MODULE
ADMINISTRATION & PROGRAMMATION SYSTÈME & RÉSEAUX

PARTIE : PROGRAMMATION SYSTÈME & RÉSEAUX

Cours intégré

- 1.5h/semaine
- Mécanismes

Travaux Pratique

- 1.5h/semaine
- Pratiques

Projet

PLAN DU COURS

1) Concepts et outils
2) Processus
3) Communication
4) Thread
5) Synchronisation
6) Fichiers
7) Nouvelles techniques de programmation système

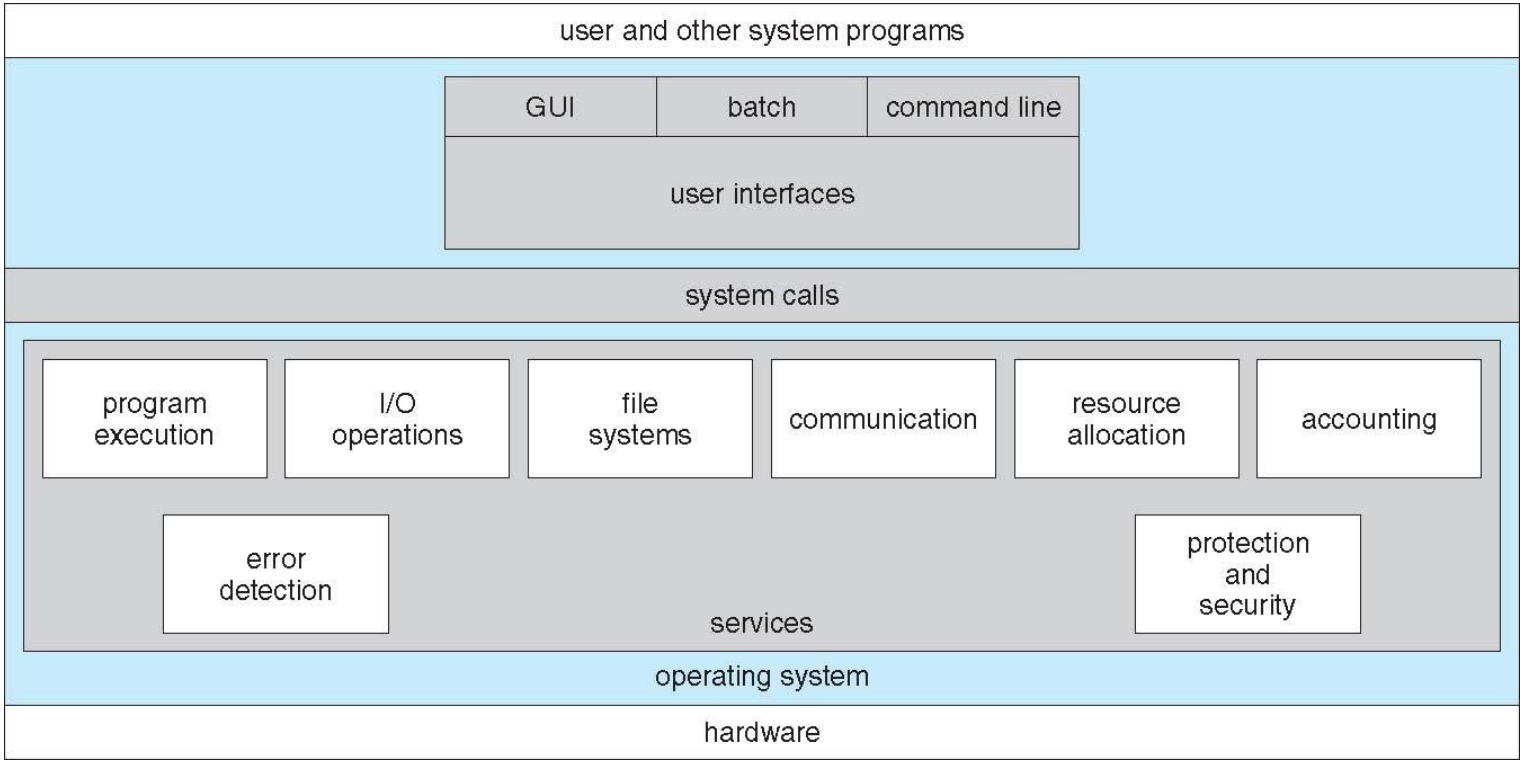
Référence : Christophe Blaess, Programmation système en C sous Linux, Edition Eyrolles

