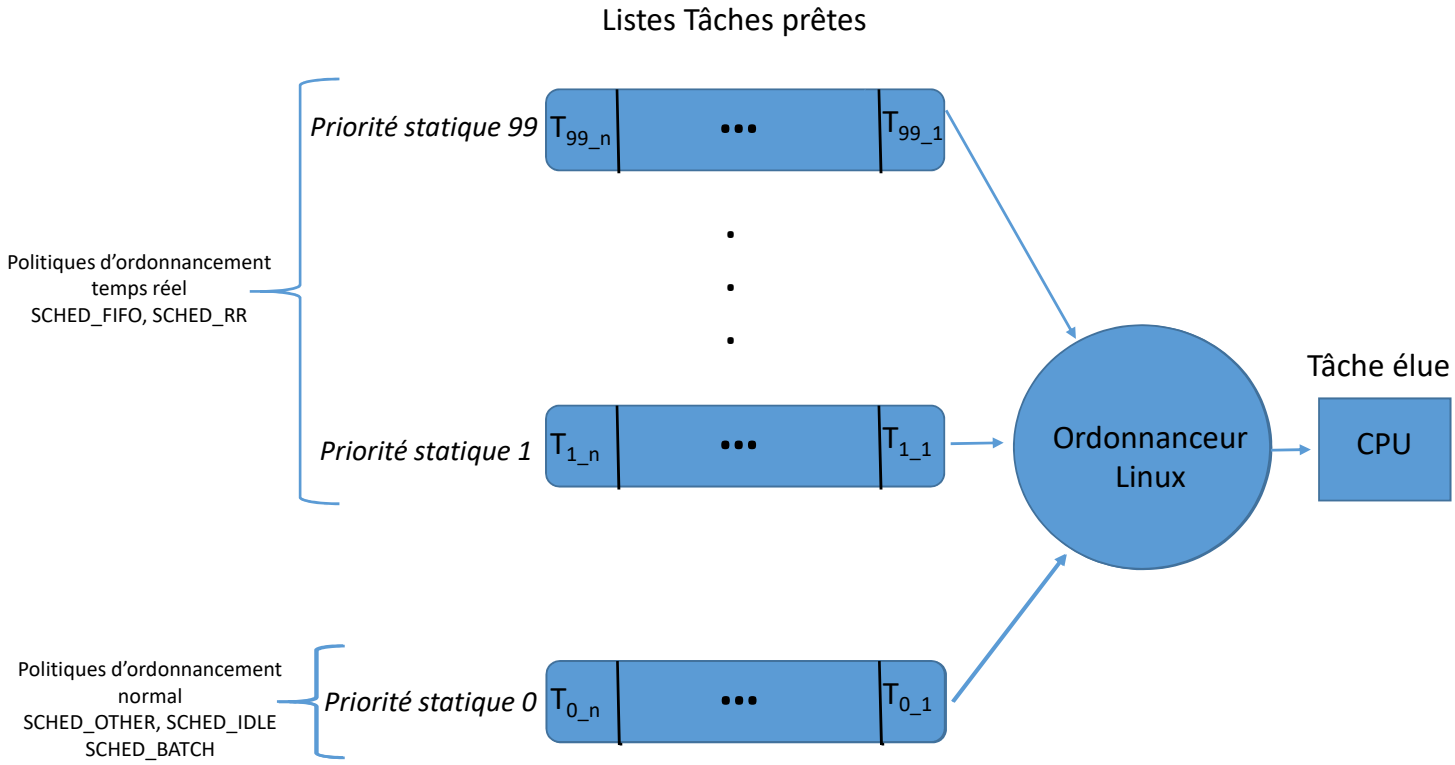
- Politiques d'ordonnancement sous Linux
  - L'ordonnanceur Linux est le composant du noyau qui décide quelle sera la tâche exécutée par la CPU
  - Chaque tâche est associée à une politique d'ordonnancement et à une priorité statique (appelée *sched_priority*)
    - L'ordonnanceur Linux prend ses décisions en tenant compte de la politique d'ordonnancement et de la priorité statique de toutes les tâches du système
  - Il existe 5 politiques d'ordonnancement sous Linux
    - 3 politiques d'ordonnancement dites normales (***SCHED_ OTHER, SCHED _IDLE et SCHED_BATCH***)
      - La priorité statique *sched priority* n'est pas utilisé dans les décisions. La valeur doit être spécifiée à 0
    - 2 politiques d'ordonnancement dites temps réel (**SCHED_FIFO et SCHED_RR**)
      - La priorité statique *sched priority* est comprise entre 1 et 99

