

Derrière GCC

- GCC signifie “GNU Compiler **Collection**”, ce n’est donc pas un programme, mais une collection d’outils, un pour chaque phase de la construction d’un programme:
- preprocessing: `cpp file.c > file.i`
- compilation: `cc1 file.i -o file.s`
- Assemblage: `as file.s -o file.o`
- Link: `collect2 file.o -o program`
- `collect2` est un wrapper GNU de `link`, qui repose sur `LD`, qui n’est pas un outil de la collection GNU. `LD` est distribué via les “binutils” et est commun à la majorité des compilateurs (intel, PGI, LLVM...)

Construction d'un programme

- Unité de compilation : un fichier source (parfois nommé TU pour « Translation Unit »)
- Préprocesseur : préparation d'une TU pour la compilation.
- Compilateur : exécution d'une TU, transformation en un set d'instructions spécifiques à l'architecture.
- Assemblage: Transformation des instructions assembleur en code machine (structure ELF)
- L'édition de liens: Assemblage des différentes TU pour créer un exécutable

Etapes de La Compilation

Preprocessing

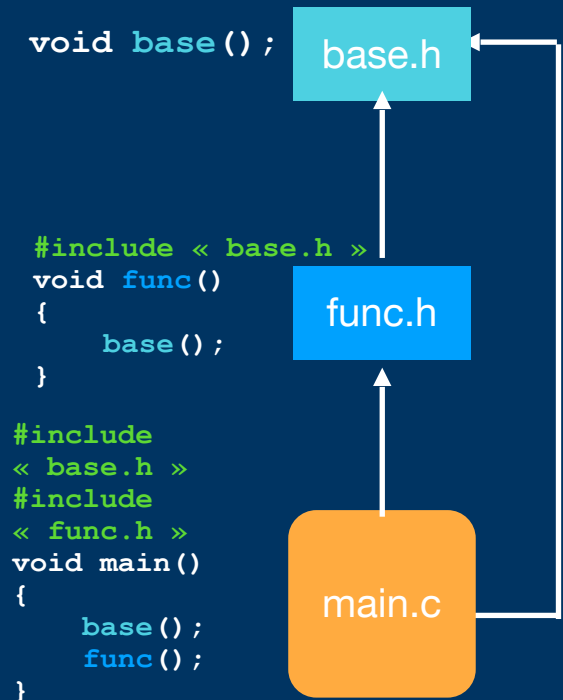
- **Outil** : `cpp`
- **Objectif** : Traiter les directives comme `#include`, `#define` et développer les macros.
- **Commande** :

```
# Invocation Manuelle  
cpp source.c -o source.i  
# Dans GCC  
gcc -E source.c -o source.i
```

Résultat : Un fichier source prétraité (`source.i`), où toutes les inclusions et macros sont résolues.

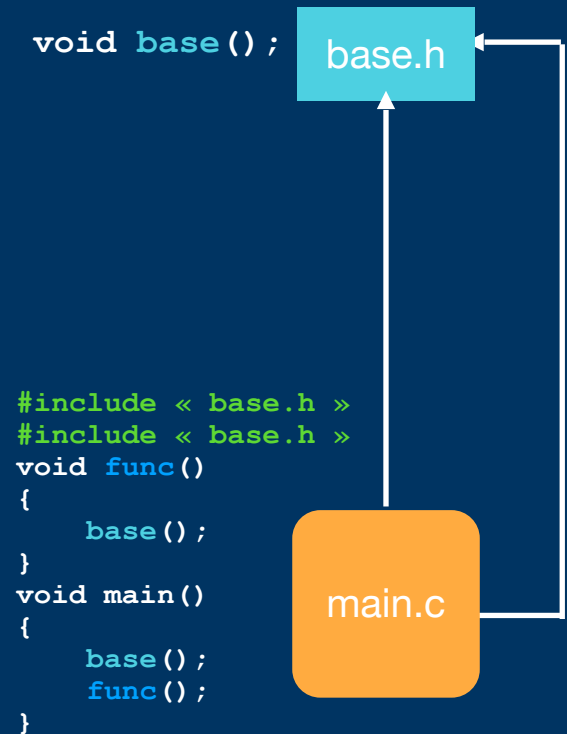
Preprocessing

- Interprétation de directives (#)
 - **#define** / **#undef** : Définition, déclaration de macro (fonctions, constantes...)
 - **#if(n)def** / **#else** / **#endif** : Compilation conditionnelle de sections de code
 - **#include** : Inclusion de fichiers récursives, nécessité des guards
 - **#error** / **#warning** / **#todo** : Influence la sortie de compilation
 - **#pragma...** : gestion compilateur
- Certaines constantes existent (**__FILE__**, **__LINE__**, **__DATE__**, **__TIME__**), et extensions selon l'architecture (**__WIN32**, **__APPLE**, **__linux__**)
- Programme : **cpp main.c main.i**, Résultat obtenu avec : **gcc -E main.c**



Preprocessing

- Interprétation de directives (#)
 - **#define** / **#undef** : Définition, déclaration de macro (fonctions, constantes...)
 - **#if(n)def** / **#else** / **#endif** : Compilation conditionnelle de sections de code
 - **#include** : Inclusion de fichiers récursives, nécessité des guards
 - **#error** / **#warning** / **#todo** : Influence la sortie de compilation
 - **#pragma...** :
- Certaines constantes existent (**__FILE__**, **__LINE__**, **__DATE__**, **__TIME__**), et extensions selon l'architecture (**__WIN32**, **__APPLE**, **__linux__**)
- Programme : **cpp main.c main.i**, Résultat obtenu avec : **gcc -E main.c**



Preprocessing

- Interprétation de directives (#)
 - **#define** / **#undef** : Définition, déclaration de macro (fonctions, constantes...)
 - **#if(n)def** / **#else** / **#endif** : Compilation conditionnelle de sections de code
 - **#include** : Inclusion de fichiers récursives, nécessité des guards
 - **#error** / **#warning** / **#todo** : Influence la sortie de compilation
 - **#pragma...** :
- Certaines constantes existent (**__FILE__**, **__LINE__**, **__DATE__**, **__TIME__**), et extensions selon l'architecture (**__WIN32**, **__APPLE**, **__linux__**)
- Programme : **cpp main.c main.i**, Résultat obtenu avec : **gcc -E main.c**

```
void base();  
void base();  
void func()  
{  
    base();  
}  
void main()  
{  
    base();  
    func();  
}
```

main.c

Preprocessing

- Invocation : `cpp main.c main.i`
- Résultat: `gcc -E main.c [-P]`
- Ajout de chemins :
 - `-I/usr/include`
 - `export C_INCLUDE_PATH`
- `-D / -undef`
- `-include`

```
# 1 "main.c"
# 1 "<built-in>" 1
# 1 "<built-in>" 3
# 361 "<built-in>" 3
# 1 "<command line>" 1
# 1 "<built-in>" 2
# 1 "main.c" 2
# 1 "./base.h" 1
void base();
# 2 "main.c" 2
# 1 "./func.h" 1
void func()
{
    base();
}
# 3 "main.c" 2
int main(int argc, char const
*argv[])
{
    func();
    return 0;
}
```

```
➤ gcc -E main.c -o main.i
main.c:1:10: erreur fatale: base.h : Aucun fichier ou dossier de ce type
#include <base.h>
      ^~~~~~
compilation terminée.
```


Compilation

- **Outil** : `cc1` (interne à GCC)
- **Objectif** : Traduire le code prétraité en langage d'assemblage.
- **Commande** :
 - > `find /usr/lib /usr/libexec /usr/local/lib -name cc1`
 - # Appel Manuel
 - `cc1 source.i -o source.s -quiet -fverbose-asm`
 - # Appel via GCC
 - `gcc -S source.i -o source.s`

Résultat : Un fichier d'assemblage (`source.s`).

