Introduction

L'objectif de ce laboratoire est d'acquérir des données audios et vidéo puis de les traiter à l'aide d'une FFT pour l'audio et d'une convolution pour la vidéo. Pour cela nous devront mettre en place un système de messagerie à l'aide de la librairie xenomai pour l'audio, puis pour la vidéo nous devront utiliser un système d'event pour la synchronisation.

Audio

Conception

Pour la conception de la partie audio de notre système, nous avons utilisé le service de message queue fourni par xenomai. Voici ci-dessous les explications de mise en place de ce système

Message entre acquisition_task et processing_task (acquisition_queue)

Pour qu'une queue soit valide il faut tout d'abord en déclarer une à l'aide de RT_QUEUE, ce dernier fonctionne plus ou moins de la même manière qu'un file descriptor mais pour les queues. Maintenant cela fait, il nous faut créer la queue à l'aide de rt_queue_create, les 2 paramètre important à définir sont la pool size ainsi que la limite maximum de message, étant donné que l'on envoie des échantillons de 512 par 512 et qu'il nous en faut plus de 16'384 il nous faut un pool pas mal grand, afin d'éviter un quelconque problème dans la queue un pool de 1000* FIFO_SIZE * NB_CHAN a été mis nous avions essayé de mettre un pool de 35 (car 16'384 / 512 = 32, puis on ajoute une marge de pour avoir 35), mais le communication ne se faisait pas correctement nous avons donc décider de prendre un grand pool afin d'éviter de quelconque problème, en effet cela n'est pas très optimal d'avoir une pool aussi grande mais pour cette expérience nous avons préféré ne pas perdre trop de temps dessus (malheureusement le temps n'est pas en notre faveur). Cela sera de même pour la limite de message nous utilisons Q_UNLIMITED fourni par rt_queue qui nous permet d'avoir un nombre illimité de message dans la queue (bien évidemment selon la limite de stockage du matériel).

Une fois la queue créer nous devons maintenant allouer le message (buffer) à envoyer pour se faire nous allons utiliser rt_queue_alloc qui nous permet de définir la taille max que prendra le message dans la pool interne comme chaque lecture de l'audio fais 512 bytes on va donc allouer 512 bytes (FIFO_SIZE * NB_CHAN).

Nous pouvons donc maintenant lire les données audios et puis les envoyer au travers de la queue à l'aide de rt_queue_send, se dernier nécessite le message ainsi que la taille du message pour la taille du message elle nous est fournie par read_samples lorsque l'on lit l'entrée audio il y a aussi de dernier paramètre à réglé qui peut être soit Q_BROADCAST qui envoie le message à tous les destinataire faisant partie de cette

queue et le message sera effacé une fois que tous les destinataire l'ont reçu, soit Q_URGENT qui envoie le message dans la queue avec un ordonnancement LIFO ou soit Q_NORMAL qui envoie le message dans la queue avec un ordonnancement FIFO, on utilisera donc ce dernier car il faut que les message soit traiter dans leur ordre d'arrivée et que l'on a qu'un seul destinataire.

Maintenant que les données sont envoyées processing_task peut les recevoir. Pour qu'il puisse les recevoir nous devons utiliser la méthode rt_queue_receive, se dernier consiste en 3 paramètre, le premier et l'adresse de la queue en question, le second est là pour récupérer le message et l'affecter à un buffer local, et le dernier le paramètre définit s'il on veut attendre de recevoir le message (TM_INFINITE) ou continuer à exécuter la suite du code sans attendre d'avoir reçu le message (TM_NONBLOCK) dans notre cas nous allons utiliser TM_INFINITE car nous avons besoin du message pour effectuer la suite.

