Ecole Temps Réel

www.lisyc.univ-brest.fr/pages_perso/babau/

Conception orientée objet appliquée à l'embarqué et au temps réel

Logiciel multitâche temps-réel POSIX

Jean-Philippe Babau

Département Informatique, UFR Sciences, UBO
Laboratoire LISyC

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

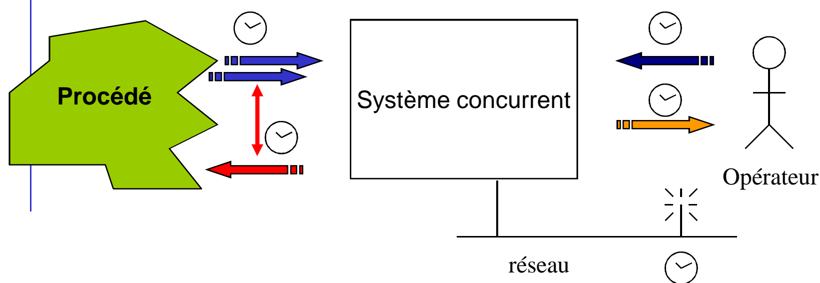
Conception multitâches

- Principes généraux sur les OS multitâches
- API POSIX
  - pthread
  - Communications
  - Gestion du temps
- Aspect réactif et concurrents
  - Exécution des actions
  - Déclenchement des threads
    - Prise en compte des événements
  - Échange de données
    - Concurrence et coopération
- Structuration
  - Création / destruction des éléments
  - Découpage

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Les systèmes temps réel embarqués communicants



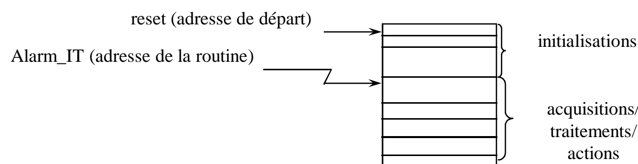
Contraintes temporelles

- acquisition, sorties, échéances

jean-philippe.babau@univ-brest.fr

Programmation

- Accès au matériel
- Langages de programmation au niveau système
  - Assembleur, C
- Programmation mono tâche (pas d'OS)
  - Programmation cyclique : une interruption (régulière, voire périodique) active un cycle de régulation
    - Acquisition via la lecture des capteurs
      - Filtrages, interprétations, mises en forme des données
    - Traitements
      - Surveillance, régulation, contrôle, réactions
    - Actions via l'écriture sur les actionneurs
      - Adaptation, protection
  - Largement répandu car simple de mise en œuvre, peu coûteux



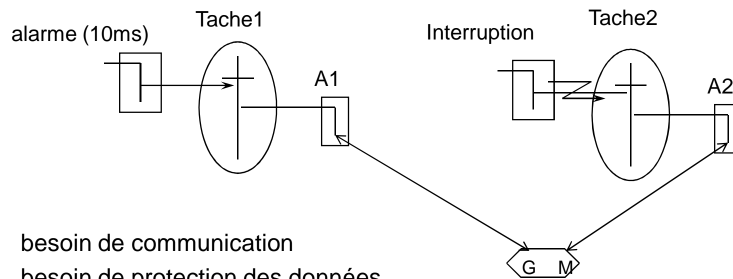
Programmation

- Programmation multitâches
  - Gestion concurrente de plusieurs interruptions
  - Gestion concurrente de plusieurs fréquences
  - Réponse en temps-réel
    - Priorité entre les actions pour respecter les délais
  - Gestion dynamique des entités concurrentes
- Système mono ou multi-processus
  - Gestion de plusieurs applications
  - Gestion de plusieurs espaces mémoire séparés
- Système monolotithiqe
  - Séparation de l'espace utilisateur et de l'espace noyau
  - Évolution du noyau
  - Intégration et gestion dynamique d'entités hétérogènes

Ecole Temps Réel

Programmation multitâches

- une activité A1 périodique (10ms)
- une activité A2 sur interruption



- besoin de communication
- besoin de protection des données

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

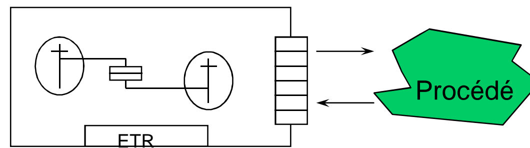
Systèmes d'exploitations multitâches

- Machine virtuelle (Java)
  - Extensions et limitations (PERC, JavaCard, EmbJava, RT-Java)
  - Cohabitation de deux systèmes concurrents (machine virtuelle et OS)
  - Gestion des IHM, de l'internet
- Système standard (**Linux**, NT)
  - Normes **POSIX**
  - Extensions temps réel (RTX, RT Linux)
  - Noyau temps réel (Windows CE)
- Système commercial
  - Adaptable, standard, librairies
- Système propriétaire
  - Coût réduit, maintenance, protection
  - Taille (petit noyau : de 4 à 40 ko ; gros noyaux : 250 ko; pour info noyau Linux : 1Mo)
- Pas d'exécutif

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Rôle de l'OS

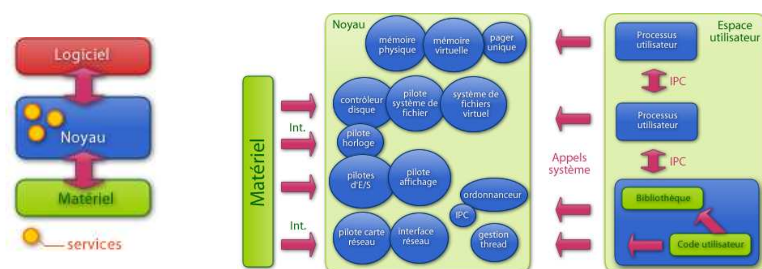


- A l'exécution
  - Gestion des interruptions et des interactions avec l'environnement (le procédé)
  - Gestion de l'état des entités concurrentes (threads)
  - Ordonnancement des threads
  - Rôle classique d'un OS
    - Gestion mémoire
    - Gestion des périphériques standards
    - Gestionnaire des fichiers
    - Communication réseau (TCP- IP, etc.)

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Architectures des systèmes monolithiques



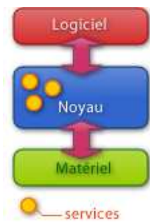
Schémas wikipédia

- Noyau non modulaire
  - Plus efficace
  - Problème de maintenance, d'évolutions, d'adaptations
- Noyaux monolithiques modulaires : **Linux**
  - Un driver = un module

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Architectures des systèmes monolithiques



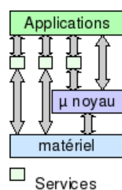
- Espace utilisateur (mode user)
  - Multi-processus et multi-thread (API POSIX)
  - Accès mémoire en mode protégé
    - Gestion par processus
- Espace noyau (mode kernel)
  - Mono-thread
  - Gestion des interruptions et des accès vers l'extérieur
  - Accès mémoire en mode non protégé
  - Programmation spécifique

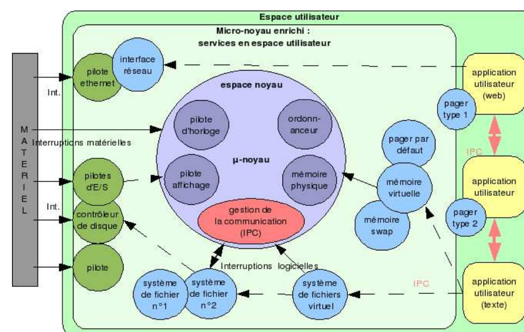
jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Architectures à micro-noyaux

