Linux et le temps

Eurogiciel

Agence de Rennes
22 rue Rigourdière - 35510 Cesson Sévigné

26 janvier 2011

1 Gestion du temps
- Notions élémentaires
- Fonctions et appels systèmes utiles
- Structures de données
- Exemples pratiques

1. Gestion du temps
   - Notions élémentaires
   - Fonctions et appels systèmes utiles
   - Structures de données
   - Exemples pratiques
2. Problématique d'ordonnancement de tâches
   - Présentation de l'ordonnanceur
   - Appels systèmes utiles pour les processus
   - Structures de données

1. Gestion du temps
   - Notions élémentaires
   - Fonctions et appels systèmes utiles
   - Structures de données
   - Exemples pratiques
2. Problématique d'ordonnancement de tâches
   - Présentation de l'ordonnanceur
   - Appels systèmes utiles pour les processus
   - Structures de données
3. Mise en œuvre du temps réel mou
   - Programmation d'un processus périodique
   - Programmation d'un thread périodique
   - Programmation de threads concurrents

1. Gestion du temps
   - Notions élémentaires
   - Fonctions et appels systèmes utiles
   - Structures de données
   - Exemples pratiques
2. Problématique d'ordonnancement de tâches
   - Présentation de l'ordonnanceur
   - Appels systèmes utiles pour les processus
   - Structures de données
3. Mise en œuvre du temps réel mou
   - Programmation d'un processus périodique
   - Programmation d'un thread périodique
   - Programmation de threads concurrents
4. Conclusion

1. Gestion du temps
   - Notions élémentaires
   - Fonctions et appels systèmes utiles
   - Structures de données
   - Exemples pratiques

Objectifs des services liés au temps

Propriétés attendues des horloges et chronomètres

Objectifs des services liés au temps

- Connaissance de la date et de l'heure courantes
- Évaluation de durées
- Définition d'échéances
- Valeurs temporelles absolues et relatives
- Manipulation et comparaison de dates, d'heures et de durées
- Définition des caractéristiques temporelles de traitements

Propriétés attendues des horloges et chronomètres

Objectifs des services liés au temps

- Connaissance de la date et de l'heure courantes
- Évaluation de durées
- Définition d'échéances
- Valeurs temporelles absolues et relatives
- Manipulation et comparaison de dates, d'heures et de durées
- Définition des caractéristiques temporelles de traitements

Propriétés attendues des horloges et chronomètres

- Résolution adaptée
- Précision et déterminisme

Sources de temps disponibles

Sources de temps disponibles

- Horloge matérielle (RTC)
- Compteur de cycles processeur (registre TSC sur x86)
- Compteur événementiel haute performance de temps global (HPET)
- Horloge haute résolution (registre IOMEM)
- Horloge réseau (NTP)

Initialisation de l'horloge système

Initialisation de l'horloge système

- Synchronisation au démarrage du système
- Lecture de la zone mémoire réservée à la RTC par la fonction `get_cmos_time()`
- Initialisation de la variable système `xtime` dont le type est une structure `timespec`
- Renseignement de champ `xtime.tv_sec` avec le nombre de secondes écoulées depuis l'Epoch (1er janvier 1970, 0h00 UTC)
- Renseignement du champ `xtime.tv_nsec` de façon à synchroniser le changement de seconde avec RTC

Périodicité du tick d'horloge (1/2)