Une fois le message affecter à une variable locale à l'aide memcpy, il nous faut maintenant dire explicitement à l'expéditeur, pour se faire nous devons utiliser rt_queue_free qui nous permettra de libérer le message de la pool de la queue, il lui faut donc comme paramètre la queue en question ainsi que le message à libérer, l'expéditeur pourra donc envoyer le message suivant sans aucun souci. Si le système s'arrête de manière abrupte à cause d'une erreur ou que le programme à fini de s'exécuter il est important de libérer la mémoire utilisée par la queue en utilisant rt_queue _free pour libérer les messages ainsi que rt_queue_delete pour supprimer la queue.

Message entre processing_task et log_task (processing_queue)

Le système de queue entre processing_task et log log_task est exactement la m

Vidéo

Conception

Pour la conception de la partie vidéo, nous avons 3 tâches à faire fonctionner qui sont la tâche vidéo normal, la tâche vidéo avec le filtre grayscale ainsi que la tâche vidéo avec le filtre de convolution. Nous avons opté de partir sur une seule et unique tâche et que le filtre appliqué serait uniquement choisi en fonction des switches. Ce choix a été motivé par une approche de simplification du code et de réduction des changements de contexte entre les différentes tâches vidéo et une synchronisation simplifiée, ce qui peut réduire le temps de traitement, tout en garantissant une gestion efficace des événements. Les buffers nécessaires pour les différentes étapes de traitement sont alloués dans le `main`. Cela permet d'éviter de faire des grosses allocations de mémoire

directement dans la tâche et de faire des appels système dans une tâche xenomai. On aurait pu utiliser aussi les fonctions fournies par l'api de xenomai pour l'allocation mais nous somme rester sur notre choix de tout faire dans le main. Pour synchroniser les tâches, nous avons dû utiliser le concept des events à la place des queues que nous avons utilisé pour les tâches audios.

Les rt_event sont des mécanismes de synchronisation fournis par Xenomai pour permettre une communication efficace entre les tâches en temps réel. Ils fonctionnent comme des drapeaux (flags) que les tâches peuvent attendre ou signaler pour coordonner leur exécution.

L'utilisation des rt_event permet de coordonner les tâches sans qu'elles aient à vérifier constamment l'état des conditions, ce qui réduit la charge sur le CPU et améliore la réactivité de notre système. Dans notre cas, étant donné que nous avons fait le choix de faire une seule tâche, rt event nous permet de signaler à la tâche de fonctionner que quand le switch numéro 1 est activé et de se mettre en état de sommeil si ce n'est pas le cas. Comme les filtres vidéo doivent fonctionner uniquement si la tâche vidéo est active, alors on n'a pas besoins de faire des cas en plus pour ces tâches spécifique.

Caractérisation des tâches et d'ordonnançabilité

Caractérisation Audio

Acquisition task

Pour la tâche d'acquisition une période à respecter nous est donné dans le code, après calcul ce dernier doit faire 2.5 ms on a donc un délai critique de 2.5 ms pour l'acquisition audio. Si nous mesurons maintenant le temps d'exécution de l'acquisition dans la while (sans la partie vidéo) nous obtenons les résultats suivants (mesure en ms) :

```
------------------------------------------- summary1.c -----
 Total of 909 values
   Minimum  = 0.018030 (position = 180)
   Maximum  = 0.132210 (position = 6)
   Sum      = 98.766460
   Mean     = 0.108654
   Variance = 0.000283
   Std Dev  = 0.016819
   CoV      = 0.154795
----------------------------------------------------------------
```

On peut voir dans cette mesure que le coût de notre tâche est généralement de 0.108654ms avec la pire valeur atteinte étant de 0.132210 ms

S'il on fait maintenant la mesure du temps que rt_task_wait_period attend on a les résultats suivants :

```
---------------------------------------------- summary1.c -----
 Total of 1474 values
  Minimum  = 0.132880 (position = 143)
  Maximum  = 31.282160 (position = 1258)
  Sum      = 4675.603500
  Mean     = 3.172051
  Variance = 22.131454
  Std Dev  = 4.704408
  CoV      = 1.483081
---------------------------------------------------------------
```

Quelques mesures ci-dessous pour mieux comprendre (ces mesures se répète) :