- Noyau : limité au gestionnaire de mémoire, ordonnanceur, communication inter-processus
- Fonctions non critiques : espace utilisateur, mode protégé





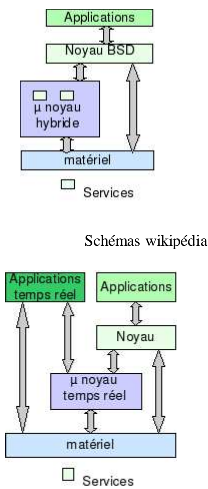
Schémas wikipédia

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Architectures hybrides et temps réel

- Noyau hybride
  - Micro-noyau + système monolithique
  - Services « utiles » implémentés dans le noyau
- Noyaux temps réel
  - Noyau pour les exécutifs simples
  - Noyau enrichi pour offrir des services
  - Noyau hybride pour l'encapsulation des services
    - RT-Linux, RTAI



Schémas wikipédia

jean-philippe.babau@univ-brest.fr

13

Ecole Temps Réel

Conception multitâches

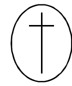
- Principes généraux sur les OS multitâches
- API POSIX
  - pthread
  - Communications
  - Gestion du temps
- Aspect réactif et concurrents
  - Exécution des actions
  - Déclenchement des threads
    - Prise en compte des événements
  - Échange de données
    - Concurrence et coopération
- Structuration
  - Création / destruction des éléments
  - Découpage

jean-philippe.babau@univ-brest.fr

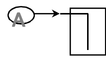
Ecole Temps Réel

LACATRE : représentation graphique d'entités

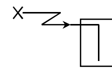
- Les « *objets* » programmables



la tâche / le thread

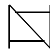


l'alarme



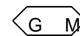
la routine d'interruption

- Les « *objets* » configurables



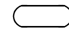
le sémaphore

la boîte aux lettres



la ressource

l'événement



message,
paramètres

jean-philippe.babau@univ-brest.fr

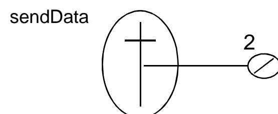
Ecole Temps Réel

LACATRE : principe

Forme graphique

sendData

2



Forme textuelle

```
TASK(sendData,132)
    FOREVER
    WAIT(2);
END_TASK
```

```
...
void * sendData(void* null)
{ /* local variables of the thread < sendData> */
    /* body of the thread <sendData> */
    while(1)
    {
        sleep(2);
    }
    pthread_exit(NULL);
}
```

Code C/POSIX

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

processus et thread Linux

- processus
  - Espace pour l'exécution d'un programme (main)
  - Process ID, répertoire de travail, …
  - Instructions (main)
  - Variables pour l'exécution
    - Etat des registres , pile, compteur de programme
  - Threads et objets de communication
  - Zone mémoire protégée
- Thread
  - Procédure (pas de retour) pour une exécution concurrente au sein du processus
  - Données spécifiques au thread
    - Variables pour permettre le changement de contexte
      - Etat des registres , pile, compteur de programme
    - Politique d'ordonnancement du thread
    - Liste de signaux en attente
    - Données applicatives

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

POSIX

- Gestion des threads
  - pthread et mutex
  - Éléments de communication
  - Gestion du temps
- IEEE POSIX 1003.1c **standard** (1995)
  - Non spécifique à Linux
  - API en langage C
- Performances
  - Mesure de 50 000 créations sur un Intel 2.4 Ghz en mode user

    → fork() 1.54 s

    → pthreads_create() 0.67 s
- Programmation et compilation

```
#include <pthread.h>
gcc –pthread SimulSensor.c -o SimulSensor
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Création de thread

- main()
  - pthread par défaut, attributs par défaut

- Création d'un nouveau thread

int pthread_create(pthread_t * threadID, const pthread_attr_t *attr, void *(*start_routine), void *arg);

Un identifiant logique
Les attributs du thread, si NULL attributs par défaut
L'adresse de la fonction à exécuter
Les arguments applicatifs utilisés lors de la création du thread

- Exemple de création d'un nouveau thread

```
void * sendData(void * null) {          …}
int main(void)
{     pthread_t sendDataID;
      int cr;
      cr = pthread_create(&sendDataID, NULL, sendData, NULL); …}
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Arrêt d'un thread

- Thread se terminant lui même
  - Appel à pthread_exit

    void pthread_exit(void *value_ptr);
  - A la fin de la fonction (au lieu de return)
    - Passage possible d'un paramètre (void *) lors de l'arrêt ou NULL
- Remarques
  - si le main se termine sur pthread_exit(), les autres threads continuent à s'exécuter
  - si le main se termine sans appeler pthread_exit(), tous les threads s'arrêtent
  - L'appel à pthread_exit() ne ferme pas les fichiers ouverts (cf. cours drivers)

- Exemple

```
void *PrintHello( void * null)
{
  printf(« Hello\n");
  phread_exit(NULL);
}
```

```
void *PrintHello( void *number)
{
  printf(« Hello\n");
  phread_exit( (void*) number);
}
```

jean-philippe.babau@univ-brest.fr

Arrêt d'un thread

- Arrêt par un autre thread

```
int pthread_cancel(pthread_t threadID);
```

  - pour le thread arrêté, appel implicite à pthread_exit(PTHREAD_CANCELED)
  - Peut être ignoré, différé ou traité immédiatement

- Arrêt du processus
  - Dans le programme par appel à exit() ou exec()
  - Commande shell d'arret du processus

Attributs de thread

- Comportement du thread caractérisé par des attributs
  - Initialisation lors de la création
    - Donnée de type pthread_attr_t myThreadAttributes;
    - Attributs par défaut (paramètre NULL à la création)
  - Modifiable après la création du thread

- Gestion des attributs avant création
  - Structure spécifique à initialiser / libérer

```
pthread_attr_init(& myThreadAttributes);
pthread_create(&sendDataID, myThreadAttributes, functionName, NULL);
pthread_attr_destroy(& myThreadAttributes);
```

  - Consultation/modification de myThreadAttributes via une API

```
int pthread_attr_getstacksize(const pthread_attr_t * attr, size_t *stacksize);
int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
```

- Gestion directe des attributs avec l'ID du thread
  - Exemple int sched_setscheduler(pid_t *pid, int policy, const struct sched_param *p);*
  - Gestion par un thread de ses propres attibuts (pid = 0)

Synchronisation sur l'arrêt d'un thread

- Thread se terminant
  – Création avec l'attribut PTHREAD_CREATE_JOINABLE

```
pthread_t threadID;
pthread_attr_t myThreadAttributes;
pthread_attr_init(& myThreadAttributes);

pthread_attr_setdetachstate(& myThreadAttributes, PTHREAD_CREATE_JOINABLE);
pthread_create(&threadID, & myThreadAttributes, functionName, NULL);
```

  – Appel à pthread_exit

- Thread se synchronisant sur l'arrêt du thread se terminant
  – Appel à pthead_join

```
int cr;
void * status
cr = pthread_join(threadID, &status);
```

