# Le traitement du temps

*Mesure*
*Horloges*
*Sommeil*
*Mise en forme*

Fabrice Harrouet
École Nationale d'Ingénieurs de Brest
harrouet@enib.fr
http://www.enib.fr/~harrouet/

# Intérêt

- ▷ **Cadencer les applications**
  - ◇ Cinématique indépendante des performances
- ▷ **Délais d'attente**
  - ◇ Éviter l'attente active
- ▷ **Dater des événements**
  - ◇ Tenir un journal
  - ◇ Faire du calcul sur les dates (statistiques)
- ▷ **Évaluer les performances**
  - ◇ Mesurer des durées

# Mesure du temps

- **Le type** `time_t`
  - Défini dans `time.h`
  - *Ansi C* : permet des opérations arithmétiques
  - *Posix* : nombre de secondes depuis *Epoch* (01/01/1970 à 0h0m0s TU)
  - Pour l'instant ≡ entier long signé (32 bits)
    - Dépassement prévu le mardi 19/01/2038 à 3h14m7s TU
      → vendredi 13/12/1901 à 20h45m52s TU
  - Nouvelle représentation pour l'avenir ?
    - Non signé → + 70 ans, 64 bits → stockage et communication !
  - Précautions à prendre dès aujourd'hui
    - Utiliser `time_t` → mise à jour ≡ recompilation
    - Stocker et communiquer à travers des `long long int` (64 bits)

# Mesure du temps

- **L'appel système** `time()` (`man 2 time`)
  - `#include <time.h>`
    `time_t time(time_t * t);`
  - Donne la date courante, résolution = 1 seconde
    - *Posix* : différence ≡ durée en secondes
    - *Ansi C* : décoder avec `localtime()` (voir plus loin)
    - Utiliser la fonction `difftime()` (`man 3 difftime`)
      `double difftime(time_t t1,time_t t0);`
      Différence `t1-t0` en secondes
  - Si `t` n'est pas nul la date courante y est également inscrite
  - Retour : la date ou `-1` si erreur (mauvais pointeur)

# Mesure du temps

- **L'appel système** `gettimeofday()` (`man 2 gettimeofday`)
  - ```
    #include <sys/time.h>
    #include <unistd.h>
    int gettimeofday(struct timeval * tv,
                     struct timezone * tz);
    ```
  - Donne la date courante, résolution = 1 $\mu s$
  - `tv` : adresse d'une structure `timeval`
    - Champ `tv_sec` (`time_t`): secondes depuis 01/01/1970
    - Champ `tv_usec` (`long`): $\mu s$ dans la seconde courante
  - `tz` : obsolète, passer un pointeur nul
  - Retour : `0` si ok, `-1` si erreur (mauvais pointeur)
    ```

# Mesure du temps

```
#include <sys/time.h>
#include <unistd.h>
#include <stdio.h>

unsigned long getTime(void) // temps en millisecondes
{
struct timeval tv;
gettimeofday(&tv,(struct timezone *)0);
return(tv.tv_sec*1000LU+tv.tv_usec/1000LU);
}

int main(void)
{
unsigned long t0=getTime();
/**** Debut du traitement a mesurer ****/
for(int i=0;i<1000;i++)
  {
  for(int j=0;j<1000000;j++) {}
  }
/**** Fin du traitement a mesurer ****/
fprintf(stderr,"%lu ms elapsed\n",getTime()-t0);
return(0);
}
```

# Mesure du temps

- **Mesure fine de l'activité du processus** (statistiques essentiellement)
  - La fonction `clock()` (`man 3 clock`)

    ```
    #include <time.h>
    clock_t clock(void);
    ```

  - Retour : temps d'activité effective du processus ($\neq$ durée perçue)
  - Différence `/ CLOCKS_PER_SEC` : durée en secondes
  - L'appel système `times()` (`man 2 times`)

    ```
    #include <time.h>
    #include <sys/time.h>
    clock_t times(struct tms * buf);
    ```

  - Retour : durée écoulée depuis le démarrage du système
  - Différence `/ CLK_TCK` : durée en secondes
  - `buf` : cumul du temps utilisateur/noyau du processus/ses enfants

# Programmer des horloges

- **Principe**
  - Utilisation
    - Préciser un délai (une période) et le traitement associé
    - Poursuivre l'application
  - Haut niveau : horloges applicatives
    - Programmation événementielle
    - Flot d'exécution non interrompu par l'expiration
    - Les traitements peuvent être longs
  - Bas niveau : horloge préemptive
    - Délivrance de signaux (voir le cours *Communication Sous Unix*)
    - Interruption des appels système *lents*
    - Interruption du flot d'exécution par la routine de traitement
    - Problèmes de réentrance $\rightarrow$ traitements minimaux si possible

# Programmer des horloges

- **L'appel système** `alarm()` (`man 2 alarm`)
  - `#include <unistd.h>`
    `unsigned int alarm(unsigned int seconds);`
  - Programme le déclanchement de `SIGALRM` dans `seconds` secondes
  - Annule et remplace la programmation précédente
  - Retour : les secondes restantes de la programmation précédente
  - Précautions
    - Interférences avec `setitimer()` et `sleep()` (voir plus loin)
    - `SIG_DFL` → fin du programme
    - `SIG_IGN` → aucun effet !
    - Éviter le déclenchement tardif avec `alarm(0)`
    - Interruption d'un appel système lent : dû à `alarm()` ?

# Programmer des horloges