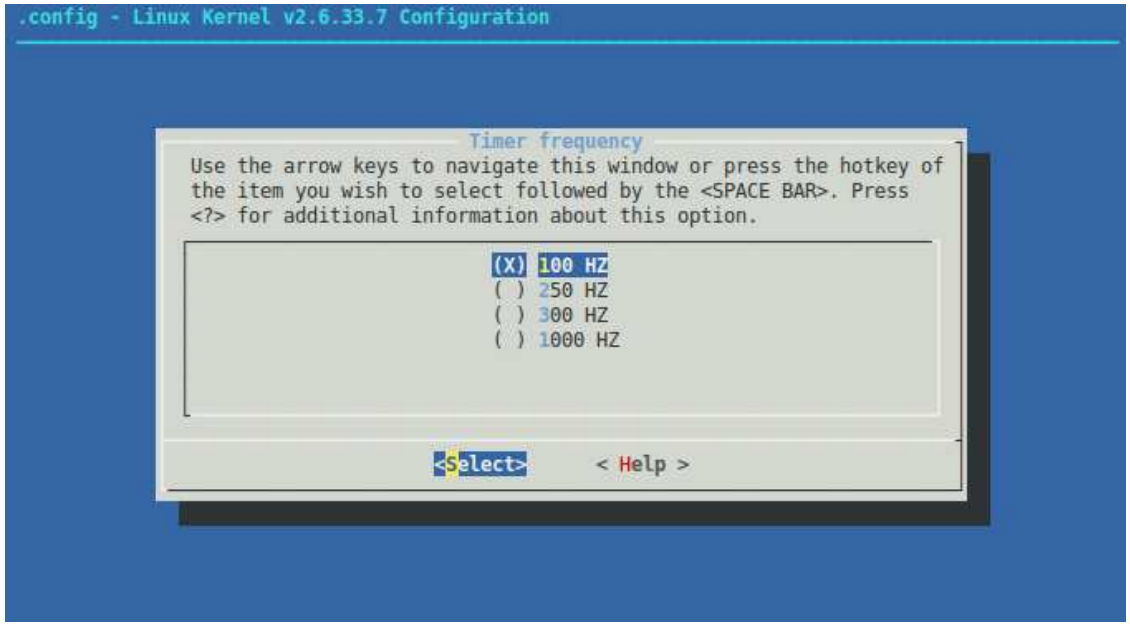
Périodicité du tick d'horloge (1/2)

- Définition de la granularité d'horloge
- Déclaration au démarrage du noyau d'un périphérique matériel capable de fournir une interruption périodique de fréquence HZ
- Définition par la constante HZ du nombre de tics d'horloge par seconde correspondant à l'intervalle de temps entre deux interruptions du HPET
  - Temps écoulé = nombre d'interruptions/HZ
  - Précision = 1/HZ
  - HZ = 100 $\rightarrow$ 10 ms
  - HZ = 250 $\rightarrow$ 4 ms
  - HZ = 300 $\rightarrow$ 3.3333 ms
  - HZ = 1000 $\rightarrow$ 1 ms
- Définition de HZ dans `include/asm-generic/param.h`

Périodicité du tick d'horloge (2/2)

Périodicité du tick d'horloge (2/2)

- Définition de la constante HZ par configuration et compilation du noyau
- Menu `Processor type and features` → `Timer frequency`



Notion de jiffies

Notion de jiffies

- Variable système global initialisée au démarrage du système
- Comptage permanent du nombre d'interruptions HPET
- Définition dans `<linux/jiffies.h>`
- Utilisation des fonctions `jiffies_to_timespec()` `timespec_to_jiffies()` définies dans `<linux/time.h>` pour la conversion
- Correspondance entre un jiffy, un tick d'horloge et un HZ
- Récupération par l'appel système `sysconf()` argumenté de la constante système `_SC_CLK_TCK`

Mécanismes de contrôle

Mécanismes de contrôle

- Gestion globale du temps par des interruptions matérielles
- Mise en œuvre par le noyau d'un gestionnaire d'interruption `timer`
- Datation par le noyau du temps écoulé depuis la première interruption
- Enregistrement l'interruption périodique dans le fichier `/proc/interrupts`
- Démonstration par la commande `(cat /proc/interrupts && sleep 2 && cat /proc/interrupts) | grep -i timer`

Rôle du gestionnaire d'interruption `timer`

Rôle du gestionnaire d'interruption `timer`

- Mise à jour des durées d'activité et d'inactivité du système
- Renseignement du fichier `/proc/uptime`
- Mise à jour de la date et de l'heure courantes
- Équilibrage des charges de traitement entre les processeurs d'un système multi-processeurs
- Exécution des temporisations arrivées à échéance

Appel système `gettimeofday()`

Appel système `gettimeofday()`

- Prototype `int gettimeofdaystruct(timeval *tv, struct timezone *tz);`
- Définition dans `<sys/times.h>`
- Mesure d'un temps réel incluant tous les temps parasites
- Précision dépendante de celle du HPET
- Renvoi dans une structure de type `timeval` le nombre de secondes (champ `tv_sec`) et de microsecondes (champ `tv_usec`) écoulées depuis l'Epoch
- Mesure d'un temps réel écoulé par la différence des résultats de deux appels avec une précision de l'ordre de la microseconde
- Disponibilité d'une fonction réciproque `int settimeofday(const struct timeval *tv, const struct timezone *tz);`

