PROCESSUS ET CONCURRENCE
DANS LES LANGAGES DE PROGRAMMATION

EXEMPLE COMPARATIF

le processus principal père crée deux processus fils; chaque processus lance une impression

---

LANGAGE DU COURS ET SÉMAPHORES

```
with SI_B; use SI_B; -- importation de méthodes
with Ada.Text_Io; use Ada.Text_Io;

procedure Main is

   -- objet de synchronisation, objet partagé
   package Fin is                    -- déclaration
      procedure Signaler;
      procedure Attendre(I : in Integer);
   end Fin;

   package body Fin is               -- implantation
      procedure Signaler is
      begin
         V(S);
      end Signaler;
      procedure Attendre(I : in Integer) is
      begin
         for J in 1..I loop P(S); end loop;
      end Attendre;
   begin
      E0(S,0); -- initialisation de l'objet partagé
   end Fin;

   -- déclaration du type des fils
   task type Un_Fils (K : Integer) is
   begin
      Put_Line("BONJOUR DU FILS "
                        & Integer'Image(K));
      Fin.signaler;
   end Un_Fils;

   -- déclaration et création des fils
   Fils1 : Un_Fils(1); Fils2 : Un_Fils (2);

-- programme du processus père
begin
   Put_Line("BONJOUR DU PERE")
   Fin.Attendre(2);
   Put_Line("FIN DU PROGRAMME");
end Main;
```

PROGRAMMATION DES PROCESSUS CONCURRENTS EN C ET POSIX

EXEMPLE COMPARATIF

le processus principal père crée deux processus fils; chaque processus lance une impression

---

En Unix-Linux-Posix avec des processus

```
#include <stdio.h>

main ()
{
   printf ("BONJOUR DU PERE\n");

   id = fork();
   if ( id == 0) {
                    /*processus fils 1*/
      printf("BONJOUR DU FILS 1 %d/n");
      exit(0);
   }
   if (id < 0) perror("erreur au fork()");

   id = fork();
   if ( id == 0) {
                    /*processus fils 2*/
      printf("BONJOUR DU FILS 2 %d/n");
      exit(0);
   }
   if (id < 0) perror("erreur au fork()");

   if ( id > 0) {
            /*le père* attend la fin des 2 fils*/
      wait((int*)0);
      wait((int*)0);
      printf("FIN DU PROGRAMME \n");
   exit(0);
   }
```

PROGRAMMATION DES PROCESSUS CONCURRENTS EN C ET API POSIX

EXEMPLE COMPARATIF

Code source en C avec les API POSIX gérant des Threads (primitives système)

```
#include <ipc.h>
        /* module de communication
#include <pthread.h>
        /* module de définition des "thread"

pthread_attr_t pthread_attr_default = 1;

void threadcode(arg)
        /* fonction appelée par les fils
{
    printf("BONJOUR DU FILS %d/n", arg);
    pthread_exit (- arg);
}

void PERE()         /* programme principal
{
    pthread_t FILS1, FILS2;
        /* déclaration des fils
    pthread_attr_t attr;
    int result;

    printf ("BONJOUR DU PERE\n");
    pthread_attr_create(&attr);
    attr.pthread_attr_prio = 6;
    pthread_attr_setstacksize(&attr, 8192);

    if(pthread_create(&FILS1, attr,
                        threadcode, 1) < 0 {
        printf(" erreur de création du FILS 1\n");
        exit(1);
    }

    if(pthread_create(&FILS2, attr,
                        threadcode, 2) < 0 {
        printf(" erreur de création du FILS 2\n");
        exit(1);
    }

    pthread_join(FILS1, NULL);
            /* attend la fin du fils1
    pthread_join(FILS2, NULL);
            /* attend la fin du fils2
    printf("FIN DU PROGRAMME\n");
    exit(0);
}
```

```
        PROGRAMMATION DES PROCESSUS
      CONCURRENTS EN C et API WINDOWS NT

Code source en C avec les API WINDOWS NT gérant
des "threads" (primitives système)

#include <windows.h>
        /* module de communication
#include <stdio.h>
        /* module de définition des "thread"

DWORD WINAPI ThreadFunc
                              (LPDWORD param)
{                   /* fonction appelée par les fils
    printf("BONJOUR DU FILS %d/n", *param);
    return 0;
}

DWORD main(void)        /* programme principal
{
    DWORD dwThreadTD1, dwThrdParam1 = 1,    /*
déclaration des fils
                  dwThreadTD2, dwThrdParam2 = 2;
    HANDLE FILS1,FILS2;

    printf ("BONJOUR DU PERE\n");

    FILS1 = CreateThread(NULL, 0, ThreadFunc,
            &dwThrdParam1, 0, &dwThreadTD1);
    if(!FILS1) {
       printf(" erreur de création du FILS 1\n");
       return 1;
    }

    FILS2 = CreateThread(NULL, 0, ThreadFunc,
              &dwThrdParam2, 0, &dwThreadTD2);
    if(!FILS2) {
       printf(" erreur de création du FILS 2\n");
       return 1;
    }

    (void) WaitForSingleObject(FILS1,
                                          INFINITE);
          /* attend la fin du fils1
    (void) WaitForSingleObject(FILS2,
                                          INFINITE);
          /* attend la fin du fils2
    printf("FIN DU PROGRAMME\n");
    return 0;
}
```