Compilation

- Objectif : Transformer un contenu d'un langage **source** vers un langage **destination** (=target)
- Un programme source doit suivre un ensemble de règles afin d'être compris par le compilateur : la **grammaire**
- Le processus de compilation se découpe en trois grosses phases principales (simplifié)



Front-end


- Réalise l'analyse grammaticale du langage source pour produire une représentation intermédiaire (IR)
1. Analyse Lexicale : lecture de la source un caractère après l'autre pour former des mots (=lexème). Toute information superflue est ignorée (espaces...)
 2. Analyse Syntaxique : Chacun de ces lexèmes est soumis à validation pour s'assurer qu'il font parti du langage
 3. Analyse Sémantique : un ensemble de lexème forme une phrase, qui doit être sémantiquement juste
- Tout un pan de l'informatique moderne s'intéresse au formalisme du langage (pour créer son propre langage : `lex` & `yacc`)

Token	Example lexeme
const	const
if	if
relop	<, <=
id	pi, count, age
num	3.14, 0
literal	"hello world"

Middle-end

- Une fois le code généré, on obtient un programme sémantiquement juste mais loin d'être optimisé. De nombreuses passes sont en jeu ici
 - Graphe de control-flow, inlining
 - Élimination de code « mort » (DCE)
 - Transformation de boucles
 - Propagation de constantes
- Ce composant est indépendant de tout langage et de toute architecture. Réutilisation infinie, tant que la grammaire fourni la même sémantique (représentation intermédiaire)

```
int foo(void)
{
    int a = 24;
    int b = 25;
    int c;
    c = a * 4;
    return c;
    b = 24;
    return 0;
}
```



Back-end

- Génération du programme pour la machine cible. Chaque architecture ayant un jeu d'instructions différent
- le code généré possède ses propres optimisations (=machine-dependent Optimisations), Vectorisation (SSE, AVX...)
- Registres, Pipelining, mode d'adressage (Absolute, PC-centric, register-*...), code redondant
- Génération des fichiers contenant le code assembleur (.s)
- Résultat de **gcc -S main.c**

```
.section      __TEXT,__text,regular,pure_instructions
.build_version macos, 10, 14
.globl _func
.p2align     4, 0x90

_func:
.cfi_startproc
## %bb.0:
pushq   %rbp
.cfi_def_cfa_offset 16
.cfi_offset %rbp, -16
movq     %rsp, %rbp
.cfi_def_cfa_register %rbp
movb     $0, %al
callq    _base
popq     %rbp
retq
.cfi_endproc

.globl _main
.p2align     4, 0x90

_main:
.cfi_startproc
## %bb.0:
pushq   %rbp
.cfi_def_cfa_offset 16
.cfi_offset %rbp, -16
movq     %rsp, %rbp
.cfi_def_cfa_register %rbp
subq     $16, %rsp
movl     $0, -4(%rbp)
movl     %edi, -8(%rbp)
movq     %rsi, -16(%rbp)
callq    _func
xorl     %eax, %eax
addq     $16, %rsp
popq     %rbp
retq
.cfi_endproc
```

Assemblage

- **Outil** : `as`
- **Objectif** : Convertir le code d'assemblage en code machine (fichier objet).
- **Commande** :

`#Avec as`

`as source.s -o source.o`

`# Avec GCC`

`gcc -c source.s -o source.o`

Résultat : Un fichier objet (`source.o`), contenant du code machine binaire.

Assemblage

Source : <https://en.wikipedia.org/wiki/Endianness>

- Transformation du code dépendant machine en code binaire
- Prise en compte de « l'Endianness » (Little / big)
- Création d'un fichier objet (.o) suivant le format ELF

■ **.text** : code défini dans le fichier

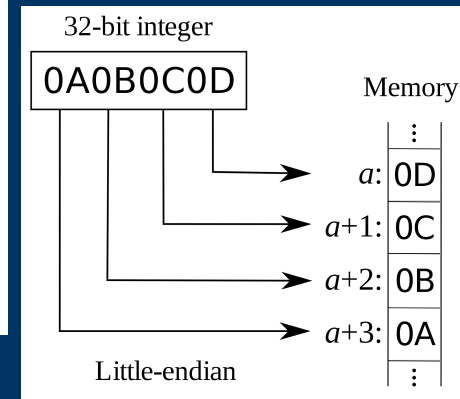
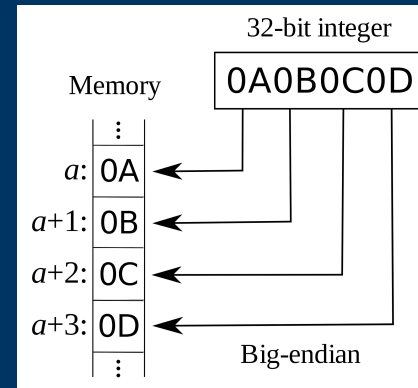
■ **.data / .bss** : variables globales du fichier initialisées / ou non

■ **.rodata** : Constantes

■ **.shstrtab** : tableau des chaînes de caractères

○ Commande : **as main.s -o main.o**

○ Résultat : **gcc -c main.c -o main.o**



```
readelf -SW main.o
Il y a 9 en-têtes de section, débutant à l'adresse de décalage 0x180:

En-têtes de section :
[Nr] Nom                Type                Adr                Décala.Taille ES Fan LN Inf Al
[ 0]                     NULL                0000000000000000 000000 000000 00  0  0  0
[ 1] .text                PROGBITS            0000000000000000 000040 000042 00 AX  0  0  1
[ 2] .data                PROGBITS            0000000000000000 000082 000000 00 WA  0  0  1
[ 3] .bss                 NOBITS              0000000000000000 000082 000000 00 WA  0  0  1
[ 4] .rodata              PROGBITS            0000000000000000 000082 00000d 00 A  0  0  1
[ 5] .comment              PROGBITS            0000000000000000 00008f 00002d 01 MS  0  0  1
[ 6] .note.GNU-stack        PROGBITS            0000000000000000 0000bc 000000 00  0  0  1
[ 7] .eh_frame             PROGBITS            0000000000000000 0000c0 000078 00 A  0  0  8
[ 8] .shstrtab             STRTAB              0000000000000000 000138 000047 00  0  0  1

Clé des fanions :
W (écriture), A (allocation), X (exécution), M (fusion), S (chaînes), I (info),
L (ordre des liens), O (traitement supplémentaire par l'OS requis), G (groupe),
T (TLS), C (comprimé), x (inconnu), o (spécifique à l'OS), E (exclu),
l (grand), p (processor specific)
```

Édition de Liens (Link)

- **Outil** : ld
- **Objectif** : Combiner les fichiers objets et bibliothèques pour produire un exécutable.
- **Commande** :

```
# Using LD
ld -m elf_x86_64 -dynamic-linker /lib64/ld-linux-x86-64.so.2 /usr/
lib/x86_64-linux-gnu/crt1.o /usr/lib/x86_64-linux-gnu/crti.o main.o
-lc /usr/lib/x86_64-linux-gnu/crtn.o -o ./a.out
# Using GCC
gcc source.o -o executable
```

Résultat : Un fichier exécutable (executable).

Édition de liens

- Addition de plusieurs fichiers objets pour créer un exécutable
- Fonction du « linker » : **ld** / **ld.gold** (version GNU)
- Fusion des sections identiques
 - « Relocations » de symboles = réarrangement de l'espace d'adressage
- Résultat : **gcc main.c**

