3. SYSTEMES TEMPS REEL - Moniteur Temps Réel MTR86W32

Introduction au Moniteur Temps Réel multitâche MTR86W32

MTR86W32 : librairie de fonctions

. création, activation, suppression de processus,

. gestion des interblocages, gestion des sémaphores

. synchronisation de processus,

. gestion des priorités entre processus

. ordonnancement des processus

Figure 1

Squelette de programme C utilisant la librairie MTR86W32

```
#include "mtr86w32.h"

/* Déclaration des variables globales */

/* Déclaration des prototypes des tâches */



/* Tâche d'initialisation */
TACHE init( void )
{
/* créer les tâches */
/* créer les sémaphores, ressources, tubes ...s'il y a lieu */
/* lancer les tâches*/
}

/* Autres tâches */



main()
{
int code;
_init_video();     /* Si utilisation sous VC++ de la librairie conio */
code = start(init, 4096);
/* lancement de l'application avec 4Ko octets de pile pour init() */
exit(code);                    /* retour au système */
}
```

Exemple 1 de programme C utilisant la librairie MTR86W32

Création, activation de 3 tâches séquentielles A, B, C

```
/* La tache A affiche une chaine de caracteres,
caractere par caractere     */

/* La tache B affiche une chaine de caracteres  */

/* La tache C affiche une chaine et attend la frappe
d'une touche au clavier*/

/* La tache A est la plus prioritaire, suivie de la tache
B puis C          */

#include       "mtr86w32.h"

/*Prototypage des taches      */
TACHE init(void);
TACHE  tacheA (void);
TACHE  tacheB (void);
TACHE  tacheC (void);

/*Prototypage des fonctions     */
void printcarparcar(char *);

/* Fonction d'affichage d'une chaine de caracteres,
caractere par caractere */
void printcarparcar(char *s)
{   while (*s != '\0' ) putch(*s++);  }
```

```
TACHE init(void)            /* Tache initiale */
{
 active(cree(tacheA, PRIORITE_NORMALE, 512));
 active(cree(tacheB, PRIORITE_NORMALE-1, 512));
 active(cree(tacheC, PRIORITE_NORMALE-2, 512));
}


TACHE  tacheA(void)
{   printcarparcar("---ExecA\r\n");  }


TACHE tacheB(void)
{  cputs("...ExecB\r\n");       }


TACHE tacheC(void)
{
  printf("===ExecC\r\n");
  while (!kbhit());
  mtr86exit(0);                  /* Fin du multitache :*/
}


int main()
{
   int retour;
   start( init, 10000);          /* Debut du multitache */
   exit(retour);
}
```

Exemple 2 de programme C utilisant la librairie MTR86W32

Création, activation de 3 tâches A, B, C concurrentes

```
/* Taches A, B et C identiques à ce qui précède */

/*   Prototypage des taches     */
TACHE init(void);              TACHE  tacheA (void);
TACHE  tacheB (void);      TACHE  tacheC (void);

/* Prototypage des fonctions   */
void printcarparcar(char *);

int premfois;

/* Affichage caractere par caractere */
void printcarparcar(char *s)
{
    while (*s != '\0' )  {
    putch(*s++);
    dort(cvrtic(100));   /* Suspension de la Tache appelante (A) de 100 ms*/
    premfois = 1;     }
}

TACHE init(void)        /* Tache initiale */
{
  active(cree(tacheA, PRIORITE_NORMALE, 512));
  active(cree(tacheB, PRIORITE_NORMALE-1, 512));
  active(cree(tacheC, PRIORITE_NORMALE-2, 512));
}
```

```
TACHE  tacheA(void)
{   printcarparcar("---ExecA\r\n");   }

TACHE tacheB(void)
{   cputs("...ExecB\r\n");    }

TACHE tacheC(void)
{
  premfois = 1;
     while (!kbhit())
     {
          if (premfois == 1)
          {
               premfois = 0;
               printf("===ExecC\r\n");
          }
      }
  mtr86exit(0);                    /* Fin du  multitache */
}

int main()
{
 int retour;

 start( init, 10000);             /* Debut du multitache */
 exit(retour);
}
```

TP 3. SYSTEMES TEMPS REEL - Moniteur Temps Réel MTR86W32

1. Documentation de MTR86W32

mtr86w32.pdf

2. Exemples de programmes multitâches MTR86W32

. Création, activation de 3 tâches séquentielles.