CONCEPTS ET OUTILS

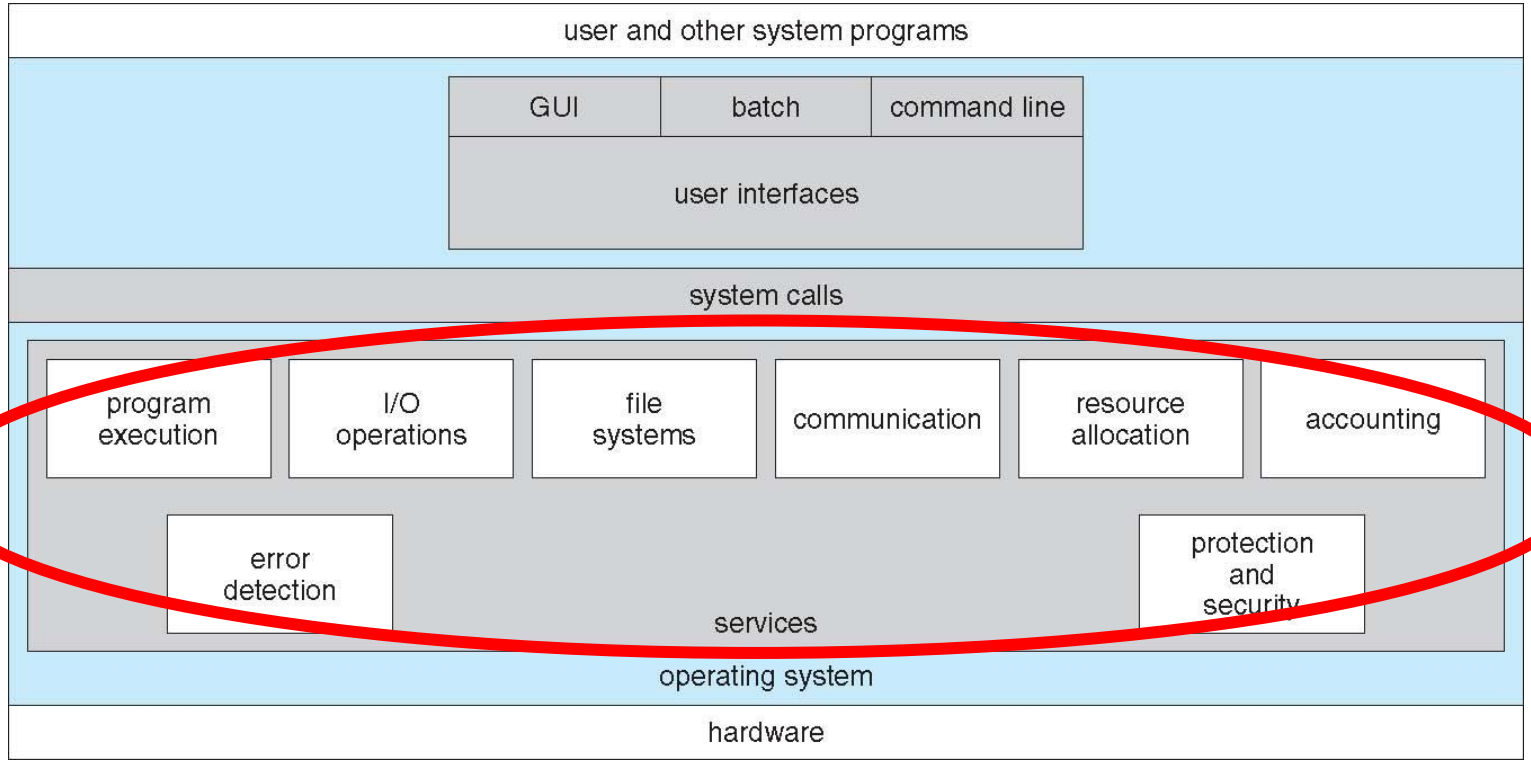
PLAN

1. Développement sous Linux
2. Outils de développement
3. Construction d'une application

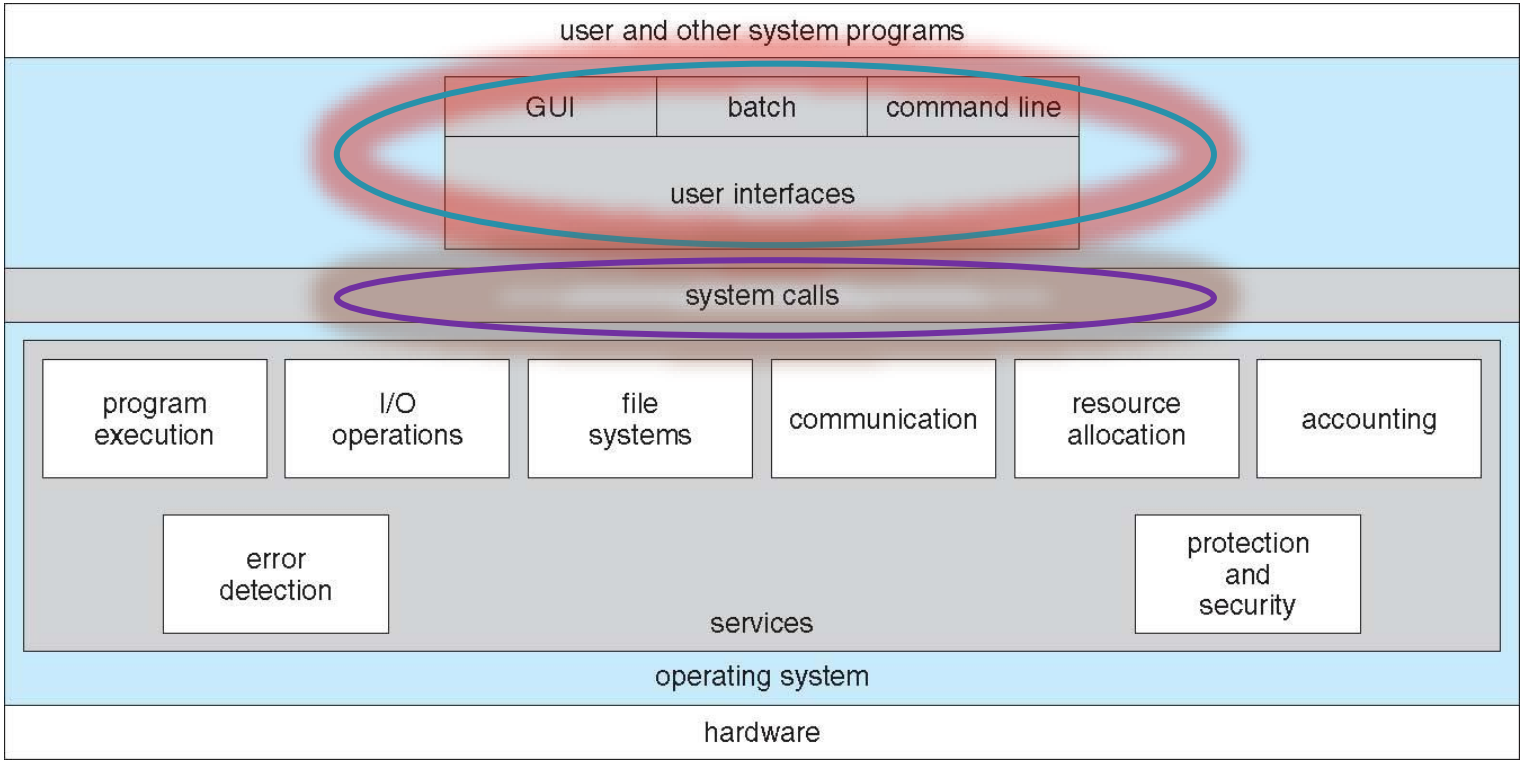
1. Structure du SE



2. Services offerts



3. Accès aux services



3. Accès aux services

a) Interfaces utilisateurs (User Interface: UI)

Les services offerts par le SE sont accessibles via des interfaces utilisateurs.

Trois formes sont possibles :

- **Interfaces de ligne de commande (Command-line Interface: CLI):** Saisie directe des commandes.
- **Interfaces batch (Batch interface):**

  Saisie des commandes dans des fichiers.
- **Interfaces graphiques (Graphical User Interface: GUI):**

  Commande dépend de l'action choisie du menu graphique (via la souris ou l'écran tactile).

