WORKLOG

Contrairement à l'autre Worklog que je remplissais de manière incrémentale en décrivant les suites d'actions à réaliser pour atteindre des objectifs, celui-ci sera plus un résumé concis et explicite des deux parties (Bare metal et distrib).
L'idée de ce document est d'expliquer le plus simplement ce que l'on a fait en cours.

BARE METAL

Première chose importante : On n'a pas d'OS.
Ça signifie qu'on n'a littéralement aucun utilitaire pour nous aider. Il faut donc composer avec deux éléments:

- Le code charger au départ. C'est en fait le code que l'on va écrire. Mais comme on n'a pas d'OS, il va falloir s'occuper d'effectuer les liens nous même avec un fichier en ".ld", que l'on passera à la compilation.

  Ce fichier permet de définir deux choses importantes : Les emplacements mémoires et leurs tailles (c'est-à-dire où sera positionné en mémoire chacune des parties de notre code, et la taille des zones mémoire utilisables) et le point d'entrée.
  Le point d'entrée est en fait le code qui sera appelé au tout début, l'idéal est que ça soit directement un code assembleur que l'on a écrit nous même pour avoir un contrôle total sur l'initialisation.

- Les zones mémoire de la carte et du processeur. Grâce à la documentation il est possible d'obtenir des informations sur l'état de la carte et d'y appliquer des modifications.
  Par exemple, nous avons une carte avec une diode et voulons l'allumer. Il faudra donc aller chercher dans la documentation de la carte l'emplacement de la carte qui contrôle l'état de la diode et ensuite modifier la valeur de cette adresse.

Il est possible grâce à tout ça de faire fonctionner un code basique. Ce dernier peut communiquer avec l'extérieur grâce à la "ligne série", une technologie de communication. Le concept est assez simple, la carte possède deux adresses mémoires que l'on pourra utiliser pour lire et écrire. De notre point de vue de développeur, ces deux emplacements se comportent en mode producteur/consommateur.
La première se mettra à jour à chaque fois que nous lirons un octet dedans, ce qui permet de lire tout un paquet d'information simplement en dépilant les octets un à un, nous savons qu'il n'y à plus d'information à lire grâce à une autre adresse qui contient des flags nous donnant des informations.
La deuxième permet d'écrire des octets un à un, qui seront envoyés à notre correspondant (dans le cas où on utilise QEMU c'est le terminal où QEMU tourne qui affiche et interprète ces octets).

Maintenant vient le cas des interruptions. Ces dernières sont des signaux levés par la carte et qui permettent d'activer des fonctions que nous avons codées pour réagir dessus.
Il faut donc tout d'abord activer les interruptions qui nous intéressent, il faut se référer à la doc et trouver les bonnes adresses mémoires et modifier les bons bits pour obtenir ce résultat.
Une fois cela fait, il faut y attacher une fonction à appeler. Cette dernière pourra donc réagir dessus. Il est intéressant d'utiliser les interruptions pour pleinement tirer partie du matériel qui va nous notifier quand on en a besoin plutôt que de faire tourner à plein régime un programme qui va interroger la carte en permanence pour réagir sur ces événements.

DISTRIBUTION

Commençons par définir ce qu'est un noyau. À l'image du petit programme bare metal dont l'on parlait dans le chapitre précédent, ce dernier communique directement avec le matériel.

On utilise GRUB, qui est le noyau linux de GNU. Ce dernier est configurable et peut prendre en entrée soit un fichier contenant directement toute notre distribution ou de manière plus simple on peut directement passer notre arborescence.
Cette arborescence de fichier contiendra notre exécutable d'entrée, mais aussi les librairies standard (dans /lib par défaut).
On pourra mettre des binaires utiles dans /bin.
Donc une manière simple d'avoir un shell basique : Mettre un binaire de shell (dans notre cas busybox) dans bin, créer un lien symbolique pointant dessus depuis /bin/sh (chemin standard du shell), Et enfin appeller /bin/sh depuis l'exécutable d'entrée.