  – Appel bloquant si le thread n'est pas fini, pas d'attente sinon
  – Récupération du paramètre passé à pthread_exit via status

jean-philippe.babau@univ-brest.fr

Passage d'argument

- Dernier argument de pthread_create
  – Passage par référence avec un cast en void*
- Exemple

```
#include <pthread.h>
#include <stdio.h>
#define MAX_THREADS 5

void *PrintHello( void *num )
{     long numero= (long) num+1;
      printf(« Hello thread  %d!\n", numero);
      phread_exit(NULL); }

int main (void)
{     pthread_t threads[MAX_THREADS];
      int cr; long threadNumber;
      for(threadNumber =0; threadNumber <MAX_THREADS; threadNumber ++)
      { cr = pthread_create(&threads[threadNumber], NULL, PrintHello, (void *) threadNumber);
              if (cr){ printf("ERROR sur pthread_create() : %d\n", cr); exit(-1); }
      } pthread_exit(NULL); }
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Ordonnancement des threads

- Principes
  - Politique par thread
    - SCHED_OTHER : processus normal, non temps réel, valeur par défaut
    - SCHED_FIFO : à priorité, FIFO pour le même niveau de priorité
    - SCHED_RR : à priorité, round-robin (configurable par niveau) pour le même niveau de priorité
  - Ordonnancement global des threads sur le système
  - Au moins 32 niveaux de priorité
    - 0 : SCHED_OTHER
    - Si 100 niveaux, la priorité la plus élevée est 99
  - Même politique pour un niveau de priorité
  - Il faut être SU pour passer en police SCHED_FIFO ou SCHED_RR

- Exemple de répartition de niveaux

  Niveaux 1-32 : SCHED_FIFO

  Niveaux 32-64 : SCHED_RR

  Niveaux 65-99 : SCHED_FIFO

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Ordonnancement des threads

- Définition de la politique d'ordonnancement du thread (#include <sched.h>)

  int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);

  - Policy : SCHED_OTHER, SCHED_FIFO, SCHED_RR

    pthread_attr_setschedpolicy(& myThreadAttributes, SCHED_RR);

  - Paramètres (priority pour SCHED_FIFO et SCHED_RR)

```
int pthread_attr_setschedparam(pthread_attr_t *restrict attr, struct sched_param *param);

struct sched_param myThreadSecheduleParameters;
 myThreadSecheduleParameters.sched_priority = myPriorityLevel;
pthread_attr_setschedparam (& threadAttributes, & myThreadSecheduleParameters)
ou  sched_setparam(0, & myThreadSecheduleParameters);

ou sched_setscheduler(0, SCHED_RR, myThreadSecheduleParameters);
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Ordonnancement des threads

- Partage d'une même politique/niveau par plusieurs threads
  - Définition d'une politique et d'un niveau de priorité
    - dans le thread qui crée les autres threads
  - Création des autres threads
    - Valeur par défaut (PTHREAD_INHERIT_SCHED) : même paramètres d'ordonnancement
    - Attribut à PTHREAD_EXPLICIT_SCHED pour ne pas hériter de la politique et du niveau de priorité

```
int pthread_attr_setinheritsched(pthread_attr_t *attr, int inheritsched);
```

- Niveaux de priorités (1-99)

```
int maxLevel = sched_get_priority_max(policy);
int  minLevel = sched_get_priority_min(policy);
```

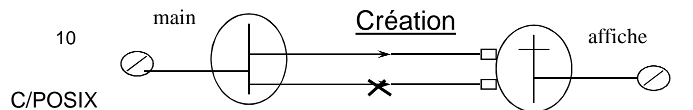
- Intervalle de temps pour SCHED_RR (~ 100 ms)

```
struct timespec tm;
sched_rr_get_interval_max(0,&tm);
tm.tv-nsec de type long
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Hello world



C/POSIX

```
#include <stdio.h>
#include <pthread.h>
pthread_t      afficheID;

void * affiche(void * null)
{ int i = 1;
        while (1)
        {
                sleep(1);
                printf("iteration : %i\n",i); i++;
        }
        pthread_exit(NULL);
}

int main(void)
{       pthread_create(&afficheID,NULL,affiche,NULL);
        sleep(10);
        pthread_cancel(afficheID);
        pthread_exit(NULL);
}
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Conception multitâches

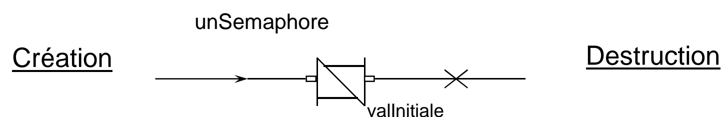
- Principes généraux sur les OS multitâches
- API POSIX
  - pthread
  - Communications
  - Gestion du temps
- Aspect réactif et concurrents
  - Exécution des actions
  - Déclenchement des threads
    - Prise en compte des événements
  - Échange de données
    - Concurrence et coopération
- Structuration
  - Création / destruction des éléments
  - Découpage

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Le sémaphore

Création



Destruction

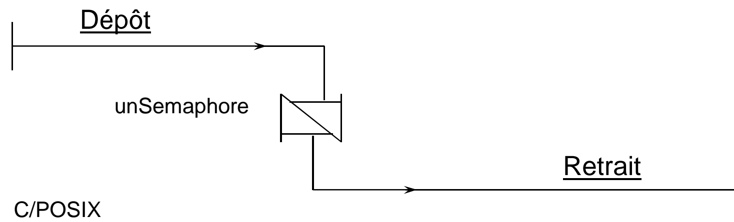
C/POSIX

```
#include <semaphore.h>

sem_t unSemaphore; int cr; unsigned int valeurInitiale; int partage;

sem_init(& unSemaphore, partage, valeurInitiale);
// si partage est nul : le sémaphore est limité aux threads du processus appelant
// si partage est non nul : le sémaphore n'est pas limité aux threads du processus appelant

cr = sem_destroy(& unSemaphore)
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Le sémaphore



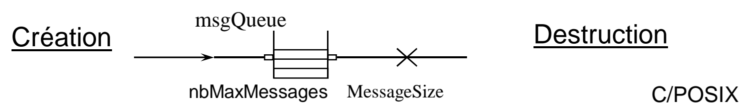
C/POSIX

```
int cr, valeur;
cr = sem_post(& unSemaphore);                   // valeur courante du sémaphore ++
cr = sem_wait(& unSemaphore);                   // attend que la valeur courante du sémaphore>0, puis --
cr = sem_trywait(& unSemaphore);                // renvoie une erreur si le sémaphore est à 0, sinon --
cr = sem_getvalue (& unSemaphore, &valeur)      // valeur est le contenu courant du sémaphore
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

La boîte aux lettres



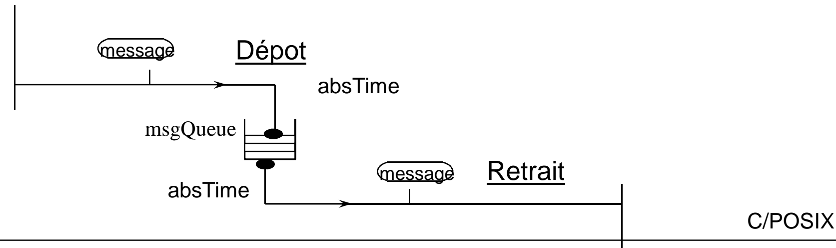
C/POSIX

```
#include <mqueue.h>
#include <fcntl.h>
#include <sys/stat.h>
mqd_t msgQueueID;
struct mq_attr msgQueueAttributes;
msgQueueAttributes.mq_maxmsg = nbMaxMessages;        // nombre maximal de messages
msgQueueAttributes.mq_msgsize = MessageSize;         // taille des messages

msgQueueID = mq_open ("/msgQueue", flag, mode, msgQueueAttributes);
// flag : O_CREAT et O_NONBLOCK, O_EXCL, O_RDONLY, O_WRONLY, O_RDWR,
ou  msgQueueID = mq_open ("/msgQueue", flag);
// ouverture de bal déjà créée, flag : O_RDONLY, O_WRONLY, O_RDWR, O_NONBLOCK
mq_close (msgQueueID);            // ferme la bal
mq_unlink ("/msgQueue“);          // supprime la bal
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

La boîte aux lettres



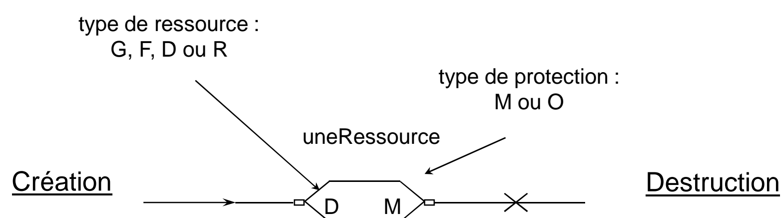
C/POSIX