31.213000
0.135210
1.162530
2.484120
2.491910
2.493630
2.495680
2.491870
2.493860
2.494610
2.495170
2.493050
2.493920
2.494410
2.496020
2.492010
2.495180
2.494280
2.495570
2.492450
2.494230
2.494500
2.495350
2.493350
2.494320
2.494560
2.495620
2.493470
2.492260
2.493750
2.495710
2.492610

2.493740
2.494370
2.495950
2.493480
On remarque dans les mesures que la période est en général respectée et ne dépasse pas les 2.5ms, mais qu'après une trentaine de mesures, ce temps est dépassé, cela est dû à processing_task qui préempte acquisition task car il a obtenu toutes les valeurs qu'il avait besoin pour effectuer la FFT, on verra par la suite que processing_task prend environ 30ms à s'effectuer.

Nous allons maintenant mesurer le temps que cela prend pour acquérir toutes les données nécessaires à processing task, cela nous donnera le cout de notre tâche d'acquisition pour qu'il collecte toutes les données nécessaires :

```
---------------------------------------------- summary1.c -----
 Total of 1000 values
  Minimum  = 82.608850 (position = 513)
  Maximum  = 85.098500 (position = 0)
  Sum      = 82621.952040
  Mean     = 82.621952
  Variance = 0.006148
  Std Dev  = 0.078407
  CoV      = 0.000949
---------------------------------------------------------------
```

On peut donc voir avec ces mesures qu'il nous faut approximativement 82.621952ms pour que toute les données soient récolté, ce qui est plutôt logique en sachant que l'on envoi 512 donnéee par 512 et qu'il nous en faut 16'384, on aura alors besoin d'environ 32 acquisition de 2.5ms, ce qui nous fait donc un temps de 80ms en théorie, la raison pour laquelle nous avons un peu plus est dû au fait que même s'il on demande à la méthode read_samples de lire 512 bytes, il peut en lire moins et donc nous aurons besoin de quelque acquisition en plus.

Processing task

Pour la tâche de traitement il n'y a pas vraiment de période à respecter, il nous faut principalement déterminer son pire temps d'exécution, voici donc les mesures :

```
---------------------------------------------- summary1.c -----
 Total of 1000 values
  Minimum  = 30.981380 (position = 421)
  Maximum  = 31.110240 (position = 253)
  Sum      = 31038.595010
  Mean     = 31.038595
  Variance = 0.000335
  Std Dev  = 0.018303
  CoV      = 0.000590
---------------------------------------------------------------
```

On voit donc avec ces résultats que le pire temps d'exécution de processing_task est de 31.110240 ms

Log task

Nous allons maintenant calculer le pire temps d'exécution pour la tâche des logs, pour que cette tâche puisse s'exécuter il faut que processing task aient fait un traitement on aura donc une addition du temps qu'il faut pour acquérir les données ains que de les traiter au niveau du temps d'exécution, voici les mesures :

```
------------------------------------------- summary1.c -----
 Total of 1000 values
   Minimum  = 109.739580 (position = 0)
   Maximum  = 115.093690 (position = 586)
   Sum      = 114970.581890
   Mean     = 114.970582
   Variance = 0.028485
   Std Dev  = 0.168775
   CoV      = 0.001468
-----------------------------------------------------------------
```

Les tâches sont donc toutes exécuter dans un intervalle de temps d'environ 114.970582ms avec le pire cas d'exécution qui est de 115.093690ms

Toute les tâches ensemble

Voici le temps d'exécution de toutes les tâches audios ensemble :

```
------------------------------------------- summary1.c -----
 Total of 996 values
   Minimum  = 113.845440 (position = 511)
   Maximum  = 116.496820 (position = 0)
   Sum      = 113460.919070
   Mean     = 113.916585
   Variance = 0.007346
   Std Dev  = 0.085710
   CoV      = 0.000752
-----------------------------------------------------------------
```

On remarquera que ces mesures sont très similaires à ceux du log task, ce qui est normal comme expliqué plus haut log task s'exécute uniquement quand processing task a fini de traiter ces tâches et l'on a vu dans la partie acquisition task que pour récolter toutes les données on mesurait en moyenne 82.621952ms puis pour le temps que prenait un traitement de toutes ces donnée nous avions en moyenne 31.038595 se qui fait un total de 113.660547ms pour que toute les tâche aient été exécutée au moins une fois, cette valeur correspond bel et bien à ce que nous obtenons dans cette mesure.

