Programme en mode User

MI074 - 8

Combien de codes

- code OS
  - kernel le noyau
  - libk les fonctions de services
  - arch spécifique à la plateforme
  - cpu spécifique au processeur

- code des bibliothèques de l'utilisateur
  - libc les fonctions de service des applications + les syscall
  - pthread la gestion des threads et des synchro
  - crt0 code de lancement d'une application

- code des applications
  - une seule application à la fois

Questions

Que va t'il falloir faire pour permettre la programmation
en mode user ?

- Quel est l'interface entre les mondes user et système ?
- Quel est l'impact sur la plateforme ?
- Où mettre le programme pour le lancer depuis le système ?
- Comment passer du mode user au mode système
- Comment compiler et faire l'édition de lien ?
- Comment lancer un thread utilisateur ?

Interface entre les deux mondes

- des appels systèmes:
  - services standardisés proposé par le système pour l'application (ou les applications).

- Un appel système c'est :
  - un numéro de service
  - des paramètres pour le service
  - un résultat qui prend la forme d'un effet de bord
  - un état de retour (0 si tout va bien le plus souvent)
  - un numéro d'erreur pour connaitre la raison de l'erreur (errno)

- exemple ?

Passage du mode usr au mode sys

- C'est presque un appel de fonction
  - **int service(typ0 a0, typ1 a1, typ2 a2, typ3 a3)**
  - avec un effet de bord sur la variable errno

- Quand on appelle une fonction en C,
  - les registres temporaires sont perdus
  - les registres persistants sont intacts

- utilisation de l'instruction syscall

```
#define SYSCALL(service,args...)              \
    int __attribute__((noinline)) service(args)
#define DO_SYSCALLc(service_num) {     \
    register int retval, errval;                \
    cpu_syscall(service_num,retval,errval); \
    errno = errval;                             \
    return retval;                              \
}
```

Passage du mode usr au mode sys

```
#define cpu_syscall(service,retval,errval)                    \
__asm__ volatile (                                            \
   ".set noreorder      # do not modify the sequence  \n"  \
   "addi    $2, $0, %2  # set the service number      \n"  \
   "syscall             # call the system            \n"  \
   "add    %0, $0, $2   # save the result            \n"  \
   "add    %1, $0, $3   # save the error number      \n"  \
   ".set  reorder       # return to normal mode      \n"  \
   :"=r" (retval), "=r" (errval)                              \
   :"i" (service)                                             \
   :"$2", "$3"                                                \
)
```

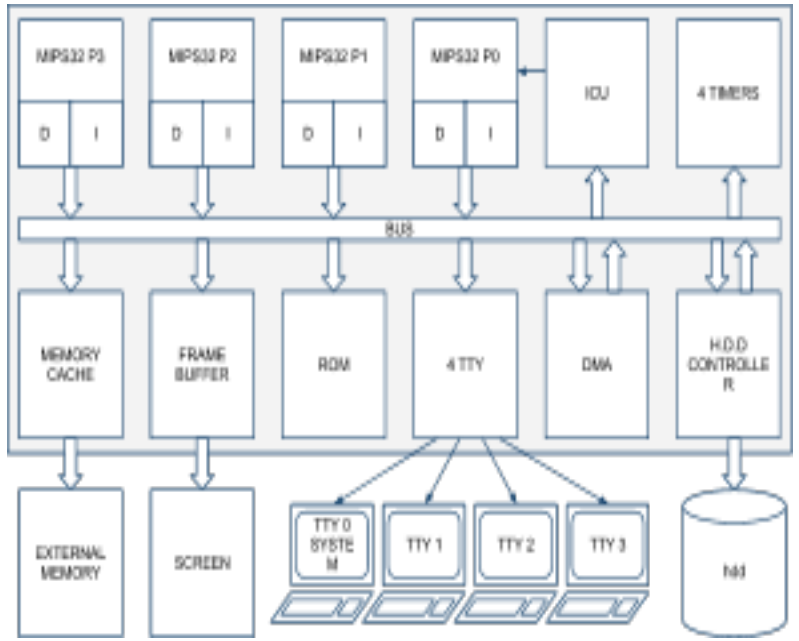
Passage du mode usr au mode sys

- entrée dans kentry
  - on teste si on est en mode user
    - si oui on change de pile
  - on teste si c'est un syscall
    - si oui
      on ajoute les paramètres serviceno et errno
      on sauve l'adresse de retour EPC
      on saute à __do_syscall
      on restaure la pile
  - on teste si c'est une interruption
    - ...

__do_syscall

- utilise une vecteur de syscall syscall_tbl[ ]
- se contente de lancer la fonction correspondant au service avec les paramètres a0 à a4

Impact sur la plateforme



- Où mettre le code ?
  - sur le disque
  - même sans système de fichiers

- Il faut compiler séparement

Comment lancer le programme

__do_init

- initialise l'architecture
- crée les thread_idle()
- charge le programme en mémoire
- lance la fonction _start()