Fonction `clock()`

Fonction `clock()`

- Prototype `clock_t clock(void);`
- Définition dans `<time.h>`
- Mesure théorique du temps processeur écoulé sous forme de ticks d'horloge durant l'exécution d'un processus
- Évaluation cumulée des temps passés en espaces utilisateur et noyau
- Conversion de la durée en secondes par division à l'aide de la constante `CLOCKS_PER_SEC`
- Obtention de résultat approximé
- Limitation de durée en raison du type `clock_t` de retour correspondant au type `long int`

Fonction `time()`

Fonction `time()`

- Prototype `time_t time(time_t *t);`
- Définition dans `<time.h>`
- Argument pouvant être nul
- Renvoi de le nombre de secondes écoulées depuis l'Epoch
- Précision à la seconde
- Application essentiellement dédiée à une gestion calendaire
- Peu d'intérêt dans le cadre d'un système temps réel

Appel système `times()`

Appel système `times()`

- Prototype `clock_t times(struct tms *buf);`
- Définition dans `<sys/times.h>`
- Renvoi dans une structure `tms` le temps passé par un processus appelant en espace utilisateur (`tms_utime`), en espace noyau (`tms_stime`) et les temps des processus fils terminés (`tms_cutime` et `tms_cstime`)
- Accès aux compteurs de temps des processus
- Mesure du temps exprimée en nombre de ticks
- Précision de la mesure de l'ordre de la milliseconde
- Exclusion des moments pendant lesquels le processus évalué est endormi
- Mesure d'un temps effectif d'exécution d'un bout de code par la différence des résultats de deux appels avec une précision de l'ordre de la milliseconde

Appels système `getitimer()` et `setitimer()` (1/2)

Appels système `getitimer()` et `setitimer()` (1/2)

- Prototype `int getitimer(int which, struct itimerval *value);`
- Prototype `int setitimer(int which, const struct itimerval *value, struct itimerval *ovalue);`
- Définition dans `<sys/time.h>`
- Lecture et écriture d'une temporisation de précision
- Limitation de la précision de temporisation à résolution d'horloge système (1/HZ)
- Mise à disposition de trois types de temporisation dont la sélection s'effectue par affectation de l'argument `which`

Appels système `getitimer()` et `setitimer()` (2/2)

Appels système `getitimer()` et `setitimer()` (2/2)

- Choix du type de temporisation
  - `ITIMER_REAL` : Décompte en temps réel de la temporisation et envoi du signal `SIGALARM` au processus au terme du décompte
  - `ITIMER_VIRTUAL` : Décrémentation de la temporisation uniquement lorsque le processus s'exécute dans l'espace utilisateur et envoi du signal `SIGVTALRM` au terme du décompte
  - `ITIMER_PROF` : Décompte de la temporisation lorsque le processus s'exécute dans l'espace utilisateur, mais aussi durant les appels système dans l'espace noyau, et envoi du signal `SIGPROF` au terme du décompte
- Utilisation de `ITIMER_REAL` plus courante en raison de sa plus grande précision

Appel système `ioctl()` pour la RTC (1/2)

Appel système `ioctl()` pour la RTC (1/2)

- Prototype `int ioctl(int fd, RTC_request, param);`
- Définition dans `<sys/rtc.h>`
- Interfaçage avec les horloges matérielles intégrées du système
- Disponibilité minimale d'une horloge matérielle
- Accessibilité via un périphérique `/dev/rtc` ou `/dev/rtc0`, `/dev/rtc1`, etc.
- Utilisation d'un descripteur de fichier `fd` donné en premier argument
- Déclaration de l'argument `param` en fonction de la requête passée en second argument

Appel système `ioctl()` pour la RTC (2/2)

Appel système `ioctl()` pour la RTC (2/2)

