C-41400
Programmation Répartie

C-41400-PR

- **Responsable : Emmanuel Saint-James**
- **Cours : Luciana Arantes**
- **TD et TME : E. Saint-James, L. Arantes, S. Dubois**

C41014 - PR

- **Cours 1 : Introduction, processus**
- **Cours 2: Signaux**
- **Cours 3 : Gestion de fichiers**
- **Cours 4 &5: Processus légers**
- **Cours 6: Tube**
- **Cours 7: IPC POSIX**
- **Cours 8 &9 : Communications distantes**
  - Sockets UDP et TCP
- **Cours 10 : Temps Réel**

Bibliographie

- **Le langage C (C Ansi)**
  Brian Kernighan, Dennis Ritchie, *Masson Prentice Hall*, 1991, 280 p.

- **Unix : Programmation et communication**
  J.-M. Rifflet, J.-B. Yunes, *Dunod,* 2003, 768 p.

- **Méthodologie de la programmation en C, Bibliothèque standard, API POSIX**
  J.-P. Braquelaire, *Dunod, 3e éd.*, Paris 2000, 556 p.

- **Programmation système en C sous Linux**
  C. Blaess, *Eyrolles, 2e éd.*, 2005, 964p.

- **Advanced programming in the Unix environment**
  W. Richard Stevens, *Addison-Wesley*, 1992, 768p.

Cours 1 – Rappels

- **La Norme POSIX**
- **Compilation / Makefile**
- **Processus**

1. La Norme POSIX

POSIX : principe

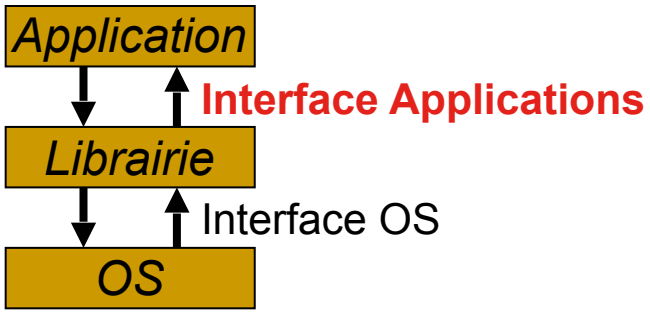
Portable Operating System Interface for Computing Environments

Document de travail

- Produit par IEEE
- Endossé par ANSI et ISO

API standard pour applications

- Définitions de services
  - définition du comportement attendu lors d'un appel de service
- Portabilité **garantie** pour les codes sources applicatifs qui l'utilisent
  - contrat *application* / *implémentation* (système)



Macro _XOPEN_SOURCE

#define **_XOPEN_SOURCE 700**

1. La Norme POSIX

- **POSIX : En fait, un ensemble de standards (IEEE 1003.x)**
  - Chaque standard se spécialise dans un domain
    - *1003.1 (POSIX.1) System Application Program Interface (kernel)*
    - *1003.2 Shell and Utilities*
    - *1003.4 (POSIX.4) Real-time Extensions*
    - *1003.7 System Administration*
- **Divisé en sections, 2 catégories de contenu :**
  - Bla-bla (Préambule, Terminologie, Contraintes, ...)
  - Regroupements de services par thème
    - Pour chaque service, une définition d'interface (*Synopsis, Description, Examples, Returns, Errors, References)*

1. La Norme POSIX

Exemple de définition d'interface

NAME

getpid - get the process ID

SYNOPSIS

```
#include <unistd.h>
pid_t getpid(void);
```

DESCRIPTION

The *getpid*() function shall return the process ID of the calling process.

RETURN VALUE

The *getpid*() function shall always be successful and no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

SEE ALSO

exec*(), *fork*(), *getpgrp*(), *getppid*(), *kill*(), *setpgid*(), *setsid*(), the Base Definitions volume of IEEE Std 1003.1-2001, *<sys/types.h>*, *<unistd.h>

2. Compilation/Makefile : Environnement de Programmation

Récupération d'arguments

Ligne de commande : `<nom_programme> <liste_arguments>`

ex. : `> myprog toto /usr/local 12`

Copiée par l'OS dans une zone mémoire accessible au processus

En C, récupération au niveau du `main`

```
main(int argc, char* argv[])
```

- `argc` nombre d'arguments (nom du programme inclus)
- `argv` tableau d'arguments (`argv[0]` = nom du programme)

ex. :

`argc`	*4*
`argv[0]`	*"myprog"*
`argv[3]`	*"12"*

2. Compilation/Makefile

- **Fichier *Makefile* (ou *makefile*)**
  - Constitué de plusieurs règles de la forme