```
#define messageSize 10
unsigned int messagePriority = 1; // 0 : priorité la plus forte, gestion FIFO si priorités égales
char  messageS[messageSize];
char  messageR[messageSize];
timespec abstime;

mq_send (msgQueueID, messageS, messageSize, messagePriority);
ou mq_timedsend (msgQueueID, messageS, messageSize, messagePriority, &absTime);
        // attente au maximum absTime (temps absolu de type timespec) si la file est pleine

mq_receive (msgQueueID, messageR, messageSize, &messagePriority);
ou mq_timedreceive(msgQueueID, messageR, messageSize, &messagePriority, &absTime);
        // attente au maximum absTime (temps absolu de type timespec) si la file est vide
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

La ressource



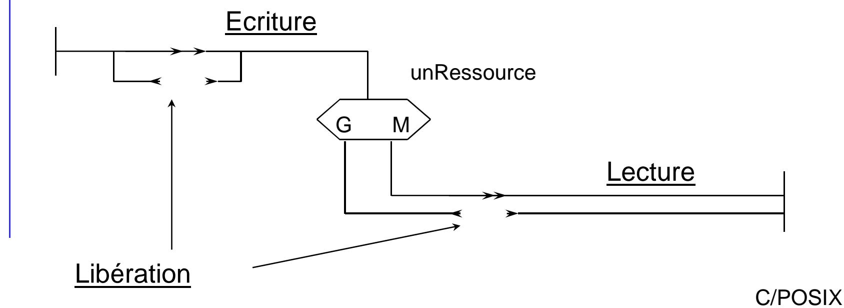
C/POSIX

```
pthread_mutex_t unMutex;  int cr;
cr = pthread_mutex_init(& unMutex, NULL);
ou pthread_mutex_t unMutex = PTHREAD_MUTEX_INITIALIZER;
// attributs par défaut, renvoie une erreur si déjà initialisé ou pas assez de mémoire

cr = pthread_mutex_destroy(& unMutex);
// renvoie EBUSY si unMutex n'est pas libre, EINVAL si unMutex n'a pas été initialisé
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

La ressource



C/POSIX

cr = pthread_mutex_lock (& unMutex); // prend unMutex, ou attend si il est déjà pris

ou cr = pthread_mutex_trylock(& unMutex); // renvoie EBUSY si unMutex n'est pas libre

cr = pthread_mutex_unlock (& unMutex); // renvoie une erreur si un autre thread a bloqué unMutex ou s 'il est déjà libre

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

POSIX: le mutex

- Attributs du mutex

int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr, int protocol);

- Protocol

PTHREAD_PRIO_NONE : gestion FIFO
PTHREAD_PRIO_INHERIT : héritage de priorité
PTHREAD_PRIO_PROTECT : protocole à priorité plafond (compatible avec SCHED_FIFO)

int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr, int prioceiling);
prioceiling entre priorité minimale et maximale de SCHED_FIFO (entre 1 et 99)

```
pthread_mutex_t unMutex;
pthread_mutexattr_t myMutexAttributes;
pthread_mutexattr_init(& myMutexAttributes);
pthread_mutexattr_setprotocol (& myMutexAttributes, PTHREAD_PRIO_PROTECT);
pthread_mutexattr_setprioceiling (& myMutexAttributes, 10);
pthread_mutex_init(&unMutex, &myMutexAttributes);
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Le mutex en rw

- Principes
  - Plusieurs lecteurs en parallèle possible (mode r)
  - En exclusion mutuelle pour l'écriture (mode w)

pthread_rwlock_t unRWMutex; int cr;

cr = pthread_rwlock_init(& unRWMutex, NULL);

ou pthread_rwlock_t unRWMutex = PTHREAD_RWLOCK_INITIALIZER;

cr = pthread_rwlock_destroy(& unRWMutex);

cr = pthread_rwlock_rdlock (& unRWMutex); // prend unRWMutex (r), ou attend si il est déjà pris (w)

ou cr = pthread_rwlock_tryrdlock(& unRWMutex);

cr = pthread_rwlock_wrlock (& unRWMutex); // prend unRWMutex (w), ou attend si il est déjà pris (r ou w)

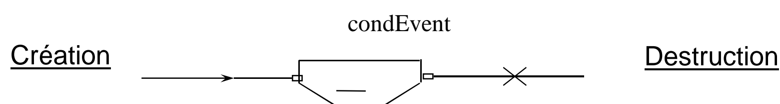
ou cr = pthread_rwlock_trywrlock(& unRWMutex);

cr = pthread_rwlock_unlock (& unRWMutex);

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

L'événement



C/POSIX

pthread_cond_t condEvent; int cr;

pthread_mutex_t unMutexEvt = PTHREAD_MUTEX_INITIALIZER;

// protection de l'atomicité de l'événement ou de la condition

cr = pthread_cond_init(& condEvent, NULL); // attribut à NULL sous Linux

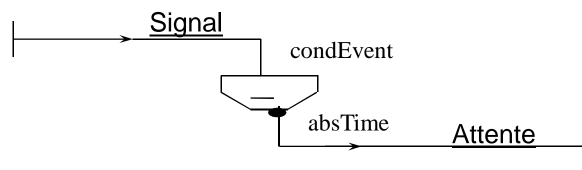
ou pthread_cond_t condEvent = PTHREAD_COND_INITIALIZER;

cr = pthread_cond_destroy(& condEvent);

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

L'événement



C/POSIX

```
cr = pthread_mutex_lock (& unMutexEvt);
    cr = pthread_cond_signal(&condEvent);        // signale l 'événement à un (?!) thread en attente, sinon pas de mémorisation
  ou cr = pthread_cond_broadcast(&condEvent);  // signale l 'événement à TOUS les threads en attente
cr = pthread_mutex_unlock (& unMutexEvt);

cr = pthread_mutex_lock (& unMutexEvt);
  cr = pthread_cond_wait(&cond, & unMutexEvt);                         // attend (indéfiniment) l 'événement
  ou  cr = pthread_cond_timedwait(&cond, & unMutexEvt, absTime);       // attend l 'événement avant absTim (temps absolu)
cr = pthread_mutex_unlock (& unMutexEvt);
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

La condition

- Association événement / mutex / donnée
  - L'événement est envoyé sur une valeur spécifique de la donnée
  - Le mutex protège l'accès à la donnée
  - On attend l'événement et la condition vrai sur la donnée

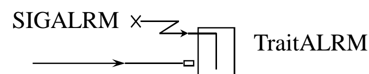
```
int data = 0;
…

cr = pthread_mutex_lock (& unMutexEvt);
    data ++;
    if (data > seuil)                      // vérification de la condition
    {     pthread_cond_signal(&condEvent);  // envoi du signal si la condition est vérifiée
    }
cr = pthread_mutex_unlock (& unMutexEvt);
…

pthread_mutex_lock (& unMutexEvt);
    while (data <= seuil)                                       // vérification de la condition
        { pthread_cond_wait(&condEvent, & unMutexEvt); }        // attente du signal
    // libération de unMutexEvt, puis reprise unMutexEvt  lorsque condEvent arrive
pthread_mutex_unlock (& unMutexEvt);
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Le signal