```
#include <signal.h>
#include <unistd.h>
#include <errno.h>
#include <stdio.h>

void doNothing(int) { }

int main(void)
{
struct sigaction action;
sigemptyset(&(action.sa_mask));
action.sa_flags=0;
action.sa_handler=doNothing;
sigaction(SIGALRM,&action,(struct sigaction *)0); // Installer le gestionnaire
fprintf(stdout,"Input ? "); fflush(stdout);
char input;
alarm(10);                                        // Programmer l'horloge
int nb=read(STDIN_FILENO,&input,sizeof(char));    // Lire la saisie
unsigned int r=alarm(0);                          // Annuler l'horloge
if((nb!=sizeof(char))&&(errno==EINTR)) fprintf(stdout,"Too late !\n");
else fprintf(stdout,"Input --> [%c] (%u seconds remaining)\n",input,r);
return(0);
}
```

# Programmer des horloges

- **L'appel système** `setitimer()` (`man 2 setitimer`)
  - `#include <sys/time.h>`
    `int setitimer(int which,`
    `const struct itimerval * nval,`
    `struct itimerval * oval);`
  - Programme le déclenchement de l'horloge `which` selon `nval`
  - Annule et remplace la programmation précédente
    - Le reste est écrit dans `oval` (si pointeur non nul)
  - `which` : type d'horloge
    - `ITIMER_REAL` : temps physique (`SIGALRM`)
    - `ITIMER_VIRTUAL` : temps en mode utilisateur (`SIGVTALRM`)
    - `ITIMER_PROF` : temps en mode utilisateur+noyau (`SIGPROF`)

# Programmer des horloges

- **L'appel système** `setitimer()`
  - La structure `itimerval` (champs de type `struct timeval`)
    - Champ `it_value` : délai avant prochain déclanchement
    - Champ `it_interval` : période pour recharger `it_value`
  - Désactivation si `it_value` est initialisé à `0`
  - `it_value` est décrémenté selon `which` jusqu'à `0`
    - Envoi du signal associé
    - Rechargement de `it_value` par `it_interval` si non nul
  - Retour : `0` si ok, `-1` si erreur (mauvais arguments)
  - Précautions
    - Interférences avec `alarm()` et `sleep()` (voir plus loin)
    - Précision $\equiv$ période du système ($\simeq$10 $ms$ physiques)
    - Mêmes précautions qu'avec `alarm()`

# Programmer des horloges

```
#include <signal.h>
#include <sys/time.h>
#include <unistd.h>
#include <stdio.h>

int flag=0;
void timerTask(int) { flag++; }

int main(void)
{
struct sigaction action;
sigemptyset(&(action.sa_mask));
action.sa_flags=0;
action.sa_handler=timerTask;
sigaction(SIGALRM,&action,(struct sigaction *)0); // Installer le gestionnaire
struct itimerval itv;
itv.it_value.tv_sec=itv.it_interval.tv_sec=0;
itv.it_value.tv_usec=itv.it_interval.tv_usec=500000; // Periode de 500 ms
setitimer(ITIMER_REAL,&itv,(struct itimerval *)0); // Programmer l'horloge
int old=flag;
while(flag<5) if(flag!=old) fprintf(stderr,"%d\n",old=flag);
return(0);
}
```

# Programmer des horloges

