GRENOBLE INP UGA Phelma

École nationale supérieure de physique, électronique, matériaux

Introduction aux systèmes d'exploitation temps réel

Partie 1 – temps réel sous Linux

SEOC 2A

N castagné, 2024-25 - version 08

Le module ISETR

Partie 1 : introduction au temps réel non critique : *contexte du multimédia* **[NOUVEAU]**

- **Nicolas Castagné**
- Avant les vacances de février
- 5 séances : CM, 2 * TP (2 groupes) ; CM, TP sur plateforme technologique (4 groupes)
- **Objectif général :**
  - Introduction aux systèmes ~~d'exploitation~~ temps-réel
  - Focus sur le contexte des boucles temps réel réactif multimédia/RV, sur système d'exploitation *general purpose* (en C++)
  - *(auto-formation au C++)*

Partie 2 : introduction aux Systèmes d'Exploitations temps réel.

- **Stéphane Mancini**
- Après les vacances de février
- 11 séances : 7 CM, 2 BE de 4h (2 groupes)
- **Objectif général :**
  - Introduction aux **systèmes d'exploitation temps réel** (notamment embarqués)
  - Focus sur le temps réel sur système embarqué, sous **FreeRTOS**

Evaluation : Examen 2h. TP non noté, mais des questions peuvent concerner les TPs.

Le module ISETR

Partie 1 : introduction au temps réel non critique : *contexte du multimédia* **[NOUVEAU]**

- CM1 :
  - Système temps réel ? Temps réel multimédia ?
  - Introduction BE qui suit
- BE#1 et #2 : 2 groupes
  - Comprendre les principes de l'audio sur PC Linux ; notion de flux audio et de traitement par buffer
  - Programmer un pipeline audio (lecture de fichiers sonores) sous Linux
  - Se confronter au C++
- CM2 :
  - Synthèse des BE prédents
  - Introduction BE qui suit
- BE#3 : 4 groupes
  - *reverse engineering* de stations temps réel synchrone de simulation multisensorielles geste-son-image pour la création artistique, sur la plateforme Art Science Technologie de G-INP.
  - Découverte de l'axe de recherche et des applications.
  - *RV sur la plateforme Art Science Technologie à côté de GI (précisions à venir)*

Accès aux supports : https://gitlab.ensimag.fr/castagnn/seoc_2a_isetr_partie1

Le module ISETR

Aujourd'hui :

I. Vous avez dit « systèmes temps réel » ?

II. Flux audio temps réel sur OS standards non temps réel

III. Objectif des séances TP #1 et #2

IV. Mes premiers pas en C++ *(ou « de Java et du langage C au C++ »)*

I. Vous avez dit « systèmes temps réel » ?

Comment définir le (calcul) temps réel ?

Un système temps réel, c'est quoi ?

Un OS temps réel, c'est quoi ?

Systèmes temps réel ?

A minima, on peut déjà distinguer :

- Les Systèmes (et logiciels) **Transformationnels** :

Système qui lit ses données d'entrée lors de son démarrage, qui effectue des calculs, fournit ses sorties (aussi vite que possible), puis meurt.

- Les systèmes **Interactifs** et **Réactifs**.

Système qui ne s'arrête « jamais » et interagissent en permanence avec l'environnement.

Systèmes temps réel ?

- System which « controls an environment by receiving data, processing them, and returning the results **sufficiently quickly** to affect the environment at that **time** »
  James Martin in "Programming Real-time Computer Systems". Englewood Cliffs, NJ: Prentice-Hall Inc. p. 4, 1965

- « Peut être qualifiée de temps réel (ou "temps contraint", ou "réactif") toute application dont le déroulement est **assujetti à l'évolution dynamique d'un procédé extérieur** qui lui est connecté et dont il doit contrôler le comportement ».
  G.D.R-T.R CNRS. Le temps réel. Technique et Science Informatiques, 1988

- System in which « the correctness [...] depends **not only** on the logical result of computation, but also on the **time at which the results are generated** »
  from Jack Stankovic, in "Misconceptions of Real-Time Computing", IEEE Computer, 21(10):10--19, 1988

- Le comportement d'un système temps réel est valide si le résultat d'un traitement :
  - Est bien sûr conforme à celui attendu
    (identique à celui qui aurait été obtenu sans la contrainte d'exécution en temps-réel)
  - Et, plus spécifiquement, **est rendu disponible avant une date limite (deadline)**

Systèmes temps réel ?