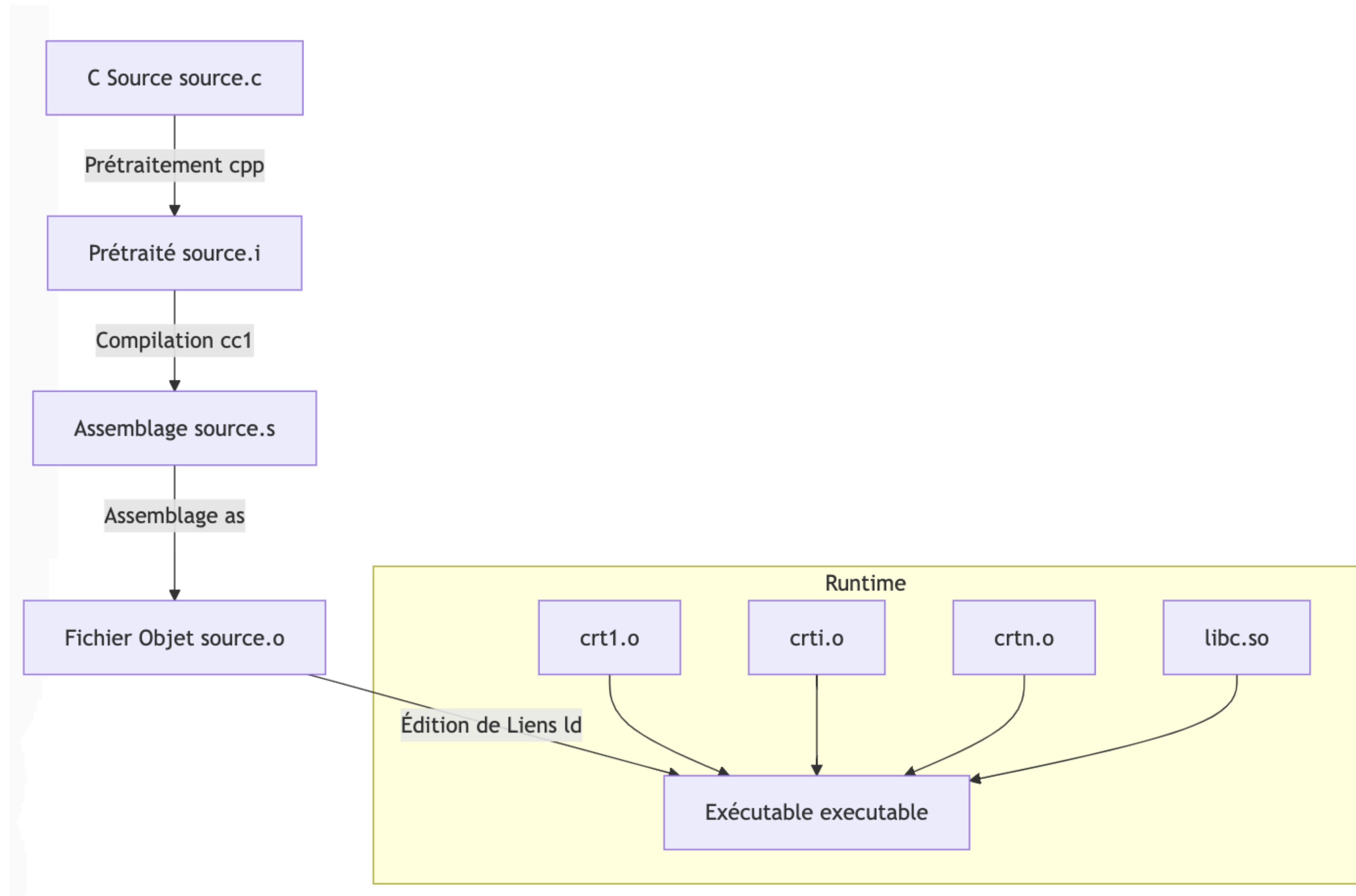
```
↳ readelf -h ./a.out
En-tête ELF:
  Magique:      7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Classe:                               ELF64
  Données:                               complément à 2, système à octets de poids faible d'abord (little endian)
  Version:                               1 (current)
  OS/ABI:                               UNIX - System V
  Version ABI:                               0
  Type:                               EXEC (fichier exécutable)
  Machine:                               Advanced Micro Devices X86-64
  Version:                               0x1
  Adresse du point d'entrée:              0x4003b0
  Début des en-têtes de programme :      64 (octets dans le fichier)
  Début des en-têtes de section :        6360 (octets dans le fichier)
  Fanions:                               0x0
  Taille de cet en-tête:                  64 (octets)
  Taille de l'en-tête du programme:       56 (octets)
  Nombre d'en-tête du programme:          9
  Taille des en-têtes de section:         64 (octets)
  Nombre d'en-têtes de section:           27
  Table d'index des chaînes d'en-tête de section: 26
```

```
Betelgeuse ~
↳ objdump -d ./a.out | grep "<_start>"
00000000004003b0 <_start>:
Betelgeuse ~
↳ objdump -d ./a.out | grep "<main>"
00000000004004ae <main>:
```

Édition de liens

Artefacts compilo-spécifiques:

- `crt1.o` / `crt0.o...` : Symboles de chargement du programme qui contient surtout le point d'entrée (fonction `_start()`) pour le déchargement du Shell.
- `crti.o` / `crti.o`: Symboles `_init` et `_fini`, constructeur et destructeur (old-style). Ils sont conservés par rétrocompatibilité avec les anciens systèmes. Remplacé par les segments des sections `.init_array` et `.fini_array`.
- `crtbegin.o` / `crtend.o`: **C++** constructeurs
- `crtbeginS.o` / `crtendS.o` : remplace son équivalent quand `-fPIC`
- `crtbeginT.o` / `crtendT.o`: remplace son équivalent quand `-static`



Bibliothèque / Module

- Une application contient rarement tout le code dont elle a besoin et repose sur l'inclusion de modules déjà implémentés : réutilisation de code
- Une déclaration de la partie publique du module. C'est le header inclus, exposant variable & fonctions (ex: /usr/include)
 - Inclusion avec `-I`, `C_INCLUDE_PATH`, `-include...`
 - Invocation avec : `#include <mymodule.h>`
- Les code du module précompilé, qui est chargé lors de l'édition de liens pour l'optimisation du binaire final (exemple : /usr/lib[64])
 - Inclusion avec `-L`, `[LD_]LIBRARY_PATH`
 - Invocation avec: `-l<nom du module>` (ex: `-lgcc` pour **libgcc.a**)
 - Pas nécessaire pour les fonctions comme `printf/scanf`, pourquoi ?

```
└─ gcc main.c -I./include -L./lib -lmylib
```

```
└─ tree -L 1 /usr/include
/usr/include
├── aio.h
├── aliases.h
├── alloca.h
├── a.out.h
├── argp.h
├── argz.h
├── ar.h
├── arpa
├── asm
├── asm-generic
├── assert.h
├── bits
├── byteswap.h
├── bzlib.h
├── c++
├── complex.h
├── cpio.h
├── crypt.h
├── ctype.h
├── cursesapp.h
├── cursesf.h
├── └─ ls /usr/lib64/*.so -l
    /usr/lib64/BugpointPasses.so
    /usr/lib64/eppic_makedumpfile.so
    /usr/lib64/ld-2.27.so
    /usr/lib64/libanl-2.27.so
    /usr/lib64/libanl.so
    /usr/lib64/libasm-0.174.so
    /usr/lib64/libbfd-2.29.1-23.fc28.so
    /usr/lib64/libBrokenLocale-2.27.so
    /usr/lib64/libBrokenLocale.so
    /usr/lib64/libbtparse.so
    /usr/lib64/libbz2.so
    /usr/lib64/libc-2.27.so
    /usr/lib64/libccl.so
    /usr/lib64/libclangAnalysis.so
    /usr/lib64/libclangApplyReplacements.
    /usr/lib64/libclangARCMigrate.so
    /usr/lib64/libclangASTMatchers.so
```

Bibliothèque / Module

- **STATIQUE (extension .a)**

- Le module est lié & injecté à l'application (archive de fichiers .o)
- Avantage : Indépendant de l'exécution
- Inconvénients : Binaire + lourd, fonction externes référencées « en dur » (pas de **dlopen()**)
- Outil : **ar** (-x : extract, -s : create, -t : list) Souvent : **ar rcs libmodule.a module.o**

- **DYNAMIQUE (extension .so)**

- Le module est référencé à la compilation et injecté à l'**exécution**
- Avantage : Binaire plus léger, rien n'est en dur dans le binaire, plus de souplesse à l'exécution
- Inconvénient : crée un overhead au runtime, dépendance entre environnement de compilation & d'exécution
- Outil : **ld**
- Via compilateur : **gcc -shared module.o -o libmodule.so**
- **-fPIC** indispensable dans 90% des cas de bibliothèques dynamiques (*Position Independent Code*)

Bibliothèque / Module

```
Betelgeuse ~  
➤ gcc -static main.c -I.  
/usr/bin/ld : ne peut trouver -lc  
collect2: error: ld a retourné le statut de sortie 1
```