- Signaux
  - Signaux classiques C
    - SIGALRM, …
  - Pas de perte
  - POSIX : nouveau signal RTSIG_MAX



C/POSIX

```
#include <signal.h>
void TraitALRM()
{
		signal(SIGALRM,TraitALRM);
		/* TraitALRM body */
}
…
int main() { …
/* activation de l'attente de signal */
signal(SIGALRM,TraitALRM);
… }
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Conception multitâches

- Principes généraux sur les OS multitâches
- API POSIX
  - pthread
  - Communications
  - Gestion du temps
- Aspect réactif et concurrents
  - Exécution des actions
  - Déclenchement des threads
    - Prise en compte des événements
  - Échange de données
    - Concurrence et coopération
- Structuration
  - Création / destruction des éléments
  - Découpage

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Gestion de l'heure

- Heure UNIX ou POSIX

```
#include <sys/timeb.h>
int ftime (struct timeb *tp);
struct timeb {   time_t time;   unsigned short millitm;   short timezone;   short dstflag; };
```

Nombre de secondes écoulées depuis le 1er janvier 1970 à 00:00 UTC (sans les ajustements)

Bogue de l'an 2038, le 19 Janvier 2038 à 3 h 14 min 7
- dépassement de capacité sur un entier signé 32 bits
- passage à 64 bits d'ici 2038

- Gestionnaire de temps à la ms ou ns

```
#include <sys/time.h>
#include <unistd.h>
int gettimeofday(struct timeval *tv, struct timezone *tz);
struct timeval {
        int   tv_sec;     /* secondes */
        int   tv_usec;   /* microsecondes */ };
struct timespec {
        time_t tv_sec;   /* seconds */
        long tv_nsec; /* nanoseconds */ };
```

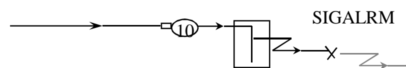
jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

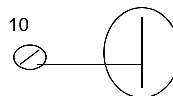
POSIX: gestion des activations temporelles (1/3)

- Alarm
  - Attente en seconde (unsigned int)
  - Déclenche l'envoi du signal SIGALRM
  - Arrêt du processus si pas de réception du signal

```
#include <unistd.h>

alarm(10);
```



- Sleep
  - Attente en seconde
  - Blocage du thread
    - Pas d'utilisation du CPU



jean-philippe.babau@univ-brest.fr

POSIX: gestion des activations temporelles (2/3)

- Timer
  - Définition d'une structure itimerspec
    - Date de départ (it_value)
    - Intervalle d'activation(it_interval)
  - Création (timer_create) avec connection à un signal (SIGALRM par défaut), activation/désactivation (timer_settime), destruction (timer_delete)
  - Compilation avec l'option –lrt, incompatible avec sleep(), alarm(), …

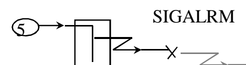
```
#include <time.h> #include <signal.h>
void timerHandler(timer_t timerId, int arg)
{ …}
int main(void)
{timer_t timerId;     struct itimerspec timerAttr;
…timerAttr.it_value.tv_sec = 5; timerAttr.it_value.tv_nsec = 0;
timerAttr.it_interval.tv_sec = 5; timerAttr.it_interval.tv_nsec = 0;
timer_create(CLOCK_REALTIME, NULL, &timerId);
timer_settime(timerId,0, &timerAttr, NULL); …}
```


POSIX: gestion des activations temporelles (3/3)

- itimer
  - Déclenche l'envoi du signal SIGALRM
  - Incompatible avec sleep(), alarm(), …

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

struct itimerval timerAttr;
timerAttr.it_value.tv_sec = 5; timerAttr.it_value.tv_usec = 0;
timerAttr.it_interval.tv_sec = 5; timerAttr.it_interval.tv_usec = 0;

setititimer(ITIMER_REAL, &timerAttr, NULL);
```

SIGALRM



Ecole Temps Réel

Conception multitâches

- Principes généraux sur les OS multitâches
- API POSIX
  - pthread
  - Communications
  - Gestion du temps
- Aspect réactif et concurrents
  - Exécution des actions
  - Déclenchement des threads
    - Prise en compte des événements
  - Échange de données
    - Concurrence et coopération
- Structuration
  - Création / destruction des éléments
  - Découpage

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Mise en place des tâches

- Spécification
  - Un événement externe ➔ séquence d'actions
  - Respecter des contraintes temps réel (fréquence et échéance)
  - Politiques de réaction aux fautes temporelles
  - Reconfiguration
- Mise en place de threads
  - Activation (fréquence d'activation)
    - En réaction à un événement en provenance de l'environnement
    - Selon une fréquence donnée
  - Réalisation des actions dans un temps borné (échéance)
    - Priorité la plus forte si échéance la plus faible (Deadline Monotonic)
    - Priorité la plus forte si fréquence la plus faible (Rare Monotonic)

jean-philippe.babau@univ-brest.fr

Type de threads

- Initiale (*main*)
  - activée au lancement de l'application
- Ordinaire
  - liée à un événement externe ou interne
  - threads matérielles
    - lié à un driver, lié à une horloge
  - threads logicielles
    - activé par un autre thread
- Thread de fond
  - toujours active
  - exécutée lorsque toutes les autres threads sont terminées

jean-philippe.babau@univ-brest.fr

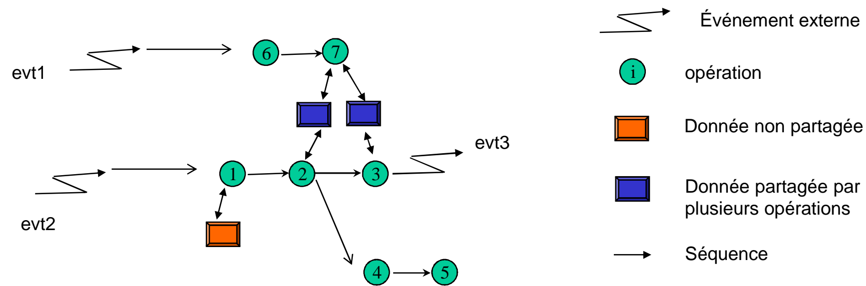
Structure d'un thread

- Initiale
  - Corps du *main*
- Ordinaire
  - Boucle infinie avec attente
    - Temporelle (*sleep*)
    - En lecture sur un driver (*read()* bloquant)
    - Sémaphore, boîte aux lettres, événement, condition
    - Mécanisme spécifique de synchronisation
      - À programmer
- Thread de fond
  - Boucle infinie sans attente
    - Priorité la plus faible