    <cible>: <liste_prérequis>

    <commandes>

    NB : chaque commande est précédée d'une tabulation

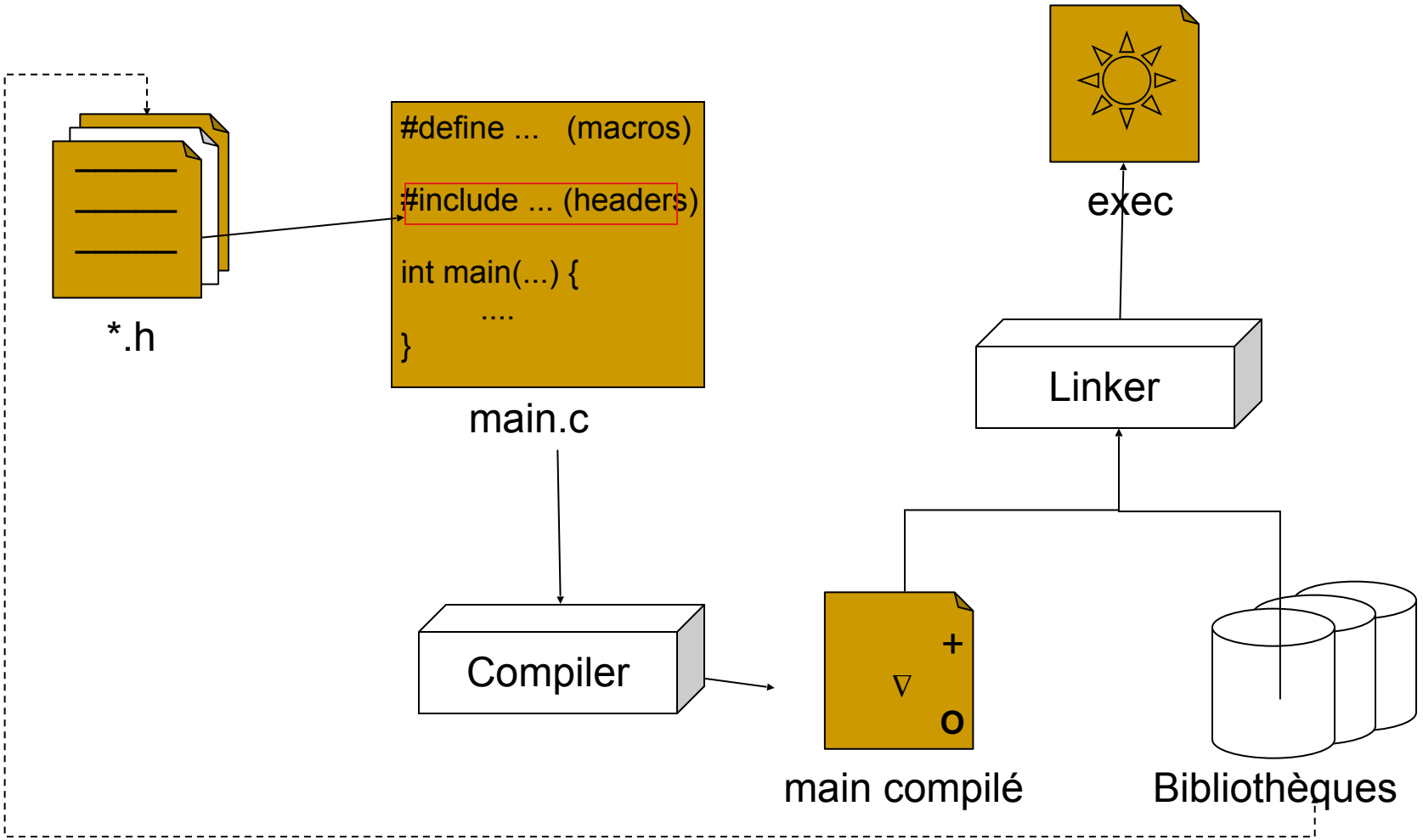
  - Prérequis

    <nom_fichier> Le fichier est-il présent ?

    <nom_cible> La règle est-elle vérifiée ?

  - Evaluation d'une règle en 2 étapes
    1. Analyse des prérequis (processus récursif)
    2. Exécution des commandes

2. Compilation/Makefile



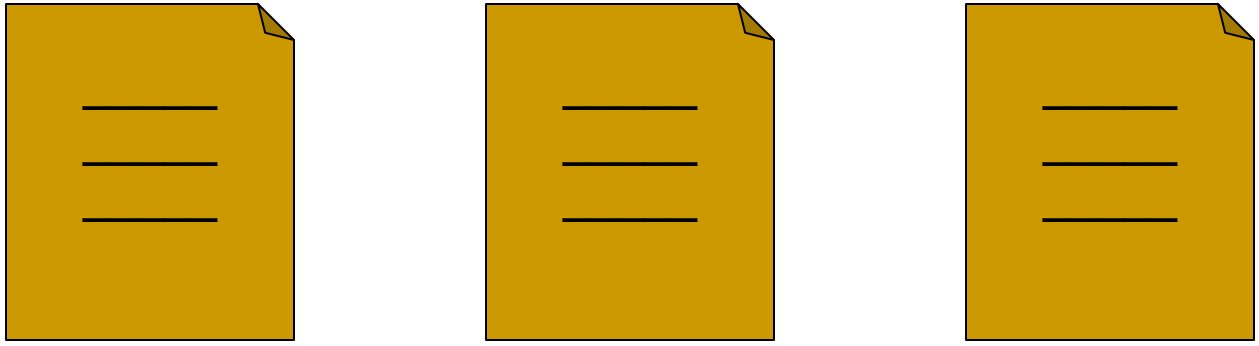
2. Compilation/Makefile

Quelques options de `gcc`

-ansi	Assurer le respect du standard ANSI
-Wall (Warning)	Afficher tous les avertissements générés
-c (cpp + cc1 + as)	Omettre l'édition de liens
-g	Produire des informations de déboguage
-D (Define)	Définir une macro
-M (Make)	Générer une description des dépendances de chaque fichier objet
-H (Header)	Afficher le nom de chaque fichier header utilisé
-I (Include)	Etendre le chemin de recherche des fichiers headers (`/usr/include`)
-L (Library)	Etendre le chemin de recherche des bibliothèques (`/usr/lib`)
-l (library)	Utiliser une bibliothèque (`lib<`*`nom_librairie`*`>.a`) durant l'édition de liens
-o (Output)	Rediriger l'output dans un fichier

2. Compilation/Makefile

Exemple : projet "HelloWorld"



2. Compilation/Makefile

Exemple : fichier 'hello.c'

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

void Hello(void){
        printf("Hello World\n");
}
```

2. Compilation/Makefile

Exemple : fichier 'hello.h'

```
#ifndef H_GL_HELLO
#define H_GL_HELLO

void Hello(void);