Caractérisation Vidéo

Pour la caractérisation de la vidéo nous avons calculer le temps de son exécution. On a une tâche vidéo qui doit s'exécuter à une fréquence de 15HZ soit une période de 66.66 ms. Avec les 1000 mesures par tâche que nous avons prises, on peut voir que pour que la tâche vidéo normal s'exécute dans les temps avec une moyenne de 66.5 ms et quelques cas d'exécution critique où on a un temps maximum de 77 ms pour la tâche de vidéo normal. Pour la tâche vidéo de greyscale, nos données montre les mêmes résultats que pour la tâche vidéo normal. C'est à dire qu'elle a aussi une moyenne de 66.5ms et un temps d'exécution critique (max) à 77.5 ms.

Cependant, les calculs nécessaires pour exécuter la tâche vidéo de convolution sont beaucoup trop compliqué pour que la vidéo puisse se faire dans les temps voulues qui sont est de 66.6 ms. A chaque exécution, la tâche est en retard de plus ou moins 40ms. On voit effectivement qu'elle prend en moyenne 106.6 ms avec une exécution maximum de 117.6 ms. Cela montre que si on veut que la tâche vidéo en général soit ordonnançable, il faut la faire sans la tâche avec le filtre de convolution sinon elle ne sera pas ordonnançable.

Tâche vidéo normal :

---------- summary1.c -----

Total of 1000values
Minimum = 18133950.000000 (position = 1)
Maximum = 77596490.000000 (position = 899)
Sum = 66502302420.000000
Mean = 66568871.291291
Variance = 5373449218116.500000
Std Dev = 2318070.149524
CoV = 0.034822

Tâche vidéo Greyscale:

---------- summary1.c -----

Total of 1000 values
Minimum = 23236010.000000 (position = 1)
Maximum = 77593460.000000 (position = 599)
Sum = 66502771690.000000
Mean = 66569341.031031

```
   Variance = 4557550162243.500000
   Std Dev  = 2134841.952521
   CoV      = 0.032069
---------------------------------------------------------------
```

Tâche vidéo Convolution:

```
---------------------------------------------- summary1.c -----
 Total of 1000 values
   Minimum  = 100915840.000000 (position = 0)
   Maximum  = 117630400.000000 (position = 900)
   Sum      = 106695319580.000000
   Mean     = 106695319.580000
   Variance = 466275714430.000000
   Std Dev  = 682843.843371
   CoV      = 0.006400
---------------------------------------------------------------
```

Mise en œuvre et condition d'ordonnançabilité

En se basant sur les mesures obtenues nous allons regarder pour chaque tâche quelle sont leur coûts ainsi que leur période/échéance

Audio Task

Acquisition task

Comme expliqué précédemment la période nous est déjà fourni, ce dernier est de 2.5ms, quant au cout de la tâche nous allons prendre le pire temps qui est de 0.132210ms

Processing task

Pour le processing task comme il n'a pas de période défini nous allons nous servir du pire temps d'exécution dans notre cas il vaudra 31.110240 et le cout moyen est de 31.038595

Log task

La tâche des logs ayant le même temps d'exécution que lorsque l'on a calculé lorsque toute les tâche ensemble, cela signifie que son coût est infime et peut donc être négligé dans cette partie

Video task

La période de notre tâche vidéo nous est aussi fourni, ce dernier est de 66.6ms et notre tâche vidéo avec la convolution à un cout de 106695319.580000ns

Condition Ordonnançabilité - Calculs

Pour vérifier l'ordonnançabilité de nos tâches, on peut la calculer avec la condition suffisante d'ordonnançabilité de Liu et Layland.

