Gestion des erreurs temporelles dans un système temps réel critique

Thomas Robert / Etienne Borde

etienne.borde@telecom-paristech.fr

thomas.robert@telecom-paris.fr

Quelles défaillances ?
Mais, j'ai passé le test d'ordonnancement… ….

Pré-requis de ce cours

- Cours Ordonnancement :
  - Modèle de tâche temps réel (i.e. périodique avec échéances implicites)
  - Test d'ordonnançabilité + hypothèses et paramètres requis
  - Analyse de pire temps d'exécution (définition et utilisation).
- Cours Tolérance au fautes
  - Vocabulaire : Faute / Erreur / Défaillance – manifestation de la défaillance
  - Tolérance aux fautes : mécanismes logiciels
    - Recouvrement avant/arrière
    - Redondance active / passive
    - Zone de confinement d'erreur

Défaillance temporelle: définir le problème

- **Défaillance temporelle**
  = violation de la spécification de l'attendu
  != violation d'une hypothèse temporelle

  Problème qu'est ce qui est de l'ordre de l'attendu et de l'hypothèse

- Cas du modèle tâches périodiques échéance implicite
  - **Attendu** : Exécution périodique, de période T, d'un job démarré à t0=Activation, fini à t=Activation + Deadline
  - **Hypothèses** :
    - Processus de conception garantissant le dimensionnement d'une certaine manière (test d'ordonnançabilité + WCET)
    - Plateforme d'exécution garantissant les événements d'activation / attente
  - **Défaillance** : non respect de l'échéance / activation périodique (retard) ….
  - **Comprendre pourquoi on distinguera différents niveau** : impact différent en fonction de la défaillance => besoin connaissance métier.

Défaillances temporelles : en détail

Spécification tâche périodique de période T :
1 séquences d'intervalles (Ij) = I0… Ik, Ij =[T0+j*T, T0+(j+1)*T[

1 Défaillance = concerne 1 job a priori.

Défaillance d'une tâche = 1 job défaillant

- **Transitoire : 1 job de temps en temps**
- **Permanente : tous les jobs à partir d'un certain instant**

Détection :

- **À l'échéance**
- **A l'activation (beaucoup plus dure – doit vérifier que la tâche est ready … pas si simple)**

Remontons la chaine causale : les erreurs

- Analyse de l'erreur : activation interne vs externe
  - Erreur dans l'état d'ordonnancement de la tâche
    - Temps d'exécution > > WCET estimé
    - Synchronisation incohérente avec les priorités
    - Primitive d'attentes temporisée erronées.
  - Erreur externe à l'état d'une tâche.
    - Propagation de l'erreur depuis une autre tâche (sur budget consommés, ou synchro)
    - Latence à l'a prise en compte d'interruptions.
    - Conception erronée (pas de garantie sur ordo)

Les fautes ….. causant l'erreur.

- Lié au processus de conception
  - Mauvais processus d'analyse (correction?)
  - Mauvaise caractérisation du code déployé
- Lié à l'environnement d'utilisation
  - Faute sur le matériel perturbant l'exécution de l'application / de l'ordonnanceur
  - Faute dans le logiciel perturbant l'exécution de l'application / de l'ordonnanceur
- Résumé : il faut prévoir que cela se passera mal et garantir un comportement sûr….

Stratégies de tolérance aux fautes
Opportunités et enjeux

Défaillance d'une tâche vs du système

- **1 système temps réel = N tâches**
- **Chaque tâche est un sous système : il faut limiter la propagation**
- **Vecteurs de propagation :**
  - **Les primitives de synchronisation**
  - **L'allocation exclusive du processeur qui change le budget effectivement disponible à chaque tâche avant leur deadline.**
  - **Propagation d'interférence sur WCET via le processeur (perturbation des caches …et don du WCET).**