#endif
```

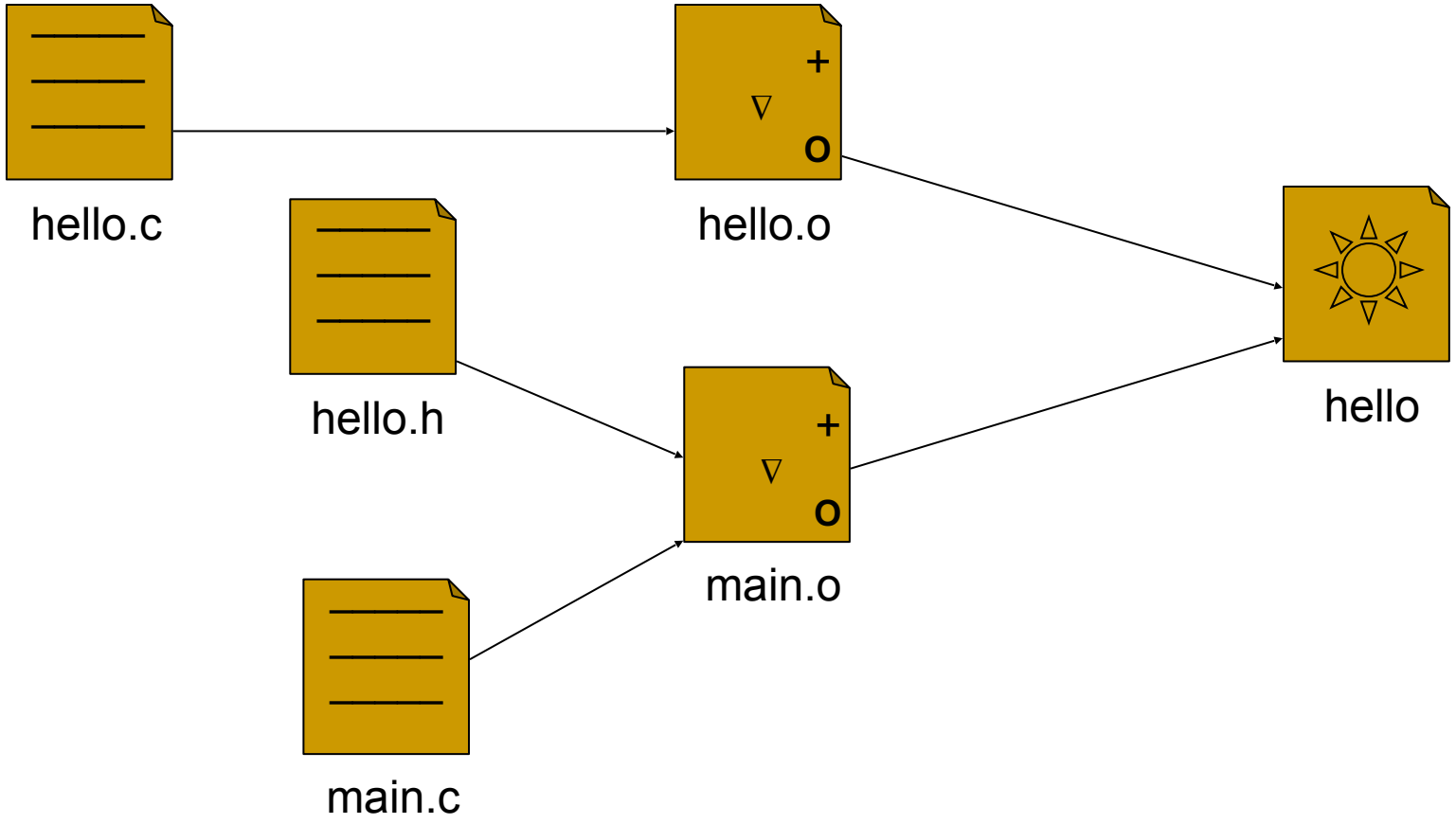
2. Compilation/Makefile

Exemple : fichier 'main.c'

```
#define _XOPEN_SOURCE 700

#include <stdio.h>
#include <stdlib.h>
#include "hello.h"

int main(int argc, char * argv[] ){
            Hello();
            return EXIT_SUCCESS;
}
```

2. Compilation/Makefile

Exemple : projet "HelloWorld" - dépendances



2. Compilation/Makefile

Exemple "HelloWorld" : fichier 'makefile' minimal

```
hello: hello.o main.o
	gcc -o hello hello.o main.o

hello.o: hello.c
	gcc -o hello.o -c hello.c -Wall -ansi

main.o: main.c hello.h
	gcc -o main.o -c main.c -Wall -ansi
```

2. Compilation/Makefile

Exemple "HelloWorld" : fichier 'makefile' enrichi - variables personnalisées

```
CC=gcc
CFLAGS=-Wall -ansi
EXEC=hello

all: $(EXEC)

hello: hello.o main.o
	$(CC) -o hello hello.o main.o

hello.o: hello.c
	$(CC) -o hello.o -c hello.c $(CFLAGS)

main.o: main.c hello.h
	$(CC) -o main.o -c main.c $(CFLAGS)

clean:
	rm -rf *.o $(EXEC)
```

2. Compilation/Makefile

Exemple "HelloWorld" : fichier 'makefile' enrichi - variables internes

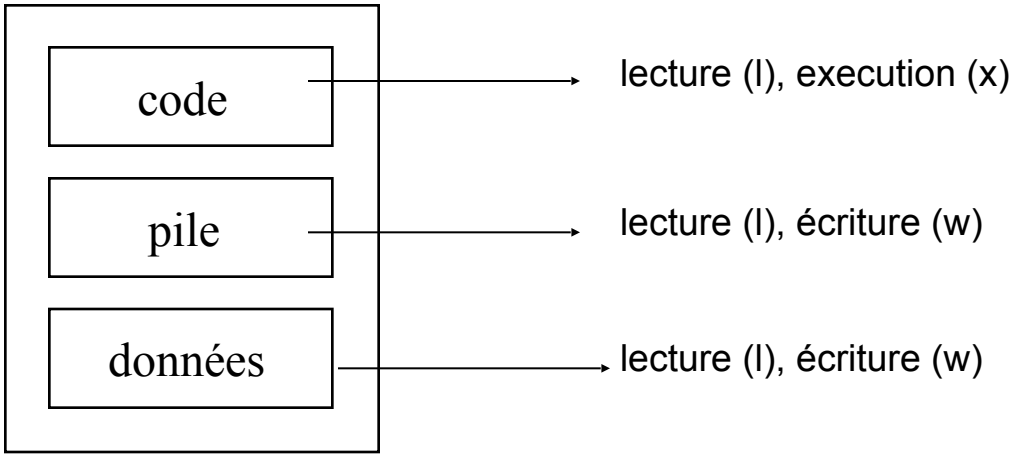
```
CC=gcc
CFLAGS=-Wall -ansi
EXEC=hello

all: $(EXEC)

hello: hello.o main.o
	$(CC) -o $@ $^

hello.o: hello.c
	$(CC) -o $@ -c $< $(CFLAGS)