Ordonnancement sous Linux (version 3.16)

- Politiques d'ordonnancement sous Linux (suite)
  - L'ordonnanceur Linux maintient une liste des tâches pour chaque valeur possible de la priorité statique
  - Pour déterminer quelle tâche sera élue, l'ordonnanceur examine la liste non vide des tâches de plus haute priorité et sélectionne la tâche tête de liste
  - Les 5 politiques d'ordonnancement sont préemptifs:
    - Si une tâche de plus haute priorité devient prête à être exécutée, la tâche courante est préemptée et retourne dans sa liste de sa priorité
  - La politique d'ordonnancement **détermine seulement <u>l'ordre d'election</u>** à l'intérieur d'une liste de tâches de même priorité

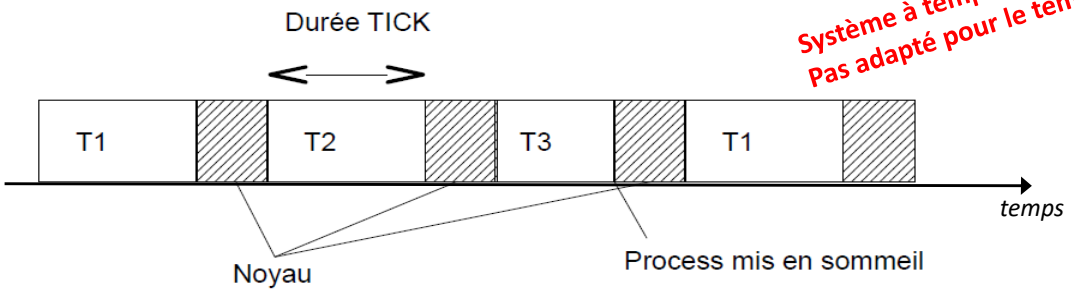
Ordonnancement sous Linux (version 3.16)

- Politiques d'ordonnancement sous Linux (suite)



Politique d'ordonnancement sous Linux: SCHED_OTHER

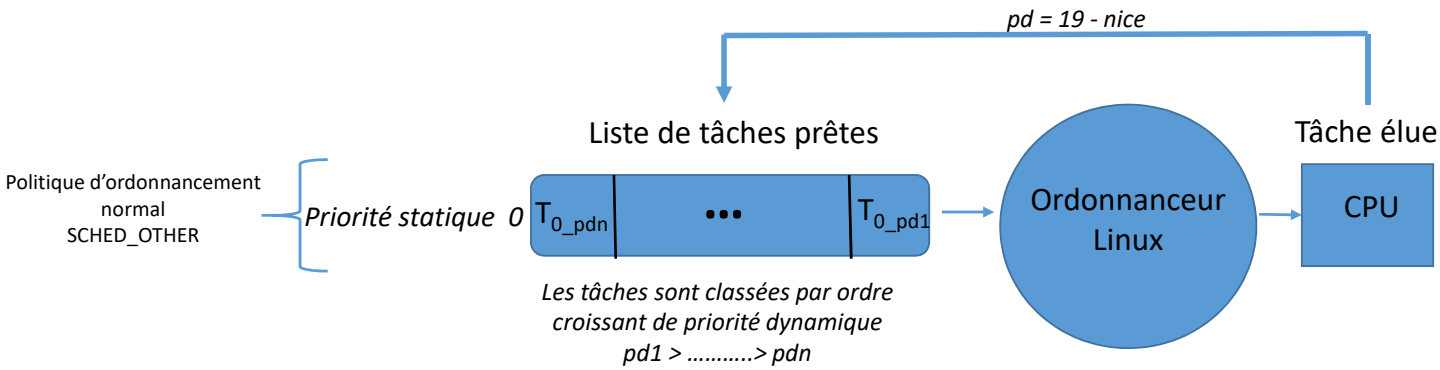
- SCHED_OTHER est la politique d'ordonnancement normal sous Linux:
  - Peut être seulement utilisé avec une priorité statique de 0
  - Découpage des tâches en quantum de temps appelé TICK (ou time slice)
  - Prise de contrôle du CPU par le noyau à chaque fin de TICK ou lorsqu'une tâche active est placée en sommeil.