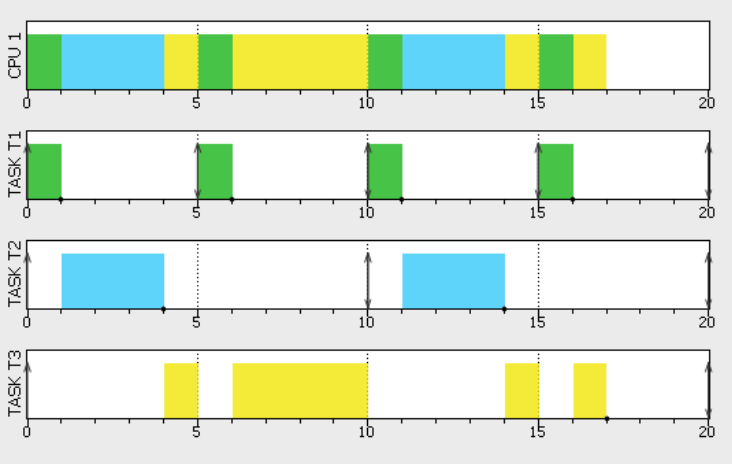
Cas simple : Détection de dépassement d'échéance et arrêt de tâche sur RMS

Pour un ordonnancement RMS d'un lot de tâche (système) :

- Détection d'erreur = dépassement d'une échéance par une tâche
- Recouvrement = arrêt de la tâche défaillante à l'échéance ratée
- Propriété de tolérance =
  - Si les temps d'exécution sont bien indépendants entre tâches indépendantes.
  - Alors prévient la propagation aux tâches plus fréquentes

Exemple :

Id	P	C
t1	5	1
t2	10	3
t3	20	7



Si t2 dépasse a un temps d'exécution = 9 au lieu de 3 => t2 et t3 ratent leurs échéances mais t1 est non affectée pour son 4ième job

Cas simple : Détection de dépassement d'échéance et arrêt de tâche – est ce une bonne idée pour EDF

Pour un ordonnancement EDF d'un lot de tâche (système) :

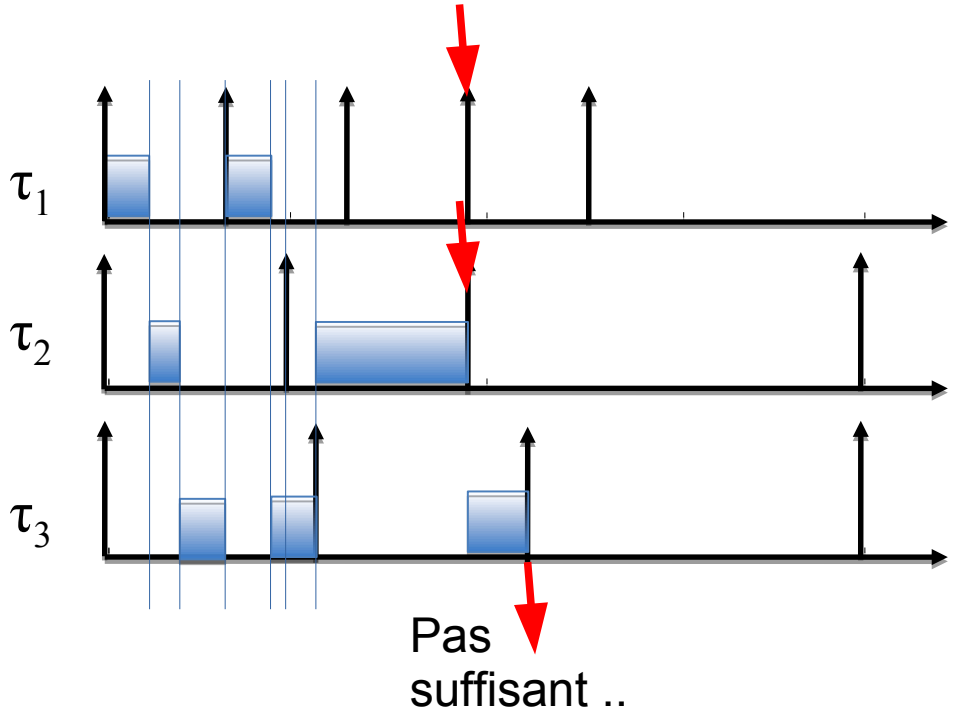
- Détection d'erreur = dépassement d'une échéance par une tâche
- Recouvrement = arrêt de la tâche défaillante à l'échéance ratée
- Propriété de tolérance = Aucune a priori ….