- Liste des principales requêtes `RTC_request` disponibles
  - `RTC_RD_TIME` : Renvoi de la date et de l'heure dans une structure `rtc_time` indiquée en troisième argument
  - `RTC_SET_TIME` : Réglage de l'horloge en fonction des paramètres définis dans la structure `rtc_time` indiquée en troisième argument
  - `RTC_PIE_ON` : Activation de l'interruption périodique et non considération du troisième argument
  - `RTC_PIE_OFF` : Désactivation de l'interruption périodique et non considération du troisième argument
  - `RTC_IRQP_READ` : Lecture de la fréquence des interruptions par seconde et récupération de la valeur par le troisième argument
  - `RTC_IRQP_SET` : Réglage de la fréquence des interruptions par seconde en fonction de la valeur donnée par troisième argument

Structure `timespec`

```
struct timespec
{
  time_t tv_sec; /* Secondes */
  long tv_nsec; /* Nanosecondes */
}
```

Structure `timeval`

```
struct timeval
{
  time_t tv_sec; /* Secondes */
  suseconds_t tv_usec; /* Microsecondes */
};
```

Structure `itimerval`

```
struct itimerval
{
  struct timeval it_interval; /* Valeur suivante */
  struct timeval it_value; /* Valeur actuelle */
};
```

Structure `timezone`

```
struct timezone
{
  int tz_minuteswest; /* Minutes par rapport au méridien de Greenwich */
  int tz_dsttime; /* Type de correction DST */
};
```

Constantes symboliques pour le DST

```
DST_NONE /* Aucun */
DST_USA /* USA */
DST_AUST /* Australie */
DST_WET /* Europe de l'ouest */
DST_MET /* Europe centrale */
DST_EET /* Europe de l'est */
DST_CAN /* Canada */
DST_GB /* Grande Bretagne et Eire */
DST_RUM /* Roumanie */
DST_TUR /* Turquie */
DST_AUSTALT /* Australie (1986) */
```

Structure `tm`

```
struct tm
{
  int tm_sec; /* Secondes */
  int tm_min; /* Minutes */
  int tm_hour; /* Heures */
  int tm_mday; /* Quantième du mois */
  int tm_mon; /* Mois (0 à 11) */
  int tm_year; /* Année (depuis 1900) */
  int tm_wday; /* Jour de la semaine */
  int tm_yday; /* Jour de l'année */
  int tm_isdst; /* Décalage horaire */
};
```

Structure `tms`

```
struct tms
{
  clock_t tms_utime; /* Durée en espace utilisateur */
  clock_t tms_stime; /* Durée en espace noyau */
  clock_t tms_cutime; /* Durée en espace utilisateur des processus fils */
  clock_t tms_cstime; /* Durée en espace noyau des processus fils */
};
```

Structure `rtc_time`

```
struct rtc_time
{
  int tm_sec; /* Seconde */
  int tm_min; /* Minute */
  int tm_hour; /* Heure */
  int tm_mday; /* Jour */
  int tm_mon; /* Mois */
  int tm_year; /* Année */
  int tm_wday; /* Non utilisé */
  int tm_yday; /* Non utilisé */
  int tm_isdst; /* Non utilisé */
};
```

Récupération du HZ

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main(void)
{
  long resultat;

  resultat = sysconf(_SC_CLK_TCK);
  printf("HZ : %ld\n", resultat);
  return EXIT_SUCCESS;
}
```

Relevé de date et heure

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

int main(void)
{
  time_t t1;

  t1 = time(NULL);

  if (t1 == (time_t) -1)
  {
    perror("time");
    return EXIT_FAILURE;
  }

  printf("%ld secondes, %s", t1, ctime(&t1));
  return EXIT_SUCCESS;
}
```

Mesure du temps écoulé (1/2)

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

#define MAX 10000000

int main(void)
{
  long clk_tck = CLOCKS_PER_SEC;
  clock_t t1, t2;
  int i;

  /* Temps initial en ticks d'horloge */
  t1 = clock();

  /* Traitement */
  for (i = 0; i < MAX; i++)
  {
    printf("=");
  }

  printf("\n");
```

Mesure du temps écoulé (2/2)

```
    /* Temps final en ticks d'horloge */
    t2 = clock();