- Par défaut, il n'y a pas de distinctions entre statique et dynamique à l'édition de liens. Possibilité de forcer un link statique : `gcc -static` (génère une erreur si la version statique n'existe pas)
- Recherche de `.so` à la compilation : `-L` / `-Wl,-rpath`
- Chargement au runtime: `LD_LIBRARY_PATH` / `LD_PRELOAD`

```
Betelgeuse ~  
➤ ldd ./a.out  
linux-vdso.so.1 (0x00007ffffdded6000)  
libc.so.6 => /lib64/libc.so.6 (0x00007ff6e0cf2000)  
/lib64/ld-linux-x86-64.so.2 (0x00007ff6e10b1000)
```

```
➤ gcc main.c -I. -Wl,-rpath=/lib64/; \  
> readelf -dW ./a.out  
Section dynamique à l'offset 0xe50 contient 21 entrées :  
Étiquettes Type Nom/Valeur  
0x0000000000000001 (NEEDED) Bibliothèque partagée: [libc.so.6]  
0x000000000000000f (RPATH) Bibliothèque rpath: [/lib64/]  
0x000000000000000c (INIT) 0x400398
```

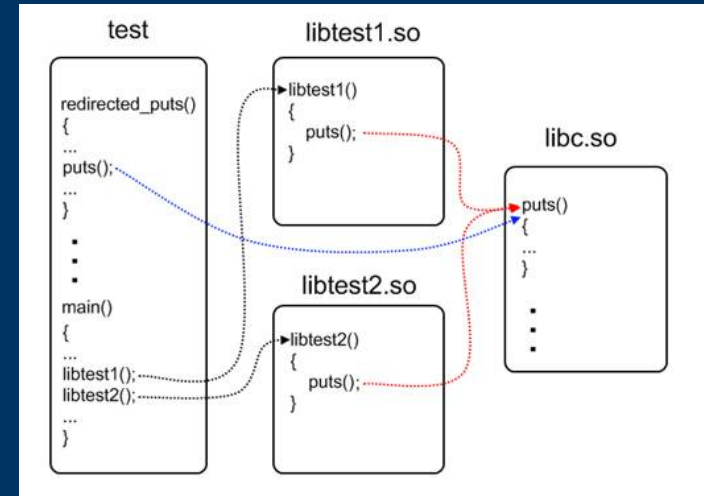
```
➤ readelf -dW ./a.out  
Section dynamique à l'offset 0xe60 contient 20 entrées :  
Étiquettes Type Nom/Valeur  
0x0000000000000001 (NEEDED) Bibliothèque partagée: [libc.so.6]  
0x000000000000000c (INIT) 0x400390  
0x000000000000000d (FINI) 0x400534  
0x0000000000000019 (INIT_ARRAY) 0x600e50  
0x000000000000001b (INIT_ARRAYSZ) 8 (octets)  
0x000000000000001a (FINI_ARRAY) 0x600e58
```

Bibliothèque / Module

- Chargement dynamique via **libdl.so**
 - **h = dlopen(« mylib.so »)** : Charge une bibliothèque (appel du loader, chargement mémoire, etc...)
 - **dlsym(h, « i »)** : Renvoie l'adresse d'un symbole chargé en mémoire (variable, fonction, etc...)
 - **dlclose(h)** : Ferme la bibliothèque, déchargement...
- Requiert une bibliothèque dynamique !

Bibliothèque / Module

- L'introspection est l'art de charger une bibliothèque à l'exécution, pour venir « écraser » les symboles existants par ceux re-définis.
- Exemple : `LD_PRELOAD=myalloclib.so ./a.out`
- Conserver la cohérence de l'application : rappeler la fonction originale via `dlsym(« func », RTLD_NEXT);`
- Le prochain symbole est déterminé par l'ordre des bibliothèques tel qu'indiqué à la compilation



```
$ ldd ./IMB-MPI1
linux-vdso.so.1 (0x00007fff493b1000)
libmpc_framework.so => $INSTALL_PATH//x86_64/x86_64//lib/libmpc_framework.so (0x00007f74b2717000)
libextls.so.0 => $INSTALL_PATH//x86_64/x86_64//lib/libextls.so.0 (0x00007f74b250d000)
libportals.so.4 => /opt/sources/portals4/INSTALL/lib/libportals.so.4 (0x00007f74b22e7000)
libm.so.6 => /lib64/libm.so.6 (0x00007f74b1f53000)
libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f74b1d34000)
librt.so.1 => /lib64/librt.so.1 (0x00007f74b1b2c000)
libsctk_arch.so => $INSTALL_PATH//x86_64/x86_64//lib/libscstk_arch.so (0x00007f74b1929000)
libhwloc.so.5 => $INSTALL_PATH//x86_64/x86_64//lib/libhwloc.so.5 (0x00007f74b16f0000)
libxml2.so.2 => $INSTALL_PATH//x86_64/x86_64//lib/libxml2.so.2 (0x00007f74b138e000)
libmpcgetopt.so.0 => $INSTALL_PATH//x86_64/x86_64//lib/libmpcgetopt.so.0 (0x00007f74b118a000)
libc.so.6 => /lib64/libc.so.6 (0x00007f74b0dcb000)
/lib64/ld-linux-x86-64.so.2 (0x00007f74b2f3a000)
libdl.so.2 => /lib64/libdl.so.2 (0x00007f74b0bc7000)
libev.so.4 => /lib64/libev.so.4 (0x00007f74b09b8000)
liblzma.so.5 => /lib64/liblzma.so.5 (0x00007f74b0791000)
```


En résumé



- Preprocessing : `gcc -E main.c (cpp)`
 - Ajout de règles : `INCLUDE_PATH= / -I / -include / -D / -undef`
- Compilation : `gcc -S main.c`
- Code objet : `gcc -c main.c (as)`
- Edition de liens : `gcc main.c (ld)`
- Ajout de bibliothèques : `-L<chemin> / -l<chemin> / <chemin absolu>`
 - Statique (.a) : `ar rcs / -static / LIBRARY_PATH`
 - Dynamique (.so) : `-fPIC -shared / -Wl,-rpath / LD_LIBRARY_PATH`

Généralités sur les IPC System V

Les IPC System V

Apparus dans Unix en 1983 ils permettent des communication inter-inter-processus (Inter-Process Communications, IPC)