PROGRAMMATION DES PROCESSUS CONCURRENTS EN ADA

Exemple : le processus principal père crée deux processus fils; chaque processus lance une impression (programmation en Ada normalisé, en C/Posix, en C/WindowsNT)

Code source en ADA 95 avec rendez-vous:

```
with TEXT_IO; use TEXT_IO;
      -- paquetage  standard d'impression

procedure PERE is
      -- création du père, programme principal

      -- déclaration de l'interface du type fils
   task type TT is
      entry START( ID : INTEGER);
      -- interface de rendez-vous
   end TT;

      -- création de deux fils serveurs
   FILS1, FILS2 : TT;

      -- déclaration de l'implantation des fils
   task body TT is
   begin
      accept START(ID : INTEGER) do
         PUT_LINE ("BONJOUR DU FILS "
                        & INTEGER'IMAGE(ID));
      end START;
   end TT;

begin
      -- début du père, client et activation des fils
   PUT_LINE ("BONJOUR DU PERE");
   FILS1.START(1);
      -- appel et réveil du fils1 avec une requête
   FILS2.START(2);
      -- appel et réveil du fils2 avec une requête
   PUT_LINE ("FIN DU PROGRAMME");
end PERE;

-- Ada est court, clair, portable sur Unix, Linux, Posix et Windows NT
-- le langage Ada est normalisé ISO/IEC 8652:1995(E)
-- structure analogue à l'appel de procédure distante
```

```
        PROGRAMMATION DES PROCESSUS
             CONCURRENTS EN JAVA

Le processus principal père crée deux processus fils;
chaque processus lance une impression

class Exemple {
   public static void main(String args[]){
      // programme principal
      System.out.println("BONJOUR DU PERE");

      // lance une instance comme premier fils
      UnProcessus a =new UnProcessus ("FILS 1");
      a.start();

      // lance une instance comme second fils
      UnProcessus b =new UnProcessus ("FILS 2");
      b.start();

      // attendre la fin des deux fils
      try {
         a.join();
         b.join();
      } catch(InterruptedException e) {
         System.out.println("un fils ne repond pas");
      }
      System.out.println("FIN DU PROGRAMME");
   }
}

      // déclaration de la classe des fils
classe UnProcessus extends Thread {
   public UnProcessus(String st){
            // mémorise un objet
      super(st);
   }
   public void run(){
               // programme du processus
  System.out.println("BONJOUR DU" + getName() );
   }
}
```

```
1        PROGRAMMATION DES PROCESSUS
2        CONCURRENTS EN ADA
3
4                  autre exemple
5              3 processus : main et deux fils
6
7  with TEXT_IO; use TEXT_IO;
8        -- paquetage  standard d'impression
9
10 procedure LES_MAUX_DE_L_IMPRESSION is
11             -- main
12
13    task type P;
14
15    task type Q;
16
17    task body P is
18             -- déclaration de corps de processus
19    begin
20       PUT ("J'ai vu la terre distribuée ");
21       PUT ("en de vastes espaces ");
22       NEW_LINE;
23       PUT ("et ma pensée ");
24       PUT ("n'est point distraite ");
25       PUT ("du navigateur.");
26       NEW_LINE;
27       PUT ("Saint-John Perse");
28       NEW_LINE (2);
29    end P;
30       ----------------
31    task body Q is
32       -- déclaration de corps de processus
33    begin
34       PUT ("L'homme est capable de faire ");
35       PUT ("ce qu'il est ");
36       PUT ("incapable d'imaginer. ");
37       NEW_LINE;
38       PUT ("Sa tête sillonne ");
39       PUT ("la galaxie de l'absurde");
40       NEW_LINE;
41       PUT ("René Char");
42    end Q;
43
44 begin
45          -- début du programme principal "main"
46    PUT ("IMPRESSION DE DEUX POEMES");
47    NEW_LINE (2);
48
49    declare
50       PP : P;  -- création d'un processus fils
51       QQ : Q;  -- création d'un processus fils
52    begin   -- activation des fils PP et QQ
53       null;
54    end;
55 end LES_MAUX_DE_L_IMPRESSION;
56
```