Pour rappel, le calcul est le suivant :

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq U_{lub_{RM}}(n) = n(2^{\frac{1}{n}} - 1) = n(\sqrt[n]{2} - 1)$$

Nous avons donc deux tâches pour l'audio, qui sont l'acquisition et le processing. Il y a aussi la tâche vidéo que nous n'allons pas prendre en compte pour le calcul.

Le calcul va dont être fait pour les deux tâches audios, il faut donc que la somme de leur taux d'utilisation du processeur soit inférieure à 0.828.

La tâche d'acquisition à un cout d'exécution d'environ 0.1ms et une période de 2.5ms, la tâche de processing à un cout de 31.1ms (dans le pire cas) et une période de 82ms et finalement la tâche vidéo à un cout de 66.6ms (106ms pour les cas extrême) et une période de 66.6 ms. On voit directement que pour la vidéo ce n'est pas ordonnançable. Disons que s'il y avait uniquement la tâche vidéo ou la tâche vidéo avec le filtre de gris appliqué, alors dans ce cas elle serait ordonnançable si on ne prenait pas en compte les cas où elle a un coût plus élevé que ça période (comme ça l'est quand on applique le filtre de convolution). Mais si on doit faire le calcul pour les trois tâches ensemble alors là on ne pourrait absolument pas avoir une condition suffisante d'ordonnançabilité.

Voici le calcul fait pour la tâche vidéo toute seul.

$$\sum_{i=1}^{1} \frac{C_i}{P_i} = \frac{66.6}{66.6} = 1 \leq 1$$

Et voici le calcul fait pour la tâche vidéo avec le filtre de convolution :

$$\sum_{i=1}^{1} \frac{C_i}{P_i} = \frac{106}{66.6} = 1.59 > 1$$

Comme la tâche vidéo n'est pas ordonnançable si la moindre autre tâche vient s'y ajouter, on peut donc faire le calcul pour les deux autres tâches :

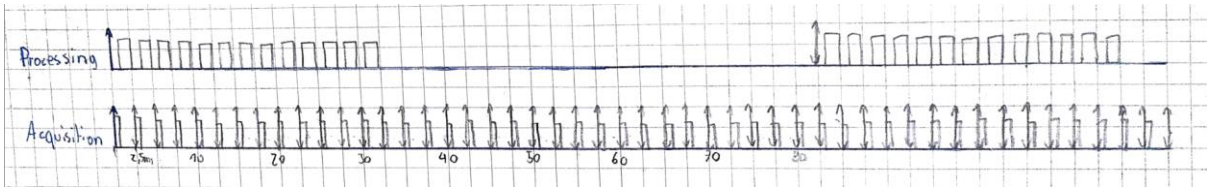
$$\sum_{i=1}^{2} \frac{C_i}{P_i} = \frac{0.1}{2.5} + \frac{3.1}{8.2} = 0.419 < 0.828$$

Et on voit que c'est ordonnançable pour les deux tâches audios.

Ordonnançabilité

L'un des principaux problèmes pour l'ordonnançabilité de la vidéo c'est qu'elle ne peut malheureusement pas l'être car comme on peut le voir dans les mesures avec une période de maximal de 66.6ms que nos délais ne sont absolument pas respectés lorsque l'on utilise la convolution, en effet avec un temps d'exécution moyen d'environ 106ms notre tâche n'est malheureusement pas ordonnançable, l'une des seules solutions à ce problème serais de diminuer le frame rate pour avoir une période plus grande, mais cela ferait que notre vidéo aurait une certaine latence visible, ce qui est déjà le cas avec notre période de base, résultat on se retrouve avec une vidéo ayant une latence similaire si on baisse le frame rate que sans la baisser, bien évidemment on pourrais aussi augmenté le nombre de CPU utilisé mais ce n'est pas ce qui est demandé pour cette expérience. La tâche vidéo ne sera donc pas pris en compte pour le calcul de l'ordonnançabilité du système et l'on ne traitera que la partie audio.