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

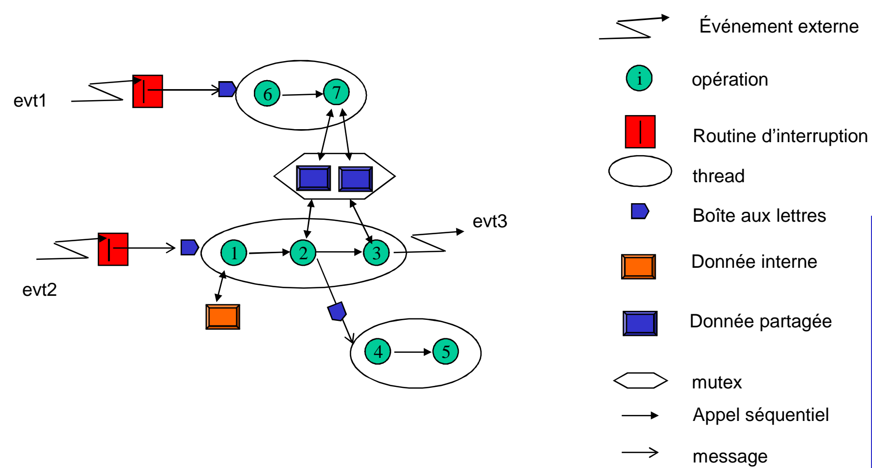
Exemple



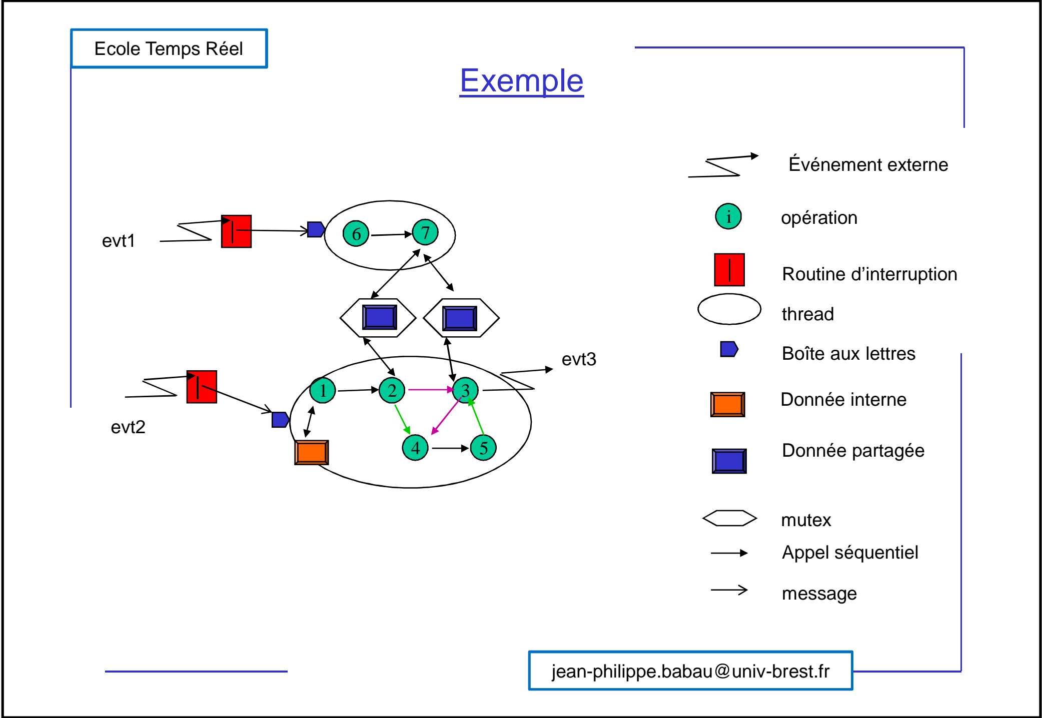
jean-philippe.babau@univ-brest.fr

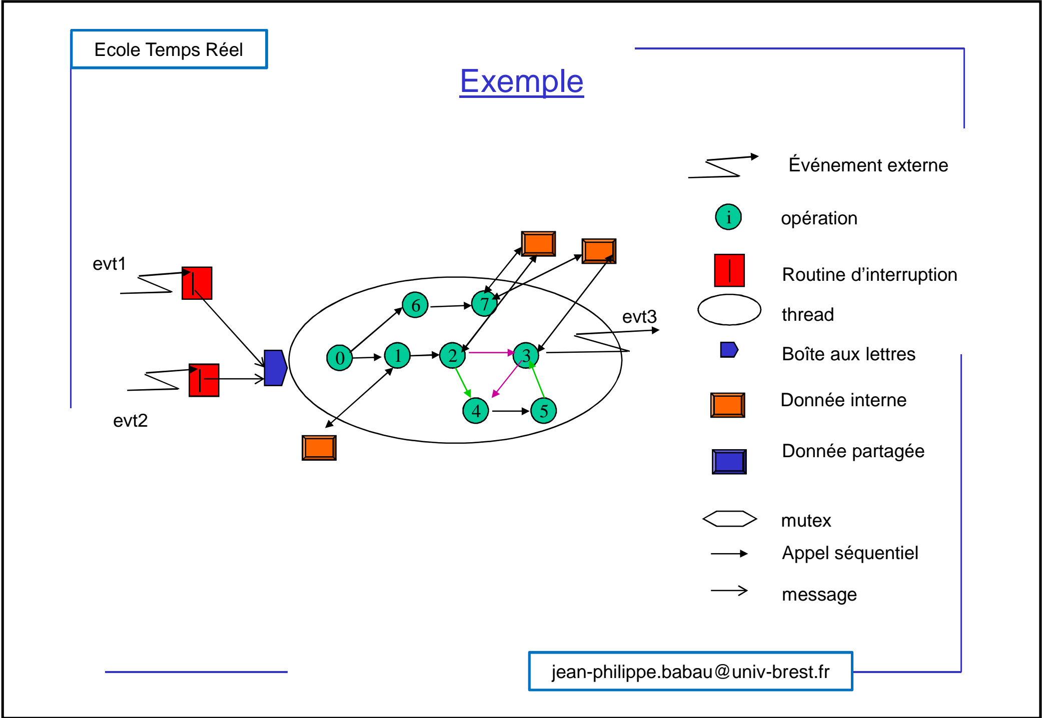
Ecole Temps Réel

Exemple



jean-philippe.babau@univ-brest.fr


Ecole Temps Réel
Exemple
evt1
6
7
evt2
1
2
3
4
5
evt3
Événement externe
i
opération
Routine d'interruption
thread
Boîte aux lettres
Donnée interne
Donnée partagée
mutex
Appel séquentiel
message
jean-philippe.babau@univ-brest.fr


Ecole Temps Réel
Exemple
evt1
evt2
0
1
2
3
4
5
6
7
evt3
Événement externe
i
opération
Routine d'interruption
thread
Boîte aux lettres
Donnée interne
Donnée partagée
mutex
Appel séquentiel
message
jean-philippe.babau@univ-brest.fr

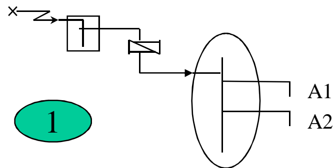
Séquencement des actions via les threads

- Actions
  - À partir d'un événement externe (alarme ou interruption) : suivi du fil d'exécution
  - Un thread = une séquence d'actions
- Gestion des contraintes de temps
  - Priorité liée à l'urgence temporelle
  - Priorités des actions -> priorités des threads

jean-philippe.babau@univ-brest.fr

threads Vs actions (1)

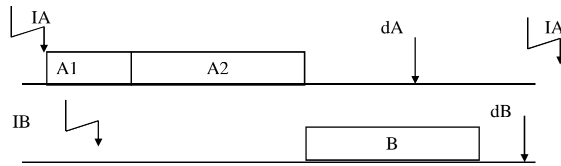
- 2 actions en séquence A1 – A2



jean-philippe.babau@univ-brest.fr

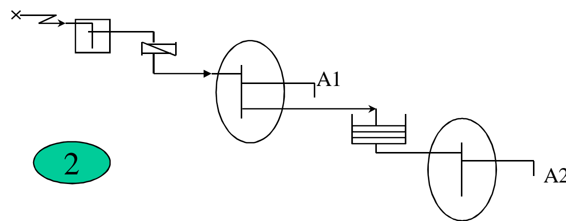
Ordonnancement des actions (1)

1

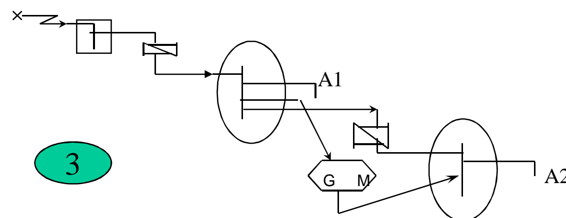


threads Vs actions (2)