PROGRAMMATION DES PROCESSUS CONCURRENTS EN ADA

```
with SI_B; use SI_B; with SCENE; use SCENE;
   -                                  - IMPORTATION de SI_B et de SCENE
procedure SIMULATION is                                  -- programme principal
   NBVOITURE: constant INTEGER := 2500;
   NBCAMION: constant INTEGER :=10;
   task type VOITURE; task type CAMION; task type GROSSISTE;

   package PLATEFORME is
         -- OBJET OU SERVICE PARTAGÉ ENTRE LES PROCESSUS
      procedure PRENDRE_LIVRAISON;          -- procédure exportable
      procedure PREPARER_LIVRAISON;
   end PLATEFORME;

   package STATIONSERVICE is
         -- OBJET OU SERVICE PARTAGÉ ENTRE LES PROCESSUS
      procedure UTILISER;
      procedure LIVRER;
   end STATIONSERVICE;

   package RONDPOINT is          -- OBJET OU SERVICE PARTAGÉ
      procedure TRAVERSER;
      procedure TRAVERSER_AVEC_CAMION_PLEIN;
   end RONDPOINT;

   package TELECOPIE is
                              -- OBJET OU SERVICE PARTAGÉ
      procedure NOTIFIER_UNE_COMMANDE;
      procedure ATTENDRE_UNE_COMMANDE;
   end TELECOPIE;

   task body VOITURE is                    -- code du type de processus
   begin
      loop
         RONDPOINT.TRAVERSER; ROULER;
          -- ROULER est importé de SCENE
         RONDPOINT.TRAVERSER; ROULER;
         STATIONSERVICE.UTILISER; ROULER;
      end loop;
   end VOITURE;

   task body CAMION is                     -- code du type de processus
   begin
      loop
         PLATEFORME.PRENDRE_LIVRAISON; ROULER;
         RONDPOINT.TRAVERSER_AVEC_CAMION_PLEIN; ROULER;
         STATIONSERVICE.LIVRER; ROULER;
         RONDPOINT.TRAVERSER; ROULER;
      end loop;
   end CAMION;

   task body GROSSISTE is  -- code du type de processus
   begin
      loop
         PLATEFORME.PREPARER_LIVRAISON; ENREGISTRER
      end loop;                    -- ENREGISTREREst importé de SCENE
   end GROSSISTE;

   package body PLATEFORME is separate;      -- Compilation séparée

   package body STATIONSERVICE is separate;

   package body RONDPOINT is separate;

   package body TELECOPIE is separate;

begin
LANCEMENT:
            --declare et lance les 2511 processus fils de SIMULATION
   declare
             -- Création des processus

      LESVOITURES: array (1..NBVOITURE) of VOITURE;

      LESCAMIONS: array (1..NBCAMION) of CAMION;

      LEGROSSISTE: GROSSISTE;

   begin
   -- activation de tous les fils du processus principal (main) SIMULATION
      null;
   end LANCEMENT;
end SIMULATION;    -- exemple avec 2512 processus
```

PROCESSUS CONCURRENTS : DIVERS ASPECTS

1. PROCESSUS SEQUENTIEL ou PROCESSUS

- unité de découpage et d'expression des activités concurrentes
- programme en exécution sur un processeur virtuel
- trace temporelle de l'exécution des instructions d'un programme
- tâche en cours d'exécution avec des données particulières
- suite d'actions considérées comme indivisibles, ou atomiques, au niveau d'observation choisi

Distinguer

programme (texte) et processus (exécution)
résultat statique (fonctionnel) et processus dynamique (calcul en cours)

2. RELATIONS ENTRE PROCESSUS

a) relations d'existence

statique : nombre fixe de processus (en particulier pour le temps réel)
dynamique : création, destruction de processus

b) relations de pouvoir

hiérarchie de processus, père, fils,...
droits d'un processus sur un autre: suspension, modification, destruction,...

c) relations de synchronisation : cohérence, coopération, compétition, ...

3. REPRESENTATION DANS UN NOYAU EXECUTIF

Les processus qui s'exécutent dans un système, en se partageant ses ressources, doivent être gérés par le noyau exécutif de ce système. En particulier, il faut noter ce qui doit être sauvegardé quand un processus quitte l'état élu et ce qui doit être restauré quand le processus reprendra cet état. Les processus sont représentés et décrits par leurs états et par leurs contextes et sont repérés chacun par un descripteur de processus ou un vecteur de contexte. La représentation dépend du système, comme le contexte.

