TP Système d'Exploitation (Operating System)

Linux –Distribution Ubuntu

://elearning-univ-skikda.dz

Niveau: L2

Année: 2021/2022

Initiation Rapide & Prise en Main de Linux-ubuntu

- Différents SE existent : Unix, Linux, OS, Windows, iOs, Android, Windows Phone, …
- OS (Mac, Apple) et Windows (Microsoft) : SE commerciaux (Copyright)
- Unix et Linux : SE open source (Copyleft) soumis à la licence GNU
- Windows et OS admettent plusieurs versions
- Linux : admet plusieurs distributions (RedHat, **Ubunto, Debian, Fedora, Mandriva, …)**
- Tous les laboratoires de recherche (en informatique), académies scientifique et universités du monde entier utilisent Linux.

Installation du Système Linux

•Différents environnements existent :PC, station de travail, serveur, …
•Chaque environnement a sa propre configuration machine : il faut en tenir compte !
•Pour Linux la commande : *linuxconfig permet d'avoir des infos sur la configuration*
•Plusieurs Modes d'installation :
–Linux seul sur une machine,
–Sous Machine virtuelle,
–Au choix au boot (au démarrage) avec un autre SE (ex. Windows ou Linux)

Types de commandes

Trois types de commandes existent :
–Pour **répertoires,**
–Pour **fichiers,**
–Pour **processus**
•Commandes pour Répertoires (directories)

cd	**pwd**	**ls**	**mkdir**	**rmdir**

•Commandes pour Fichiers (Files)

rm	**cp**	**mv**	**find**	**ln**	**cat**

•Commandes pour Processus, ex. : **ps -user**

Commandes pour Répertoires ls / mkdir / rmdir / cd / pwd

Ces commandes s'appliquent uniquement sur les répertoires

•Commande ls

ls : permet de lister le contenu d'un répertoire (liste d'un répertoire)

•Beaucoup d'options sont disponibles pour cette commande, parmi elles, certaines sont intéressantes

•ls (sans option) : liste les fichiers en plusieurs colonnes

•ls –l : liste les fichiers et répertoires avec les droits d'accès

•ls –a : liste tous les fichiers (même ceux commençant par un point)

•ls | more : liste écran par écran

•NB: La touche flèche ''haut'' du clavier vous permet de retrouver les commandes que vous avez déjà tapé.

Commandes pour Répertoires ls / mkdir / rmdir / cd / pwd

- Commande mkdir/rmdir

- **mkdir : (make directory) création d'un répertoire**

  mkdir <nom_répertoire> : permet de créer un répertoire

- **rmdir : (remove directory) destruction d'un répertoire**

  rmdir <nom_répertoire> : permet de supprimer un répertoire

Commandes pour Répertoires ls / mkdir / rmdir / cd / pwd

•Commande cd

La commande cd vous permet de se déplacer dans les répertoires tant que les permissions l'autorisent

•Tapez cd

•cd <sans paramètre> vous ramène dans votre répertoire personnel (rép. par défaut de l'utilisateur : ~ ou $HOME)

•cd /home : entrer dans le répertoire /home s'il existe !!

•Tapez cd .. (puis pwd), que constateriez-vous ? (pour remonter au rép. parent)

•Tapez cd /tmp (puis pwd), que constateriez-vous ?

•Tapez cd / que constateriez-vous ? (pour remonter au rép. racine)

•Exemple : Créer un répertoire avec : mkdir SE1G0

puis cd SE1G0

puis cd .. ou cd /

puis cd (rép. par défaut user ~ ou $home) puis rmdir SE1G0

Commandes pour Répertoires ls / mkdir / rmdir / cd / pwd

•**Commande pwd / cp / set**

pwd : permet de savoir dans quel répertoire vous êtes ! (affiche le rép. courant)

•Copiez les fichiers d'un répertoire /tmp/exos dans votre répertoire courant avec la commande :

cp /tmp/exos/*. (le point final ‘‘.’’ indique le répertoire courant)

•Utilisez ls et ls –al (notez les différences d'affichages)

•Tapez pwd. Notez le répertoire dans lequel vous êtes.

•Tapez cd /tmp puis pwd. Notez le répertoire dans lequel vous êtes

Commandes pour Fichiers cp / mv / find / chmod / ln / cat / more / less / diff / echo / rm

Ces commandes ne concernent que les fichiers.