- La notion de systèmes temps réel ne renvoie **pas** à la notion du calcul rapide ou « haute performance ». *Exécution temps réel* ne veut **pas** dire *exécution rapide* !

- Calcul haute performance (*high performance computing*)
  - Minimisation du temps ***moyen*** nécessaire pour exécuter un ensemble de tâches
  - Pas de contrainte de *deadline*
  - Un super-calculateur n'est (a priori) pas un système temps-réel
- Calcul temps réel
  - **Garantie** d'une réponse **en temps contraint** (*deadline*)
  - Le temps de réponse varie en fonction des contraintes du système physique en interaction
  - Il existe des systèmes temps réel temporalité lente !

- **Néanmoins**, les systèmes temps-réel bénéficient bien souvent des techniques du calcul haute performance (par ex. : *multi-core processing*)

Systèmes temps réel ?

- **Notion de criticité des contraintes temps réel**

Contrainte dure – « *hard real time* »

Ne pas respecter une deadline équivaut à un **dysfonctionnement majeur** ou empêche le système de continuer de fonctionner.
… et peut mener à des conséquences désastreuses !

Exemples de contextes :
- Contrôle de vol
- Contrôle d'un bras robotique
- Système de freinage ABS
- Interaction gestuelle à retour d'effort
- Réacteur nucléaire
- …

Contrainte *soft* – « *soft real time* »

L'utilité du résultat d'un traitement **diminue** lorsque qu'on s'écarte de la deadline, mais le système continue de fonctionner (avec une dégradation de qualité)

Exemples de contextes :
- Application interactives sur PC
- Système de réservation de billet ou ticket
- *Streaming* musical ou vidéo, réalité virtuelle
- Requête Internet
- Communications téléphoniques, visio-conférence
- …

Systèmes temps réel ?

- **Notion de criticité des contraintes temps réel**

Contrainte dure – « *hard real time* »

Nécessite (souvent) une implantation sur dispositif de calcul **embarqué**
Nécessite (souvent) un **système d'exploitation temps réel**
Forte interaction matériel – logiciel

La correction du système doit être strictement vérifiée avant sa mise en œuvre
Mise en œuvre de :

- Validation **statistique**, en jouant sur les conditions d'entrée et d'exécution (charge, etc.)
- Validation **formelle** en prenant en compte l'ensemble des composant matériels et logiciels [difficile !]

Contrainte *soft* – « *soft real time* »

Implémentation (souvent) envisageable sur des **plateformes a priori non dédiées au temps-réel** : PC, smartphone, système d'exploitation usuel « *general purpose* »,

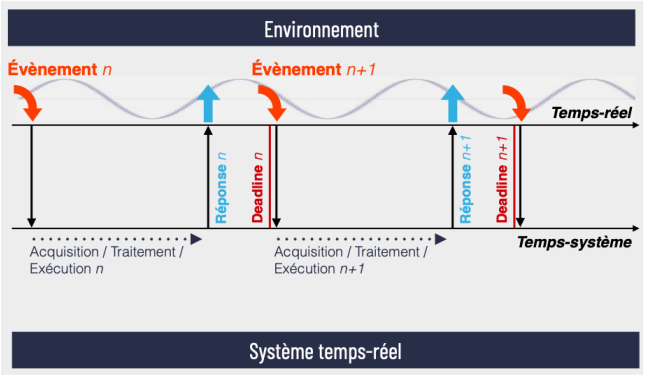
Une validation statistique suffit en général pour ce type de cas

Systèmes temps réel ?

- **Notion de modèle synchrone vs. asynchrone**

Système temps réel *synchrone*

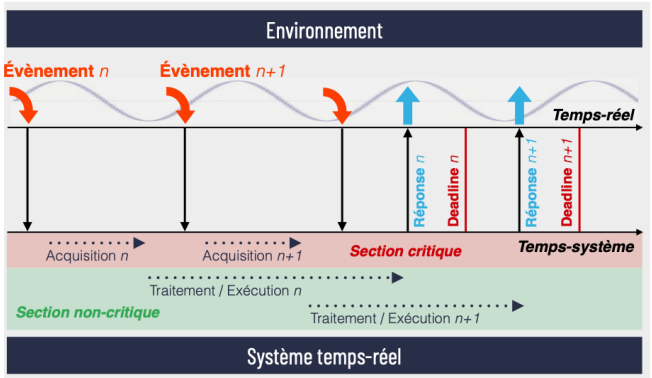
Temporalité basée sur une **horloge, souvent matérielle** qui cadence les traitements.