4. CONTEXTE D'UN PROCESSUS POUR UN NOYAU EXECUTIF

a). ressources : informations, données, programmes, matériels qui sont nécessaires à l'évolution d'un processus et qui doivent être mises en place pour le déroulement correct d'un processus actif.

b). contexte : état instantané des ressources utilisées par le processus, et de toutes les informations qui servent à le distinguer des autres processus du système. Par exemple:

-le contexte du processeur: registres spéciaux, registres généraux, compteur ordinal, mot d'état du programme, clés d'accès à la mémoire, masques d'interruption, anneau de protection,

-son espace d'adressage (mémoire logique ou virtuelle) : segments de code, de données, piles d'exécution,...

-le contexte des canaux ou des fichiers: répertoire courant, répertoire racine, liste des fichiers ouverts, descripteurs de ces fichiers, ports de communication vers le réseau,...

-les attributs distinctifs du processus: nom externe, nom interne, priorité, échéance, date limite d'exécution, appartenance à un groupe, droits ou capacités,...

La nature précise du contexte dépend de l'organisation de chaque système.

5. ETATS D'UN PROCESSUS POUR UN NOYAU EXECUTIF

a) états et transitions intrinsèques : bloqué <-> actif

hypothèse simplificatrice: autant de processeurs que de processus, ce qui s'exprime encore par : **un processeur virtuel par processus**

état bloqué : processus en attente d'un événement, d'un signal d'un autre processus, d'une donnée préparée par un autre processus, d'une date, d'une entrée-sortie, d'un rendez-vous ou d'un réveil...

état actif : processus en exécution, non bloqué, sur son processeur

b) prise en compte du partage des processeurs réels :

sous-états de l'état actif : élu <-> prêt

état prêt : processus actif mais il attend un processeur réel : son activité se déroule mais à vitesse nulle

état élu : processus actif auquel un processeur est alloué : son activité se déroule à vitesse non nulle

c) autres états pour gestion des autres ressources partagées (mémoire physique)

6. DÉFINITION DES PROCESSUS DANS LES LANGAGES DE PROGRAMMATION

Définition procédurale et instantiation : "task type", "task" en Ada, Thread en Java.

Définition par clonage et recouvrement de code : fork, exec, exit, wait en Posix, Unix, Linux

7. MODELISATION DES PROCESSUS CONCURRENTS

files d'attentes (clients et ressources)

réseaux de Petri (synchronisation des actions)

compteurs abstraits (accès concurrent aux objets)

8. EVOLUTION DES NOTIONS

Le coût de la commutation de processus lors du partage de processeur et le développement des systèmes répartis ont contribué à faire évoluer le partage et les notions associées.

On a introduit les notions **d'acteur** et de **processus léger** (activité, "thread").

Un **acteur** est un allocataire de ressources et d'espace d'adressage (mémoire virtuelle), regroupe les contextes pour ces ressources allouées et regroupe plusieurs processus légers. (objet passif)

Un **processus léger** utilise les ressources de l'acteur, en particulier son espace d'adressage, qu'il partage avec d'autres processus légers. (objet actif)

La commutation de processus léger se limite à la commutation du contexte du processeur et est moins coûteuse que la commutation d'acteur qui nécessite la commutation des espaces d'adressage et des autres contextes.

On a aujourd'hui un glissement de sens dans les systèmes
(Mach, Amoeba, Unix, AS400, Windows NT)

acteur : on utilise "process", processus ou tâche

activité : processus léger, "thread", proléger

A quand des **processus lourds**, nomades ou par morceaux sur le réseau ?

POUR LE COURS SYSTÈME ET L'ETUDE DE LA SYNCHRONISATION

On prend le terme processus ("process"ou "task") au sens de séquence d'actions, donc activité, "thread", "task"(Ada) et non au sens de réceptacle de ressource.

.