    /* Affichage des différents temps */
    printf("Nb ticks/seconde : %ld\n", clk_tck);
    printf("Nb ticks initial : %ld\n", (long)t1);
    printf("Nb ticks final : %ld\n", (long)t2);
    printf("Temps ecoule (sec.) : %lf\n", (double)(t2-t1) / (double)clk_tck);
    return EXIT_SUCCESS;
}
```

Mesure précise du temps écoulé (1/2)

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

#define MAX 1000

int main(void)
{
  struct timeval tv1,tv2;
  struct timezone tz;
  long long diff;
  int i;

  /* Temps initial */
  gettimeofday(&tv1, &tz);

  /* Traitement */
  for (i = 0; i < MAX; i++)
  {
    printf("=");
  }

  printf("\n");
```

Mesure précise du temps écoulé (2/2)

```
    /* Temps final */
    gettimeofday(&tv2, &tz);

    /* Calcul et affichage du temps ecoule */
    diff = (tv2.tv_sec - tv1.tv_sec) * 1000000L + (tv2.tv_usec-tv1.tv_usec);
    printf("Temps ecoule (microsec.) : %lld\n", diff);
    return EXIT_SUCCESS;
}
```

Évaluation des durées d'activité et d'inactivité du système (1/2)

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

/* Fonction de conversion et d'affichage */
void affichage (char *libelle, long delai)
{
  const long minute = 60;
  const long heure = minute * 60;
  const long jour = heure * 24;

  printf("%s: %ld jours, %ld:%02ld:%02ld\n", libelle, delai / jour,
         (delai % jour) / heure, (delai % heure) / minute, delai % minute);
}

/* Routine principale */
int main(void)
{
  FILE *fp;
  double activite, inactivite;

  if ((fp = fopen("/proc/uptime", "r")) < 0)
  {
    perror("fopen");
    return EXIT_FAILURE;
  }
```

Évaluation des durées d'activité et d'inactivité du système (2/2)

```
    fscanf(fp, "%lf %lf\n", &activite, &inactivite);
    fclose(fp);

    affichage("Temps d'activite ", (long)activite);
    affichage("Temps d'inactivité ", (long)inactivite);
    return EXIT_SUCCESS;
}
```

Mesure de consommation de temps processeur (1/2)

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

#define MAX 100000000
#define SIZE 100000

int main(void)
{
  long int i;
  double duree_utilisateur, duree_noyau;
  int *temp;
  struct utsname noyau;
  struct tms mesure;

  /* Traitement en espace utilisateur */
  for (i = 0 ; i < MAX ; i++)
  {
    temp = (int *)malloc(sizeof(int) * SIZE);
    free(temp);
  }
```

Mesure de consommation de temps processeur (2/2)

```
  /* Traitement en espace noyau */
  for (i = 0 ; i < MAX ; i++)
  {
    uname(&noyau);
  }

  /* Calcul des temps ecoules */
  times(&mesure);
  duree_utilisateur = (mesure.tms_utime);
  duree_noyau = (mesure.tms_stime);

  /* Affichage des resultats */
  printf("Temps CPU en espace utilisateur : %f\n", duree_utilisateur);
  printf("Temps CPU en espace noyau : %f\n", duree_noyau);
  return EXIT_SUCCESS;
}
```

2 Problématique d'ordonnancement de tâches
- Présentation de l'ordonnanceur
- Appels systèmes utiles pour les processus
- Structures de données

Rôles principaux

Considérations complémentaires

Rôles principaux

- Sélection de la tâche la plus prioritaire et attribution du processeur
- Mise en place d'un mécanisme de répartition de la ressource processeur
- Garantie d'exécution de toutes les tâches
- Recherche du meilleur rendement d'exécution et l'enchaînement optimal des tâches
- Possibilités d'application de plusieurs disciplines d'ordonnancement

Considérations complémentaires

Rôles principaux

- Sélection de la tâche la plus prioritaire et attribution du processeur
- Mise en place d'un mécanisme de répartition de la ressource processeur
- Garantie d'exécution de toutes les tâches
- Recherche du meilleur rendement d'exécution et l'enchaînement optimal des tâches
- Possibilités d'application de plusieurs disciplines d'ordonnancement

Considérations complémentaires

- Minimisation des permutations de contexte
- Prise en compte des contraintes de précédence entre les tâches
- Choix du processeur en cas de système multi-processeurs