main.o: main.c hello.h
	$(CC) -o $@ -c $< $(CFLAGS)

clean:
	rm -rf *.o $(EXEC)
```

$@	*target name*
*$^*	*list of dependencies*
$<	*name of 1st dependency*

3. Processus

- **Processus: entité active du système**
  - correspond à l'exécution d'un programme binaire
  - identifié de façon unique par son numéro : **pid**
  - possède 3 segments :
    - code, données et pile
  - Exécuté sous l'identité d'un utilisateur :
    - propriétaire réel et effectif
      - Groupe réel et effectif
  - possède un répertoire courant

3. Processus

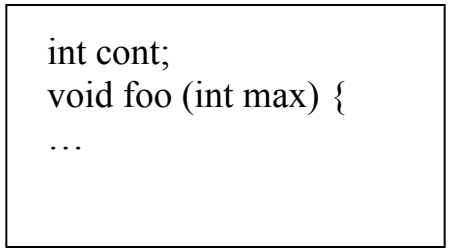


Processus

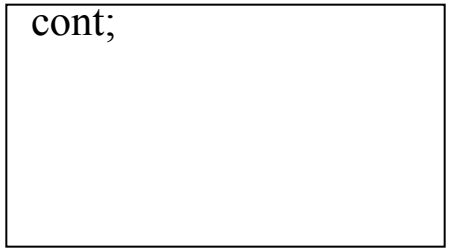
- **Chaque processus est indépendant**
  - Deux processus peuvent être associés au même programme (code)
  - Synchronisation entre processus (communication)
- **CPU est partagé (temps partagé)**
  - Commutation entre les processus

3. Processus : Execution Programme

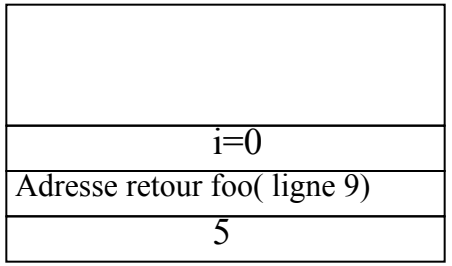
```
1: int cont;

2: void foo (int max) {
3:  int i;
4:  for (i=0; i++; i<max)
5:     printf ("%d \n", i);
6:}

 7: int main (int argc, char* argv []) {
 8: cont=5;
 9:  foo(cont) ;

10: return EXIT_SUCCESS;
11: }
```



code

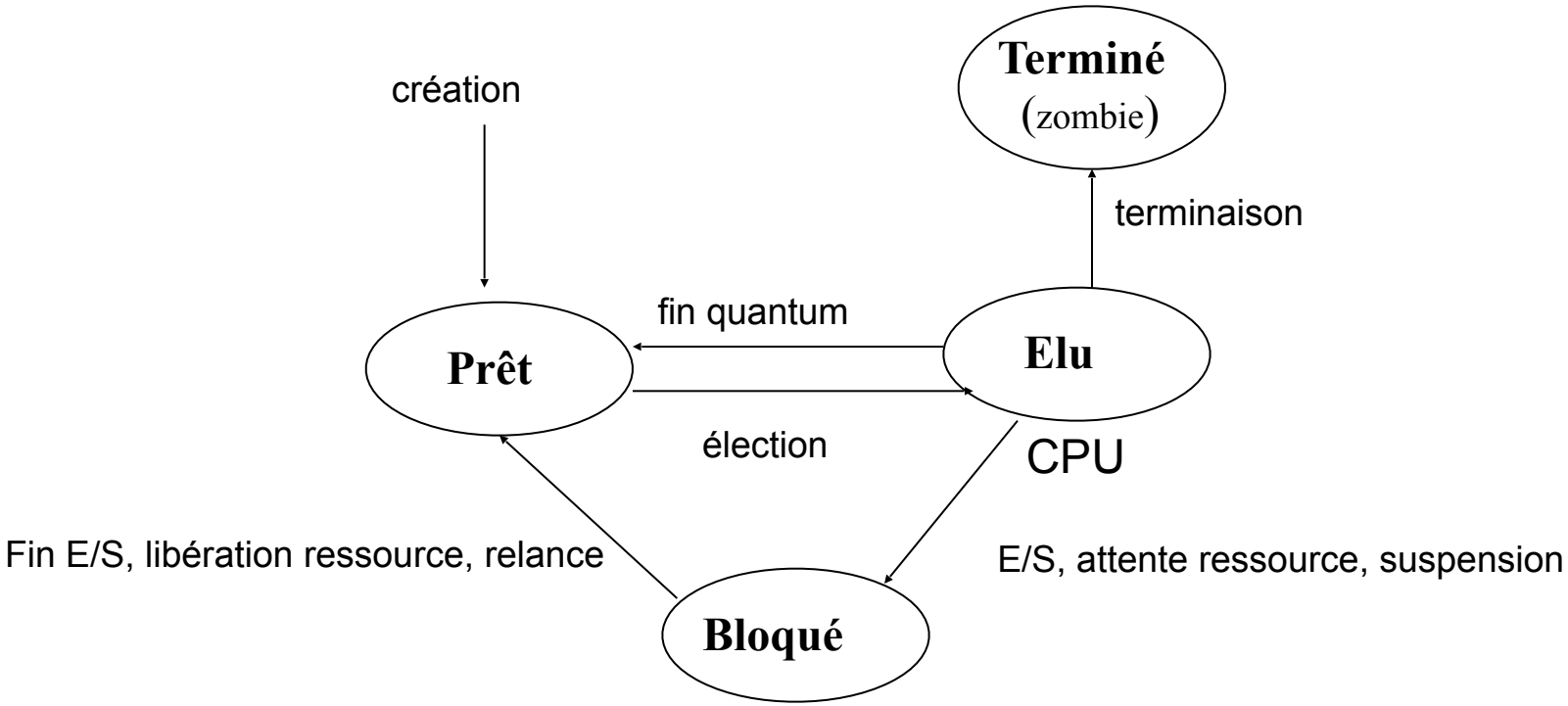


donnée



pile

3. Processus: Etats d'un processus



- **Quantum** : durée élémentaire (e.g. 10 à 100 ms)

3. Processus: Attributs d'un processus

- **Identité d'un processus:**
  - pid: nombre entier
    - POSIX: type *pid_t*
      - <unistd.h> : fichier à inclure
    - **pid_t getpid (void)** :
      - obtention du pid du processus

- **Exemple:**