•Un fichier est un texte ASCII (code ou script) stocké sur un emplacement physique (sur le disque) et admet

–*un lien physique (adresse ou adresse physique) et*

–*un lien logique (son nom ou nom logique)*

Commandes pour Fichiers cp / mv / find / chmod / ln / cat / more / less / diff / echo / rm

•*Commande* **cp / mv / find**

cp : permet de copier un ou plusieurs *fichiers d'un emplacement (répertoire) vers un autre répertoire (dont il faut préciser le chemin d'accès : "path")*

cp <nom_fichier> /home/perso

Copie le fichier dans un autre répertoire, par exemple **user/ftls**

•mv : permet de déplacer un ou plusieurs *fichiers d'un répertoire vers un autre*

mv <nom_fichier> /home/perso

•find : recherche de fichiers

•Cette commande permet de trouver des fichiers depuis une racine spécifiée suivant plusieurs critères (**-name : nom; -mode : dernière modification; …)**

•Exemple : dites que fait cette commande

find . / -name "*f*" -print

Commandes pour Fichiers cp / mv / find / ln / cat / more / less / diff / echo / rm

- **Commandes ln / cat :**

La commande **ln : ajout de liens sur les fichiers**

- ***ln permet d'ajouter un lien physique ou symbolique sur un fichier.***
- ln <fichier.origine> <Nom.lien> : Crée un lien physique
- ln -s <fichier.origine> <Nom.lien> : Crée un lien symbolique
- Cette fonction est intéressante, elle permet d'avoir un seul fichier physique sur le disque et de le désigner sous plusieurs noms logiques.
- Cela peut, dans certains cas, faciliter les opérations de mises à jour ou de maintenance d'applications.
- **cat : affichage ininterrompu d'un fichier**
- **cat permet d'afficher sans interruption un fichier ou plusieurs fichiers:**

cat fichier1 fichier2 (affiche fichier1 et fichier2)

Commandes pour Fichiers cp / mv / find / ln / cat / more / less / diff / echo / rm

- **Commande rm:**

rm : permet de détruire (supprimer) un ou plusieurs *fichiers (liens logiques) ou liens physiques du répertoire courant (dans certains cas il faut préciser le chemin "path")*

NB : Un fichier supprimé sous Linux ne peut pas être récupéré

- **rm <nom_fichier> : efface un fichier**
- **rm * : efface tous les fichiers du répertoire**
- **rm –i <fichier1> <fichier2> <fichier3> : –i demande confirmation de l'effacement de chaque fichier**

Commandes pour Fichiers cp / mv / find / ln / cat / more / less / diff / echo / rm

- **Commande rm:**

rm : permet de détruire (supprimer) un ou plusieurs *fichiers (liens logiques) ou liens physiques du répertoire courant (dans certains cas il faut préciser le chemin "path")*

NB : Un fichier supprimé sous Linux ne peut pas être récupéré

- **rm <nom_fichier> : efface un fichier**
- **rm * : efface tous les fichiers du répertoire**
- **rm –i <fichier1> <fichier2> <fichier3> : –i demande confirmation de l'effacement de chaque fichier**

Le Multi-Processing (Multi-tâches)

- Multi-(processing/traitement/Utilisateur/Users/tâches/tasks/ processus/threads …)
- Unix/Linux est un SE multi-tâche multi-utilisateurs
- Un utilisateur peut **lancer plusieurs tâches (programmes, processus ou commandes) en parallèle et même en concurrence**
- Pour comprendre le concept de multitâche sous Linux, lancer la commande :

ps *-eaf*

Le Multi-Processing (Multi-tâches)

On devrait obtenir des infos sous la forme :

UID	PID	PPID	C	STIME	TITY	TIME	CMD
root	1	0	0	08:27	?	00:00:04	Init [5]

•Maintenant, lancer la commande : ***gedit puis ps -eaf (sur 2 fenêtres différentes)***
•Chaque tâche (processus ou Cmd) appartient à quelqu'un (utilisateur) identifié par son **UID**
•Chaque processus est identifié par son numéro, c'est son **PID**
•Chaque processus est le fils d'un autre processus qui est identifié (le processus père) par son **PPID**
•Le reste des infos sont des ***infos de contrôle qui servent au scheduler (ordonnanceur de tâches)***
•Pour n'avoir que les processus qui concernent l'utilisateur actuel, lancer la commande suivante ***: ps -eaf | grep login_utilisateur***

Commandes sur les Processus

Lister vos processus à l'aide de la commande : ***ps -user***

–Notez le **PID de celui qui contient la chaine *–bash***

•Tapez la commande: **kill -9 N°PID**

–où **N°PID est le numéro du processus que vous avez relevé.**

–Que s'est-il passé ?