```
#include <signal.h>
#include <sys/time.h>
#include <unistd.h>
#include <stdio.h>

int nbReal=0,nbVirtual=0,nbProf=0;
void timerTask(int signum)
{
if(signum==SIGALRM) nbReal++;
if(signum==SIGVTALRM) nbVirtual++;
if(signum==SIGPROF) nbProf++;
}

int main(void)
{
struct sigaction action;
sigemptyset(&(action.sa_mask));
action.sa_flags=0;
action.sa_handler=timerTask;
sigaction(SIGALRM,&action,(struct sigaction *)0);
sigaction(SIGVTALRM,&action,(struct sigaction *)0);
sigaction(SIGPROF,&action,(struct sigaction *)0);
```

```
$ ./prog
R:1  V:0  P:0
R:1  V:1  P:1
R:2  V:1  P:1
R:2  V:2  P:2
R:3  V:2  P:2
R:3  V:3  P:3
R:4  V:3  P:3
R:4  V:4  P:4
R:5  V:4  P:4
...
R:597  V:574  P:583
R:597  V:575  P:583
R:597  V:575  P:584
R:598  V:575  P:584
R:598  V:576  P:584
R:598  V:576  P:585
R:599  V:576  P:585
R:599  V:577  P:585
R:599  V:577  P:586
R:600  V:577  P:586
$
```

# Programmer des horloges

```
struct itimerval itv;
itv.it_value.tv_sec=itv.it_interval.tv_sec=0;
itv.it_value.tv_usec=itv.it_interval.tv_usec=100000; // Periode de 100 ms
setitimer(ITIMER_REAL,&itv,(struct itimerval *)0);
setitimer(ITIMER_VIRTUAL,&itv,(struct itimerval *)0);
setitimer(ITIMER_PROF,&itv,(struct itimerval *)0);    // Programmer les 3 horloges
int oldReal=nbReal,oldVirtual=nbVirtual,oldProf=nbProf;
while(nbReal<600) // Afficher la progression pendant 1 minute
  {
  int show=0;
  if(oldReal!=nbReal) { oldReal=nbReal; show=1; }
  if(oldVirtual!=nbVirtual) { oldVirtual=nbVirtual; show=1; }
  if(oldProf!=nbProf) { oldProf=nbProf; show=1; }
  if(show) fprintf(stderr,"R:%d  V:%d  P:%d\n",nbReal,nbVirtual,nbProf);
  }
return(0);
}
```

# Mise en sommeil du processus

- **La fonction** `sleep()` (`man 3 sleep`)
  - `#include <unistd.h>`
    `unsigned int sleep(unsigned int seconds);`
  - Suspend le flot d'exécution courant pendant `seconds` secondes
  - Interruptible par les signaux
  - Retour : le nombre de secondes restantes
  - Précautions :
    - Interférences avec `alarm()` et `setitimer()` (`SIGALRM`)
    - Arrondi sur les secondes restantes

# Mise en sommeil du processus

- **La fonction** `usleep()` (`man 3 usleep`)
  - `#include <unistd.h>`
    `void usleep(unsigned long usec);`
  - Suspend le flot d'exécution courant pendant `usec` $\mu s$
  - N'utilise pas `SIGALRM`
  - Interruptible par les signaux
  - Pas de retour $\rightarrow$ durée restante inconnue
  - Précision $\simeq$ période du système
  - Non *POSIX* (origine *BSD*)

# Mise en sommeil du processus

- **L'appel système** `nanosleep()` (`man 2 nanosleep`)
  - `#include <time.h>`
    `int nanosleep(const struct timespec * req,`
    `struct timespec * rem);`
  - Suspend le flot d'exécution courant selon `req`
  - Indique éventuellement le temps restant dans `rem`
  - Structure `timespec` :
    - Champ `tv_sec` (`time_t`) : nombre de secondes
    - Champ `tv_nsec` (`long`) : nombre de nanosecondes
  - N'utilise pas `SIGALRM`

# Mise en sommeil du processus

- **L'appel système** `nanosleep()`
  - Interruptible par les signaux
  - Retour : `0` si ok, `-1` si erreur
  - Causes d'erreur (consulter `errno`) :
    - Mauvais arguments
    - `EINTR` → temps restant dans `rem` (si pointeur non nul)
  - En cas d'interruption → relance immédiate avec `rem`
    - Éviter les dérives trop importantes
  - Précision de l'ordre de la $ns$ illusoire ! (période du système)

# Mise en sommeil du processus