```
#define _XOPEN_SOURCE 700
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main(int argc, char **argv) {
        printf (" pid du processus : %d \n", getpid()) ;
 return EXIT_SUCCESS;
}
```

3. ProcessusAttributs d'un processus (cont)

- **Un processus est lié à un utilisateur et son groupe**
  - Réel : Utilisateur (groupe):
    - Droits associé à l'utilisateur (groupe) qui lance le programme
  - Effectif: utilisateur (groupe):
    - Droits associé au programme lui-même
      - identité que le noyau prend effectivement en compte pour vérifier les autorisations d'accès pour les opérations nécessitant une identification
      - Exemple: ouverture de fichier, appel-système réservé.
  - UID (User identifier) GID (group identifier)
    - #include <sys/types.h>
    - Types *uid_t* et *gid_t*

3. Processus : création d'un processus

- **Primitive *pid_t fork (void)***
  - permet la création dynamique d'un nouveau processus (*fils*) qui s'exécute de façon concurrente avec le processus qui l'a créé (*père*).

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork (void)
```

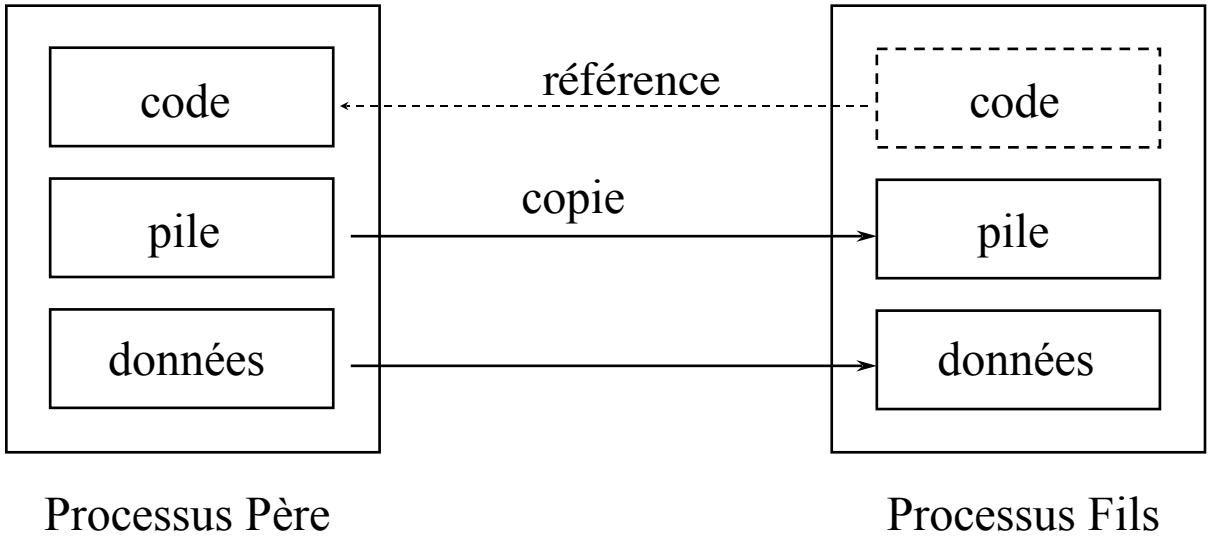
  - Processus fils créé est une copie du processus père

3. Processus: création d'un processus

- **Chaque processus reprend son exécution en effectuant un retour d'appel fork**
  - un seul appel à ***fork***, mais deux retours dans chacun des processus. Valeurs de retour diffèrent selon le processus
    - **0** : renvoyé au processus fils
    - **pid du processus fils** : renvoyé au processus père
    - **-1** : appel à la primitive a échoué
      - errno <errno.h> :
        - ENOMEM : système n'a plus assez de mémoire disponible
        - EAGAIN : trop de processus créés
  - pid_t getppid (void)
    - obtenir le pid du père

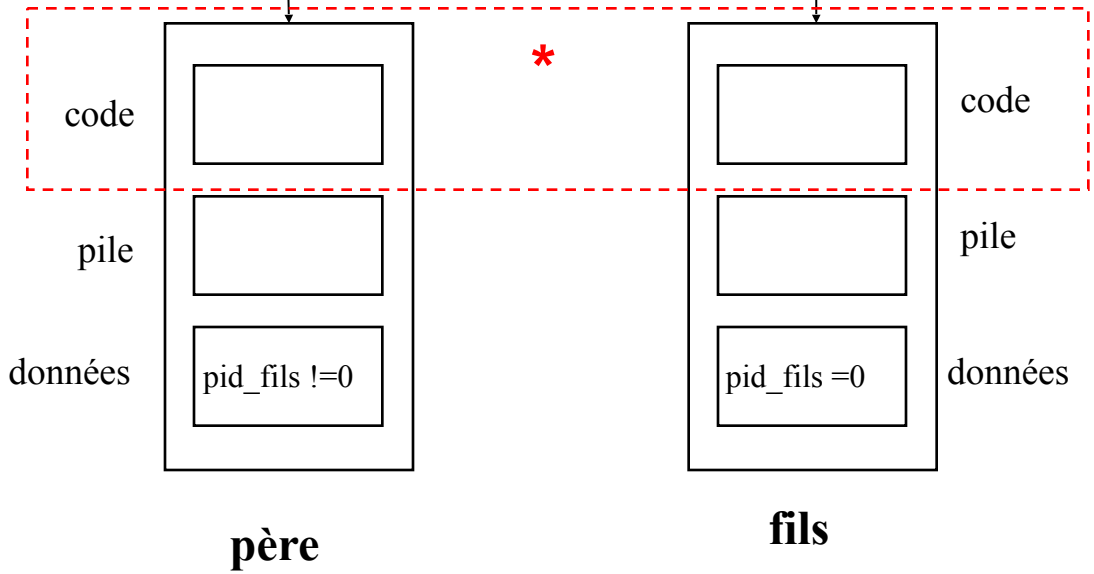
3. Processus: fork

- Les deux processus partagent le même code physique.
- Duplication de la pile et segment de données :
  - variables du fils possèdent les mêmes valeurs que celles du père au moment du *fork* ;
  - toute modification d'une variable par l'un des processus n'est pas visible par l'autre.