Exemple :

Id	P	C
t1	4	1.5
t2	6	1
t3	7	3

U=0.97
comportement FIFO pour échéances identiques

Si t2 a un temps d'exécution défaillant et s'exécute jusqu'à détection de l'échéance => t1 et t3 défaillent aussi



Détection avancée : identifier l'activartion des fautes

- Pb : peut on éviter la défaillance d'une tâche en détectant l'erreur à l'origine de sa défaillance
- Idée 1 : Comment détecter que le temps d'exécution est/sera supérieur au WCET
  - Solution 1 : approche basée budget consommé
  - Solution 2 : approche basée progression relative (a-t-on assez progressé dans le graphe de contrôle par rapport au budget qui reste
- Idée 2 : Comment détecter que les fréquences d'activation sont / seront supérieures à celles prévues :
  - Solution 1 : mesure du temps inter-arrivée
  - Solution 2 : estimation de la variation du temps inter-arrivée
- Et après la détection que fait-on ?

Définition du principe de robustesse

- Vous avez 1 spécification du comportement attendu A, garantie tant que certaines hypothèses sont vérifiées H.
- Vous avez 1 design D (du code + matériel avec tolérance aux fautes usuellement).
- Principe de la robustesse = même si H devient fausse, dans certaines limites D fournit toujours A.
- Ex1 : contrôleur de vitesse robuste à des erreurs de mesure de la vitesse réelle (<10 %).
- Ex2 : ordonnancment robuste à une erreur de détermination de WCET.

Formalisation de la robustesse

- **Robustesse inconditionnelle (cas précédant)**
  - Conception du système D assure :
    si Hypothèse 1 => Comportement A
  - Robustesse : même si Hypothèse 1 faux, Comportement A
- **Robustesse par priorisation**
  - Conception du système D assure :
    Si Hypothèse 1 => Comportement A & Comportement B
  - Robustesse : même si Hypothèse 1 faux, Comportement B dans tous les cas (i.e. A est sacrifiable)

Reconfiguration et mode de fonctionnement

- Objectif prouver que la propagation peut être bornée.
  => bien caractériser la manifestation
  => borner la nature la manifestation de la faute.

- Introduction de la notion de mode opérationnel
  - 1 mode = Hypothèses comportementales, environnement + description d'une structure
  - 1 mode est activable si non contredit par l'état courant
  - Changement de mode == passage d'une structure à une autre

Robustesse passive / active

- **Robustesse passive = confinement sans changement de structure**

Ex : Allowance = quantité de temps d'exécution qui peut être consommée sans compromettre l'ordonnançabilité pour 1 lot de tâche et 1 algorithme d'ordo

- **Robustesse active = confinement par changement de la structure du système (i.e. vous vous souvenez, structure = ce qui ne change pas)**

Ex : Criticité mixte avec défausse = modèle pour valider la robustesse du système en cas de temps d'exécution> WCET + robustesse par priorisation.

Allowance : robustesse intrinsèque du tandem (lot tâche, algorithme d'ordonnancement)

- 2 grands types d'analyses d'ordonnançabilité : basée utilisation, ou basée temps de réponse => inéquations.
- Un lot de tâche + algorithme d'ordonnancement => condition d'ordonnançabilité $F(p1, \ldots pk) < G(q1, \ldots qr)$
- Principe d'Allowance = regarder, pour les paramètres de type WCET, l'incrément Δ max applicable aux WCETs sans que l'inéquation ne devienne fausse.
- Exemple Ordonnancement EDF mono coeur, lot de tâche
  - T1 : P= 10 , C=3
  - T2 : P= 15 C=5
  - T3 : P= 20 C=5, de combien le C de chaque tâche peut il augmenter sans défaillance ? même valeur ?

Méthode de calcul et réponse à l'exercice

- Cas simple des conditions orientés utilisation pour 1 tâche
  - ∑ Ci/Pi ≤ B (B est une constante paramétrée parfois par le nombre de tâches-- cas de RMS).
  - Allowance Ai d'une tache i : $A_i = \limsup \{ x, (x < P_i * (B - \sum_{j \neq 0} \frac{C_j}{P_j})) \}$
  - Pour EDF B=1 donc dans l'exercie Ai=1…3
    - U1=3/10, U2= 1/3, U3=1/4 => U = 53/60
    - A1= 10* 7/60 unités temps, (env. 1,16)
    - A2=15*7/60 unités temps (1.75)
    - A3= 20*7/60 unités temps ~ 2,33
    - Remarque : allowance doit représenter à chaque fois le même incrément de taux d'utilisation !