Appel système `nice()`

Appel système `nice()`

- Prototype `int nice(int inc);`
- Définition dans `<unistd.h>`
- Changement du niveau de courtoisie d'un processus
- Plage de -20 à 19 pour le superutilisateur (root)
- Plage de 0 à 19 pour un utilisateur
- Augmentation de l'argument `inc` correspondant à un abaissement de la priorité du processus

Appels système `getpriority()` et `setpriority()`

Appels système `getpriority()` et `setpriority()`

- Prototype `int getpriority(int which, int who);` et `int setpriority(int which, int who, int prio);`
- Définition dans `<sys/time.h>` et `<sys/resource.h>`
- Récupération ou affectation de la priorité d'un processus, d'un groupe de processus ou d'un utilisateur
- Définition de la portée en fonction de l'argument `which` qui peut être `PRIO_PROCESS`, `PRIO_PGRP`, `PRIO_USER`
- Interprétation de l'argument `who` relativement à l'argument `which`

Appels système `sched_getscheduler()` et `sched_setscheduler()`

Appels système `sched_getscheduler()` et `sched_setscheduler()`

- Prototype `int sched_getscheduler(pid_t pid);` et `int sched_setscheduler(pid_t pid, int policy, const struct sched_param *p);`
- Définition dans `<sched.h>`
- Récupération ou affectation de la discipline d'ordonnancement
- Affectation de la discipline d'ordonnancement par les valeurs `SCHED_FIFO`, `SCHED_RR`, `SCHED_OTHER` ou `SCHED_BATCH`
- Affectation de paramètres complémentaires par une structure de type `sched_param`
- Considération du processus courant lorsque l'argument `pid` est égal à `0`

Appel système `sched_yield()`

Appel système `sched_yield()`

- Prototype `int sched_yield(void);`
- Définition dans `<sched.h>`
- Abandon de l'exécution du processus courant sans blocage
- Déplacement du processus en fin de file de sa priorité

Appels système `sched_get_priority_min()` et `sched_get_priority_max()`

Appel système `sched_rr_get_interval()`

Appels système `sched_get_priority_min()` et `sched_get_priority_max()`

- Prototype `int sched_get_priority_min(int policy);` et `int sched_get_priority_max(int policy);`
- Définition dans `<sched.h>`
- Récupération des niveaux de priorité minimal et maximal pour la discipline passée par l'argument `policy`
- Renseignement de la discipline d'ordonnancement concernée par `SCHED_FIFO`, `SCHED_RR`, `SCHED_OTHER` ou `SCHED_BATCH`

Appel système `sched_rr_get_interval()`

Appels système `sched_get_priority_min()` et `sched_get_priority_max()`

- Prototype `int sched_get_priority_min(int policy);` et `int sched_get_priority_max(int policy);`
- Définition dans `<sched.h>`
- Récupération des niveaux de priorité minimal et maximal pour la discipline passée par l'argument `policy`
- Renseignement de la discipline d'ordonnancement concernée par `SCHED_FIFO`, `SCHED_RR`, `SCHED_OTHER` ou `SCHED_BATCH`

Appel système `sched_rr_get_interval()`

- Prototype `int sched_rr_get_interval(pid_t pid, struct timespec *interval);`
- Définition dans `<sched.h>`
- Récupération du quantum de temps utilisé pour la discipline `SCHED_RR`
- Renvoi du résultat dans un structure de type `timespec`

Structure sched_param

```
struct sched_param
{
  int32_t sched_priority; /* Nouvelle priorité */
  int32_t sched_curpriority; /* Priorité courante */
  union
  {
    int32_t reserved[8];
    struct
    {
      int32_t __ss_low_priority;
      int32_t __ss_max_repl;
      struct timespec __ss_repl_period;
      struct timespec __ss_init_budget;
    } __ss;
  } __ss_un;
};
```

3 Mise en œuvre du temps réel mou
- Programmation d'un processus périodique
- Programmation d'un thread périodique
- Programmation de threads concurrents

Condition préalable

Gestion de la périodicité

Gestion de l'ordonnancement

Condition préalable

- Passage en mode `root`

Gestion de la périodicité

Gestion de l'ordonnancement

Condition préalable

- Passage en mode `root`

Gestion de la périodicité

- Lecture du fichier `/dev/rtc`
- Contrôle via l'appel système `ioctl()`
  - Définition de la fréquence d'interruption par `RTC_IRQP_SET`
  - Activation de l'interruption par `RTC_PIE_ON`
  - Désctivation de l'interruption par `RTC_PIE_OFF`