- 2 actions en séquence A1 – A2



2

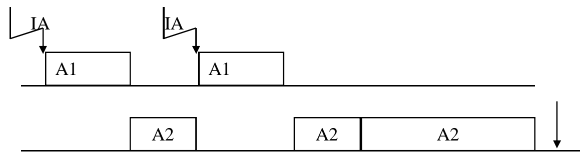


3

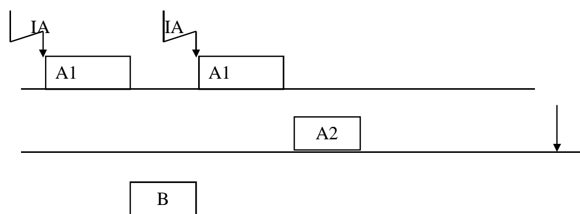
Ecole Temps Réel

Ordonnancement des actions (2)

2



3



jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Conception multitâches

- Principes généraux sur les OS multitâches
- API POSIX
  - pthread
  - Communications
  - Gestion du temps
- Aspect réactif et concurrents
  - Exécution des actions
  - Déclenchement des threads
    - Prise en compte des événements
  - Échange de données
    - Concurrence et coopération
- Structuration
  - Création / destruction des éléments
  - Découpage

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Politique de déclenchement

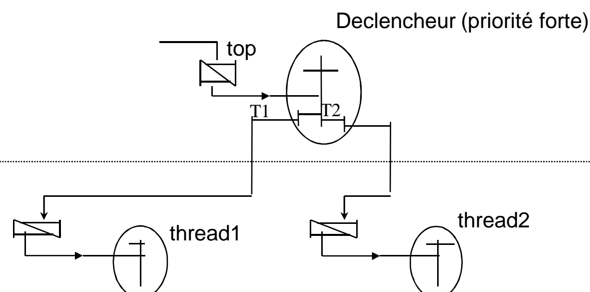
- Traitement à date fixe
- Traitement périodique
  - Traitement régulier
  - Traitement du signal
  - Scrutation d'un procédé ou de l'environnement
- Traitement lié à une interruption
  - Réaction à un événement
  - Traitement immédiat (réservé aux urgences) ou différé (priorités et planification)
- Nombres d'activation
  - Si plusieurs événements (interruption ou alarme) arrivés
- Synchronisation des déclenchement des threads
  - Plusieurs périodes

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Déclenchement synchronisé

Contôle (~exécutif)

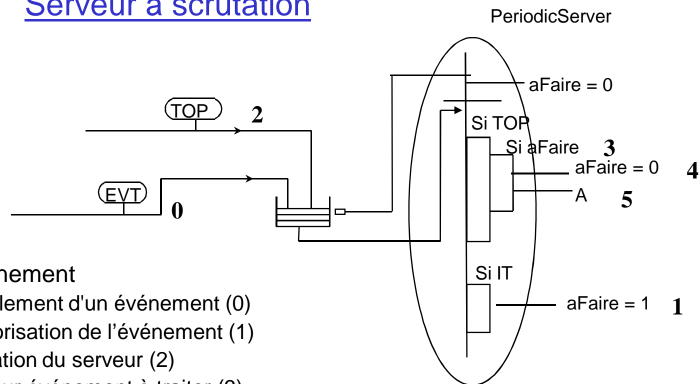


Application

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Serveur à scrutation



- Fonctionnement
  - Signalement d'un événement (0)
  - Mémorisation de l'événement (1)
  - Activation du serveur (2)
  - Test sur événement à traiter (3)
  - Acquittement logiciel (4)
  - Traitement (5)
- Variantes
  - Afaire ++, aFaire A(n)

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Scrutation / événementiel

- Événementiel
  - Evaluation de l'intervalle minimum entre deux événements (*dmin*)
    - dynamique du procédé
    - valeur imposée
  - Pas de perte de temps
- Scrutation
  - L'application définit son rythme de travail
  - Prédictibilité
  - Obligatoire dans certains domaines (certification)
    - Certification
    - OSEK Time

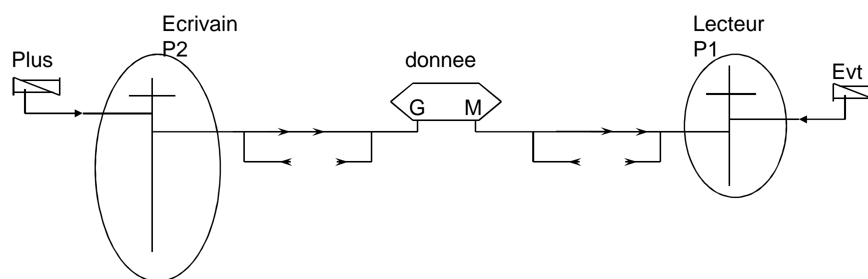
jean-philippe.babau@univ-brest.fr

Conception multitâches

- Principes généraux sur les OS multitâches
- API POSIX
  - pthread
  - Communications
  - Gestion du temps
- Aspect réactif et concurrents
  - Exécution des actions
  - Déclenchement des threads
    - Prise en compte des événements
  - Échange de données
    - Concurrence et coopération
- Structuration
  - Création / destruction des éléments
  - Découpage

jean-philippe.babau@univ-brest.fr

Donnée partagée (LACATRE)



jean-philippe.babau@univ-brest.fr

Donnée partagée (POSIX)

```
typedef struct
{
        int heure ;
        int minute ;
} horaire;

horaire donnee ;

pthread_mutex_t protHeure = PTHREAD_MUTEX_INITIALIZER;

void Lecteur()
{
    while(1)
    {   sem_wait(& Evt);
        pthread_mutex_lock (& protHeure );
            printf("%i %i\n", donnee.heure, donnee.minute ) ;
        pthread_mutex_unlock (& protHeure );
    }
}
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Donnée partagée (POSIX)

```
void Ecrivain()
{
    while(1)
    {   sem_wait(& Plus);