. Création, activation de 3 tâches concurrentes.

3. Programme MTR86W32 : Horloge - chronomètre

Enoncé du problème

. *horloge* : *h* autorise le réglage de l'horloge.

. *chronomètre* : *c* déclenche le chronomètre (démarrage)

s provoque l'arrêt du chronomètre.

i affiche un temps intermédiaire (3 s)

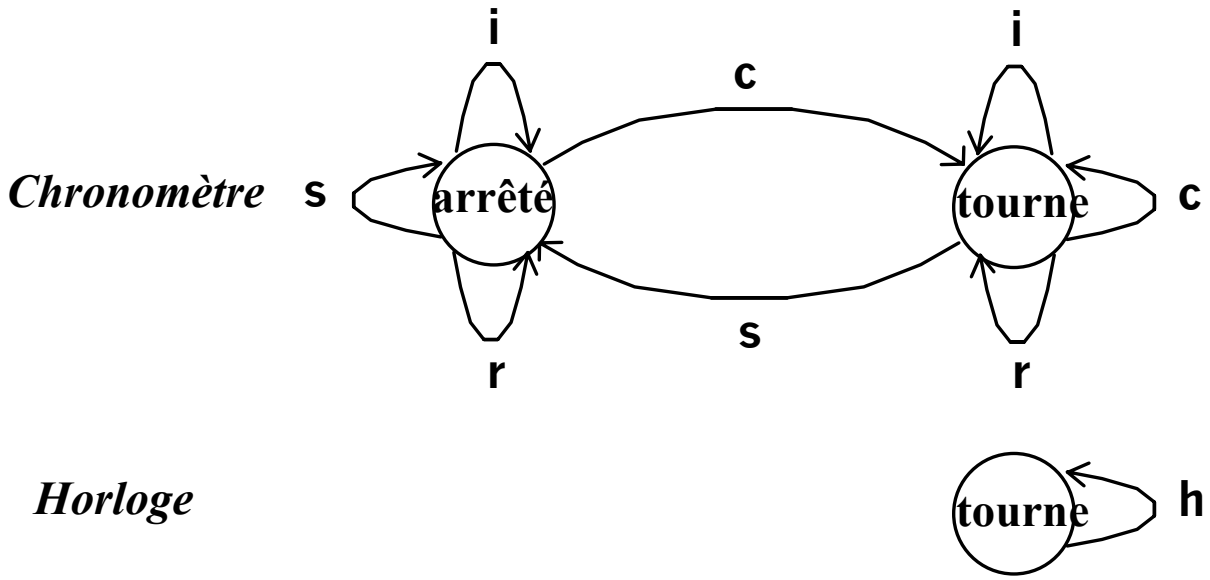
r remise à zéro (reset) du chronomètre

. *programme* : *f* provoque la sortie du programme.

