Cours 1 :

Compatibilité entre le source , compilateur, linker, fichier .h, même libc (compilé pour certain version du noyau), le noyau offre une ABI (syscall)
Version ABI différentes de noyau, et il faut qu'elle soit compatible.
Le noyau est compatible pour un processeur donné (32 bits, 64 bits, arm…) qui Offre une interface (ABI)
Les appelles systèmes sont wrappé par une bibliothèque chargé dynamiquement par le noyau, libc > vdso ce qui permet pour certains appelle système de ne pas descendre au noyau et de directement appelé ca a partir d'une mémoire partagé que le noyau écrit dedans.

Bus load value from address / store a value in address
Le dialogue entre processeur et contrôleur se fait par des lectures et écritures a une adresse (pas forcément dans une adresse mémoire)
S'il 'adressa va dans un contrôleur mémoire, on lit et on écrit dans la mémoire
Si l'adresse est routé sur Un autre Contrôleur : dialogue avec un périphérique
Du point de vue de la processeur on lit et on écrit de la mémoire
On écrit pas dans la mémoire et on pas de la mémoire

La fonction première du bus c'est de router, voit une adresse => je parle a qui?
Pour chaque zone mémoire, le bus c'est a qui s'adresser principalement grâce à des intervalles adresse définit pour chaque contrôleur (Memory Map).
Le processeur démarre dans une addresse codé dans son matériel (pour le arm démarre à l'adresse 0). Il fait un load de l'adresse 0 pour récupérer cette instruction.
Rien n'est configuré niveau matérielle (pour avoir la flexibilité de modifier le comportement selon le besoin), sauf un support persistant qui est configuré et linké pour s'exécuter à l'adresse 0. Ce bout de code est mis en place par le fabricant.
Ce Code configure la ddr (contrôleur) pour pouvoir lire et écrire dans la ram, puis copie des choses dans la ram après il peut allouer de la mémoires, il se copie ailleur et se mets a s'exécuter ailleur dans la mémoire, dans le processeur arm à l'adresse 0 il y a un vecteur d'interruption, il faut l'écrire ce vecteur pour prendre le contrôle et tant que EPROM est mappé à l'adresse 0 c'est impossible c'est pour ca qu'il faut déplacer.
Réinitialisation du matérielle 3 fois

Le contrôleur permet de router le processeur grâce a un intervalle adresses mémoire vers le bon composant. Donc le processeur est toujours en communication avec un contrôleur et pas directement la mémoire.

Individual wires :
1 octet = 8 bits (suite de 0 ou 1) = 8 fils matérial 1 v ou 5v le matériel reçoit une combinaison de 1v ou 5v en fonction de combinaison dans les 8 bits

FIFO queue :
Une pile en matérielle de 32 bits cad 32 valeurs de 0 ou 1.

Signo bit : chaque bit a une signification, exemple : reset, power on
Le devise communique aussi en retournant des bits que le développeur ou le système doit interpréter, exemple : rj45 détaché…

En général il y a un FIFO (un registre matérielle), on pousse les données en vérifiant le bit qui informe si le FIFO n'est pas plein, le devise consomme à partir de cette FIFO
Communication avec le matérielle : Mettre des 0 et 0 dans un fil et lire à partir d'une fil , ou mettre des valeurs dans un FIFO
Les autres registre matérielles en écrit des bits et on lit des bits (puis on interprète)

UART controller : circuit qui permet d'avoir une ligne de serie RS232

Le terminal est une version logicielle d'une ligne série, stdin et stdout fonctionnent de la même manière que UART matérielle.

Un shell dialogue via la ligne RS232 qui passe dans la ligne usb
En bare metal on utilise une version logiciel de RS232, le logiciel qui est coté du materiel pour communiquer il va lire dans la ligne pour obtenir un character et écrire dans ligne pour envoyer des characters.
Addresse plus l'offset dans une adresse dans la mémoire, chaque adresse correspond a un registre différent et chaque registre a un sens différent.