- Files de messages
- Segment de mémoire partagée
- Sémaphores

Le noyau est chargé de la gestion des ressources associées via des commandes

Files de Messages



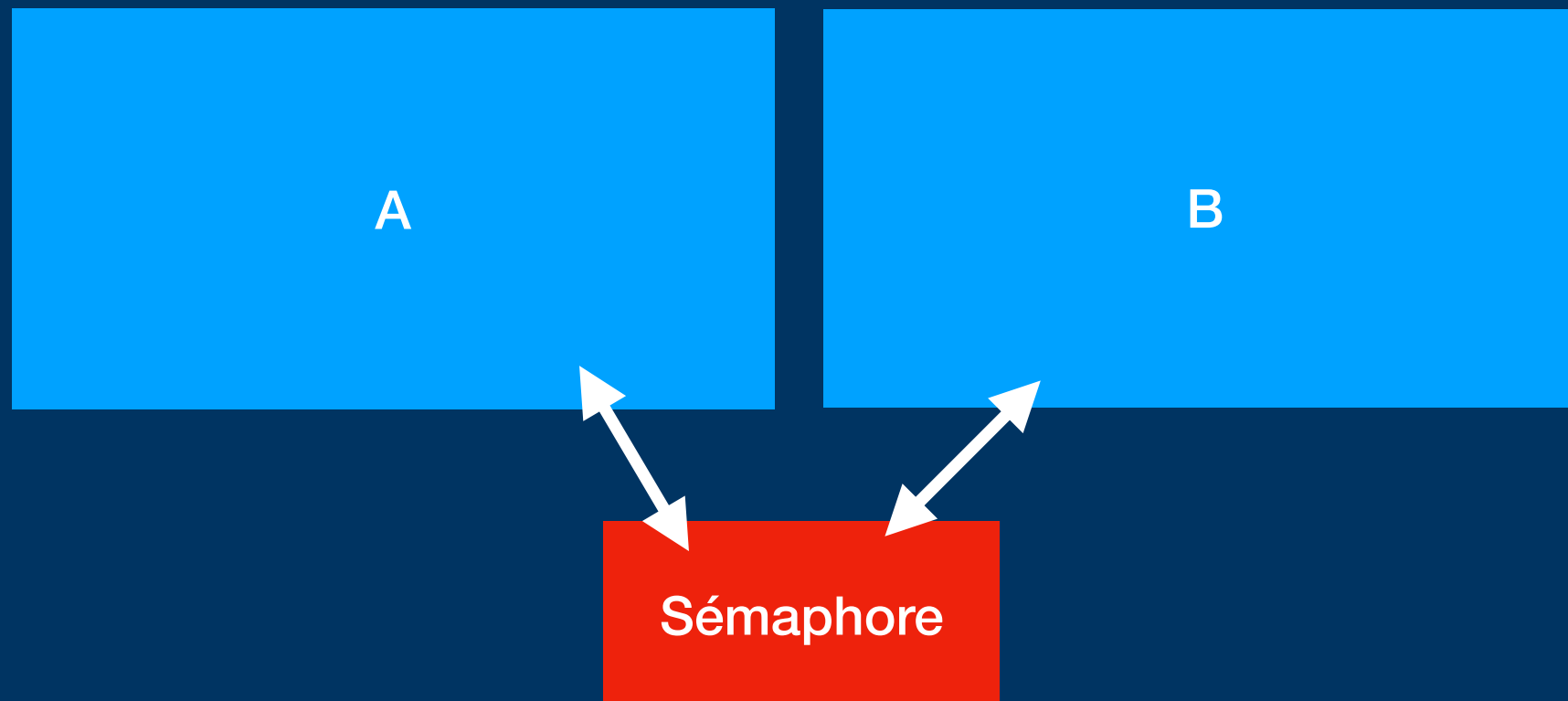
- *ftok*: génération d'une clef IPC
- *msgget*: Récupère un identificateur de file de message
- *msgrecv*: Réception d'un message depuis une file
- *msgsend*: Envoi d'un message dans une file
- *msgctl*: Contrôle de la file de messages

Segment de mémoire partagée



- *ftok*: génération d'une clef IPC
- *shmget*: Récupère un identificateur de segment shm
- *shmat*: Projection d'un segment SHM
- *shmdt*: Supression d'un segment SHM
- *shmctl*: Contrôle du segment SHM

Sémaphore IPC



- *ftok*: génération d'une clef IPC
- *semget*: Récupère un identificateur de sémaphore
- *semop*: Fait une opération sur le sémaphore
- *semctl*: Contrôle du sémaphore

Les IPC System V

\$ ipcs

----- Files de messages -----

clef	msqid	propriétaire	perms	octets utilisés	messages
------	-------	--------------	-------	-----------------	----------

----- Segment de mémoire partagée -----

clef	shmid	propriétaire	perms	octets	nattch	états
0x00000000	42729472	jbbsnard	600	1048576	2	dest
0x00000000	39616513	jbbsnard	600	524288	2	dest

----- Tableaux de sémaphores -----

clef	semid	propriétaire	perms	nsems
------	-------	--------------	-------	-------

Les IPC System V

```
$ ipcrm -h
```

Utilisation :

```
ipcrm [options]  
ipcrm shm|msg|sem <id> ...
```

Supprimer certaines ressources IPC.

Options :

-m, --shm-id <ident.>	retirer le segment de mémoire partagée par ident.
-M, --shm-key <clef>	retirer le segment de mémoire partagée par clef
-q, --queue-id <ident.>	retirer la file de messages par identifiant
-Q, --queue-key <clef>	retirer la file de messages par clef
-s, --semaphore-id <id.>	retirer le sémaphore par identifiant
-S, --semaphore-key <clef>	retirer le sémaphore par clef
-a, --all[=shm msg sem]	tout retirer (dans la catégorie indiquée)
-v, --verbose	expliquer les actions en cours
-h, --help	afficher cette aide et quitter
-V, --version	afficher les informations de version et quitter

Consultez ipcrm(1) pour obtenir des précisions complémentaires.

Les IPC System V

```
$ ipcmk -h
```

Utilisation :
ipcmk [options]

Créer diverses ressources IPC.

Options :

-M, --shmem <taille>	créer un segment de mémoire partagée de taille <taille>
-S, --semaphore <nsems>	créer un tableau de sémaphores à <nsems> éléments
-Q, --queue	créer une file de messages
-p, --mode <mode>	droits de la ressource (0644 par défaut)
-h, --help	afficher cette aide et quitter
-V, --version	afficher les informations de version et quitter

Consultez ipcmk(1) pour obtenir des précisions complémentaires.

Resources

Les ressources IPC sont indépendante des processus

- Il est possible de laisser des scories si l'on ne fait pas attention
- Un processus peut se « rater » à un segment lors de son redémarrage par exemple
- Les processus partagent des segments avec un mécanisme de clef qui est un secret « a priori » pour la sécurité

Clefs pour les IPCs System V

La Clef

Un IPC (de tout type) est partagé par une clef:

- C'est un entier qui doit être le même entre tous les processus partageant la resource;
- On peut la connaître a priori avec risque de conflit (un peut comme un port TCP);
- Une clef spéciale IPC_PRIVATE crée une file limité à un processus et l'ensemble de ses descendants;
- On peut la créer avec une fonction « ftok » qui repose sur un fichier et un nom de projet.

Ftok

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
```

```
key_t ftok(const char *pathname, int proj_id);
```

DESCRIPTION

The `ftok()` function uses the identity of the file named by the given `pathname` (which must refer to an existing, accessible file) and the least significant 8 bits of `proj_id` (which must be nonzero) to generate a `key_t` type System V IPC key, suitable for use with `msgget(2)`, `semget(2)`, or `shmget(2)`.

The resulting value is the same for all `pathnames` that name the same file, when the same value of `proj_id` is used. The value returned should be different when the (simultaneously existing) files or the project IDs differ.

RETURN VALUE

On success, the generated `key_t` value is returned. On failure `-1` is returned, with `errno` indicating the error as for the `stat(2)` system call.



Création / Récupération de ressources

Une fois que l'on a une clef de type *key_t* on peut retrouver/créer une resource:

- File de message : *msgget*
- Segment de mémoire partagée: *shmget*
- Sémaphore: *semget*

Les Files de Messages

IPC SYSTEM V

Files de Messages pour une Communication entre Processus sur un Même Noeud.

Le message sera toujours de la forme:

```
Struct XXX {  
    long id; // Toujours > 0 !  
    ... DATA ...  
    // Taille max sans le long MSGMAX (8192 Octets)  
};
```

Lors de l'envoi et de la réception d'un message la taille et TOUJOURS sans le long qui définit le type de message. Cette même valeur (ici id) doit TOUJOURS être supérieure à 0.

En pratique on crée une struct statique sur la pile car l'allocation d'un objet avec piggybacking demande plus de code.

Créer Une File de Messages

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
```

```
int msgget(key_t key, int msgflg);
```

- Key : Une clef, soit manuelle, soit via ftok ou bien IPC_PRIVATE
- msgflg: mode de création de la file et ses droits UNIX
 - ➡ IPC_CREAT crée une file s'il y en a aucune associée à cette clef
 - ➡ IPC_EXCL échoue s'il existe déjà une file sur la clef indiqué (toujours combiné avec IPC_CREAT!)
 - ➡ 0600 droit UNIX en octal (important car si omis 0000 et la file et moins pratique !)