Fork : création d'un processus

```
int pid_fils;
 main (int arg, *argv []) {
 ….
 if ( ( (pid_fils= fork ( )) == 0)
 {/* fils */
   …
 }
 else {
 …
 }
```



3. Processus: Fork - exemple

test-fork1.c

```
#define _XOPEN_SOURCE 700
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char* argv []) {
    int a= 3; pid_t pid_fils;
        a *=2;
        if ( (pid_fils = fork ( ) )  == -1 ) {
              perror ("fork"); exit (1); }
        else
              if (pid_fils == 0) {
                  a=a+3;
                  printf ("fils : a=%d \n", a); }
        else
            printf ("pere : a=%d \n", a);
return EXIT_SUCCESS;
}
```

3. Processus: Fork exemple

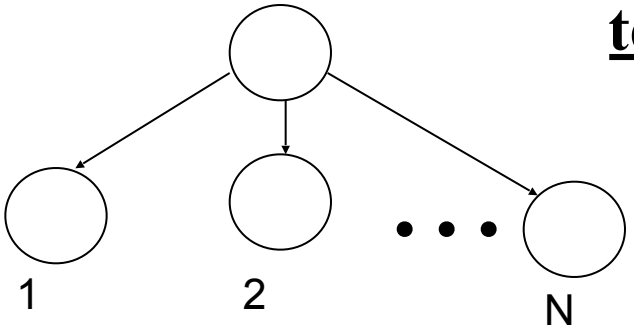
Combien de processus sont-ils créés?

test-fork2.c

```
#define _XOPEN_SOURCE 700
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char* argv []) {
   int i =0 ;
   while (i <3) {
     printf ( "%d ", i);
     i ++;
     if (fork ( ) == -1)
        exit (1);
   }
   printf( "\n ");
   return EXIT_SUCCESS;
}
```

3.Processus : fork exemple

- **Un processus crée N fils**



test-fork3.c

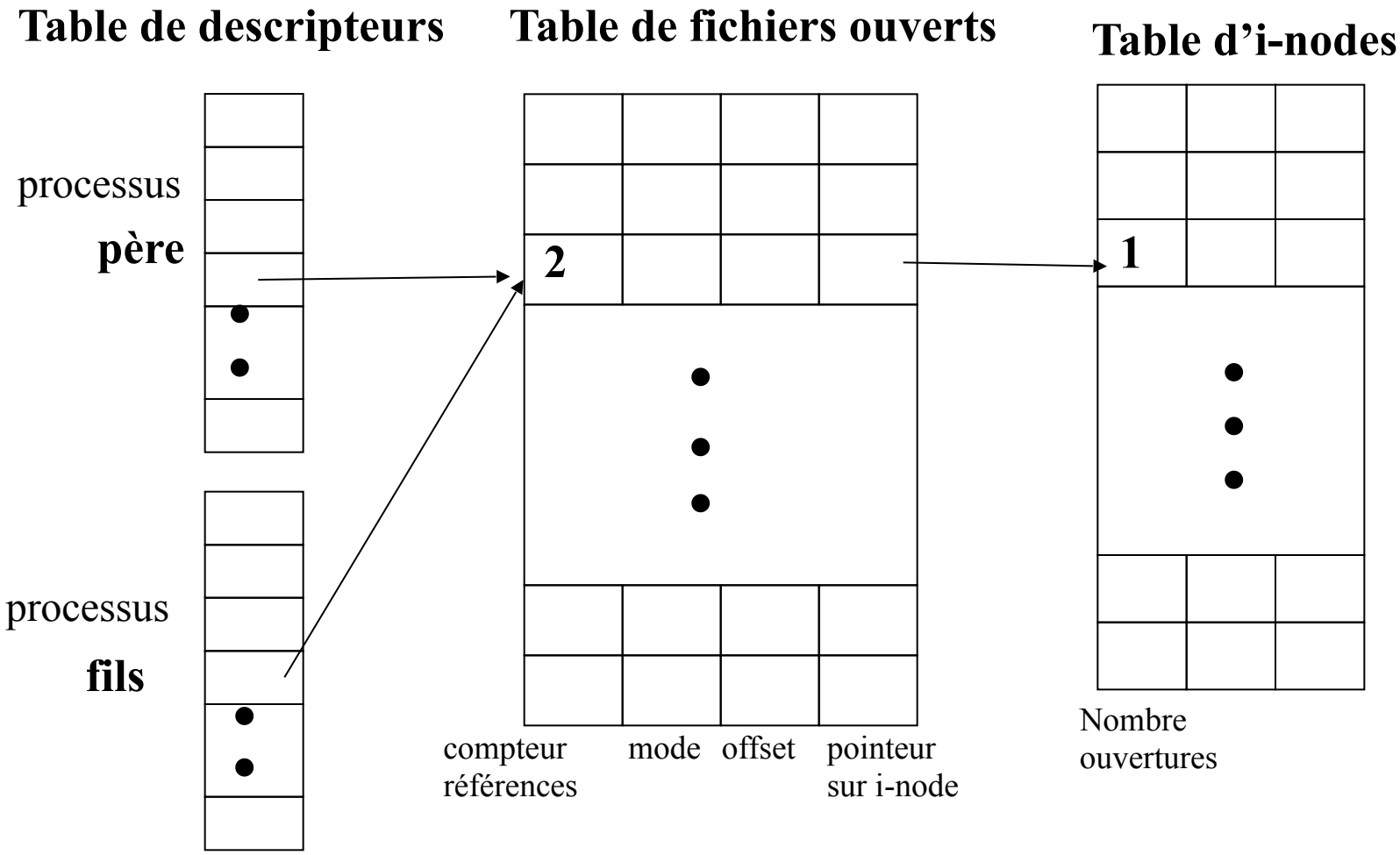
3. Processus: Fork - Héritage

- **Un processus hérite de(s) :**
  - ID d'utilisateur et ID de groupe
    - (réel et effectif)
  - ID de session
  - Répertoire de travail courant
  - Les bits de *umask*
  - Masque de signal et les actions enregistrées
  - Variables d'environnement
  - Mémoire partagée attachée
  - Les descripteurs de fichiers ouverts
  - Valeur de *nice*
  - ...