Le périphérique doit être configurer, le code fourni dans le cours est (presque) configuré.
Configurations : A quelle vitesse l'envoie des données est fait, encodage…

Pour faire un print de hello world par exemple :
Le processeur démarre et un moment donné il va se retrouver sur notre code source :
Fonction étant donnée une adresse de uart choisie (uart 0 par exemple) on veut envoyer un character (donc 8 bits)
Boucle qui lit le registre est dit si le FIFO de sortie (send) est plein ou pas s'il est plein il faut attendre et une fois qu'elle vide il faut écrire dans l'autre registre celui de la fifo et donc il prend (contrôleur??) les 8 bits et les mets dans la fifo et s'occupe de le transfert de l'autre côté comme une carte réseau (version bloquante) parce que il pas un système comme linux qui ordonne les processus et qui permet de notifier notre programme que la fifo est prête.

Taper au clavier, ca passe dans les deux fils, arrive par derrière, le circuit comprend ce qui arrive et traduit ça par des bits (1 octets), FIFO de réception vide? Boucle > si il y a quelque chose a lire dans le fifo du matériel il est retourné.

Bios initialise un certains nombre de périphériques.
MS dos : convention pc de 16 bits premier secteur de ce disque la lire et mettre en mémoire et exécuter.

Bootloader : multiboot choisir linux, windows surcouche indépendante de fabricant, hyperviseur dans cette étape, bootloader charge le noyau dans la mémoire dans une addresse spécifier dans la convention.

Exercice 1 : Notre bootloader et notre noyau.
Exercice 2 : Grub chargé le noyau linux et le noyau démarre la distribution
Un fichier formaté comme un disque, un disque c'est une séquence de secteurs, secteur = 512 o, le premier contient notre bootloader.
La signature permet au bios de la vérifier et d'essayer de lancer le système.
Le bios charge le bootloader dans la mémoire, le processeur est en mode 16 bits il faut qu'il passe en 32 bits.
Le boot loader cherche le noyau, et dans /boot (dans le système de fichier pour la première partition), pour pouvoir le lire il faut savoir lire le système de fichiers et avoir le driver, comprendre le format.
Mbr (qui fait partie de bootloader stage 1) 512o comprend un certain nombre de format, ou le noyau est stocké

On va construire un noyau qui est stocké dans le premier secteur disk qui sera chargé et lancé, il fait un console avec un echo, taper des character > partir par ligne de série > le logiciel va écrire les caractères dans l'autre sens et ca va réapparaître dans l'écran.

Une board deux fils un câble usb et machine
Tout ce que je tape transcrit en 0 et 1 et partira sur le fil, lue par mon logiciel sur board, renvoyé en 0 et 1, lue par mon logiciel (en c) sur mon pc, et ferait une printf.

logiciel lue un des registres pour recevoir les octets et écrit pour retourner des octets

Par le protocole RS232, si je tape 'a' dans le terminal l'outil graphique de ce terminal lie 'a' et le mets dans le fil comme un socket (Il faut bien encoder le character 'a' parce que la machine ne comprend pas un character 'a' mais plutôt des bits donc encodage ASCII) puis le récupérer et interpréter les bits et réaffiche.

Un board cable, Lancer un terminal taper des caractères et lire des caractères pour lue il doit recevoir.

Qemu remplace le board, et permet d'exécuter un code arm alors que ma machine et un intel, compatible avec gdb et donc on peut déboguer le bios, le bootloader
Créer un disk, donner le disk a Qemu, exécuter le bios booter et chargé l'image.
Donne le support de ligne de série donc avoir un shell, la fenêtre en question devient une ligne série.

Disk.img est considéré comme un disque dur
Le terminal comme une ligne série.
De la memoire, memory map, register, UART

Ctrl A + c pour accéder au console qemu et sortir du shell, dans ce mode on peut allumer/éteindre des périphériques, contrôler la virtualisation (le board), décrire le matérielle Du board simuler. Info
Pour trouver la liste des devises existants, leurs noms : info qtree
Et donc Pour trouver l'adresse UART : Info qtree.

Cours 2 :