Deadline = date du prochain top horloge
Durée (Acquisition + Traitement + Sortie) < Durée entre N top horloge
Exemples : traitement audio ; interaction à retour d'effort…

Système temps réel *évènementiel*

La réaction du système est déclenché en réaction à un **évènement** détecté dans l'environnement.



Deadlines non imposées par la cadence des événements.
Les temps de réponses attendus peuvent varier suivant la nature de l'évènement.
Exemples : GUI interactive ; échanges client/serveur sur Internet…

Systèmes temps réel ?

- **Notion de modèle synchrone vs. asynchrone**

Système temps réel *synchrone*

A chaque *top horloge* faire
- Lecture de la mémoire des entrées
- Calcul des ordres à envoyer au procédé
- Émission des ordres en attente de sortie (parfois au top horloge suivant)

Remarque :
- Lorsque la ***latence*** entre entrée et sortie est limitée à un unique échantillon, on parle de *single step delay*
- Mais la ***latence*** est souvent plus importante, de plusieurs échantillon.
- *Exemple : latence entre entrée et sortie audio dans un système de traitement audio*

Système temps réel *évènementiel*

A chaque *interruption* ou *évènement* faire
- Lecture de l'information arrivée
- Activation du traitement correspondant
- Émission des ordres issus de ce traitement

Mais :
- que faire si une interruption survient alors que le système est en train de traiter une interruption précédente ?
- ➢ notion de priorité des interruptions
- ➢ notion de "tâche" associée à un plusieurs types d'interruptions
- ➢ mécanisme de préemption et de reprise de tâche
- ➢ gestion de l'exécution concurrente des tâches (ordonnancement), souvent avec priorités entre tâches

Systèmes **d'exploitation** et temps réel ?

Système d'exploitation *general purpose*

Linux, Windows, MacOS, Android…

Le système d'exploitation cherche à optimiser les temps de réponse **moyens** aux évènements (par exemple aux interruptions système) de (nombreuses) tâches en cours d'exécution, mais ces temps de réponse sont **non déterministes.**

Pour implanter une boucle temps réel, on s'appuie sur la puissance générale de la station de calcul … en espérant qu'elle soit suffisante et en prenant de la marge…

BE#1 et #2 de cette première partie du cours, avec N Castagné

Système d'exploitation *general purpose* amélioré

RTLinux RTAI, Windows IoT Entreprise, RedHawk …

Ajoutent aux systèmes d'exploitation *general purpose* usuels divers mécanismes qui visent à améliorer la réactivité et le déterminisme temporel.
Exemples :

- politiques d'ordonnancement révisées et plus paramétrables (e.g. niveaux de priorité)
- isolation de processeurs ou cœurs
- niveaux de priorité entre tâches bloquées sur un mutex
- déterminisme du temps de réponse à une une interruption système
- …

BE#3 de cette première partie du cours, avec N Castagné

Système d'exploitation temps réel

FreeRTOS, Azure Real-Time système, VxWorks, QNX …

Permet de prévoir et garantir le déterminisme du temps d'exécution d'une tâche TR (déterminisme temporel).
Mais n'offre pas par défaut tous les services usuels des OS *general purpose* tel que le partage de l'accès aux ressources.