3. Processus: Fork - Héritage

- **<u>Un processus n'hérite pas de(s) :</u>**
  - identité (pid) du processus père
  - temps d'exécution
  - signaux pendants
  - verrous de fichiers maintenus par le processus père
  - alarmes ni temporisateurs
    - fonctions *alarm*, *setitimer*, …

3. Processus: Fork - Héritage de descripteurs de fichier



3. Processus: Fork - Héritage de descripteurs de fichier

test-fork3.c

```
#define _XOPEN_SOURCE 700
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <sys/wait.h>

#define SIZE_TAMPON 100
 char tampon [SIZE_TAMPON]; int status;

 int main (int argc, char* argv []) {
   int fd1, fd2;    int n,i;
  if ((fd1 = open (argv[1], O_RDWR| O_CREAT |
                 O_SYNC,0600)) == -1) {
       perror ("open \n");
      return EXIT_FAILURE;
}
if (write (fd1,"abcdef", strlen ("abcdef")) == -1) {
   perror ("write");
   return EXIT_FAILURE; }
if (fork () == 0) {
   /* fils */
     if ((fd2 = open (argv[1], O_RDWR)) == -1) {
       perror ("open \n");
       return EXIT_FAILURE;
     }
     if (write (fd1,"123", strlen ("123")) == -1) {
       perror ("write");
       return EXIT_FAILURE;
     }
     if ((n= read (fd2,tampon, SIZE_TAMPON)) <=0) {
       perror ("fin fichier\n");
      return EXIT_FAILURE;
     }
     for (i=0 ; i<n; i++)
       printf ("%c",tampon [i]);
     printf("\n");
     exit (0);
  }
  else /* père */
    wait (&status);
  return EXIT_SUCCESS;
}
```

```
>test-fork3 fich
  abcdef123
>cat fich
  abcdef123
```

3. Processus : Fork - Héritage de variable d'environnement

```
#define _XOPEN_SOURCE 700
#include <sys/types.h>
#include <sys/unistd.h>
#include <stdio.h>
#include <stdlib.h>

char* env; pid_t pid_fils;
```

```
int main (int argc, char* argv []) {
    env=getenv ("PATH");
    printf ("PERE: PATH=%s\n\n", env);
                if ( (pid_fils = fork ( ) )  == -1 ) {
                        perror ("fork"); exit (-1); }
                else
                  if (pid_fils == 0) {
                     printf ("FILS: PATH %s \n", env);
                     setenv("PATH",strcat (env,":./"),1);
                     env=getenv ("PATH");
                     printf ("FILS: PATH=%s \n\n", env); }
                  else {
                    sleep (1);
                    env=getenv ("PATH");
                     printf ("PERE: PATH=%s\n", env);
                  }
                  return EXIT_SUCCESS;
}
```

test-fork4.c

```
> test-fork4.c
PERE: PATH=/usr/bin: /usr/local/bin

FILS: PATH=/usr/bin: /usr/local/bin
FILS: PATH=/usr/bin: /usr/local/bin:/.

PERE: PATH=/usr/bin: /usr/local/bin
```

3. Processus : Terminaison d'un processus

- **Fonction *exit(int val)* ou *return val***
  - val: valeur récupérer par le processus père
  - Possible d'employer les constantes:
    - EXIT_SUCESS
    - EXIT_FAILURE
  - Processus lancé par le shell se termine, code d'erreur disponible dans la variable $?
    - echo $?

3. Processus : Terminaison d'un processus

- **Processus zombie:**
  - Etat d'un processus terminé tant que son père n'a pas pris connaissance de sa terminaison.
- **Synchronisation père/fils:**
  - En se terminant avec la fonction *exit* ou *return* dans main, un processus affecte une valeur à son *code de retour* :
    - processus père peut accéder à cette valeur en utilisant les fonctions *wait* et *waitpid*.

3. Processus: Wait - Synchronisation père/fils

- **Primitive pid_t wait (int* status)**

```
#include <sys/types.h>
#include <sys/wait.h>
 pid_t wait (int* status)
```

  - Si le processus appelant :
    - possède au moins un fils *zombie* :
      - la primitive renvoie l'identité de l'un de ses fils zombies et si le pointeur *status* n'est pas NULL, sa valeur contiendra des informations sur ce processus fils.
    - possède des fils, mais aucun n'est dans l'état zombie :
      - Le processus est bloqué jusqu'à ce que:
        - un de ses fils devienne *zombie*
        - il reçoive un signal .
    - ne possède pas de fils
      - l'appel renvoie -1 et errno = ECHILD.

3. Processus: Wait - Synchronisation père/fils

- **Interprétation de la valeur de retour - *int*status***
  - Utilisation des ***macros*** pour des questions de portabilité :
    - Type de terminaison
      - WIFEXITED : non NULL si le processus fils s'est terminé normalement.
      - WIFSIGNALED : non NULL si le processus fils s'est terminé à cause d'un signal
      - WIFSTOPPED : non NULL si le processus fils est stoppé (option WUNTRACED de waitpid)
    - Information sur la valeur de retour ou sur le signal
      - WEXITSTATUS : code de retour si le processus s'est terminé normalement
      - WTERMSIG : numéro du signal ayant terminé le processus
      - WSTOPSIG : numéro du signal ayant stoppé le processus

3. Processus: Wait - Synchronisation père/fils

- **Exemple :**

test-wait.c

```
#define _XOPEN_SOURCE 700
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>

int main(int argc, char **argv) {

  pid_t pid_fils; int status;
  if (fork () == 0) {
    printf ("FILS:  pid = %d \n",
              getpid());
    exit (2);
  }
else {
    pid_fils = wait(&status);
    if (WIFEXITED (status) ) {
      printf ("PERE: fils %d termine, status : %d \n",
                pid_fils,  WEXITSTATUS (status));
      return EXIT_SUCCESS;
  }
    else
      return EXIT_FAILURE;
  }
}
```

```
>test-wait
   FILS: pid= 3254
    PERE : fils 3254 termine, status : 2
```