Créer Une File de Messages

- Créer une file pour un processus et ses fils
 - ➡ `file = msgget(IPC_PRIVATE, 0600);`
- Créer une file pour accéder à une file potentiellement existante:
 - ➡ `file = msgget(key , IPC_CREAT | 0600);`
- Pour être sûr de créer une nouvelle file en lecture écriture pour soi et en lecture seule pour les autres utilisateurs:
 - ➡ `file = msgget(key, IPC_CREAT | IPC_EXCL | 0622);`
- Utiliser uniquement une file existante précédemment créée par un serveur:
 - ➡ `file = msgget(key, 0);`

Envoyer un Message

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
```

```
int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

- msqid : file de message à utiliser, créée avec msgget
- msgp : pointeur vers les données à envoyer (comprend forcément un long qui est l'ID de message)
- size : taille du message **SANS** le long qui est l'ID du message
- msgflg: mode d'envoi du message
 - ➡ IPC_NOWAIT ne pas bloquer si la file est pleine (renvoie EAGAIN dans errno)
 - ➡ 0 en général

Recevoir un Message

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
```

```
ssize_t msgrcv(int msqid,
               void *msgp, size_t msgsz, long msgtyp, int msgflg);
```

- msqid : file de message à utiliser, créée avec msgget
- msgp : pointeur vers les données à envoyer (comprend forcément un long qui est l'ID de message)
- size : taille du message **SANS** le long qui est l'ID du message
- msgtyp : type de message à recevoir:
 - ➡ 0 : prochain message de la file
 - ➡ 0 < TYP prochain message avec l'ID donné
 - ➡ TYP < 0 prochain message avec un ID inférieur ou égal à TYP, utilisé pour gérer des priorités de messages
- msgflg: mode de réception du message:
 - ➡ IPC_NOWAIT ne pas bloquer si pas de message du TYP donné (renvoie ENOMSG dans errno)
 - ➡ MSG_EXCEPT renvoie un message d'un TYP différent de celui donné (seulement pour TYP > 0)
 - ➡ MSG_NO_ERROR permettre au message d'être tronqué à la réception (à la différence du comportement de base)

Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), 2, 0);
```

Quel message ??

Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), 2, 0);
```

Quel message ??

2

Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), -10, 0);
```

Quel message ??

Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), -10, 0);
```

Quel message ??

9

Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), 99, 0);
```

Quel message ??

Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), 99, 0);
```

Quel message ??

L'appel reste bloqué indéfiniment si un message 99 n'est jamais posté.

Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), 99, IPC_NOWAIT);
```

Quel message ??

Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), 99, IPC_NOWAIT);
```

Quel message ??

L'appel renvoie -1 et met errno à ENOMSG

Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), 11, MSG_EXCEPT);
```

Quel message ??

Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), 11, MSG_EXCEPT);
```

Quel message ??

9

Contrôler une File

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
```

```
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

- msqid : ID de la file à contrôler
- cmd: commande à appliquer à la file
 - ➡ IPC_STAT récupères les informations sur la file dans la *struct msqid_ds* (voir man)
 - ➡ IPC_SET permet de régler certains attributs en passant une *struct msqid_ds*
 - ➡ **IPC_RMID supprime la file toute les opérations courantes ou future échouent (avec la possibilité non gérée qu'une nouvelle file soit créée avec la même clef). La synchronisation et à la charge du programmeur.**
 - ➡ ... il existe d'autre flags voir man

PENSEZ à SUPPRIMER VOS FILES !!!

```

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <unistd.h>
#include <sys/wait.h>

#include <sys/time.h>

double get_time(){
    struct timeval val;
    gettimeofday(&val, NULL);
    return (double)val.tv_sec + 1e-6 * val.tv_usec;
}

#define SIZE 16

struct msg_t{
    long type;
    int data[SIZE];
};

#define NUM_MSG 65536

int main( int argc, char ** argv ){
    int file = msgget(IPC_PRIVATE, IPC_CREAT | 0600);

    if( file < 0 ){
        perror("msgget");
        return 1;
    }

    int i;
    struct msg_t m;
    m.type = 1;

    int pid = fork();

    if( pid == 0 )
    {
        int stop = 0;

        while(!stop)
        {
            msgrcv(file, &m, SIZE*sizeof(int), 0, 0);
            /* Notify end */
            if( m.data[0] == 0 )
                stop = 1;
            m.type = 1;
            msgsnd(file, &m, SIZE*sizeof(int), 0);
        }
    }
}

```

```

    else
    {
        double total_time = 0.0;

        for( i = 2 ; i <= NUM_MSG ; i++)
        {
            m.data[0] = i;
            m.type = i;

            double start = get_time();
            int ret = msgsnd(file, &m, SIZE*sizeof(int), 0);

            if( ret < 0 )
            {
                perror("msgsend");
                return 1;
            }

            double end = get_time();
            total_time += end - start;

            msgrcv(file, &m, SIZE*sizeof(int), 1, 0);
        }

        m.data[0] = 0;
        msgsnd(file, &m, SIZE*sizeof(int), 0);

        wait( NULL );

        msgctl( file, IPC_RMID, NULL);

        fprintf(stderr, "Pingpong takes %g usec Bandwidth is %g MB/s :
                    total_time/NUM_MSG*1e6,
                    (double)(SIZE*NUM_MSG*sizeof(int))/
                    (total_time*1024.0*1024.0));

    }

    return 0;
}

```


Les Segments SHM

IPC SYSTEM V

Partager une Zone Mémoire entre Deux Processus

SHM = SHared Memory

Les avantages:

- Communication directe sans recopie mémoire;
- Pas de passage par l'espace noyau à la différence des files messages (context switch et recopie);
- Latences plus faible (même mémoire)

Les inconvénients:

- Il faut manuellement synchroniser les communications (lock ou sémaphore)
 - ➡ *Comprenez qu'il est possible de mettre un lock dans cette zone mémoire, un spin lock directement, un mutex avec le bon attribut (PTHREAD_PROCESS_SHARED). Ou bien un sémaphore des IPC.*
- La structuration des données est à la charge du programme

Créer le Segment SHM

```
#include <sys/ipc.h>
#include <sys/shm.h>
```

```
int shmget(key_t key, size_t size, int shmflg);
```

- key : Une clef, soit manuelle, soit via ftok ou bien IPC_PRIVATE
- Size: taille du segment SHM en octet (arrondie à la page supérieure).
Donc mapper un int est un gros gâchis de mémoire (une page fait 4 KB).
- shmflg: mode de création de la file et ses droits UNIX
 - ➡ IPC_CREAT crée une file s'il y en a aucune associée à cette clef
 - ➡ IPC_EXCL échoue s'il existe déjà une file sur la clef indiqué (toujours combiné avec IPC_CREAT!)
 - ➡ 0600 droit UNIX en octal (important car si omis 0000 et la file et moins pratique !)

Projeter le Segment SHM

```
#include <sys/types.h>
#include <sys/shm.h>
```

```
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

- shmid : le descripteur du segment SHM
- shmaddr: une adresse où mapper le segment, alignée sur une frontière de page. **NULL si indifférent.**
- shmflg: options relative à la projection du segment
 - ➡ SHM_RND arrondis l'adresse passée par *shmaddr* à une frontière de page
 - ➡ SHM_RDONLY partager le segment en lecture seule
 - ➡ ... il existe d'autres flags voir man

Retirer le Segment SHM

```
#include <sys/types.h>
#include <sys/shm.h>

int shmdt(const void *shmaddr);
```

- shmaddr: adresse renvoyée par shmat

Tous les processus doivent retirer le segment de leur mémoire autrement la suppression avec shmctl n'est pas effective. Si un processus se termine il détache la mémoire mais cela ne marque pas le segment pour suppression.