- Réglage fin de l'ordonnanceur.
- Préemptibilité. **Prévisibilité**. Etc.
- Plutôt pour des processeurs embarqués.
- **Forte imbrication matériel / logiciel.**

Cours et BEs de la seconde partie du cours, avec S Mancini

II. Flux audio temps réel sur OS standards (non temps réel)

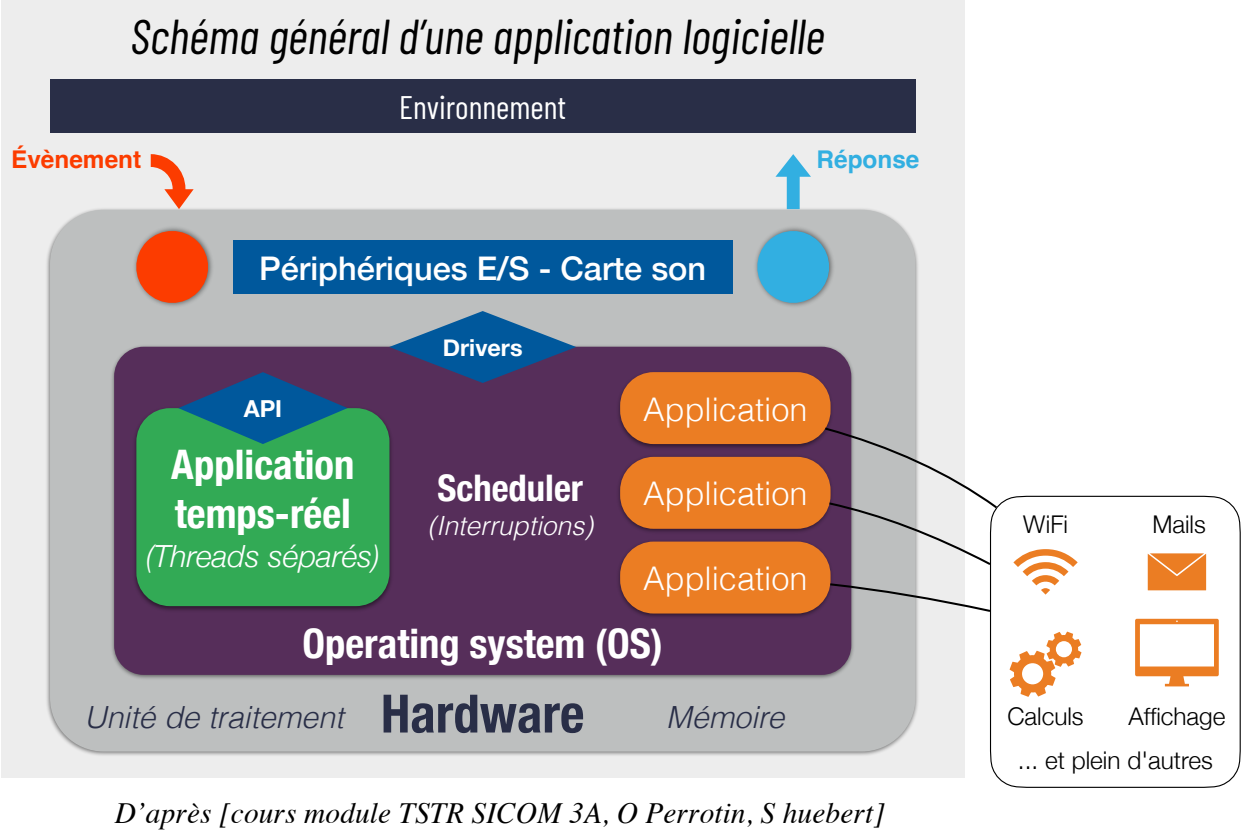
Du matériel au logiciel

L'application dans son ensemble doit exécuter des traitements diversifiés et possiblement complexe (IHM, calculs, réseau, affichage graphique…)

La couche audio temps réel est gérée au moyen d'une API dédiée.