        pthread_mutex_lock (& protHeure );
            if (donnee.minute < 59 )
            {   donnee.minute ++ ;}
            else
            {   if (donnee.heure < 23)
                {       donnee.heure ++ ;
                        donnee.minute = 0 ; }
                else
                {       donnee.heure = 0 ;
                        donnee.minute = 0 ; }
            }
        pthread_mutex_unlock (& protHeure );
    }
}
```

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

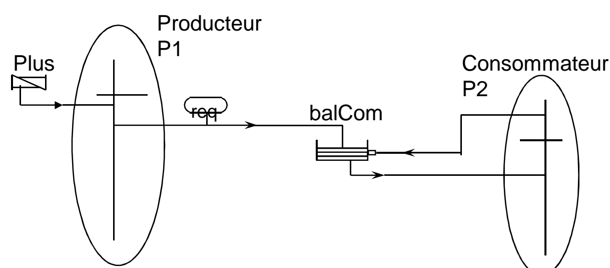
Communication Client / Serveur

- Le serveur crée une bal de réception des requêtes
- Le client envoie une requête dans la bal de réception des requêtes
  - mode d'adressage ou de nommage
- Si réponse nécessaire
  - création d'une bal de réponse
  - envoi avec la requête de la bal de réponse
  - attente de la réponse

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

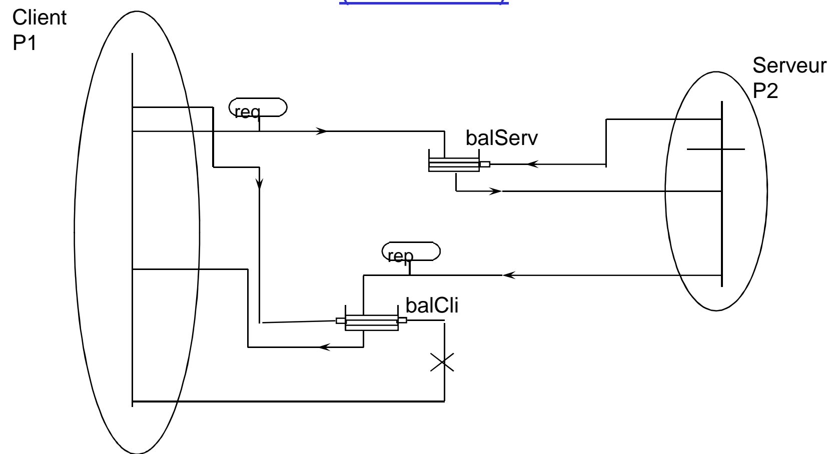
Client/serveur avec une boîte aux lettres (LACATRE)



jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Client/serveur avec une boîte aux lettres de réponse (LACATRE)

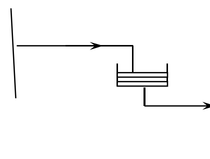


jean-philippe.babau@univ-brest.fr

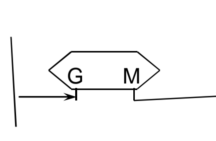
Ecole Temps Réel

Echange d'informations

- Messages
  - Besoins
    - Requêtes
    - Invocation distante
    - Serveurs
    - Objets
  - Mise en oeuvre
    - Boite aux lettres
    - Canal



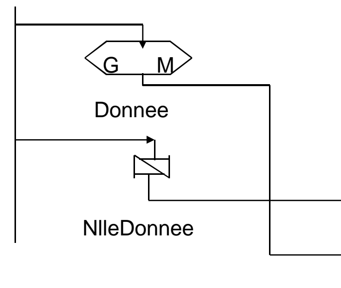
- Données partagées
  - Besoins
    - Données du problème
  - Mise en oeuvre
    - Variables globales
    - Fichiers
    - Mémoire partagée



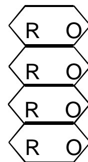
jean-philippe.babau@univ-brest.fr

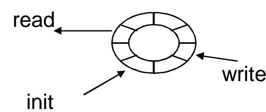
Echange d'informations

- Données synchronisées
  - Gestion de la dernière valeur
  - Attente de nouvelle valeur



- Nombre maximal fixe de données
  - buffer matériel (réseau)
  - File circulaire





Echanges de données

- Zones de stockage
  - Taille d’une zone de stockage (producteur /consommateur)
  - Politique de stockage
    - FIFO, LIFO, politique d’écrasement
  - Datation d’acquisition des données stockées
  - Validité des données : age, valeurs correctes
- Du point de vue de l’écrivain
  - Si la zone de mémorisation est pleine
    - Perte ou attente
    - Perte de la donnée la plus récente ou de la plus ancienne
    - Mise en attente (timeout)
  - Evénement lié à l’écriture
- Du point de vue du lecteur
  - Toutes les données doivent être reçues
  - Dernière(s) donnée(s) reçue(s)
  - Attente si pas de données, attente avec timeout

Conception multitâches

- Principes généraux sur les OS multitâches
- API POSIX
  - pthread
  - Communications
  - Gestion du temps
- Aspect réactif et concurrents
  - Exécution des actions
  - Déclenchement des threads
    - Prise en compte des événements
  - Échange de données
    - Concurrence et coopération
- Structuration
  - Création / destruction des éléments
  - Découpage

Création / destruction

- Principe de base : accès à un objet initialisé (créé)
- Gestion statique (main)
  - Ordre de création
    - Objets de communication
    - Threads
    - Activation des drivers (IT, alarmes)
  - Ordre de destruction
    - Ordre inverse
- Gestion dynamique
  - Thread « propriétaire »
  - Le thread crée ses objets de communication : bal, …
- Gestion pseudo-dynamique
  - Création statique d'un pool
  - Création dynamique dans la limite du pool

Ecole Temps Réel

Statique Vs dynamique

- Statique
  - Prédictible
  - Gestion du pire cas
  - Peut être coûteux
- Dynamique
  - Optimisation
  - Adaptation
    - Environnement inconnu
    - Changements de mode de fonctionnement (reconfiguration)
- Semi dynamique
  - Allocation limitée à une borne supérieure
- Statique et dynamique
  - Application programmée en statique
  - Superviseur dynamique

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Conception multitâches

- Principes généraux sur les OS multitâches
- API POSIX
  - pthread
  - Communications
  - Gestion du temps
- Aspect réactif et concurrents
  - Exécution des actions
  - Déclenchement des threads
    - Prise en compte des événements
  - Échange de données
    - Concurrence et coopération
- Structuration
  - Création / destruction des éléments
  - Découpage

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Découpage de l'application

- Approche modulaire
  - Encapsulation
  - Réutilisation
    - Application, pilote, couches de communication, services …
- Découpage en phases
  - Phases
    - Création / destruction / suspension / réactivation
    - Moteur / exception
  - Mise en place
    - Plusieurs processus distincts
      - Lien entre les processus
        - » Evénement déclenchant la fin d'un processus
        - » Événement déclenchant la création d'un processus
        - » Gestion des transitions (sauvegarde, rafraîchissements, liens avec le procédé)
        - » Suspension / activation des activités en cours / nouvelles
        - » Gestion du cycle de vie des activités du système
      - Partage des données entre processus
    - Plusieurs modes de fonctionnement par thread
      - Mode : information partagée de manière atomique

jean-philippe.babau@univ-brest.fr

Ecole Temps Réel

Découpage de l'application

- Découpage en processus serveurs
  - Un processus offre des services aux autres processus
    - Modèle de communication à préciser (cf. serveur ci-avant)
      - Classiquement un thread + une file de message
  - Problème de gestion de la concurrence d'accès
    - Inversion de priorité possible via la file des messages
    - Priorité des services pour des événements concurrents
      - Gestion de la file et association de plusieurs threads à la file
  - Gestion des priorités
    - Priorité liée à l'action
      - Gestion dynamique de la priorité du thread si messages de priorités distinctes
    - Affecter une priorité haute aux services
      - Par exemple pour les services I/O
    - Affecter une priorité basse aux services
      - Traitements sans échéance

jean-philippe.babau@univ-brest.fr

Définition des threads

- Approche événementielle
  - Un thread est mis en place pour une séquence d'action liée à un événement externe et possédant une priorité liée à l'urgence temporelle de l'événement
- Approche structurelle
  - Un thread par objet, un thread par service
  - Un thread pour un ensemble d'objets ou de services
- Approche système
  - Un thread à haute priorité pour exécuter des services de haut niveau non fournis par l'OS

Conclusion

- POSIX
  - API riche et complexe : bien lire les spécifications pour bien utiliser l'API
- Conception multithread
  - Découpage en thread
    - Réduire la concurrence au strict nécessaire
  - Activation sur événement externe ou temporisation
  - Structuration pour la réutilisation de services
- Communications avec l'extérieur
  - Drivers

Liens utiles et utilisés pour ce cours

- http://www.linux-kheops.com/doc/man/manfr/man-html-0.9/
- https://computing.llnl.gov/tutorials/pthreads/#Abstract
- Bertrand Dupouy “Systèmes temps réel et POSIX temps réel”
  - © http://www.infres.enst.fr/~dupouy/pdf/TRAM/1-SystTR-INF342.pdf