•Tapez

–la commande **clear pour nettoyer l'écran puis**

–la commande **top**

Commandes sur les Processus : ps

ps : fournit la liste des processus actifs

•Tapez : ***ps***

Cette commande écrite seule permet de lister les processus courant

•**ps -user**

Liste les processus appartenant à l'utilisateur **user**

•**ps -aux**

Liste de tous les processus du **système**

•**jobs -l**

Liste des jobs avec leur numéro de job ainsi que leur numéro de processus système

•**NB : la commande top : permet de connaître les processus gourmands en puissance de calcul.**

•Ce qui nous intéresse c'est la première colonne PID, qui est le numéro des processus. C'est ce numéro qui est utilisé par **kill**

Commandes sur les Processus : kill

kill : cette commande permet de détruire (tuer, supprimer) un processus

•Exemple : si le processus cible est le numéro (PID) **546**

•**kill 546**

Tente de détruire le processus 546

•**kill -9 546**

Force la destruction du processus 546

Éditeur de texte sous Linux

Comment écrire son programme ?

•Comme l'écriture de programmes et de scripts passe par l'utilisation d'un éditeur de texte, on se familiarisera avec l'éditeur *gedit*

Programmation en C sous Linux-Ubuntu

Compiler un programme C

- Le Compilateur **GCC = GNU (OpenSource Project) C Compiler**
- C'est le **Compilateur C/C++ pour Linux**
- **gcc s'exécute à partir de l'interface en ligne de commande.**

Ecrire et Compiler mon premier programme C

Ecriture du programme

•À partir du shell (terminal) taper la commande: gedit hello.c

•Ecrire le programme suivant:

```
#include <stdio.h>
int main() {
printf ("Hello word ! \n");
return 0;
}
```

• Compiler et exécuter le programme

- pour Compiler et exécuter ce programme on tape la commande:
  gcc hello.c
- Puis la commande : ./a.out

```
[sysexploi@local]$ gcc  hello.c
[sysexploi@local]$ ./a.out
Hello word !
```

Quelques Options Utiles

-o filename : permet de changer le nom du fichier de sortie (output).

```
[sysexploi@local]$ gcc  hello.c -o Test
[sysexploi@local]$ ./Test
Hello word !
```