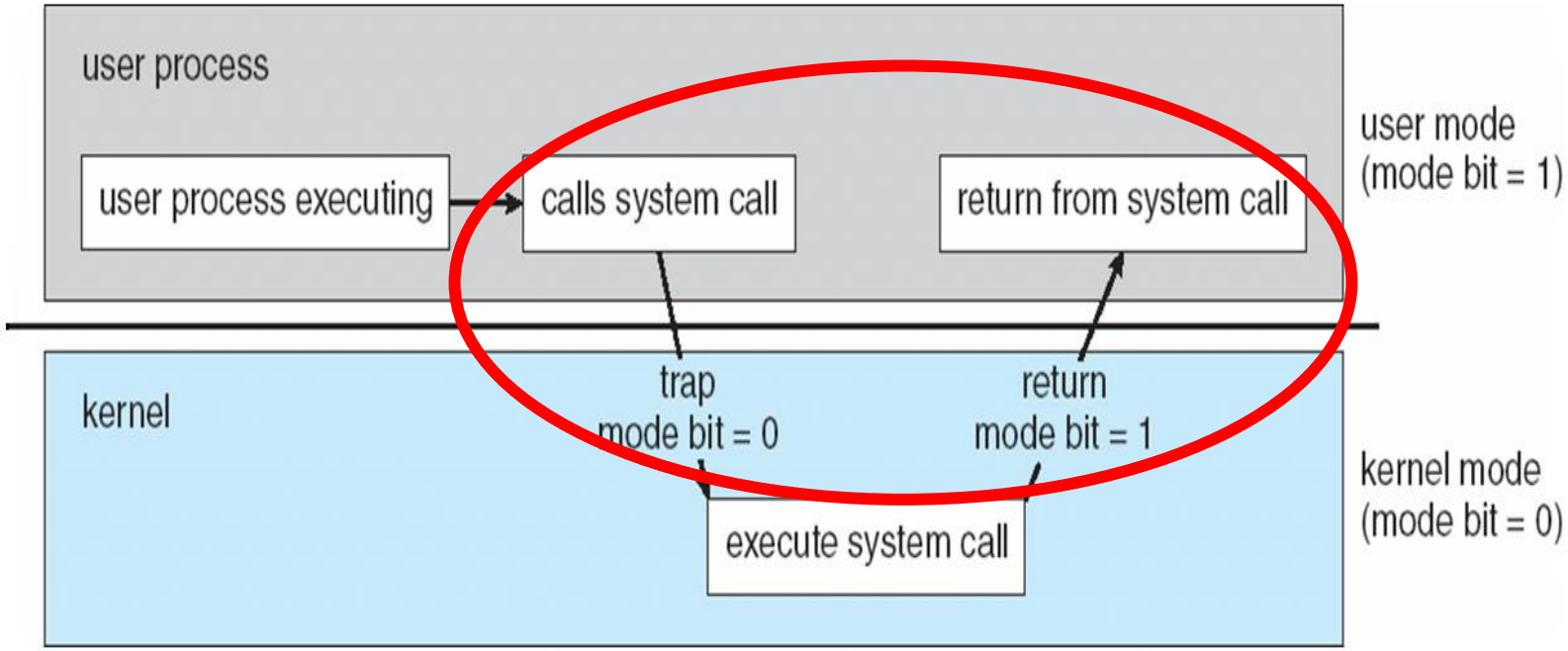
3. Accès aux services

b) Appel Système

- Le processeur a deux modes de fonctionnement :
  - **Mode noyau**
  - **Mode utilisateur**
- Ce mode est mémorisé dans un bit de PSW (Program Status Word).
- Ce bit est dit **mode bit,** il est fournit par le matériel pour distinguer le mode d'exécution (exécution du code de l'utilisateur ou du code du SE).

3. Accès aux services

b) Appel Système



3. Accès aux services

b) Appel Système

Différents types d'appels système:

- **Gestion des processus**
- **Gestion des Fichiers**
- **Gestion des Interruptions**
- **Communication et stockage de l'information**

3. Accès aux services

b) Appel Système

- **Point d'entrée offert par le noyau qui permet au programmeur d'invoquer des services variés**
- **Normalisé POSIX**
  - **description des fonctions d'appel système fournis par le noyau**
  - **portabilité des applications**
  - **offre un jeu d'appels système commun**

3. Accès aux services

b) Appel Système

- Pour assurer des applications vers des environnement non POSIX
  - Ajout d'une couche au noyau : Bibliothèque C
    - Cette bibliothèque regroupe des fonctionnalités complémentaires de celles qui sont assurées par le noyau. Exemple toutes les fonctions mathématiques (le noyau n'utilise jamais les nombres réels).
    - La bibliothèque C permet aussi d'encapsuler les appels système dans des routines de plus haut niveau, qui sont donc plus aisément portables d'une machine à l'autre.

3. Accès aux services

b) Appel Système

- **Le programmeur dispose donc :**
  - **des appels système, implémentés par le noyau et offrant un accès de bas niveau aux fonctionnalités du système,**
  - **des routines de bibliothèques qui peuvent compléter les possibilités du noyau, mais aussi l'encadrer pour le rendre plus simple et plus portable.**

3. Accès aux services

b) Appel Système

- L'invocation d'un appel système est une opération coûteuse (commutation du processus, manipulations des registres, …).
- L'appel d'une fonction de bibliothèque au contraire est un mécanisme léger, équivalent à l'appel d'une sous-routine du programme (sauf bien entendu quand la fonction de bibliothèque invoque elle-même un appel système).

OUTILS DE DÉVELOPPEMENT

1. Processus de développement C

❖ Editeur de texte

permet de créer et de modifier les fichiers sources, exemple :

- **Emacs :** éditeur de texte qui a des possibilités d'extension par l'intermédiaire de scripts Lisp
- **vi,** beaucoup plus léger mais il est disponible sur toutes les plates-formes Unix et peut être utilisé même sur un système très réduit pour réparer des fichiers de configuration.
- **Nedit** est très intuitif il a une puissance surprenante, tant dans la création de macros que dans le lancement de commandes externes (make, spell, man…), ou la manipulation de blocs de texte entiers.
- **Kde et Gnome** proposent tous deux un éditeur de texte parfaitement incorporé dans l'ensemble des applications fournies. Malheureusement ces éditeurs ne sont pas vraiment très appropriés pour le programmeur

OUTILS DE DÉVELOPPEMENT

1. Processus de développement C

❖ Compilation et édition de lien

– Le compilateur utilisé est **gcc** (Gnu C Compiler)

- On peut l'invoquer par gcc ou cc (gCC ou g++ si le code est en C++)
- Par défaut la compilation d'un fichier se fait dans un fichier **a.out** (si le nom du fichier exécutable n'est pas précisé)

