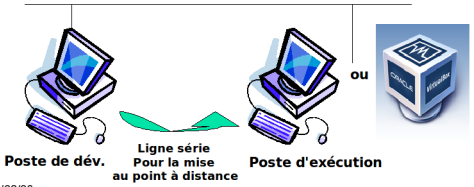
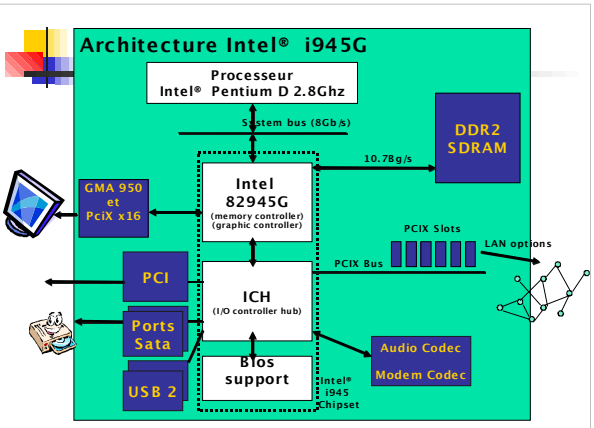
Plateforme d'expérimentation du projet système

Environnement de travail

- Environnement de développement sur un PC
  - Linux (Ubuntu), **`ensipsys01-96`**
- Exécution du noyau sur un autre PC ou dans une machine virtuelle (VirtualBox)

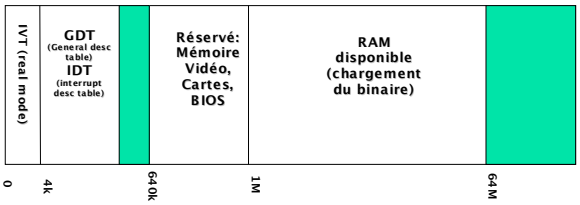




Mémoire

- Mémoire volatile
  - 1Go de SDRAM sur les PC de l'école
    - *DDR2 Synchronous Dynamic RAM*
  - 64Mo recommandés pour VirtualBox
  - Certaines plages d'adresses sont réservés
    - tables du système, …
  - D'autres sont inaccessibles : mémoire vidéo, périphériques, BIOS, …
  - La plus grande partie est disponible pour vos programmes

Cartographie de la mémoire




```
movb $2, 0x100000; mettre la valeur 2 à l'adresse 100 000 (en RAM)

char *p;                    ; définition d'un pointeur
p= (char *)0x100000;        ; positionne ce pointeur à l'adresse 1M
*p = 2;                     ; Mettre 2 dans cette adresse mémoire
```


Fonctionnement d'un poste d'exécution

- PC
  - Brancher une clé USB avec le noyau
  - Démarrer la machine et appuyer sur F12
  - Sélectionner *USB Device* puis laisser le chargeur installé sur la clé lancer le noyau
  - Appuyer sur le bouton d'extinction quand le test est terminé
- VirtualBox
  - Copier le noyau sous : ~/.VirtualBox/TFTP/kernel.bin
  - *Lancer* la machine virtuelle – VM – (configurée pour démarrer par le réseau)
  - *Eteindre* la VM quand le test est terminé

Détails de configuration sur le Wiki

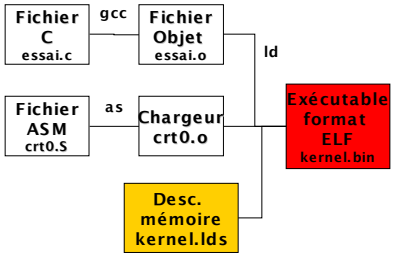
Démarrage: chargeur et Crt0

- Au démarrage de la machine, un chargeur place le noyau en mémoire et lance son exécution
  - *Grub* sur la clé USB ou *Noyau.pxe* pour VirtualBox
  - Le format du noyau doit respecter la « Multiboot Specification » : nécessaire pour un chargement correct. Cf manuel Grub
- Le démarrage commence par Crt0
  - Initialisation de la mémoire, configuration des tables système
  - Débogage par la ligne série via des outils standards (**gdb**, **ddd**, …)

Écrire un noyau

- Programmation en C ou en Assembleur IA32
- Utilisation des outils GNU sur un poste de développement:
  - **gcc**, **as**, **ld**, …
- Production de l'exécutable directement chargeable en mémoire de la machine d'exécution
  - Binaire x86 avec directives de chargement dans le fichier *kernel.lds*
  - Respect de la « Multiboot Specification »

Chaîne de développement

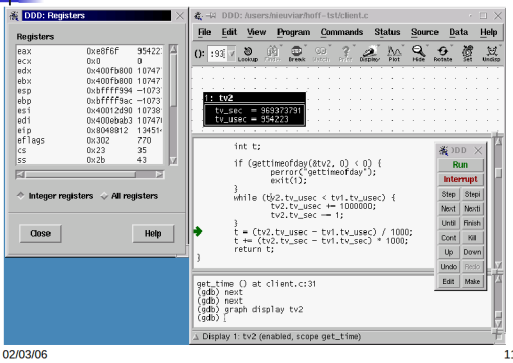



```
# édition de liens du noyau (symbole de début d'execution : entry)
kernel.bin: kernel.lds $(OBJS) userdata.o
	$(LD) -e entry -Tkernel.lds $(OBJS) userdata.o -o $@
```


Mise au point à distance

- Utilisation d'outils classiques sur le poste de développement
  - **gdb** ou l'interface graphique **ddd**
- Communication par la ligne série
  - Protocole **gdb stubs**
- Principes d'établissement d'une connexion
  - Chargement d'un noyau
    - Le placer en attente d'une connexion avec le débogueur
  - Connexion de **gdb** avec le poste d'exécution
    - Commande **target remote /dev/ttyS0** ***(NE PAS FAIRE RUN !!!)*** (VirtualBox : /tmp/vbox-sys)
  - Débogage avec **gdb** ou **ddd**

Interface DDD



Code fourni

- Squelette de l'architecture + Makefile fournis
- Possibilité de repartir de votre code du TP de PSE (cours de 1er semestre)
- Fonctions de traitement de chaînes de caractères, copie de mémoire

Code fourni

- Paquetage de liste avec priorité (inspiré de celui Linux)
  - un exemple et la doc sont sur la page Web et dans le fichier source queue.h
  - Ensemble de macros C
    - pas des fonctions !
  - Arguments contenant le nom des champs utilisés de la structure listée
    - le champ de type link (suivant, précédent)
    - le champ priorité (un entier)

Code fourni

- Implantation du printf()
  - utilise une fonction d'affichage que vous avez à écrire :
    - console_putbytes(char *buf, int len) défini dans console.h et dont le code est à écrire dans les deux fichiers console.c (kernel et user)