Criticité mixte modèle avec défausse

Hypothèses : WCET dépendent de paramètres (accès mémoire, cache hit/miss) => 2 modes = 2 jeux d'hypothèses == 2 WCET par tâche.

- Mode 1 : optimiste dit LO mode tache i => Ci(LO)
- Mode 2 : pessimiste (ou dit sûr) Ci(HI) > Ci(LO)
- Transition mode 1 → mode si un tâche non terminé après avoir consommé son budget de mode 1.
- **Problème peut on garantir les échéances pour T= {t1,….,tk}=Taches_critiques U Taches_sacrifiables**
  - **T est ordonnançable en mode 1,**
  - **Taches_critiques ordonnançable en mode 2**
  - **Le passage du mode 1 à 2 garantit les échéances**

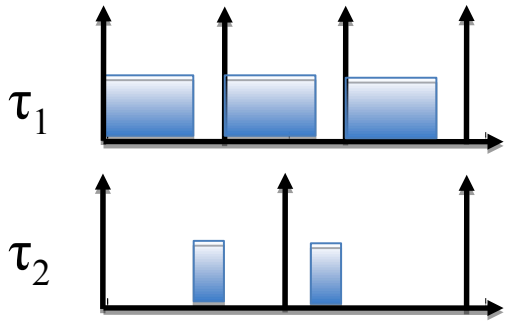
Le changement de mode : comprendre le challenge

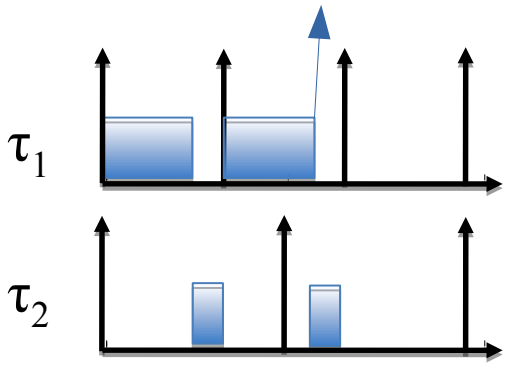
2 tâche 1 coeur, t1 de niveau 1 (P=4, C1=3)
t2 de niveau 2 (P=4, C1= 1 C2= 3)

Mode 1

OK

U≤1



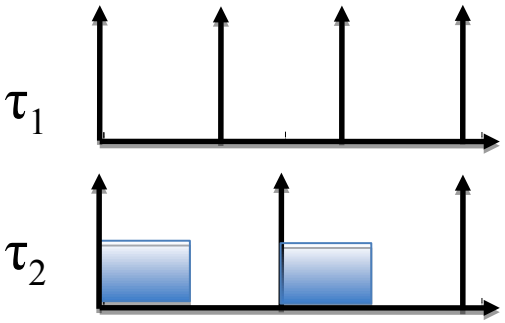


Pas assez

Mode 2

Ok

U<1



Les modes 1 et 2 peuvent tout à fait être ordonnançable séparément mais impossible à enchainer …

Et pour ordonnancer cela ?

Serveurs d'exécution

* Réserver le budget pour les tâches Hi dans des serveurs
* Allouer le budget non consommé en mode 1 dans les serveur par les tâche de mode 2 pour exécuter celles de niveau 1

Paramètres virtuels (priorité définies par modes)

* Utiliser des échéances virtuelle pour éviter le cas où une tâche de niveau 2 ne s'est pas exécuté suffisamment avant l'occurrences du changement de mode
* Au changement de mode changer d'échéance virtuelles

Technologie pour le programmeur

Ce dont nous parlerons….

- Mécanismes de détection / confinement usuels.
  - Plateforme ARINC
  - Posix
- Focus spécial sur watchdog pour détection dépassement d'échéance manquée.

Exemple au niveau OS: ARINC653 Health Monitoring

Health Monitoring (HM)

- Le standard ARINC653 fournit une section dédiée au « health monitoring », c'est à dire à la détection et au traitement des erreurs.
  - Remarque: le HM couvre un périmètre beaucoup plus large que les erreurs temporelles. Nous ne donnons ici que des principes généraux avec un focus sur les fautes temporelles.