  - La durée de possession du CPU par un processus est constante sauf en cas de mise en veille du processus par lui-même (wait, usleep...) cette durée s'appelle un TICK (time slice)
  - Un timer système interrompt le processus en cours et donne la main au noyau
  - Le noyau détermine le processus qui doit prendre le processeur

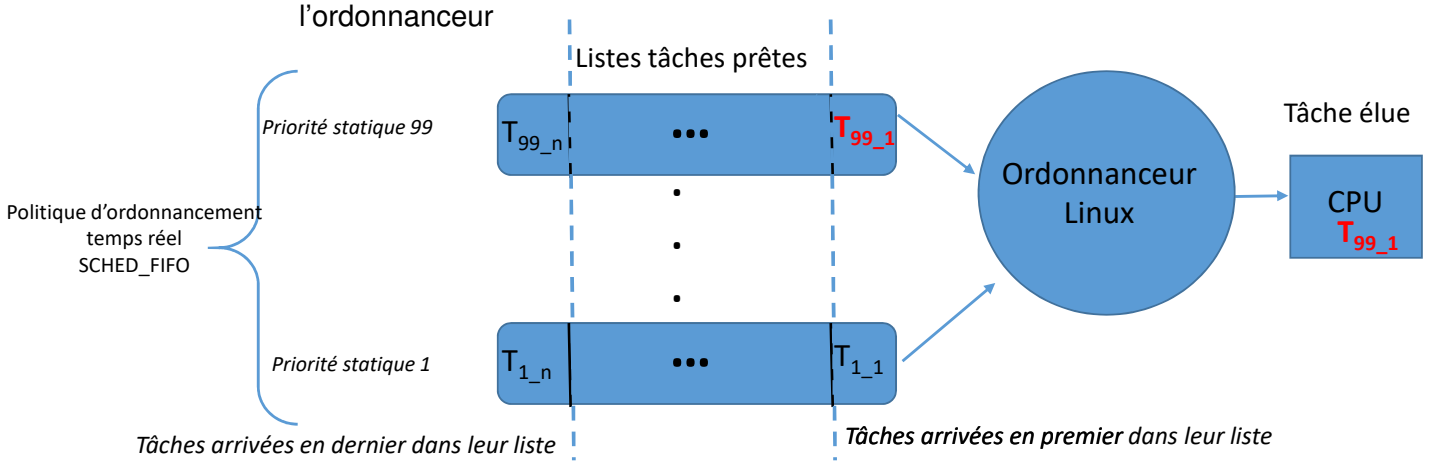
Politique d'ordonnancement sous Linux: SCHED_OTHER

- Comment SCHED_OTHER détermine l'ordre des tâches à exécuter à l'intérieur de la liste des tâches de priorité statique 0 ?
  - **par une priorité dynamique pd**
    - définie au départ par la valeur nice (gentillesse): ***pd = 19 – nice*** *avec -20 < nice <19 (nice=0 par défaut donc pd=19)* → un processus est « gentil » avec une valeur de nice positive
    - dans le temps cette priorité pd est augmentée de 1 à chaque TICK quand la tâche n'est pas allouée au processeur par le scheduler
- Quand la tâche repasse de l'état élue à l'état prêt, elle reprend comme valeur de priorité dynamique *pd = 19 - nice*