Initiation Rapide & threads

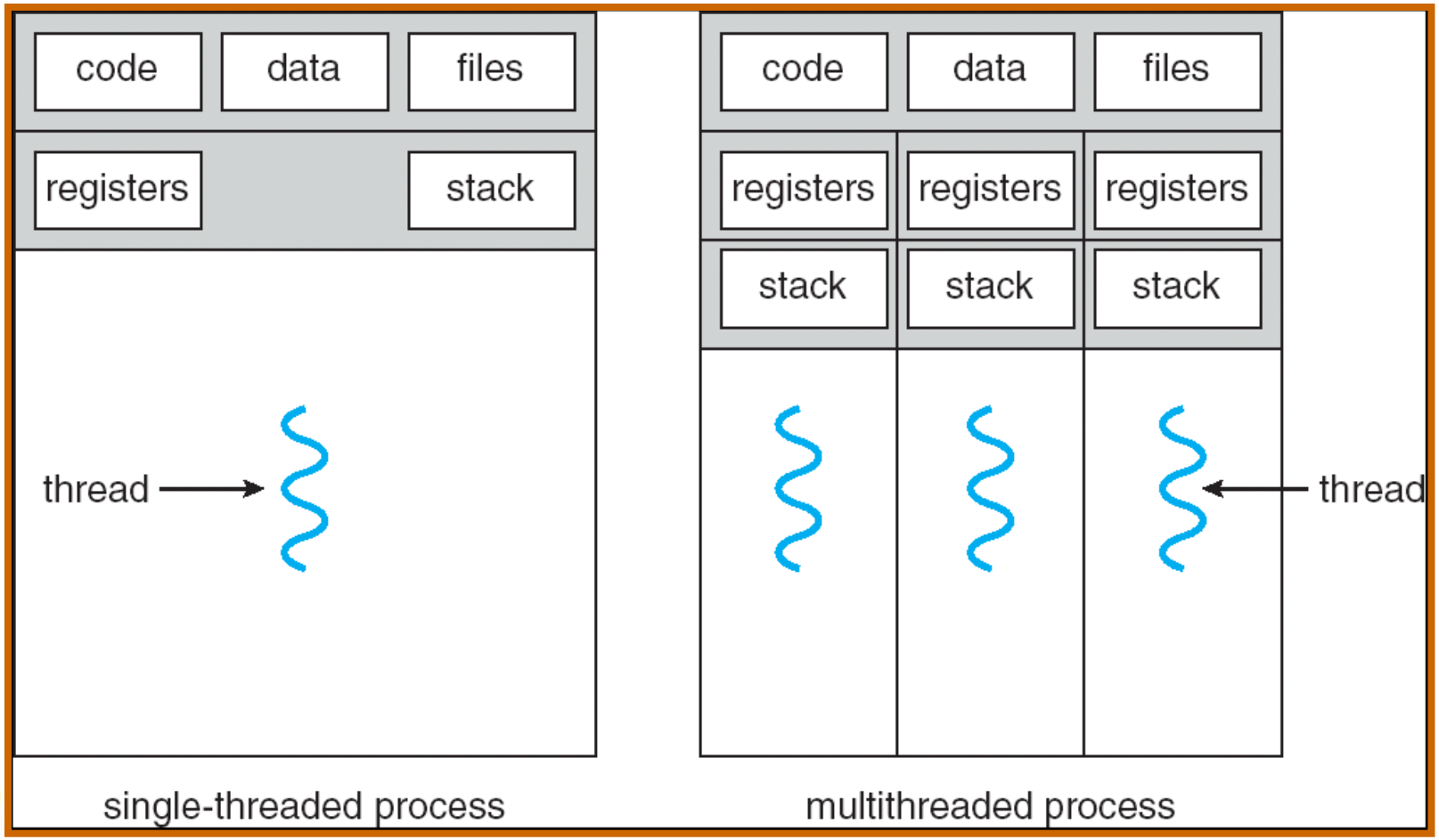
threads = lightweight processes(processus léger), activité, tâche, fil d'exécution

- **Un thread est une subdivision d`un processus**
  - Un fil de contrôle dans un processus
- **Les différents threads d 'un processus partagent l'espace adressable et les ressources d'un processus**
  - lorsqu'un thread modifie une variable (non locale), tous les autres threads voient la modification
  - un fichier ouvert par un thread est accessible aux autres threads (du même processus)

Exemple

- **Le processus MS-Word implique plusieurs threads:**
  - Interaction avec le clavier
  - Rangement de caractères sur la page
  - Sauvegarde régulière du travail fait
  - Contrôle orthographe
  - Etc.
- **Ces threads partagent tout le même document**