OUTILS DE DÉVELOPPEMENT

1. Processus de développement C

❖ Compilation et édition de lien

– ***gcc File.c***

➢ ***Pour lancer l'exécutable***

./a.out

– ***gcc File.c -o Fich***

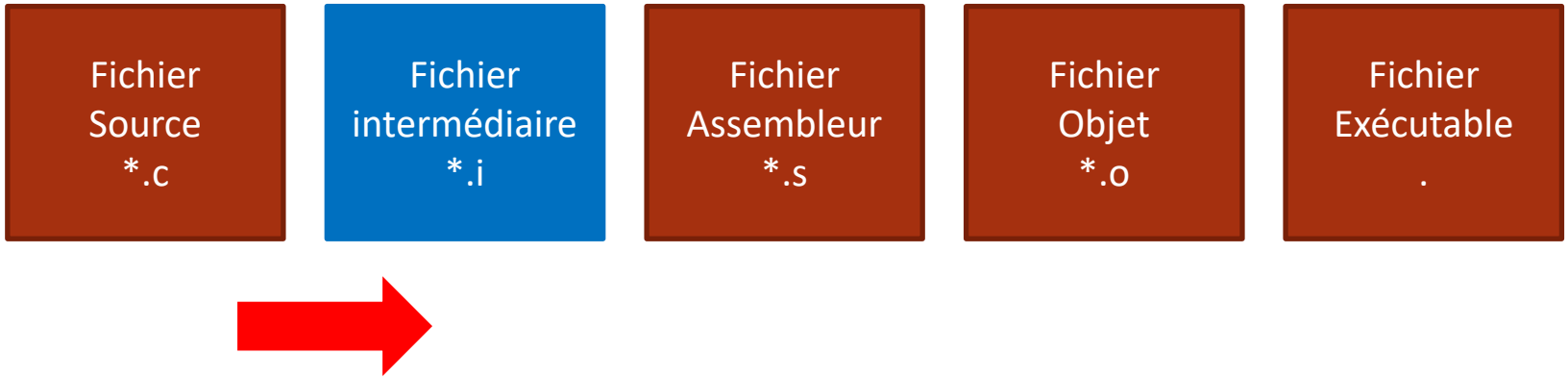
➢ ***Pour lancer l'exécutable***

./Fich

2. Compilateur gcc

Fichier Source *.c	Fichier intermédiaire *.i	Fichier Assembleur *.s	Fichier Objet *.o	Fichier Exécutable .

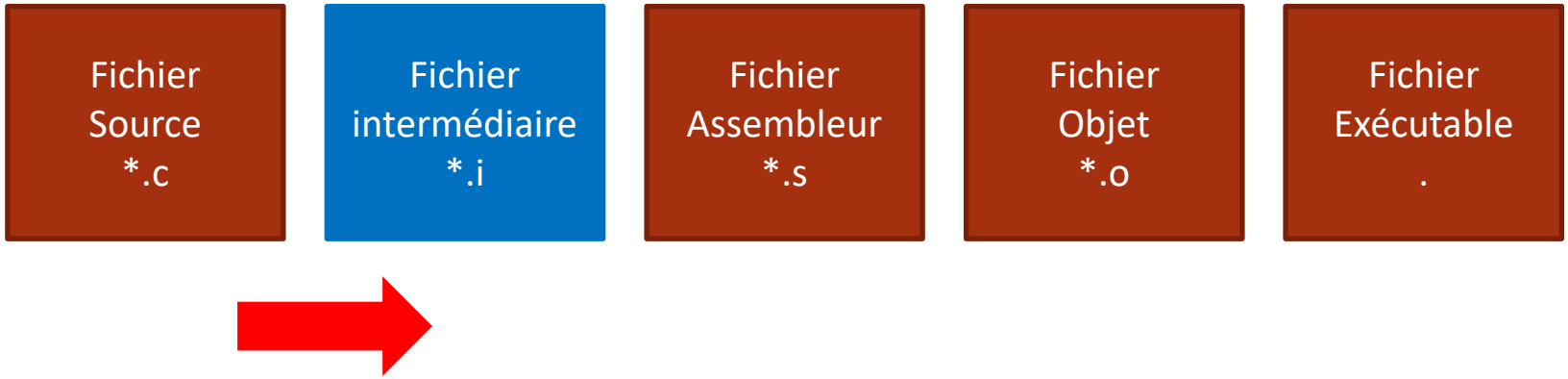
2. Compilateur gcc



Préprocessing : réalise plusieurs opérations de **substitution :**

- Suppression des commentaires
- Inclusion des fichiers .h
- Traitement des directives de compilation : #define, #if …
- Contrôle le syntaxe