Gestion de l'ordonnancement

Condition préalable

- Passage en mode `root`

Gestion de la périodicité

- Lecture du fichier `/dev/rtc`
- Contrôle via l'appel système `ioctl()`
  - Définition de la fréquence d'interruption par `RTC_IRQP_SET`
  - Activation de l'interruption par `RTC_PIE_ON`
  - Désctivation de l'interruption par `RTC_PIE_OFF`

Gestion de l'ordonnancement

- Définition de la priorité par le champ `sched_priority` de la structure `sched_param`
- Définition de la discipline d'ordonnancement par l'appel système `sched_setscheduler()`

Code (1/3)

```
#include <stdio.h>
#include <linux/rtc.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <sys/io.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <sched.h>

#define FREQ 64
#define LOOP 100

int main(void)
{
  int fd, i = 0;
  unsigned long data;
  struct sched_param sp ;
  struct timeval tv,tv2;
  struct timezone tz;
```

Code (2/3)

```
/* Definition de priorite "temps reel" */
sp.sched_priority = 60 ;
sched_setscheduler(0, SCHED_FIFO, &sp);

/* Programmation du RTC a FRQ Hz */
if ((fd = open("/dev/rtc", O_RDONLY)) == -1)
{
  perror("open");
  return EXIT_FAILURE;
}

ioctl(fd, RTC_IRQP_SET, FREQ);
ioctl(fd, RTC_PIE_ON, 0);

while (i < LOOP)
{
  /* Evalaution temporelle et attente de l'interruption suivante */
  gettimeofday(&tv, &tz);
  read(fd, &data, sizeof(unsigned long));
  gettimeofday(&tv2, &tz);

  /* Affichage */
  printf("%d %f millisec.\n",i, (tv2.tv_sec - tv.tv_sec) * 1000.0 +
         (tv2.tv_usec - tv.tv_usec) / 1000.0);

  i++;
}
```

Code (3/3)

```
    /* Suppression de l'interruption periodique */
    ioctl(fd, RTC_PIE_OFF, 0);
    close(fd);
    return EXIT_SUCCESS;
}
```

Gestion de la périodicité

Traitement de la tâche

Gestion de la périodicité

- Définition d'une temporisation périodique
- Déclenchement périodique d'un signal

Traitement de la tâche

Gestion de la périodicité

- Définition d'une temporisation périodique
- Déclenchement périodique d'un signal

Traitement de la tâche

- Déclaration d'une routine
- Exécution à chaque envoi du signal périodique

Code (1/4)

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

#define SIGNAL SIGALRM
#define TIMER ITIMER_REAL
#define PERIODE 10000

int latence_max = 0;
int avance_max = 0;

/* Gestion du signal d'interruption */
void interruption(int cause, siginfo_t *HowCome, void *ptr)
{
  static struct timeval tv;
  unsigned long long valeur_precedente, valeur_courante;
  int periode;
  static int flag = 0;
```

Code (2/4)

```
    if (flag == 0)
    {
      gettimeofday(&tv, NULL);
      flag = 1;
    }
    else
    {
      valeur_precedente = tv.tv_sec * 1000000 + tv.tv_usec;
      gettimeofday(&tv, NULL);
      valeur_courante = tv.tv_sec * 1000000 + tv.tv_usec;
      periode = (int)(valeur_courante - valeur_precedente);
      printf("%d microsec.\n", periode);

      if(periode - PERIODE > latence_max)
        latence_max =  periode - PERIODE;

      if (PERIODE - periode > avance_max)
        avance_max = PERIODE - periode;
    }
  }
```

Code (3/4)

```
/* Fonction du thread */
void *thread(void *nom)
{
  struct itimerval tempo;
  struct sigaction action;

  /* Déclaration du signal */
  action.sa_sigaction = interruption;
  sigemptyset(&action.sa_mask);
  action.sa_flags = SA_SIGINFO;

  if (sigaction(SIGNAL, &action, 0))
  {
    perror("sigaction");
    exit(1);
  }

  /* Parametrage de la temporisation */
  tempo.it_interval.tv_sec = 0;
  tempo.it_interval.tv_usec = PERIODE;
  tempo.it_value.tv_sec = 0;
  tempo.it_value.tv_usec = PERIODE;
  setitimer(TIMER, &tempo, NULL);
```

Code (4/4)