- Il s'appuie sur
  - Une table de configuration (fichier de configuration XML)
  - Quelques services fournis par l'OS

- Le HM définit trois niveaux d'erreurs: process, partition, et module.
  - Cela correspond au trois niveaux où des erreurs logicielles peuvent se produire ou être détectée.

HM ARINC653: la détection des erreurs

- Les erreurs sont détectées au niveau
  - ⌘ hardware (memory violation, stack/heap overflow, zero divide…)
  - ⌘ noyau (problème de configuration, **dépassements d'échéances**…)
  - ⌘ Applicatif (valeur inattendu, trop ancienne, appel système qui échoue…)

- La liste complète des erreurs qui peuvent être détectées par un OS est « implementation specific »

HM ARINC653: le traitement des erreurs

- Le traitement d'une erreur dépend du niveau où elle se produit

- La table de configuration donne une correspondance entre le niveau de l'erreur, son type, et l'action à mener en cas d'occurrence.
  - Comme les types d'erreurs et les actions possibles sont dépendants de l'implémentation, le contenu de la table est aussi dépendant de l'implémentation.
  - Le standard donne cependant des exemples:
    - Ignorer l'erreur
    - Arrêter/Redémarrer le processus fautif
    - Arrêter/Redémarrer la partition fautive
    - Arrêter/redémarrer le module

HM ARINC653: types et services définis par le standard (1/2)

- Pour les erreurs au niveau applicatif, le standard définit des services et des types de données.

- Types de données
  - ERROR_CODE_TYPE, un enum: **DEADLINE_MISSED**, APPLICATION_ERROR, NUMERIC_ERROR, ILLEGAL_REQUEST, STACK_OVERFLOW, MEMORY_VIOLATION, HARDWARE_FAULT, POWER_FAIL
  - ERROR_STATUS_TYPE une structure qui contient :
    - Un champ ERROR_CODE de type ERROR_CODE_TYPE pour identifier la nature de l'erreur
    - MESSAGE de type ERROR_MESSAGE_TYPE, une chaine de caractères
    - LENGTH la longueur de MESSAGE
    - FAILED_PROCESS_ID de type PROCESS_ID_TYPE pour identifier le processus fautif
    - FAILED_ADDRESS : SYSTEM_ADDRESS_TYPE adresse d'occurrence de la faute

HM ARINC653: types et services définis par le standard (2/2)

- Pour les erreurs au niveau applicatif, le standard définit des services et des types de données.
- Les principaux services
  - RAISE_APPLICATION_ERROR, pour signaler une erreur depuis le code d'un process ou d'une partition.
  - CREATE_ERROR_HANDLER, pour créer un process (prio MAX) ARINC qui est réveillé sur occurrence d'une erreur au niveau application. Par exemple, pour un dépassement d'échéance, la détection se fait au niveau noyau (dans l'ordonnanceur) mais c'est bien une erreur au niveau application.
  - GET_ERROR_STATUS, pour récupérer les informations relatives à l'erreur (message, processus et partition fautives, ...)
- Le redémarrage d'une partition ou d'un processus se fera en utilisant les services de gestion des partitions et processus (SET_PARTITION_MODE, START, STOP, etc.)

Par rapport au cours précédent

- Dans le cours précédent, on vous a parlé de forward recovery et backward recovery

- Table → forward recovery seulement (pas de sauvegarde d'état)
  - ⌘ Dans le cas de ARINC653, cela concerne des erreurs au niveau partition/module

- CREATE_ERROR_HANDLER → forward ou backward…
  - ⌘ Dans le cas de ARINC653, cela concerne des erreurs au niveau applications

Quid de la gestion des erreur temporelles avec RT-POSIX?

RT-POSIX: erreurs de dépassement d'échéance

- Il n'y a pas de support pour détecter les dépassements d'échéance en RT-POSIX.

- Il est donc à la charge du développeur de développer un *watchdog*: un mécanisme de temporisation qui vérifie que chaque tâche termine avant une certaine date.