Analyse des états et transitions du système

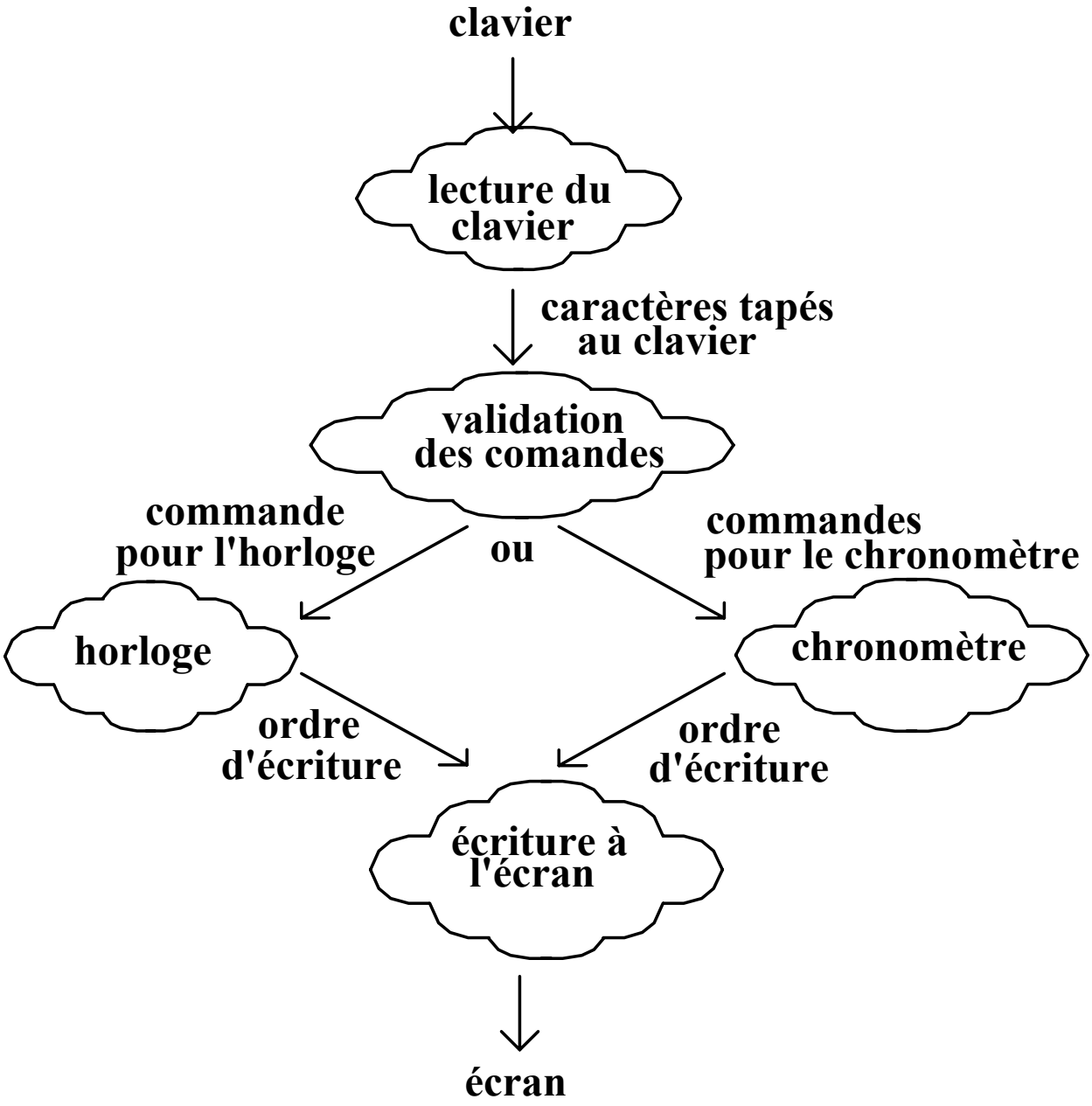
horloge : - tourne (≡ en fonctionnement)

chronomètre : - tourne
- arrêté.



Transitions (arcs) d'états (cercles) de l'horloge et du chronomètre

Analyse du flot des données



Flot des données (arcs) - transformateurs de données (bulles)

Décomposition en processus

Transformateur de données → processus : 6 critères

- *dépendance d'entrée-sortie*

- *fonction critique en temps*

- *fonction intensive en calcul*

- *cohésion fonctionnelle*

  des transformateurs de données avec interactions fortes doivent être groupés en un seul processus pour éviter le coût de communication entre processus

- *cohésion temporelle*

  des transformateurs de données qui doivent effectuer leur fonction à des moments identiques doivent être groupés pour constituer un seul processus.