On avait pu remarquer que la partie acquisition étais préempter lors du traitement dans la partie processing et que la partie de lecture de l'audio ne durait que 0.1ms il nous faut donc réussir à faire en sorte d'avoir la tâche de processing qui s'exécute durant les 2.4ms où la tâche d'acquisition n'est pas en cours d'exécution, pour se faire voici à quoi devrait ressembler le rate monotonic si les priorités ont été faites correctement. En sachant que notre processing_task doit attendre de recevoir toutes les données avant de pouvoir les traiter on peut dire que processing_task à une période d'environ 82.5 ms si l'on regarde les mesures plus haut (attention la première période ne fait rien puisqu'il n'y a pas encore de donnée dans la queue, mais les suivantes oui, la première période ne sera pas illustrée dans le chronogramme car la place disponible sur ma feuille est limitée) :



Ce chronogramme serait la situation parfaite afin d'éviter qu'acquisition task se fasse préempter. Pour que cela puisse être mis en place dans notre système nous devons modifier les priorités de nos tâches pour se faire la tâche log ne changera pas, la tâche acquisition aura la plus haute priorité (99) et afin d'être certain que notre système fonctionne à tout moment processing_task aura une priorité de 0.

Voilà donc ci-dessous les nouvelle mesure avec ces priorités :

Acquisition task

```
------------------------------------------- summary1.c -----
 Total of 3717 values
   Minimum  = 2.454360 (position = 35)
   Maximum  = 2.523750 (position = 34)
   Sum      = 9264.024610
   Mean     = 2.492339
   Variance = 0.000032
   Std Dev  = 0.005645
   CoV      = 0.002265
----------------------------------------------------------------
```

Processing task

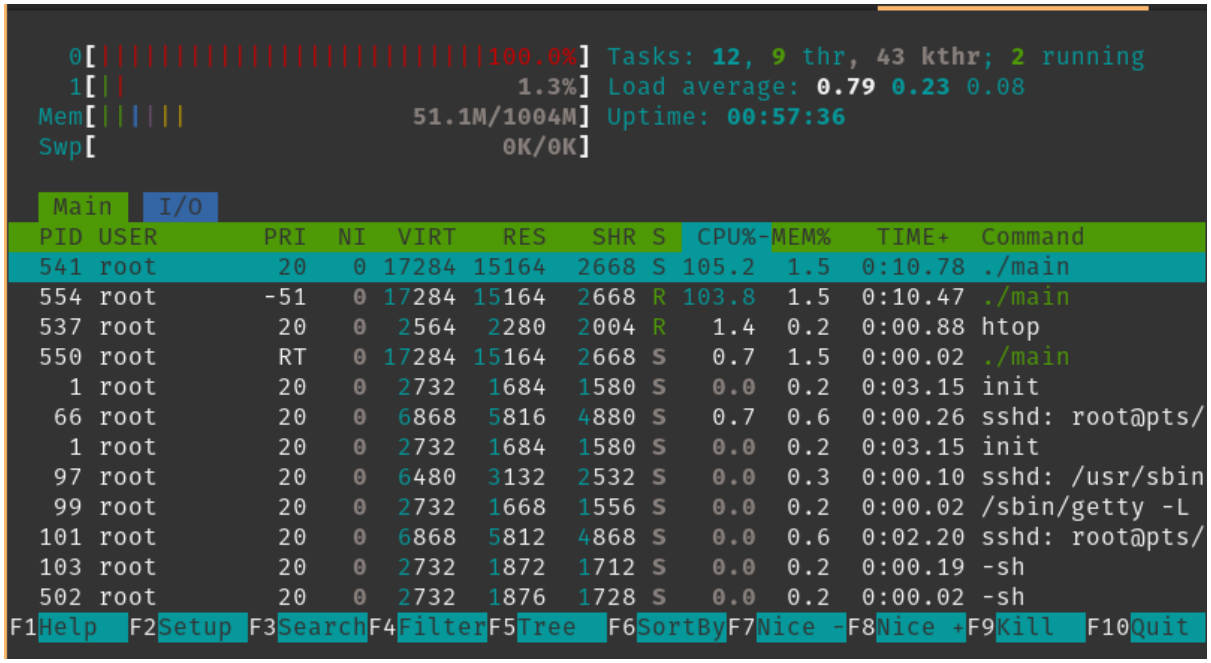
```
------------------------------------------- summary1.c -----
 Total of 990 values
   Minimum  = 33.131480 (position = 54)
   Maximum  = 37.681440 (position = 65)
   Sum      = 32999.792250
   Mean     = 33.333123
   Variance = 0.083259
   Std Dev  = 0.288547
   CoV      = 0.008656
----------------------------------------------------------------
```