- Peut servir pour détecter des
  - ⌘ Dépassements d'échéance
  - ⌘ Dépassements de WCRT

Idée générale

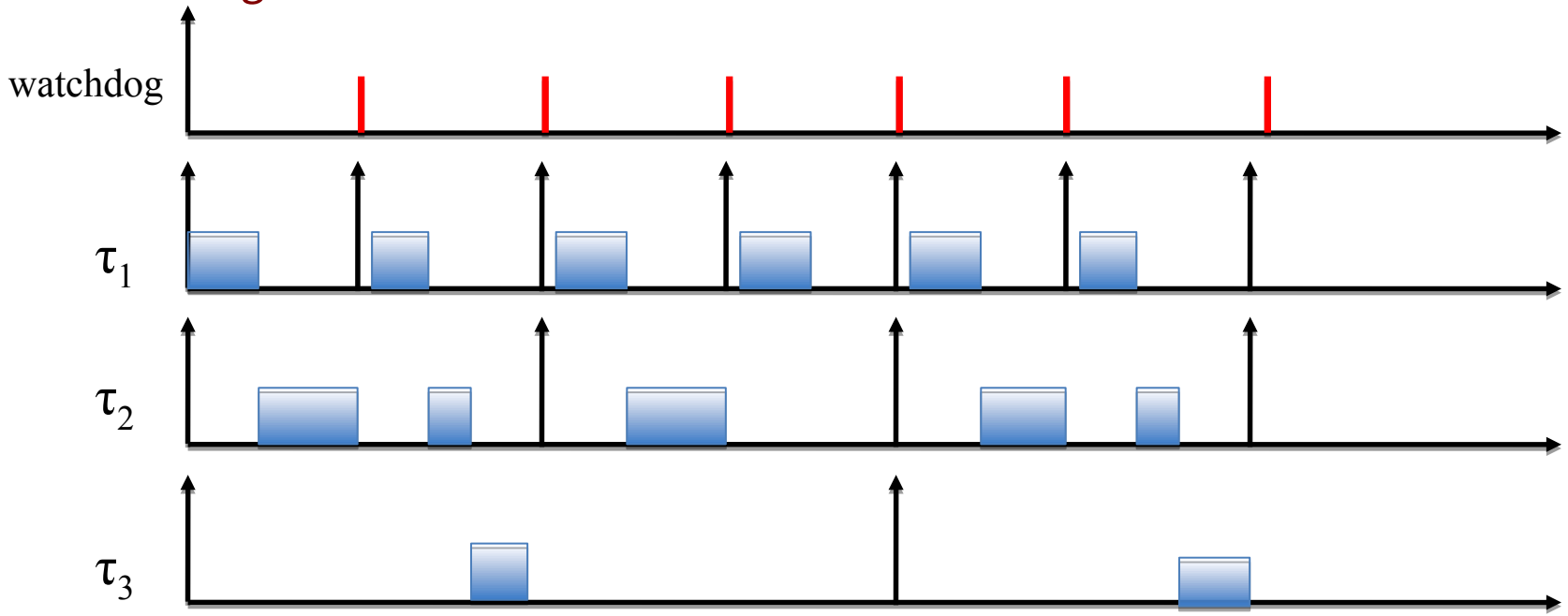
- En s'inspirant du pattern définit dans ARIN653, on peut créer une tâche *watchdog* de très forte priorité, avec un timer armé avec la date de la prochaine échéance. Exemple d'implémentation (que vous ferez en TP):
  - Chaque tâche arme un timer au début de son exécution, et le désarme à la fin de son exécution échéance du timer = échéance de la tache
  - Si la tâche n'a pas désarmé le timer au moment de son échéance, un signal est émis à la tâche en charge du signal (tâche watchdog).
  - La tâche de watchdog est en attente et se réveille sur occurrence de ce signal

Exemple: schéma d'ordonnancement

- Exemple simple: ordo RMS, 3 tâches (τ_1, τ_2, et τ_3) et un verrou (v_1).