Supprimer le Segment SHM

```
#include <sys/ipc.h>
#include <sys/shm.h>
```

```
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

- shmid : ID du segment à contrôler
- cmd: commande à appliquer à la file
 - ➡ IPC_STAT récupères les informations sur la file dans la *struct shmid_ds* (voir man)
 - ➡ IPC_SET permet de régler certains attributs en passant une *struct shmid_ds*
 - ➡ **IPC_RMID marque le segment SHM pour destruction cela ne se produira que quand tout les processus l'ayant projeté se seront détachés**
 - ➡ ... il existe d'autre flags voir man particulièrement IPC_INFO et SHM_INFO utiles pour connaitre les limites sur le système cible

PENSEZ à SUPPRIMER VOS Segments !!!

Totalement arbitraire

```
#include <sys/ipc.h>
#include <sys/shm.h>
```

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

```
int main(int argc, char **argv)
{
    int shm = shmget(19999, 2 * sizeof(int),
                    IPC_CREAT | IPC_EXCL | 0600 );
```

```
    if( shm < 0 )
    {
        perror("shmget");
        return 1;
    }
```

```
    int *val = (int*) shmat(shm, NULL, 0);
```

```
    if( !val )
    {
        perror("shmat");
        return 1;
    }
```

```
    /* valeur de départ */
    val[0] = 1;
    val[1] = 0;
```

```
    while(val[0])
    {
        sleep(1);
        val[1]++;
    }
```

```
    /* Unmap segment */
    shmdt(val);
    /* Server marks the segment for deletion */
    shmctl(shm, IPC_RMID, NULL);
```

```
    return 0;
}
```

Serveur

```
#include <sys/ipc.h>
#include <sys/shm.h>
```

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

```
int main(int argc, char **argv)
{
    int shm = shmget(19999, 2 * sizeof(int), 0 );
```

```
    if( shm < 0 )
    {
        perror("shmget");
        return 1;
    }
```

```
    int *val = (int*) shmat(shm, NULL, 0);
```

```
    if( !val )
    {
        perror("shmat");
        return 1;
    }
```

```
    /* valeur de départ */
    int last_val = -1;
    while(1)
    {
        if( val[1] != last_val ){
            printf("Val is %d max is 60\n", val[1]);
            last_val = val[1];
```

```
            /* Stop condition */
            if( 60 <= val[1] )
            {
                val[0] = 0;
                break;
            }
        }
        else
        {
            usleep(100);
        }
    }
```

```
}
```

```
    /* Unmap segment */
    shmdt(val);
```

```
    return 0;
}
```

Client

```
$ ./serveur &
```

```
$ ipcs -m
```

```
----- Segment de mémoire partagée -----
clef      shmid      propriétaire perms      octets      nattch      états
0x00004e1f 42827778      jbbesnard  600          8            1
```

```
$ ./client
```

```
Val is 0 max is 60
Val is 1 max is 60
(...)
Val is 7 max is 60
Val is 8 max is 60
Val is 60 max is 60
```

```
[2]+  Fini
```

```
./server
```

```
$ ipcs -m
```

```
----- Segment de mémoire partagée -----
clef      shmid      propriétaire perms      octets      nattch      états
```

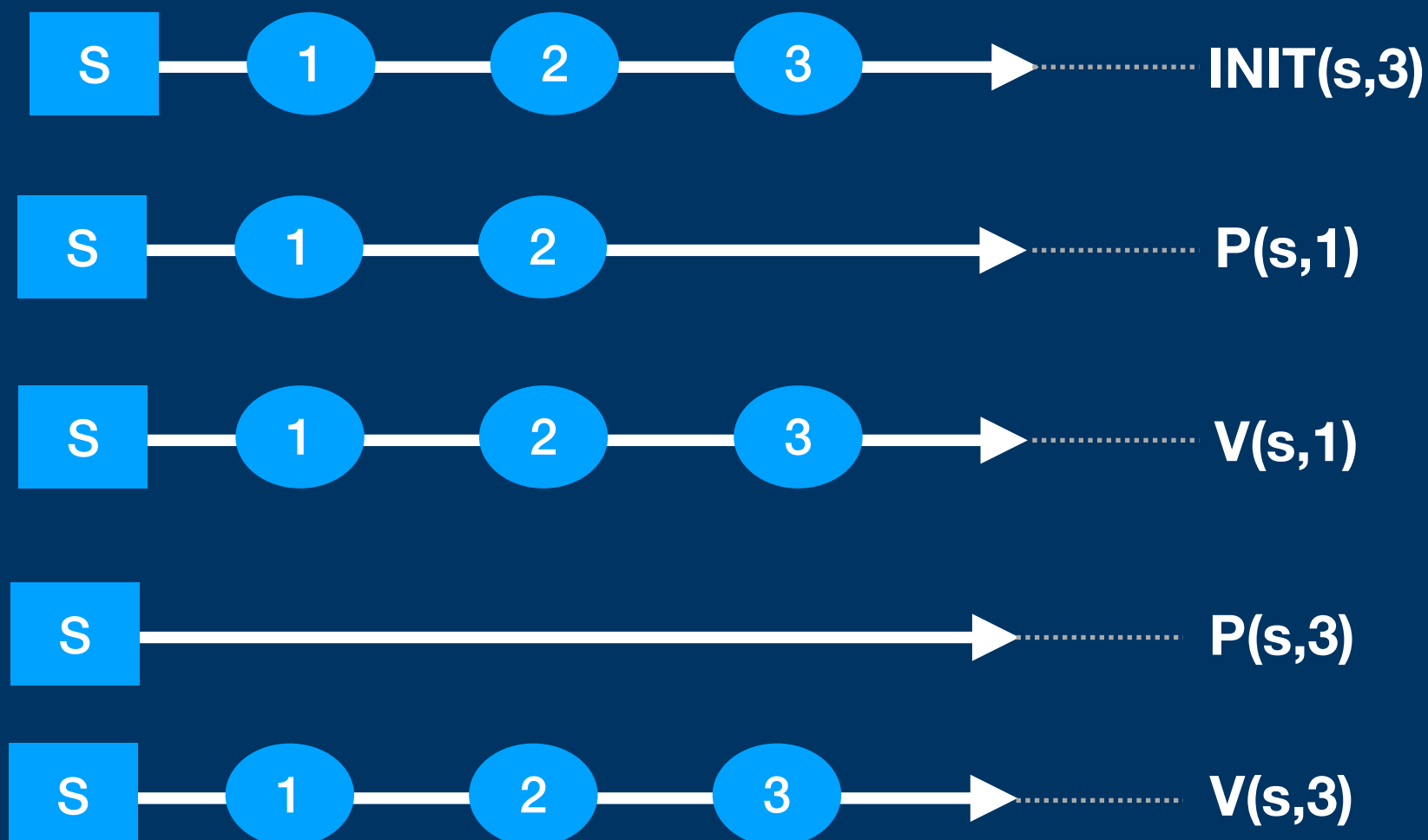

Les Sémaphores

IPC SYSTEM V

Notion de Sémaphore

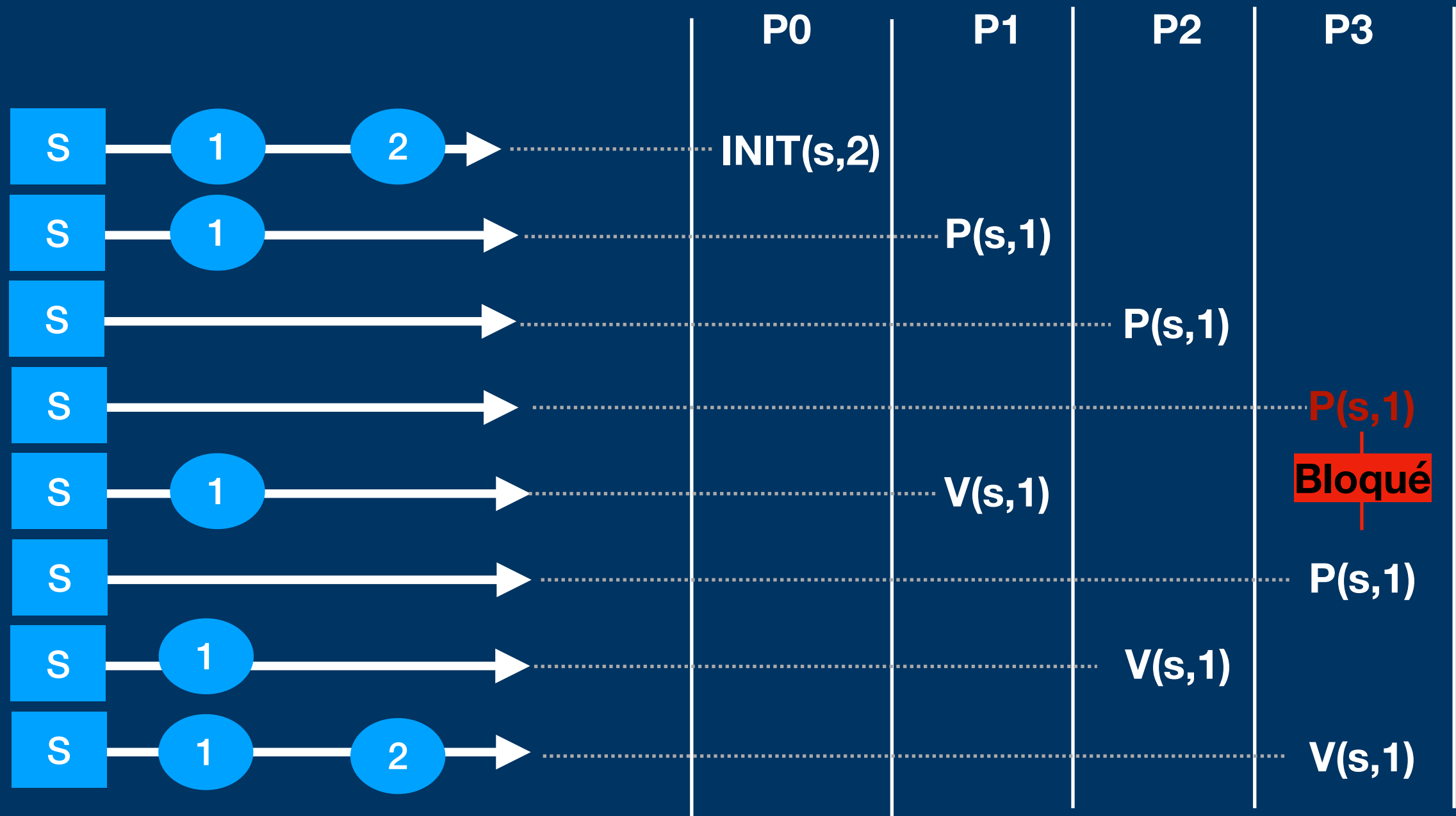
Un sémaphore est un élément de synchronisation qui permet de partager un ensemble de ressources. Il existe des sémaphores pour la programmation en mémoire partagée. Ici les sémaphore System V sont inter-processus. On définit classiquement deux opérations:

- $P(s,n)$: « Tester » (de l'allemand *passering* du fait de Dijkstra)
- $V(s,n)$: « Relâcher » (de l'allemand *vrijgave* du fait de Dijkstra)



Synchronisation avec des Sémaphores

- $P(s,n)$: « Tester »
- $V(s,n)$: « Relâcher »



Créer des Sémaphores

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```