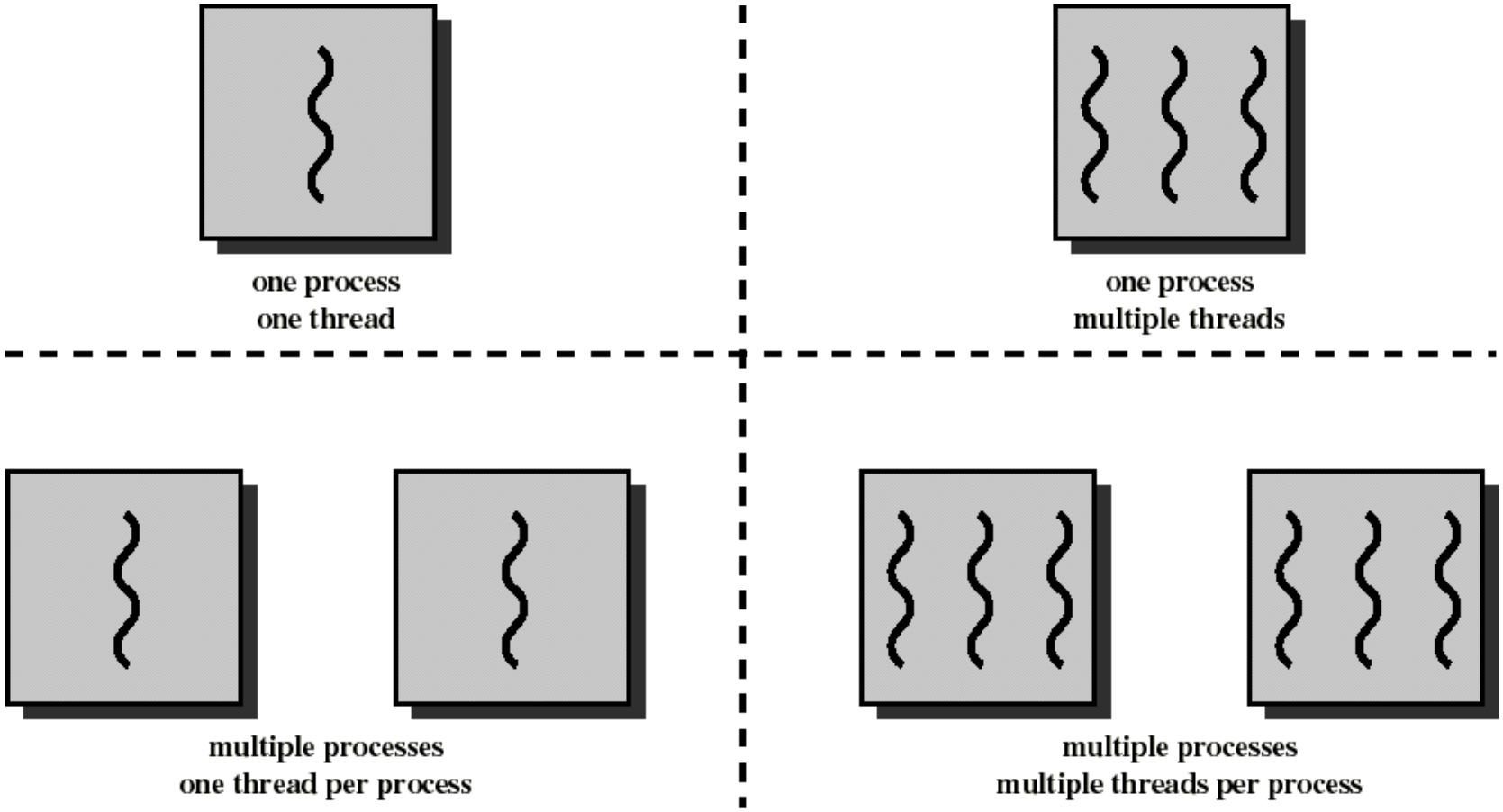
Processus à un thread et à plusieurs threads