On peut voir à l'aide de ces mesures que la tâche d'acquisition, respecte maintenant son échéance de 2.5ms, mise à part quelque valeur la dépassant comme on peut le voir pour la valeur maximum de 2.52ms. En revanche on remarque que la tâche de traitement met plus de temps à s'exécuter, alors que l'on avait une moyenne de 31.038595ms nous avons maintenant une moyenne de 33.333123ms, cela ne pose toutefois aucun problème puisque la tâche de traitement doit s'exécuter environ tous les 82ms.

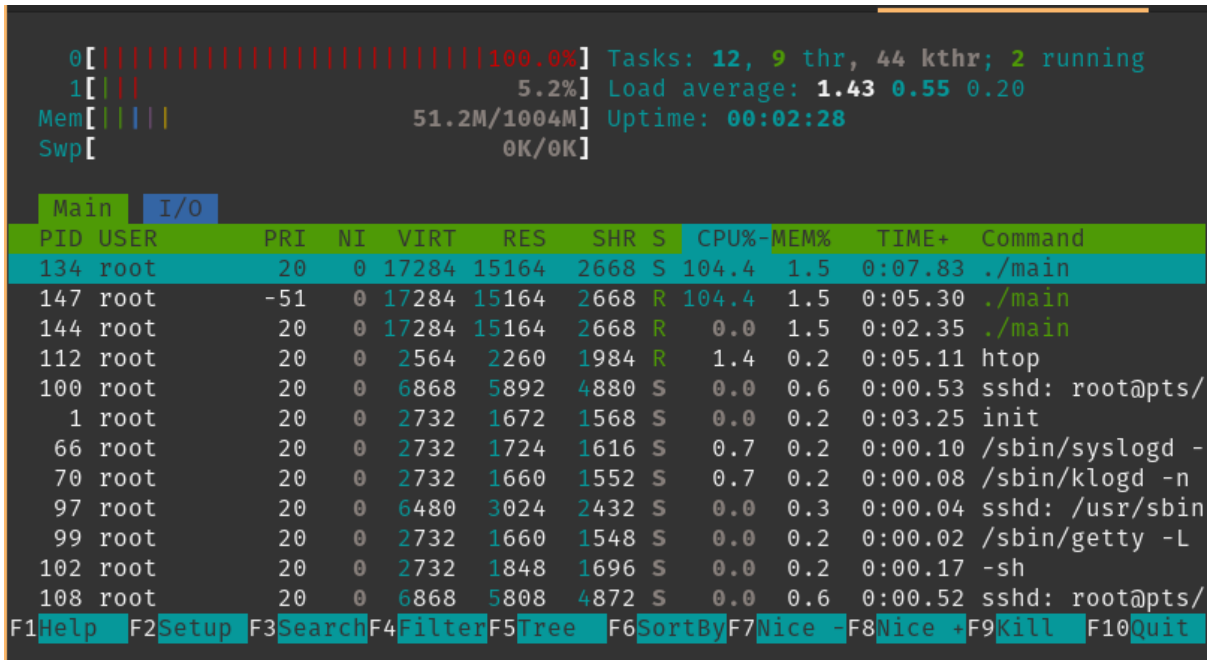
Traitement vidéo

On sait que la tâche vidéo n'est malheureusement pas ordonnançable, l'une des solutions que nous avons proposées était de diminuer le framerate afin d'obtenir un système fonctionnel malgré la lenteur de la vidéo pour ce faire voici les différentes valeurs du htop avec différent framerate:

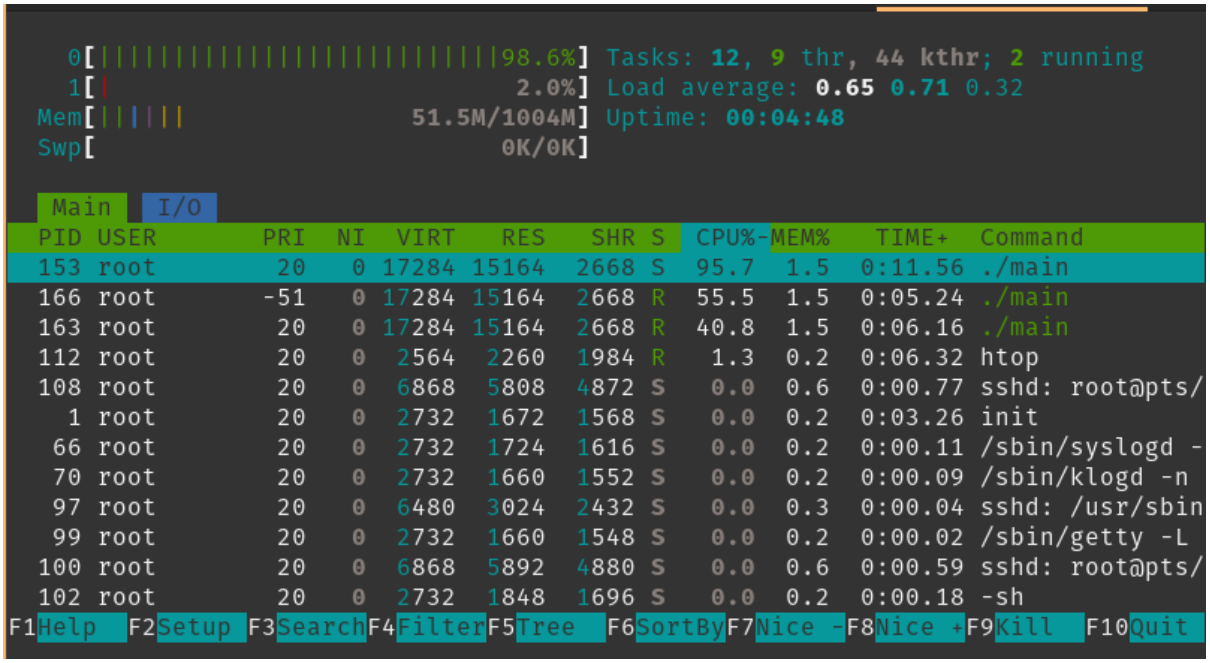
Framerate = 15



Framerate = 6



Framerate = 5



Pour que notre système entier (audio + vidéo) soit fonctionnel il nous faut baisser le framerate à une valeur de 5 framerates par secondes.

Conclusion

Ce laboratoire nous a permis de concevoir une communication entre tâche et d'utiliser des méthodes de synchronisation comme les queues et les events. On a aussi pu conclure que faire fonctionner toutes les tâches ensemble n'est pas possible même selon l'algorithme de rate monotonique et que pour que ce soit le cas, il nous faudrait baisser le nombre d'images par secondes pour que toute les tâches puissent fonctionner correctement. On a finalement réussi à montrer l'ordonnançabilité des tâches audios et de leur trouver les priorités pour quel soient ordonnançable selon l'algorithme de Rate Monotonic.