```
int semget(key_t key, int nsems, int semflg);
```

- key : Une clef, soit manuelle, soit via ftok ou bien IPC_PRIVATE
- nsem: nombre de sémaphores à créer
- shmflg: mode de création de la file et ses droits UNIX
 - ➡ IPC_CREAT crée une file s'il y en a aucune associée à cette clef
 - ➡ IPC_EXCL échoue s'il existe déjà une file sur la clef indiqué (toujours combiné avec IPC_CREAT!)
 - ➡ 0600 droit UNIX en octal (important car si omis 0000 et la file et moins pratique !)

Opération sur des Sémaphores

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```

```
int semop(int semid, struct sembuf *sops, size_t nsops);
```

- *semid* : identifiant du sémaphore
- *sembuf*: opération(s) à effectuer via un tableau

```
struct sembuf {
    unsigned short sem_num; /* semaphore number */
    short          sem_op;  /* semaphore operation */
    short          sem_flg; /* operation flags */
};
```

➡ *sem_num*: numéro du sémaphore

➡ *sem_op*: opération à effectuer

▸ *sem_op* > 0 : V(s)

▸ *sem_op* < 0 : P(s)

▸ *sem_op* == 0 : attente de la valeur 0 -> utile pour synchroniser les processus

➡ Drapeau à utiliser :

▸ *IPC_NOWAIT*: non-bloquant et renvoie EAGAIN si l'opération avait dû bloquer

▸ *IPC_UNDO*: demande au noyau d'annuler l'opération si le processus se termine en cas d'arrêt intempestif

- *nsops*: nombre d'opérations à effectuer (elle sont faites de manière atomique)


Contrôle du Sémaphore

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```

```
int semctl(int semid, int semnum, int cmd, ...);
```

- *semid* : identifiant du sémaphore
- *semnum*: identifiant du sémaphore
- *cmd*: commande à appliquer au sémaphore

Non défini dans les headers !



```
union semun {
    int          val; /* Value for SETVAL */
    struct semid_ds *buf; /* Buffer for IPC_STAT, IPC_SET */
    unsigned short *array; /* Array for GETALL, SETALL */
    struct seminfo *__buf; /* Buffer for IPC_INFO
                           (Linux-specific) */
};
```

➡ IPC_STAT récupère les informations sur le sémaphore

➡ SETALL définit la valeur du sémaphore (prend un tableau de unsigned short int en paramètre additionnel)

➡ **IPC_RMID** supprime immédiatement le sémaphore et débloque les processus en attente

➡ ... il existe **BEAUCOUP** d'autres flags voir man

```

#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>
#include <unistd.h>
#include <sys/wait.h>

int main( int argc, char ** argv ){
    int sem = semget(IPC_PRIVATE, 1, IPC_CREAT | 0600);

    if( sem < 0 ){
        perror("msgget");
        return 1;
    }

    unsigned short val = 1;
    if( semctl(sem, 0, SETALL, &val) < 0 ){
        perror("semctl");
        return 1;
    }

    int pid = fork();
    struct sembuf p;

    p.sem_num = 0;
    p.sem_op = -1;
    p.sem_flg = SEM_UNDO;

    struct sembuf v;

    v.sem_num = 0;
    v.sem_op = 1;
    v.sem_flg = SEM_UNDO;

    if( pid == 0 ) { /* Child */
        while(1){
            if( semop(sem, &p, 1) < 0 ){
                printf("Child: SEM deleted\n");
                return 0;
            }

            printf("CHILD holding the sem\n");
            sleep(1);
            semop(sem, &v, 1);
        }
    }
}

```

Suite ...

```

    else
    {
        /* Parent */
        int i = 0;
        while(i < 5)
        {
            semop(sem, &p, 1);

            printf("PARENT holding the sem\n");
            sleep(1);
            semop(sem, &v, 1);
            i++;
        }

        /* Parent delete the sem and unlock the child */
        semctl(sem, 0, IPC_RMID);

        wait( NULL );
    }

    return 0;
}

```

Sortie du Programme

```
$ ./a.out  
PARENT holding the sem  
CHILD holding the sem  
PARENT holding the sem  
CHILD holding the sem  
PARENT holding the sem  
CHILD holding the sem  
PARENT holding the sem  
CHILD holding the sem  
PARENT holding the sem  
CHILD holding the sem  
Child: SEM deleted
```


IPCs POSIX

IPC POSIX

Le standard POSIX plus récent propose également les même mécanismes:

- Files de messages
 - Segment de mémoire partagée
 - Sémaphores
-
- Il sont plus fiables en termes de libération et de partage de la ressource;
 - Enfin l'ensemble de l'interface est thread-safe;
 - Les objets sont demandés par nom et non avec une valeur donnée;
 - Ces appels sont un peu moins portable et sont à attendre plus sur des LINUX que des UNIX au sens large;
 - On les décrit généralement comme plus simples à utiliser.

Files de Message POSIX

À vous de jouer avec le man:

- mq_open
- mq_close
- mq_send
- mq_receive
- mq_unlink

Portez l'exemple SYS-V

Que pensez-vous de *mq_notify* ?

Segment SHM POSIX

À vous de jouer avec le man:

- shm_open
- shm_unlink
- mmap

Portez l'exemple SYS-V

Sémaphore IPC POSIX

Aussi « sémaphore nommé » à ne pas confondre avec les sémaphore « anonymes » de la NPTL (libpthread) qui sont dans le même header.

Rappel (ou pas) pour un sémaphore «anonyme »:

- sem_init
- sem_destroy
- **sem_post**
- **sem_wait**

À vous de jouer avec le man pour un sémaphore nommé:

- sem_open
- sem_close
- **sem_post**
- **sem_wait**
- sem_unlink

Portez l'exemple SYS-V

Peut-on l'implémenter avec un sémaphore anonyme et pourquoi ?

**Préférez vous POSIX
ou SYS-V ?**