Mono-flot

Multi-flots

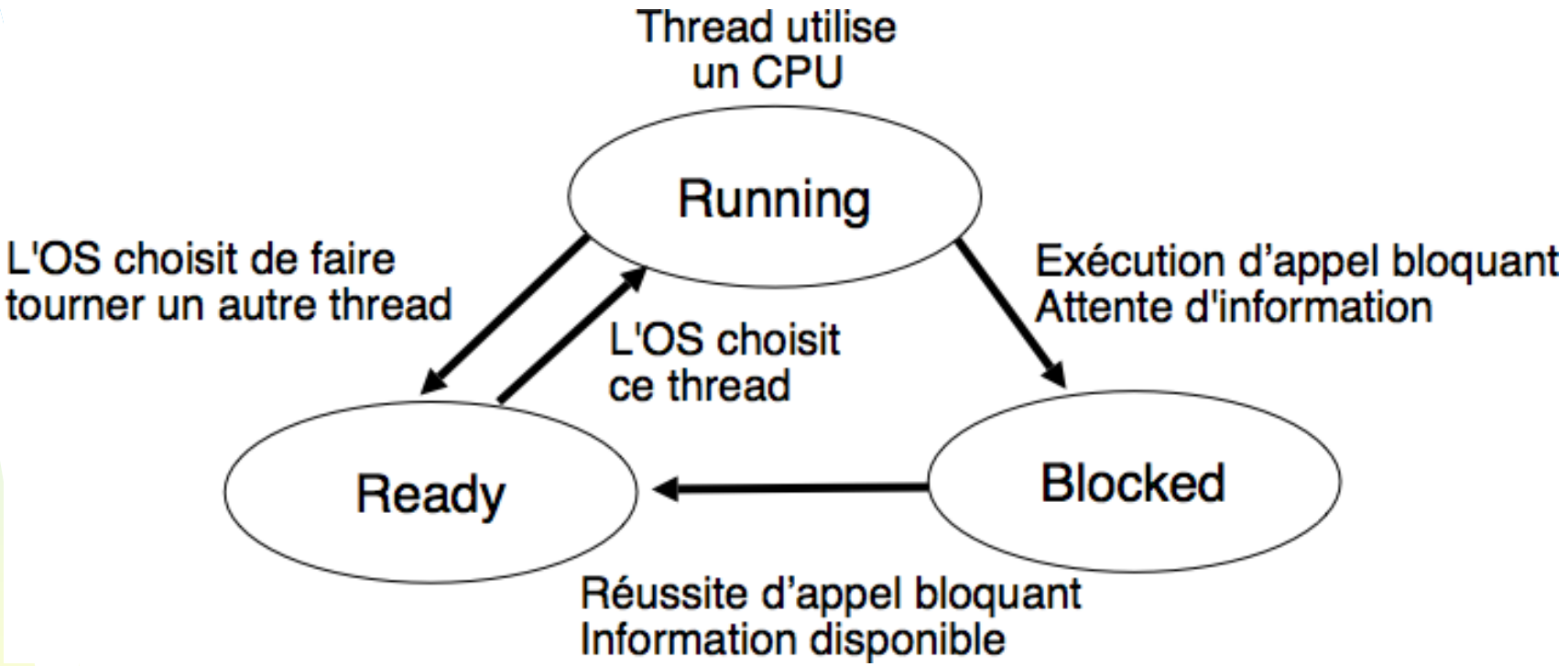
Threads et processus



Thread

- **Possède un état d'exécution (prêt, bloqué…)**
- **Possède sa pile et un espace privé pour variables locales**
- **A accès à l'espace adressable, fichiers et ressources du processus auquel il appartient**
  - En commun avec les autres threads du même processus

Thread


Thread utilise
un CPU
Running
L'OS choisit de faire
tourner un autre thread
L'OS choisit
ce thread
Exécution d'appel bloquant
Attente d'information
Ready
Blocked
Réussite d'appel bloquant
Information disponible

Pourquoi les Threads

Les threads sont un moyen populaire d'améliorer l'application par le parallélisme.(permettent de dérouler plusieurs suites d'instructions, en PARALLELE, à l'intérieur d'un même processus. Un thread exécutera donc une fonction).

Les threads fonctionnent plus rapidement que les processus pour les raisons suivantes :

1) La création de threads est beaucoup plus rapide.

2) Le passage d'un contexte à l'autre est beaucoup plus rapide.

3) Les threads peuvent être terminés facilement

4) La communication entre thread est plus rapide

Pthreads

- **Une norme POSIX (IEEE 1003.1c) d'un API pour la création et synchronisation de thread**
- **Commun dans les systèmes d'exploitation (OS) UNIX (Solaris, Linux, Mac OS X)**
- **Fonctions typiques:**
  - pthread_create (&threadid,&attr,start_routine,arg)
  - pthread_exit (status)
  - pthread_join (threadid,status)
  - pthread_attr_init (&attr)

192.168.57.2 - siteDev* - SSH Secure Shell

File Edit View Window Help

Quick Connect Profiles

```
PTHREAD_CREATE(3)                                        PTHREAD_CREATE(3)

NAME
       pthread_create - create a new thread

SYNOPSIS
       #include <pthread.h>

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

DESCRIPTION
       pthread_create creates a new thread of control  that  executes  concur-
       rently  with  the  calling  thread. The new thread applies the function
       start_routine passing it arg as first argument. The new  thread  termi-
       nates  either explicitly, by calling pthread_exit(3), or implicitly, by
       returning from the start_routine function. The latter case  is  equiva-
       lent  to calling pthread_exit(3) with the result returned by start_rou-
       tine as exit code.

       The attr argument specifies thread attributes to be applied to the  new
       thread.   See  pthread_attr_init(3)  for  a  complete  list  of  thread
       attributes. The attr argument can also be NULL, in which  case  default
:
```