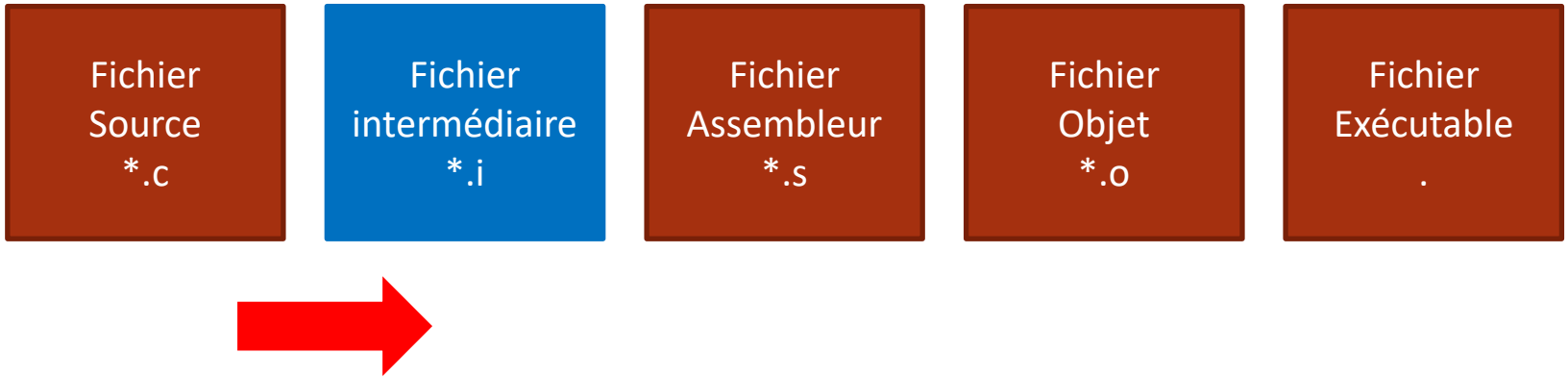
2. Compilateur gcc



Préprocessing :

- Les erreurs de compilation sont bloquantes, mais pas les avertissements (warning).
- Pour les afficher, il faut utiliser l'option -Wall (W pour Warning).

2. Compilateur gcc



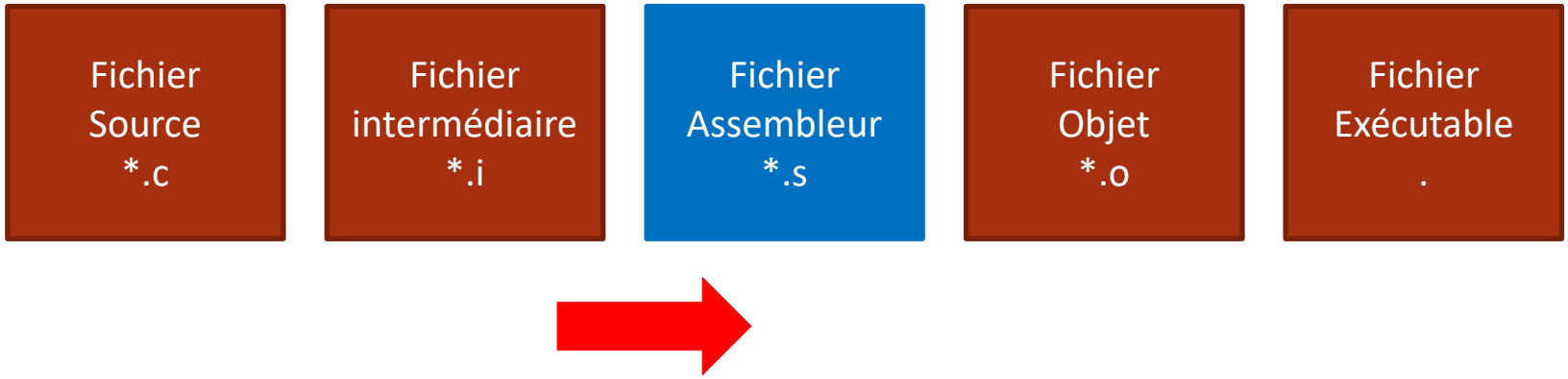
Préprocessing :

- **gcc -E Fichier.c > Fichier.i**.

L'option **-E** de gcc permet d'interrompre la compilation après la première phase.