3. Processus: Wait - Synchronisation père/fils

test-wait2.c

```
#define _POSIX_SOURCE 1
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>

int main(int argc, char **argv) {

  pid_t pid_fils; int status;
  if (fork () == 0) {
    printf ("FILS:  pid = %d \n",
                getpid());
   pause ();
    exit (2);
  }
```

```
else {
        pid_fils = wait(&status);
        if (WIFEXITED (status) ) {
             printf ( "PERE: fils %d termine avec status %d \n",
                        pid_fils, WEXITSTATUS (status));
              return EXIT_SUCCESS;
        }
       else
              if (WIFSIGNALED (status) ) {
                   printf ( "PERE: fils %d termine par signal %d \n",
                              pid_fils, WTERMSIG (status));
                    return EXIT_SUCCESS;
              }
        return EXIT_FAILURE;
   }
}
```

```
>test-wait2 &
   FILS: pid= 4897
> kill –KILL 4987
    PERE : fils 4987 termine par signal 9
```

3. Processus: Wait - Synchronisation père/fils

test-wait3.c

```
#define _POSIX_SOURCE 1
#include <sys/types.h>
#include <unistd.h>

#define N 3

int cont =0;
```

Quelle est la valeur affichée de *cont* ?

```
int main (int argc, char* argv []) {
   int i=0;  pid_t pid;
    while (i <N) {
       if ((pid=fork ( ) )== 0) {
         cont++;
         break;
       }
        i++;
     }

    if (pid != 0) {
      /* pere */
       for (i=0; i<N; i++)
        wait (NULL);
    printf ("cont:%d \n", cont);
   }
      return EXIT_SUCCESS;
}
```

3. Processus: Waitpid - Synchronisation père/ fils

- **Primitive *pid_t waitpid (pid_t pid, int* status, int opt)***

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t waitpid (pid_t pid, int* status, int opt )
```

  - en bloquant ou non le processus selon la valeur de *opt*, *waitpid* permet de tester la terminaison d'un processus fils d'identité *pid* ou qui appartient au groupe |pid|.
    - *status* possède des informations sur la terminaison du processus en question.

3. Processus: Waitpid - Synchronisation père/ fils

- **Valeur du paramètre *pid***

  > 0 du processus fils

  0 d'un processus fils quelconque du même groupe que l'appelant

  -1 d'un processus fils quelconque

  < -1 d'un processus fils quelconque dans le groupe |pid|

- **Valeur du paramètre *opt***
  - WNOHANG : appel non bloquant
  - WUNTRACED : processus concerné est stoppé dont l'état n'a pas été encore informé depuis qu'il se trouve stoppé.
- **Code renvoi**
  - -1 : erreur
  - 0 : en cas non bloquant, si le processus spécifié n'a pas terminé
  - *pid* du processus terminé

3. Processus: Waitpid - Synchronisation père/ fils

- **Exemple**

```
#define _XOPEN_SOURCE 700
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>

int main(int argc, char **argv) {
    pid_t pid_fils; int status;
  if ((pid_fils=fork ()) == 0) {
    printf ("FILS: pid=%d \n", getpid());
    sleep (1);
    exit (2);
  }
```

test-waitpid1.c

```
else {
    if (waitpid(pid_fils,&status,WNOHANG) == 0) {
      printf ("PERE: fils n'a pas terminé \n");
      return EXIT_SUCCESS;
      }
    else
      if WIFEXITED (status) {
              printf ("PERE: fils  %d terminé, status= %d \n",
              pid_fils, WEXITSTATUS (status));
              return EXIT_SUCCESS;
      }
      else
       return EXIT_FAILURE;
  }
}
```

>test-waitpid1
FILS : pid =19078
PERE: fils n'a pas terminé

3. Processus : exec - exécution de nouveaux programmes

- **Primitive exec: recouvrement**
  - ➢ permet de remplacer le programme qui s'exécute par un nouveau programme, dont le nom est passé en argument. Le nouveau programme sera exécuté au sein de l'espace d'adressage du processus appelant.
    - Si l'appel à *exec* **réussit**, il ne rend jamais le contrôle au processus appelant.
    - Exemple d'erreur (*errno*):
      - EACCES : pas de permission d'accès au fichier
      - ENOENT : fichier n'a pas été trouvé
      - …

3. Processus : exec - exécution de nouveaux programmes

- **Six fonctions de la famille exec**
  - *préfixe* = exec
  - plusieurs *suffixes* :
    - Forme sous laquelle les arguments *argv* sont transmis:
      - ***l* : argv** sous forme de liste
      - ***v* : argv** sous forme de tableau (v - vector)
    - Manière dont le fichier à exécuter est recherché par le système:
      - ***p***: fichier est recherché dans les répertoires spécifiés par *$PATH*. Si ***p*** n'est pas spécifié, le fichier est recherché soit dans le répertoire courant soit dans le *path* absolu passé en paramètre avec le nom du fichier.
    - Nouvel environnement
      - ***e***: nouvel environnement transmis en paramètre. Si ***e*** n'est pas spécifié, l'environnement ne change pas.

3. Processus : exec - exécution de nouveaux programmes

- **argv sous forme de liste :**

  int execl (const char *path, const char *arg, …);

  int execlp (const char *file, const char *arg, …);

  int execle (const char *path, const char *arg, …, char * const envp[]););

- **argv sous forme de tableau :**

  int execv (const char *path, char * const argv[]);

  int execvp (const char *file, char * const argv[]);

  int execve (const char *file, char * const argv[], char * const envp[]);

  - Dernier argument doit être NULL

3. Processus : exec - exécution de nouveaux programmes

- **Exemple : execl**

```
#define _XOPEN_SOURCE 700
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char **argv) {
  execl ("/usr/bin/wc","wc", "-w", "/tmp/fichier1", NULL);
  perror ("execl");
  return EXIT_SUCCESS;
}
```

3. Processus : exec - exécution de nouveaux programmes

- **Exemple : execlp**

```
#define _XOPEN_SOURCE 700

#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char **argv) {
  execlp ("wc","wc", "-w", "/tmp/fichier1", NULL);
  perror ("execlp");
  return EXIT_SUCCESS;
}
```