Tâche	Période	WCET
τ_1	5 ms	2 ms
τ_2	10 ms	4 ms
τ_3	20 ms	2 ms

- Scénario d'ordonnancement avec illustration des réveils potentiels de la tâche *watchdog*

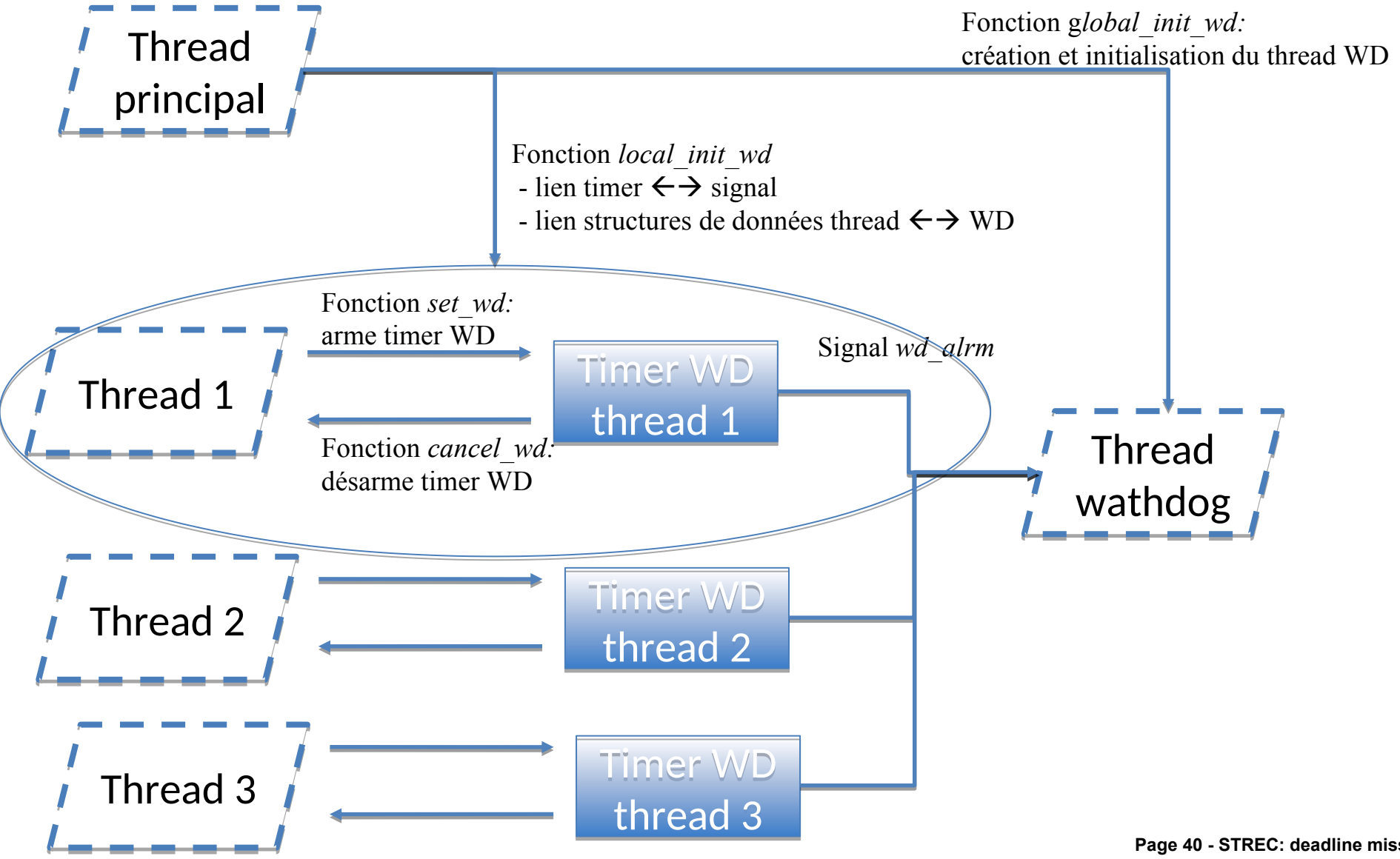


RT-POSIX: API et design pattern fourni

- thread_dipactch.c => pattern tâches temps réel
- Thread_part.c => pattern comm et synchro
- Wdlib.c => fonction de base pour initialisation et « animation » du watchdog.
- Coté lib posix et signaux :

Traitement et réception : sigwaitinfo, sigemptyset, et sigaddset.
Envoi : timer_create, timer_settime, timer_delete,

Architecture code du TP



Conclusion

Correspondance du vocabulaire tolérance aux fautes et systèmes temps réel

- Remarque n° 1 : la structure d'un STR n'inclus pas la tolérance aux fautes => doit être intégrée aux tâches
- Remarque n°2 : garanties de temps de réponse pour 1 structure => introduction des modes opérationnel et du changement de mode.