Le **>** indique qu'il faut stocker le résultat du preprocessing dans

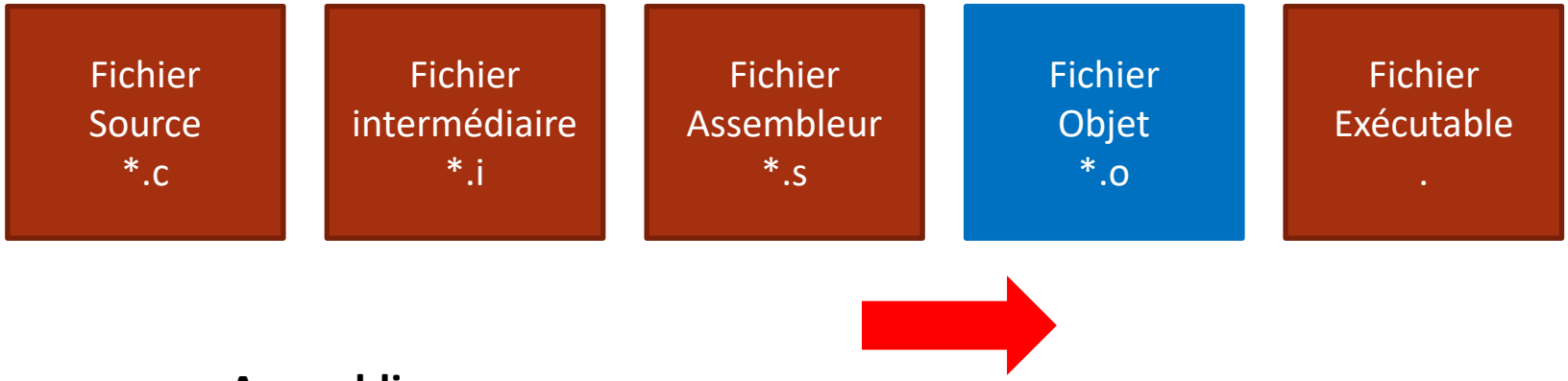
2. Compilateur gcc



Compiling :

- Le code du langage C est transformé en code Assembleur
- Il faut donner à gcc l'option –S : **gcc -S Fichier.i**
- Le fichier produit est un texte lisible en assembleur *.s

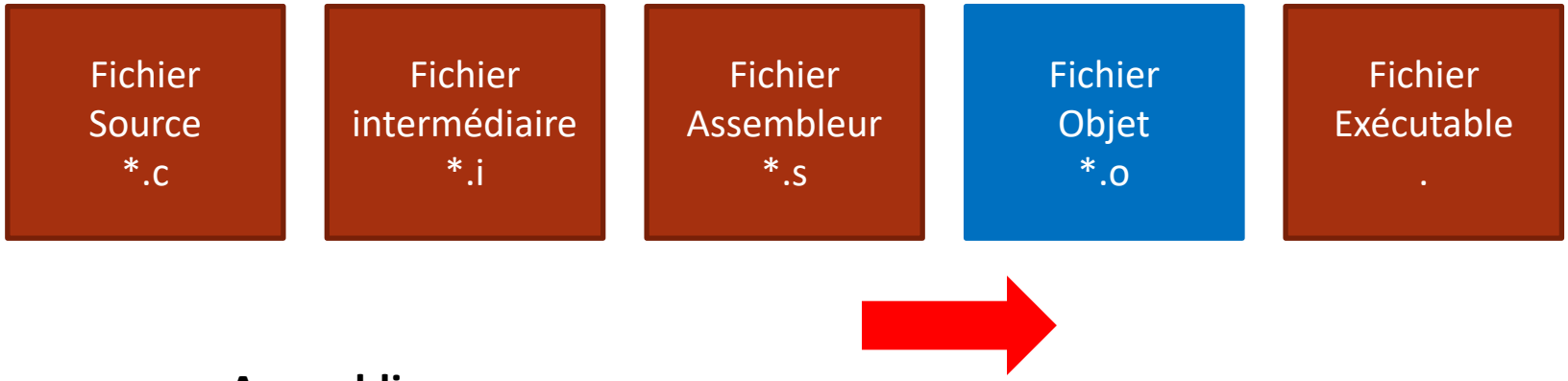
2. Compilateur gcc



Assembling :

- Le code assembleur est transformé en code machine binaire
- Il faut donner à gcc l'option -o
- Le fichier objet *.o est binaire donc non lisible mais peut être lu par **od -x *.o** (avec od : octal dump qui affiche sur la sortie standard les *octets* en hexadécimal avec l'option - x est la base 16)

2. Compilateur gcc

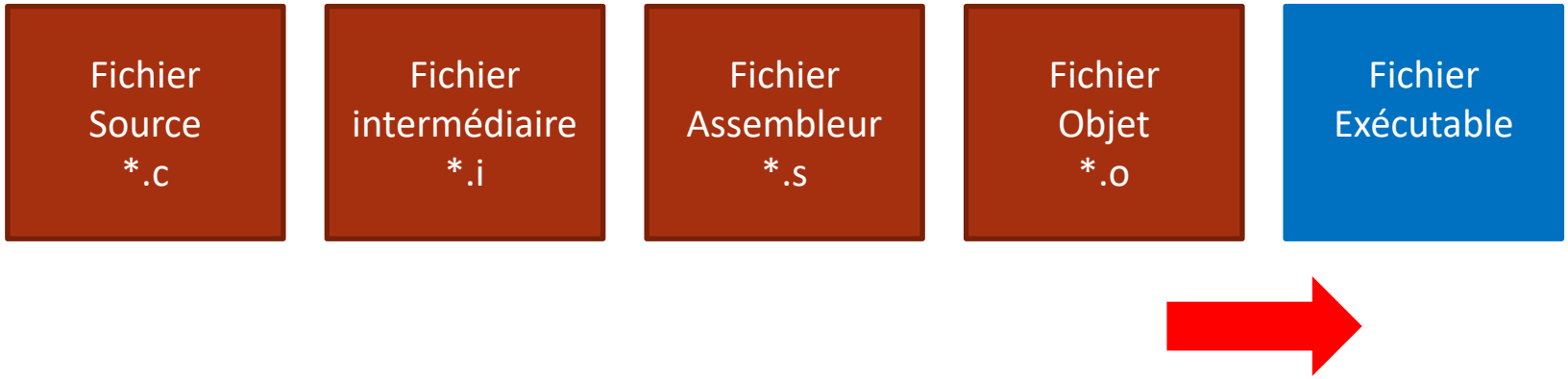


Assembling :