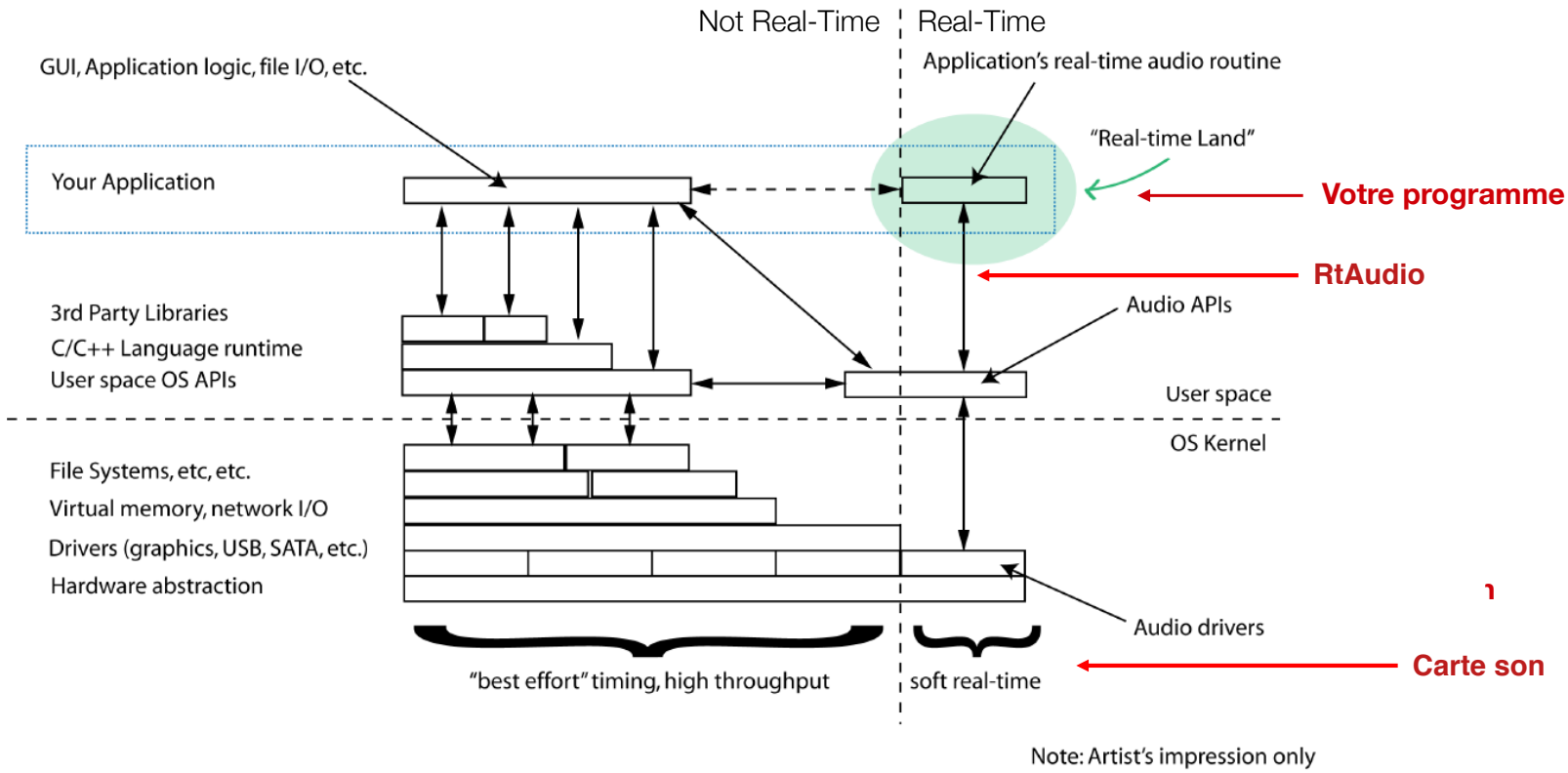
Les traitements audio temps réel sont exécutés au sein d'un *thread* dédié, initié par l'API audio, et doté d'une priorité haute dans l'OS.

Rq : il survient parfois des problèmes au lancement du flux audio (*thread)*, le temps qu'il soit correctement pris en compte par le *scheduler et que les accès mémoire se stabilisent.*



D'après [cours module TSTR SICOM 3A, O Perrotin, S huebert]