```
#include <signal.h>
#include <sys/time.h>
#include <unistd.h>
#include <errno.h>
#include <stdio.h>

unsigned long getTime(void) // Temps en millisecondes
{
struct timeval tv;
gettimeofday(&tv,(struct timezone *)0);
return(tv.tv_sec*1000LU+tv.tv_usec/1000LU);
}

void timerTask(int) { } // Juste pour interromptre

int main(int argc,char ** argv)
{
if(argc>1) // Si argument, lancer l'horloge
  {
```

```
$ ./prog
9504 ms elapsed (intr=0)
$
$ ./prog arg
9513 ms elapsed (intr=95)
$
```

# Mise en sommeil du processus

```
  struct sigaction action;
  sigemptyset(&(action.sa_mask));
  action.sa_flags=0;
  action.sa_handler=timerTask;
  sigaction(SIGALRM,&action,(struct sigaction *)0);
  struct itimerval itv;
  itv.it_value.tv_sec=itv.it_interval.tv_sec=0;
  itv.it_value.tv_usec=itv.it_interval.tv_usec=100000; // Interrompre toutes les 100 ms
  setitimer(ITIMER_REAL,&itv,(struct itimerval *)0);
  }
int intr=0;
struct timespec ts;
ts.tv_sec=9;               // Sommeil de 9.5 secondes demande
ts.tv_nsec=500*1000*1000;
unsigned long t0=getTime();
while((nanosleep(&ts,&ts)==-1)&&(errno==EINTR)) intr++; // Relancer si interrompu
fprintf(stderr,"%lu ms elapsed (intr=%d)\n",getTime()-t0,intr);
return(0);
}
```

# Mise en forme de la date et de l'heure

- **Expliciter le contenu de `time_t`**
  - Obtenu par l'appel système **`time()`** mais peu explicite tel quel
  - Fonctions de mise en forme déclarées dans **`time.h`**
  - La structure **`tm`** (que des **`int`**) définie dans **`time.h`**
    - **`tm_sec`** : seconde de la minute dans [0;59] ou [0;61]
    - **`tm_min`** : minute de l'heure dans [0;59]
    - **`tm_hour`** : heure du jour dans [0;23]
    - **`tm_mday`** : jour du mois dans [1;31]
    - **`tm_mon`** : mois de l'année dans [0;11]
    - **`tm_year`** : année après 1900 ($101 \equiv 2001$)
    - **`tm_wday`** : jour de la semaine après dimanche dans [0;6]
    - **`tm_yday`** : jour de l'année dans [0;365]
    - **`tm_isdst`** : >0 si heure d'été, 0 si heure d'hiver, <0 si inutile

# Mise en forme de la date et de l'heure

- **La fonction** `localtime()` (`man 3 localtime`)
  - `struct tm * localtime(const time_t * date);`
  - Décode `date` selon le fuseau horaire courant (heure locale)
  - Retour : adresse de la structure `tm` allouée **statiquement**
  - Recopier la structure ou ses champs avant tout nouvel appel
- **La fonction** `gmtime()` (`man 3 gmtime`)
  - `struct tm * gmtime(const time_t * date);`
  - Décode `date` selon le temps universel (ignore le fuseau horaire)
  - Retour : adresse de la structure `tm` allouée **statiquement**
  - Recopier la structure ou ses champs avant tout nouvel appel

# Mise en forme de la date et de l'heure

- **La fonction** `mktime()` (`man 3 mktime`)
  - `time_t mktime(struct tm * tmptr);`
  - Compose un `time_t` à partir de la date `tmptr` (heure locale)
  - Les champs `tm_wday` et `tm_yday` sont ignorés mais mis à jour
  - Retour : la date calculée ou `-1` si erreur
    - Limites de `time_t` ou `time_t` négatif (dépend du système)
    - `-1` peut être valide (1 seconde avant 01/01/1970) !
    - Inscrire une mauvaise valeur dans `tm_wday` ou `tm_yday`
    - Si déroulement ok ces valeurs sont corrigées
  - Procédé très permisif
    - Les données hors plage sont autorisées
    - Recadrage et ajustement des autres champs
    - Facilite le calcul sur les dates

# Mise en forme de la date et de l'heure

```
$ ./prog
Maintenant ... (j:221) jeudi 9/8/2001 16:27:32
Il y a 12 j 20 h et 42 m ... (j:208) vendredi 27/7/2001 19:45:32
70 ans avant ... (j:208) lundi 27/7/1931 19:45:32
30 ans avant ... ???
Maxi : (j:19) mardi 19/1/2038 3:14:7
Mini : (j:347) vendredi 13/12/1901 20:45:52
$
```