od -x Fichier.o

```
0000600 382f 3138 0a00 4743 433a 2028 474e 5529
0000620 2032 2e39 352e 3220 3139 3939 3130 3234
0000640 2028 7265 6c65 6173 6529 0000 0000 0000
0000660 0000 0000 0000 0000 0000 0000 0000 0000
```

2. Compilateur gcc



Linking:

- Le fichier objet est incomplet, il ne contient pas le code de la fonction printf par exemple
- L'édition des liens va réunir le fichier objet et les fonctions contenues dans les bibliothèques, pour produire le programme complet

2. Compilateur gcc

Les options de gcc sont :

Option	Argument	But
`-E`		Arrêter la compilation après le passage du préprocesseur, avant le compilateur.
`-S`		Arrêter la compilation après le passage du compilateur, avant l'assembleur.
`-c`		Arrêter la compilation après l'assemblage, laissant les fichiers objet disponibles.
`-W`	Avertissement	Valider les avertissements (*warnings*) décrits en arguments. Il en existe une multitude, mais l'option la plus couramment utilisée est `–Wall`, pour activer tous les avertissements.
`-pedantic`		Le compilateur fournit des avertissements encore plus rigoureux qu'avec `–Wall`, principalement orientés sur la portabilité du code.
`-g`		Inclure dans le code exécutable les informations nécessaires pour utiliser le débogueur. Cette option est généralement conservée jusqu'au basculement du produit en code de distribution.
`-O`	0 à 3	Optimiser le code engendré. Le niveau d'optimisation est indiqué en argument (0 = aucune). Il est déconseillé d'utiliser simultanément l'optimisation et le débogage.

2. Débogage

Lorsqu'une application a été compilée avec l'option –g, il est possible de l'exécuter sous le contrôle d'un débogueur :

gdb (Gnu Debugger) est un utilitaire qui fonctionne en ligne de commande, avec une interface assez rébarbative. Il a certaines version frontales

ddd (Data Display Debugger) est un autre frontal est également disponible sous Linux, nommé plus agréable visuellement.

3 CONSTRUCTION D'UNE APPLICATION

1. Makefile

Alternative 1:

- gcc Fichier1.c Fichiers2.c Fichiers3.c -o Fichier
- Si un fichier est modifié cette commande recompile tous les fichiers

CONSTRUCTION D'UNE APPLICATION

1. Makefile

Alternative 2:

- Makefile
- Si un fichier est modifié, cette commande recompile seulement ce fichier

CONSTRUCTION D'UNE APPLICATION

1. Makefile

Alternative 2:

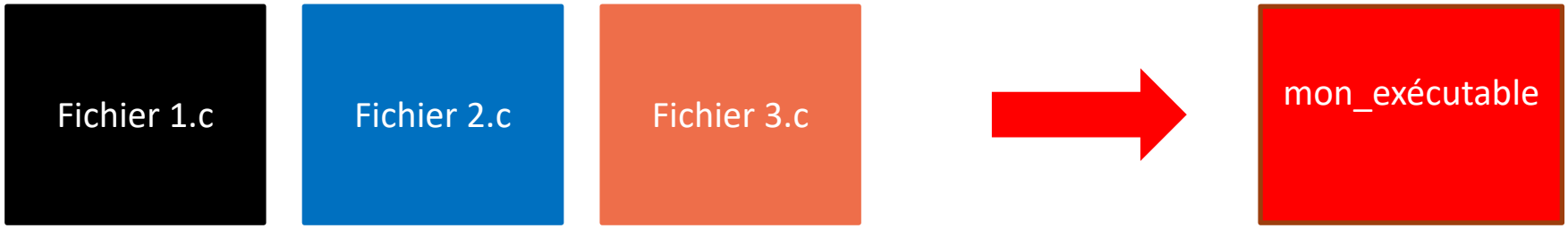
- Makefile
- *nom_cible : liste_dependances*

<TAB>commande

<TAB>commande

CONSTRUCTION D'UNE APPLICATION

1. Makefile




```
mon_executable: Fichier1.o Fichier2.o Fichier3.o
	gcc -o mon_executable Fichier1.o Fichier2.o Fichier3.o
Fichier1.o: Fichier1.c
	gcc -c Fichier1.c
Fichier2.o: Fichier2.c
	gcc -c Fichier2.c
Fichier3.o: Fichier3.c
	gcc -c Fichier3.c
```

Lien pour un exemple de makefile : https://www.youtube.com/watch?v=-riHEHGP2DU