Du matériel au logiciel



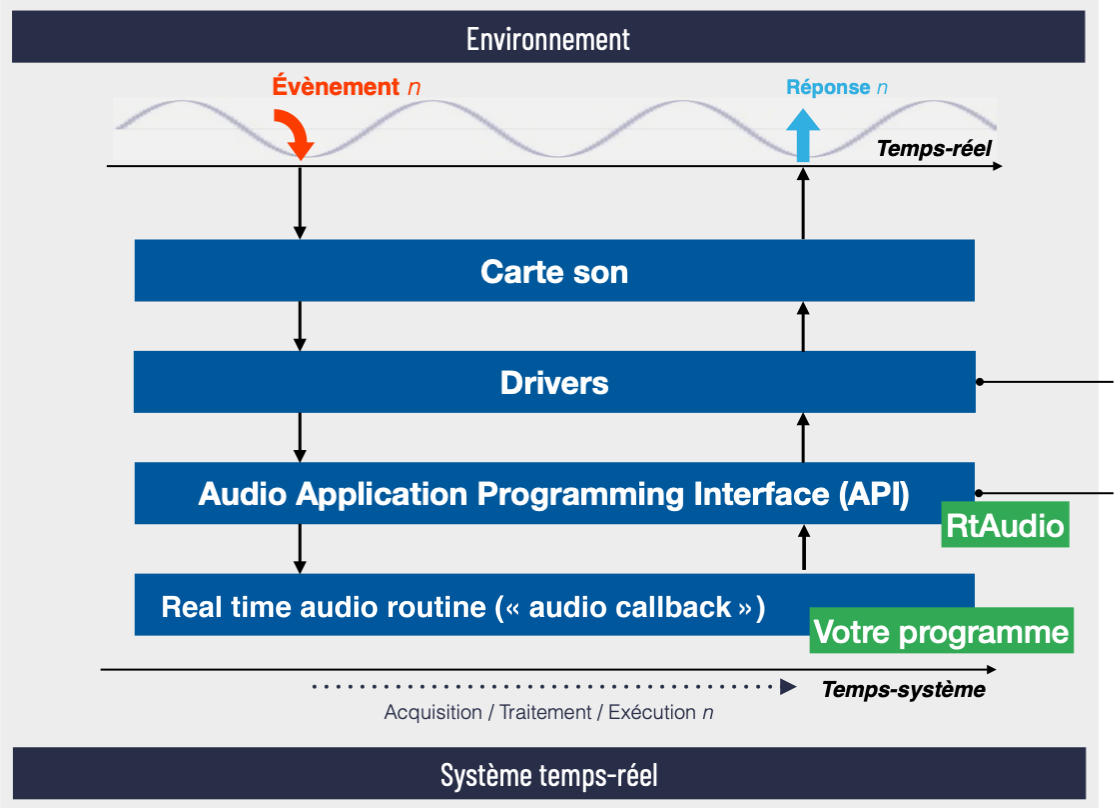
D'après [cours module TSTR SICOM 3A, O Perrotin, S Huebert]
Voir aussi http://www.rossbencina.com/code/real-time-audio-programming-101-time-waits-for-nothing

Du matériel au logiciel

- Le système d'exploitation fournit une API audio propriétaire
  - Windows : Audiograph
  - MacOS : CoreAudio
  - Linux : Alsa, OSS, Jack
  - Etc.

- Il existe des API portables (*cross-platform*) construites au-dessus des précédentes :
  - **RTAudio**, portaudio, etc