```
  /* Duree d'execution */
  sleep(5);

  printf("Latence maximale : %d\n", latence_max);
  printf("Avance maximale : %d\n", avance_max);
  return NULL;
}

/* Routine principale */
int main(void)
{
  pthread_t id_thread;
  pthread_attr_t attributs;

  pthread_attr_init(&attributs);
  pthread_create(&id_thread, &attributs, thread, NULL);
  pthread_join(id_thread, NULL);

  return EXIT_SUCCESS;
}
```

Condition préalable

Gestion de l'ordonnancement

Condition préalable

- Passage en mode `root`

Gestion de l'ordonnancement

- Définition de la priorité par le champ `sched_priority` de la structure `sched_param`
- Définition de la discipline d'ordonnancement par l'appel système `pthread_setschedparam()`

Code (1/4)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <errno.h>
#include <sched.h>

#define HAUTE_PRIORITE 5
#define BASSE_PRIORITE 1
#define FIFO SCHED_FIFO
#define RR SCHED_RR
#define MAX 100000000

pthread_t id_thread_hp, id_thread_bp;
unsigned long long hp = 0, bp = 0;

/* Fonction du thread de haute priorite */
void *thread_hp(void *arg)
{
  int discipline = FIFO;
  struct sched_param param;

  param.sched_priority = HAUTE_PRIORITE;
```

Code (2/4)

```
  if (pthread_setschedparam(pthread_self(), discipline, &param))
  perror("Thread HP : pthread_setschedparam");

  if (!pthread_getschedparam(pthread_self(), &discipline, &param))
  printf("HP -- Execution %s/%d\n", (discipline == SCHED_FIFO ? "FIFO" :
         (discipline == SCHED_RR ? "RR" :
         (discipline == SCHED_OTHER ? "OTHER" : "inconnu"))),
         param.sched_priority);
  else
    perror("Thread HP : pthread_getschedparam");

  sleep(1);

  do
  {
    hp++;
  } while (hp < MAX);

  printf("HP -- Compteur HP : %lld - Compteur BP : %lld - Ratio : %f\n",
         hp, bp, (double)hp / (double)bp);

  pthread_cancel(id_thread_bp);
  return NULL;
}
```

Code (3/4)

```
/* Fonction du thread de basse priorite */
void *thread_bp(void *nom)
{
  int discipline = FIFO;
  struct sched_param param;

  param.sched_priority = BASSE_PRIORITE;

  if (pthread_setschedparam(pthread_self(), discipline, &param))
    perror("Thread BP : pthread_setschedparam");

  if (!pthread_getschedparam(pthread_self(), &discipline, &param))
    printf("BP -- Execution  %s/%d\n", (discipline == SCHED_FIFO ? "FIFO" :
           (discipline == SCHED_RR ? "RR" :
           (discipline == SCHED_OTHER ? "OTHER" : "inconnu"))),
           param.sched_priority);
  else
    perror("Thread BP : pthread_getschedparam");

  sleep(1);
}
```

Code (4/4)

```
  do
  {
    bp++;
  } while(bp < MAX);

  printf("BP -- Compteur HP : %lld - Compteur BP : %lld - Ratio : %f\n",
         hp, bp, (double)hp / (double)bp);

  pthread_cancel(id_thread_hp);
  return NULL;

/* Routine principale */
int main(void)
{
  pthread_create(&id_thread_hp, NULL, thread_hp, NULL);
  pthread_create(&id_thread_bp, NULL, thread_bp, NULL);
  pthread_join(id_thread_hp, NULL);
  pthread_join(id_thread_bp, NULL);
  return EXIT_SUCCESS;
}
```

4 Conclusion

Bilan

Bilan

- Restriction d'exécution temps réel mou avec le noyau standard
- Considération en tant que tâche d'un processus ou d'un thread
- Pas de fonction, d'appel système ou de structure de données dédiés à la définition de la périodicité d'une tâche
- Performances relativement correctes de l'exécution temps réel du noyau standard

Questions

?