NOTION DE PROCESSUS

DÉFINITIONS

PROCESSUS

• séquence d'actions programmées
-- ou sa représentation interne

ÉTATS D'UN PROCESSUS

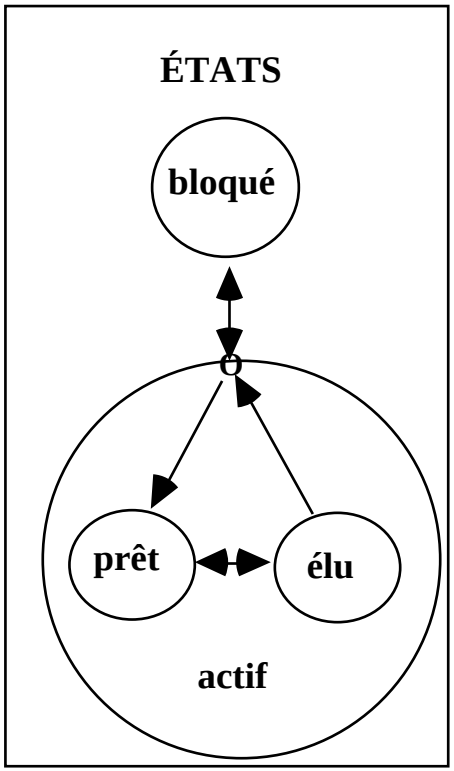
• intrinsèques : états de synchronisation

bloqué <-> actif

• sous-états d'ordonnancement du processeur

états pour le partage du processeur (ou des processeurs)

prêt <-> élu



Commandes Linux utilisées dans l'ED n°1

• gcc : compilation de programme

(GNU projet compilateurs C et C++ (v2.7))

gcc -c : produit uniquement un fichier objet

gcc -static prog -o : produit un fichier exécutable en faisant une liaison statique avec la bibliothèque système

gcc -o prog : idem et prépare une liaison dynamique (par défaut)

gcc -g : fournit des informations de mise au point en format système natif (stabs, COFF, XCOFF, or DWARF) (GDB peut fonctionner avec cette information de mise au point)

• ldd : imprime les dépendances de la bibliothèque partagée

• size : donne les tailles des sections et la taille totale

• objdump : fournit les informations disponibles sur les fichiers objets

• nm : fournit la liste des symboles issus des fichiers objets

• strace : trace les appels systèmes et les signaux

• gdb : le débrouilleur GNU ("GNU Debugger")

• ps : fournit l'état du processus

• sleep : met en attente pour une durée spécifiée

• ls : liste les éléments du répertoire courant

// Le programme de l'ED n°1

(fichier calcul_somme.c)

```
#include <stdio.h>
int VAR1 = 1;
int VAR2 = 1;

int calcul1( int n)
{
   int i, res = 0;
   for(i=0; i<=n; i++)
      res+=i;
   return res;
}

int calcul2( int n)
{
   int i, res = 0;
   for(i=0; i<=n; i++)
         res+=i*i;
   return res;
}

int main(int argn, char *argv[], char*env[])
{
   int nb_pas;
   int i;
   char *zone;

   for (i=0; i < 6; i++)
      printf("variable environnement:%s\n", env[i]);

   zone = (char *) malloc(1024);

   if (argn != 2){
      printf("il faut 1 argument : le nb de pas de
      calcul\n"); exit (-2);
   }

sscanf(argv[1], "%d", &nb_pas);

printf("\n\n\n\nnombre de pas = %d \n", nb_pas);
printf("somme = %d\n", calcul1(nb_pas));
printf("somme = %d\n", calcul2(nb_pas));
printf("Adresse de   main      = %09lx\n", main);
printf("Adresse de   VAR1      = %09lx\n", &VAR1);
printf("Adresse de   VAR2      = %09lx\n", &VAR2);
printf("Adresse de   nb_pas    = %09lx\n", &nb_pas);
printf("Adresse de   zone      = %09lx\n", zone);
printf("Adresse de   argv[1]   = %09lx\n", argv[1]);

sleep(2000);
exit(nb_pas);
}
```

// compilation et commandes de listage

```
#gcc -c calcul_somme.c
#ls -l calcul_somme.c
// visualiser des composantes du fichier objet
#nm calcul_somme.o
```

// édition de liens statique (liaison statique)

```
#gcc -static calcul_somme.c -o calcul_somme.st
#ls -l calcul_somme.st
#ldd calcul_somme.st
// visualiser les composantes du fichier executable
#nm  calcul_somme.st|egrep'printf|scanf|exit|calcul1'
```

// liaison dynamique

```
#gcc calcul_somme.c -o calcul_somme
#ls -l calcul_somme
// visualiser des composantes du fichier exécutable
#nm calcul_somme| egrep 'printf|scanf|exit|calcul1'
//imprimer les bibliothèques partagées
#ldd calcul_somme
// visualiser des noms de la bibliothèque partagée
#nm /lib/libc.so.6 | egrep 'printf|scanf|exit'
```

// exécution et trace du processus

```
// le processus après liaison statique

#strace calcul_somme.st 100

// le processus après liaison dynamique

#strace calcul_somme 100
```