Connected to 192.168.57.2 SSH2 - aes128-cbc - hmac-md5 - none 80x24

Exemple de programmation avec Threads

(cas avec 1 seul thread)

```
/* onethread.c */
#include <stdio.h>
#include <stdlib.h>   // conversion des nombres, gestion mémoire...
#include <unistd.h>  //Header file for sleep().
#include <pthread.h>

void *ThreadFun()
{
    sleep(1);
    printf("Hello Je suis un Thread \n");
    return NULL;
}

int main()
{
    pthread_t thread_id;
    printf("Avant Thread\n");
    pthread_create(&thread_id, NULL, ThreadFun, NULL);
    pthread_join(thread_id, NULL);
    printf("Aprés Thread\n");
    exit(0);
}
```

Explication du code

Dans main() nous déclarons une variable appelée thread_id, qui est de type pthread_t, qui est un entier utilisé pour identifier le Thread dans le système.

Après avoir déclaré thread_id, nous appelons la fonction pthread_create() pour créer un thread. pthread_create() prend 4 arguments.

- **Le premier argument est un pointeur vers le thread_id qui est défini par cette fonction.**
- **Le deuxième argument spécifie les attributs (exe, la taille de la pile, la politique d'ordonnancement, etc.). Si la valeur est NULL, alors les attributs par défaut sont utilisés.**
- **Le troisième argument est le nom de la fonction à exécuter pour le Thread à créer.**
- **Le quatrième argument est utilisé pour passer des arguments à la fonction, ThreadFun.**
- **La fonction pthread_join() fait attendre la fin d'un thread(comme wait() pour les processus).**

Compilation programme avec thread

pour compiler ce programme il faut établir un lien avec la bibliothèque des threads :

```
gcc –pthread onethread.c –o test1
./test1
```

Cas avec 2 threads

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>  //Header file for sleep(). man 3 sleep for details.
#include <pthread.h>

void *fun1(void *arg)
{
    sleep(1);
    printf("Hello Je suis le Thread 1\n");
    return NULL;
}

void *fun2(void *arg)
{

    Sleep(1);
    printf("Hello Je suis le Thread 2\n");
    return NULL;
}

int main()
{
    pthread_t t1;
    pthread_t t2;
    pthread_create(&t1, NULL, fun1, NULL);
    pthread_create(&t2, NULL, fun2, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    return 0;
}
```

cas avec 2 threads partageant une variable (1/2)

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

int VAL = 0; /* variable glabale partagée */


void* Thread1 (void* arg)
{
 int i;
            for(i=0;i<100;i++ ){
                    VAL=VAL+1;
           printf("thread 1: %d \n", VAL);
          }

          pthread_exit(NULL);
}


void* Thread2 (void*arg)
{
           int i;
            for(i=0;i<100;i++ ){

                    VAL=VAL+1;
                printf("thread 2: %d \n", VAL);
          }

          pthread_exit(NULL); /* Fin du thread */
}
```

cas avec 2 threads partageant une variable (2/2)

```
int main (void)
{

/* Déclaration de variable de type thread */
     pthread_t t1;
     pthread_t t2;

/* Création et lancement des threads 1 et 2 */
          pthread_create (&t1, NULL,Thread1, (void*)NULL);
          pthread_create (&t2, NULL, Thread2, (void*)NULL);

/* Attendre la fin des threads pour terminer le  main */
     pthread_join (t1, NULL);
     pthread_join (t2, NULL);

/* Fin Normale du programme */
     return 0;
}


int main (void)
{

/* Déclaration de variable de type thread */
     pthread_t t1;
     pthread_t t2;

/* Création et lancement des threads 1 et 2 */
          pthread_create (&t1, NULL,Thread1, (void*)NULL);
          pthread_create (&t2, NULL, Thread2, (void*)NULL);

/* Attendre la fin des threads pour terminer le  main */
     pthread_join (t1, NULL);
     pthread_join (t2, NULL);

/* Fin Normale du programme */
     return 0;
```