```
        #include <time.h>
        #include <stdio.h>

        void showDate(struct tm * date)
        {
        char * days[]={"dimanche","lundi","mardi","mercredi",
                       "jeudi","vendredi","samedi"};
        fprintf(stdout,"(j:%d) %s %d/%d/%d %d:%d:%d\n",
                date->tm_yday+1,days[date->tm_wday],
                date->tm_mday,date->tm_mon+1,date->tm_year+1900,
                date->tm_hour,date->tm_min,date->tm_sec);
        }
```

# Mise en forme de la date et de l'heure

```
int main(void)
{
time_t now=time((time_t *)0);
struct tm date=*(localtime(&now));
fprintf(stdout,"Maintenant ... "); showDate(&date);
fprintf(stdout,"Il y a 12 j 20 h et 42 m ... ");
date.tm_mday-=12; date.tm_hour-=20; date.tm_min-=42;
date.tm_wday=-1; mktime(&date);
if(date.tm_wday<0) fprintf(stdout,"???\n"); else showDate(&date);
fprintf(stdout,"70 ans avant ... "); date.tm_year-=70;
date.tm_wday=-1; mktime(&date);
if(date.tm_wday<0) fprintf(stdout,"???\n"); else showDate(&date);
fprintf(stdout,"30 ans avant ... "); date.tm_year-=30;
date.tm_wday=-1; mktime(&date);
if(date.tm_wday<0) fprintf(stdout,"???\n"); else showDate(&date);
now=0x7fffffff; date=*(gmtime(&now));
fprintf(stdout,"Maxi : "); showDate(&date);
now=0x80000000; date=*(gmtime(&now));
fprintf(stdout,"Mini : "); showDate(&date);
return(0);
}
```

# Mise en forme de la date et de l'heure

- **La fonction** `asctime()` (`man 3 asctime`)
  - `char * asctime(const struct tm * tmptr);`
  - Rédige la date décrite par `tmptr`
  - Format figé, en anglais
  - Retour : Une chaîne allouée **statiquement**
  - Recopier la chaîne ou l'afficher avant tout nouvel appel
- **La fonction** `ctime()` (`man 3 ctime`)
  - `char * ctime(const time_t * date);`
  - Rédige la date décrite par `date` (ignore le fuseau horaire)
  - Format figé, en anglais
  - Retour : Une chaîne allouée **statiquement**
  - Recopier la chaîne ou l'afficher avant tout nouvel appel
  - Équivalent à `asctime(localtime(date))`

# Mise en forme de la date et de l'heure

- **La fonction** `strftime()` (`man 3 strftime`)
  - `size_t strftime(char * buf,size_t length,`
    `const char * format,`
    `const struct tm * tmptr);`
  - Rédige dans `buf`, selon `format`, la date décrite par `tmptr`
  - `buf` doit pouvoir contenir `length` octets
  - `format` évoque le format de `fprintf()` mais est très différent
    - `"%F"` ≡ `"%Y-%m-%d"` : année-mois-jour
    - `"%A"` : le nom du jour
    - `"%B"` : le nom du mois
    - De nombreux autres . . . (voir le manuel)
    - Prend en compte la localisation
  - Retour : Nombre de caractères (sans `'\0'`) ou `0` si `buf` trop petit

# Mise en forme de la date et de l'heure

```
#include <time.h>
#include <stdio.h>
#include <locale.h>

int main(int argc,char ** argv)
{
if(argc>1) setlocale(LC_ALL,argv[1]);
char buffer[0x100];
time_t now=time((time_t *)0);
struct tm date=*(localtime(&now));
strftime(buffer,0x100,"%F",&date);
fprintf(stdout,"%s\n",buffer);
strftime(buffer,0x100,"%A %d %B",&date);
fprintf(stdout,"%s\n",buffer);
strftime(buffer,0x100,"%c",&date);
fprintf(stdout,"%s\n",buffer);
return(0);
}
```

```
$ ./prog
2001-08-09
Thursday 09 August
Thu Aug  9 17:08:57 2001
$ ./prog en
2001-08-09
Thursday 09 August
Thu 09 Aug 2001 05:09:03 PM CEST
$ ./prog fr
2001-08-09
jeudi 09 aout
jeu 09 aou 2001 17:09:05 CEST
$
```

# Mise en forme de la date et de l'heure