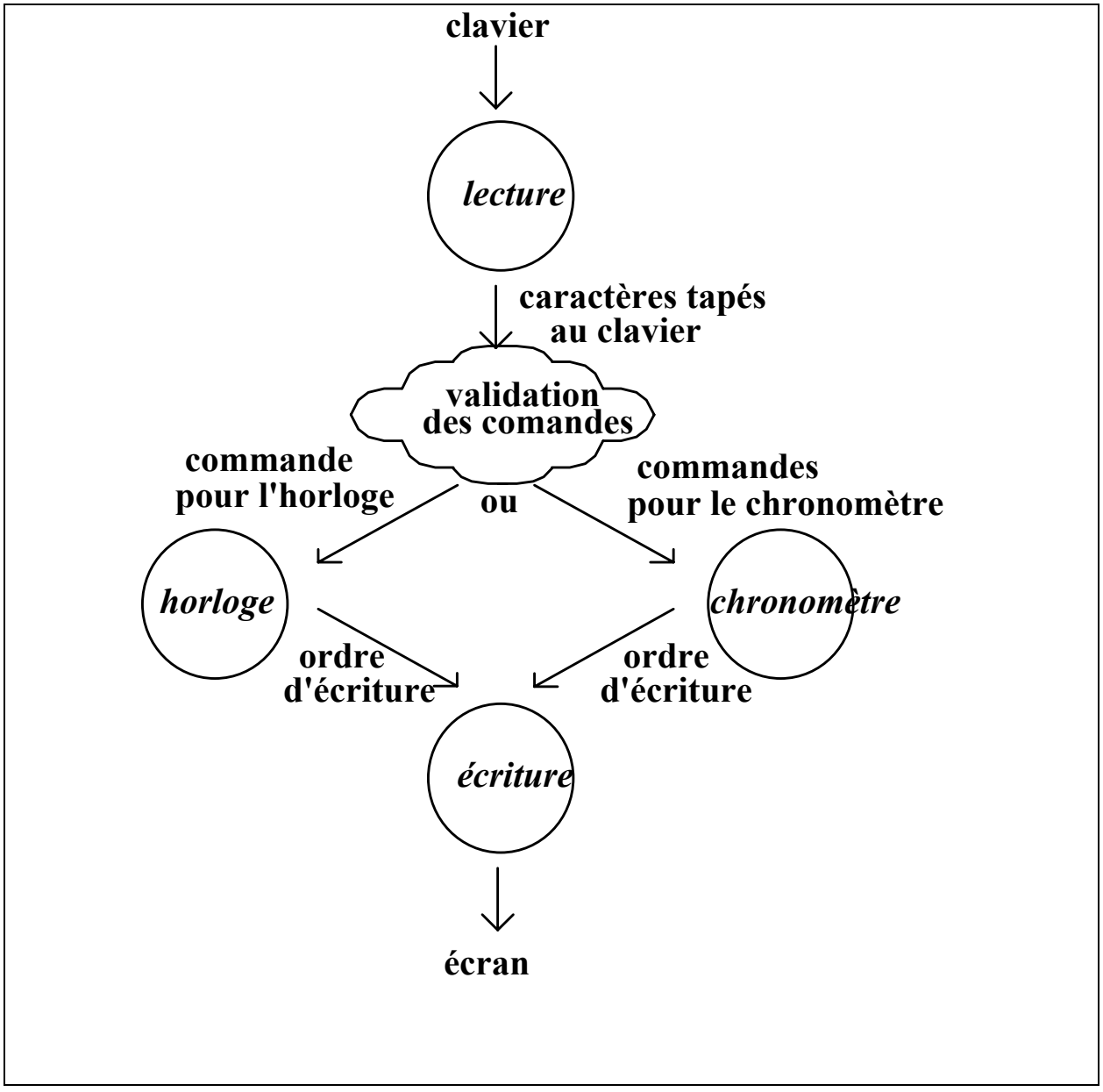
- *exécution périodique*

Ici : Dépendance d'entrée-sortie :

- lecture clavier
- écriture à l'écran (éventuellement)

Critère de l'exécution périodique

- horloge
- chronomètre.



Priorités entre les processus

L'ordre de priorité respecte la rapidité des processus.

Ainsi, on a, par ordre de priorité décroissante :

- processus *chronomètre* : processus le plus rapide
- processus *horloge*
- processus *écriture*
- processus *lecture* : processus le moins rapide.