Du matériel au logiciel

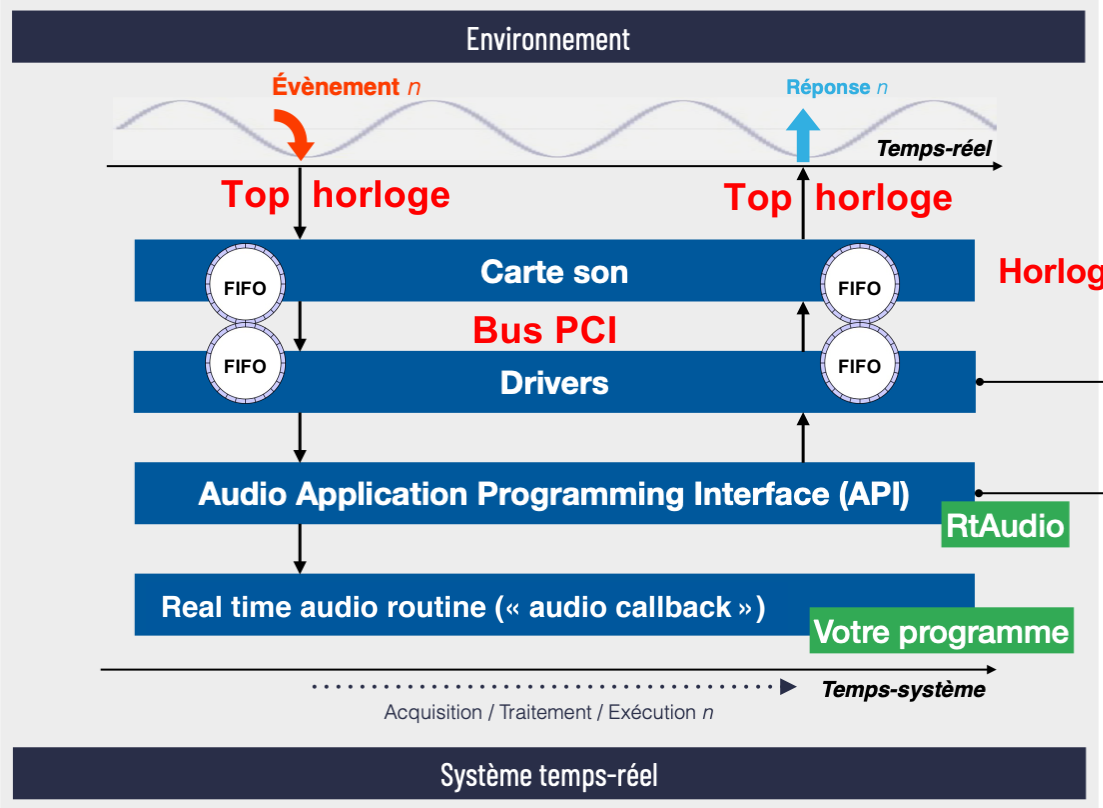


Dépend de l'OS

Chargé au sein du noyau de l'OS

D'après [cours module TSTR SICOM 3A, O Perrotin, S Huebert]

Du matériel au logiciel



Les échantillons audio transitent par plusieurs FIFO
et sont traités par paquets (buffers).

Horloge hardware à fréquence audio, ~40 KHz

Dépend de l'OS

Chargé au sein du noyau de l'OS

D'après [cours module TSTR SICOM 3A, O Perrotin, S Huebert]

Du matériel au logiciel

- **La fonction callback audio**, qui est écrite par le développeur est chargée des traitement des échantillons audio en provenance / à destination de la carte son.
- Elle est appelée (+/-) régulièrement par l'API audio, dans le contexte (pile d'appel) du *thread* temps réel, lorsqu'un paquet de N d'échantillons est devenu disponible en entrée (et/ou doit être fournis en direction de la sortie).
- La fonction callback a la responsabilité de traiter ***intégralement*** le paquet de N échantillons se présentant en entrée et/ou de fournir ***intégralement*** le paquet de N échantillons en sortie, ce **avant qu'elle soit rappelée** (*deadline*).
- Pendant que le *callback audio* traite un paquet de N échantillons, d'autres échantillons sont accumulés par le *hardware* et l'API audio, pour préparer le paquet suivant (réciproquement : sont acheminés vers la carte audio).
- **Il résulte inévitablement de l'ensemble une certaine *latence* incompressible entre entrée et sortie, qui dépend notamment de la taille N des buffers.**
    - De ~50 ms avec les PC et cartes audio les performants (environ 100 échantillons)
    - Souvent bien plus, > 20 ms (1000 échantillons)
- *A titre de comparaison, les systèmes de traitement audio professionnels embarqués peuvent fonctionner avec des latences de moins de 1 ms !*

Du matériel au logiciel

Modèle producteur-consommateur

Illustration dans le cas de l'enregistrement audio

Les échantillons audio transitent par plusieurs FIFO et sont traités par paquets (buffers).
La taille N des paquets est (+/-) paramétrable dans l'API audio.

Elle doit être **suffisante** pour assurer raisonnablement la stabilité temporelle du traitement et éviter *over-run* et *under-run*.

- **Over run** : L'application ne lit pas les données en entrée assez rapidement, le buffer est écrasé avec de nouvelles données
- **Under run** : L'application n'écrit pas les données en sortie assez rapidement, certains échantillons écrits sur les CNA sont nuls, voire aléatoires.

A l'inverse, elle doit être **suffisamment faible** pour obtenir une latence globale raisonnable.