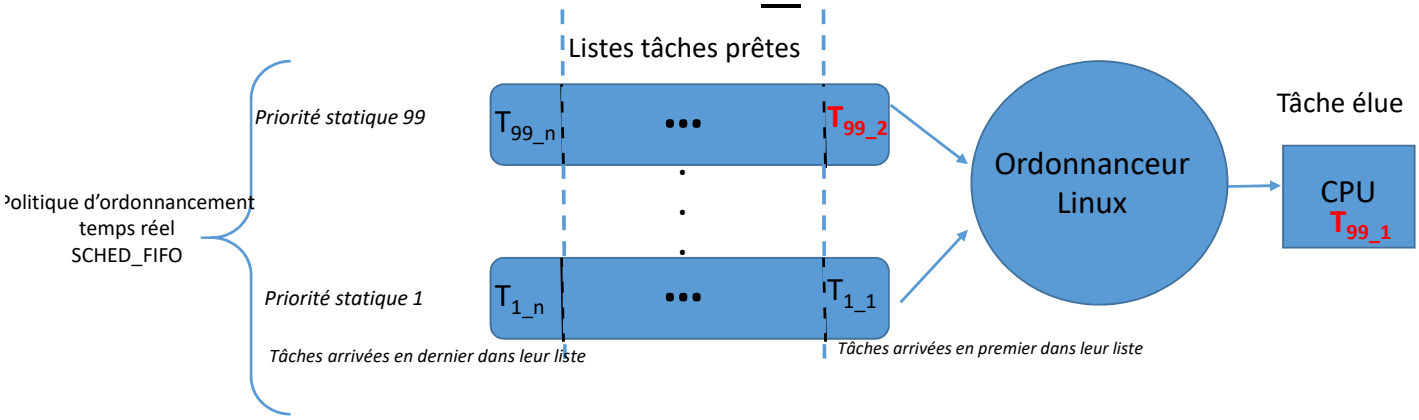


Politique d'ordonnancement sous Linux: SCHED_FIFO

- SCHED_FIFO est une politique d'ordonnancement **temps réel** sous Linux:
  - peut être utilisé avec des tâches avec des priorités statiques supérieures à 0 allant de 1 à 99 (ceci concerne peu de tâches, la majorité des tâches ont une priorité statique de 0 et gérée par la politique SCHED_OTHER)
  - FIFO (First in First out):
    - premier arrivé, premier sorti dans sa liste de tâches !!!
      - à priorité égale la tâche arrivée en premier est la première dans la liste
      - la tâche de priorité statique la plus haute et première dans sa liste est celle élue par l'ordonnanceur

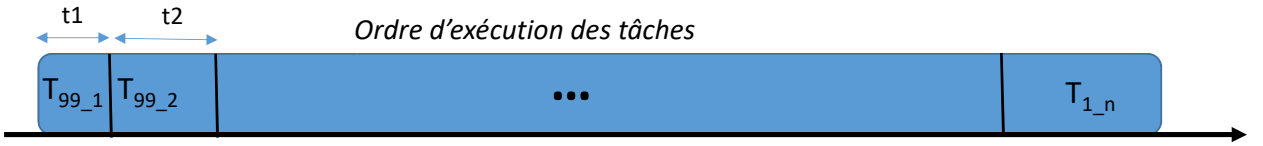


Politique d'ordonnancement sous Linux: SCHED_FIFO



- T_{99_1} est exécutée complètement par le CPU, puis T_{99_2} et ainsi de suite, les tâches sont balayées de la tâche de plus haute priorité vers la plus basse priorité.

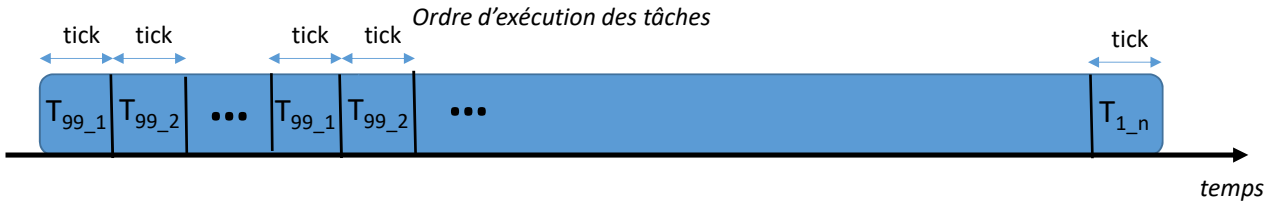
- Dans cette politique l'exécution des tâches n'ont pas la même durée



- Les tâches SCHED_OTHER sont exécutées après
- Pas de quantum de temps, pas de TICK
- Si une tâche est bloquée, elle laisse sa place à la prochaine tâche exécutable et retrouve tout de suite sa place de premier dans la liste **ET** la CPU quand elle n'est plus bloquée

Politique d'ordonnancement sous Linux: SCHED_RR

- SCHED_RR est une autre politique d'ordonnancement **temps réel** sous Linux:
  - Amélioration de SCHED_FIFO pour répartir mieux dans le temps les tâches de priorité statique allant de 1 à 99
  - SCHED_FIFO + quantum de temps: même algorithme que SCHED_FIFO mais chaque tâche est allouée à la CPU pour un temps donné



  - A chaque qu'une tâche est interrompue par le tick, elle revient dans la liste des tâches prêtes au début de la liste (file circulaire → RR: Round Robin)