Du matériel au logiciel

- Si la fonction callback audio n'est pas en situation de traiter tous les N échantillons se présentant en entrée (ou réciproquement de fournir tous les N échantillons attendus en sortie), il se produit des artefacts dans le signal audio *(**glitchs**) :*
  - Clics ou pauses ou bruit dans le signal acquis en entrée
  - Clics ou pauses ou bruit dans le signal émis en sortie sur les hauts parleurs
  - Etc.
- Pour éviter autant que possible ces glitchs, i.e. assurer une exécution *en temps* du callback audio, **il faut éviter *tous les appels systèmes* et tout risque *d'inversion de priorité.***
- **Ainsi, la fonction callback audio (*i.e.* le *thread* audio) ne doit PAS, par exemple :**
  - **Ouvrir/fermer un fichier ou faire une lecture ou écriture sur disque ou sur le réseau**
  - **Faire d'allocation mémoire**
  - **Acquérir un verrou (mutex, sémaphore…) sur une section critique**
  - **Allouer ou libérer de la mémoire**
  - **Créer un *thread***
  - **Travailler avec le réseau**
  - **Etc.**
- Ni bien sur exécuter du code qui ferait ce genre de chose…

Oui, mais…

Sur OS standard (Linux, MacOS, Fenêtre…), toutefois,

quel que soient les précautions prises dans le paramétrage et dans le code,
nos efforts seront bien précaires…

Il demeure par principe impossible
de garantir à 100% le déterminisme temporel des boucles TR

donc en particulier, pour les flux audio temps réel, l'absence de glitch

Introduction à RTAudio

RTAudio est une API audio portable écrite en C++, implantée pour l'essentiel dans une classe C++ nommée `RTAudio`. **Quatre méthodes essentielles de la classe `RTAudio` sont :**

```
// Opens a stream with the specified parameters.
RtAudioErrorType     openStream (RtAudio::StreamParameters *outputParameters,
                                 RtAudio::StreamParameters *inputParameters,
                                 RtAudioFormat format,
                                 unsigned int sampleRate,
                                 unsigned int *bufferFrames,
                                 RtAudioCallback callback,
                                 void *userData=NULL,
                                 RtAudio::StreamOptions *options=NULL);

// Starts the audio stream.
RtAudioErrorType     startStream (void);

// Stop the stream, allowing any samples remaining in the output queue to be played.
RtAudioErrorType     stopStream (void);

// Closes a stream and frees any associated stream memory.
void     closeStream (void);
```

Introduction à RTAudio

Le paramètre `callback` de la méthode `openStream` est un *pointeur de fonction*.
Il permet de préciser à RTAudio votre fonction *callback audio* à utiliser.

Le prototype de la fonction callback est imposé par l'API.
Voici un exemple de fonction callback :

```
// Audio callback fundtion. Buffers are interleaved
int my_super_audio_callback(  void *outputBuffer,
                              void *inputBuffer,
                              unsigned int nBufferFrames,
                              double streamTime,
                              RtAudioStreamStatus status,
                              void *data ) {
      // ICI mon code de traitement audio !!!
}
```

III. Objectif des séances TP #1 et #2

Objectif des séances TP #1 et #2

- Objectif général : lire simultanément plusieurs fichiers audio monophoniques et les mixer (volume, panoramique, …)
  - En programmant en C++… *donc en se confrontant à ce langage !*
  - En utilisant RTAudio et d'autres ressources fournies
  - En progressant par étape

- Etape 1 : prendre en main, comprendre, tester un exemple basique de RTAudio
- Etape 2 : l'adapter pour jouer deux sons sinusoïdaux de fréquence réglable, un par canal audio (droite/gauche)
- Etape 3 : charger *tous* les échantillons de plusieurs fichiers audio en mémoire, puis les rendre accessible depuis la fonction *callback audio* qui va les envoyer progressivement sur la carte son
- Etape 4 : monter un mécanisme de type producteur -> consommateur entre lecture des fichiers par petits bouts par le *thread* principal (fonction main, partie non temps réel de l'application), et envoi sur la carte son par le *thread* audio (fonction callback audio)
- Etape 5 : prise en main et compréhension du code (fourni) d'une petite **interface graphique de lecture et mixage de fichiers audio** (utilisant la librairie C++ **Qt**)

IV. Mes premiers pas en C++

(ou « de Java et du langage C au C++ »)

Voir le fichier Java2Cpp_*.html dans le dépôt git du module