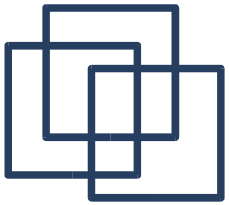
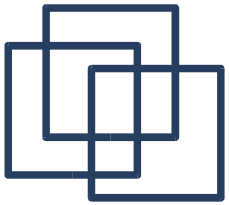


(System V)

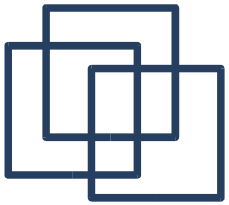
Inter-process Communication

Emmanuel Fleury
B1-201
fleury@cs.aau.dk

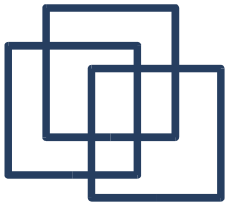


Outline

- Inter-Process Communication
- Semaphores
- Message Queues
- Shared Memory Segments



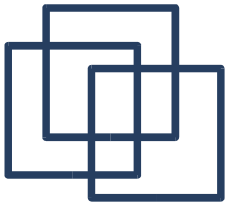
Inter-Process Communication



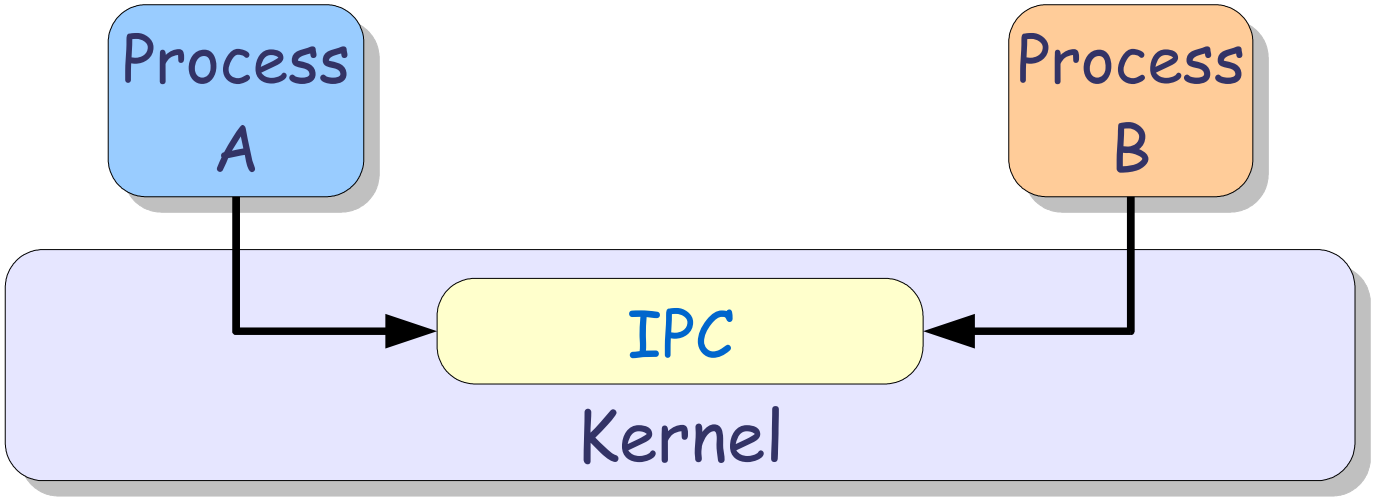
What is (System V) IPC ?

IPC is **live** communication between processes !

- What is IPC ?
  - All processes are **active** at communication time
  - Processes resides in **different** protected domains
- What is **NOT** IPC ?
  - Persistent data communication (files, pipes)
  - Process/Kernel communication (signals)

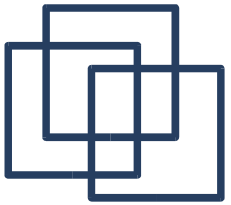


What is (System V) IPC ?



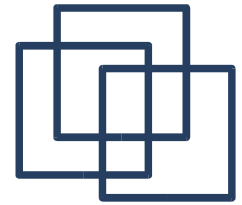
Three IPC mechanisms (sys/ipc.h):

- Semaphores (sys/sem.h)
- Message Queues (sys/msg.h)
- Shared Memory Segments (sys/shm.h)

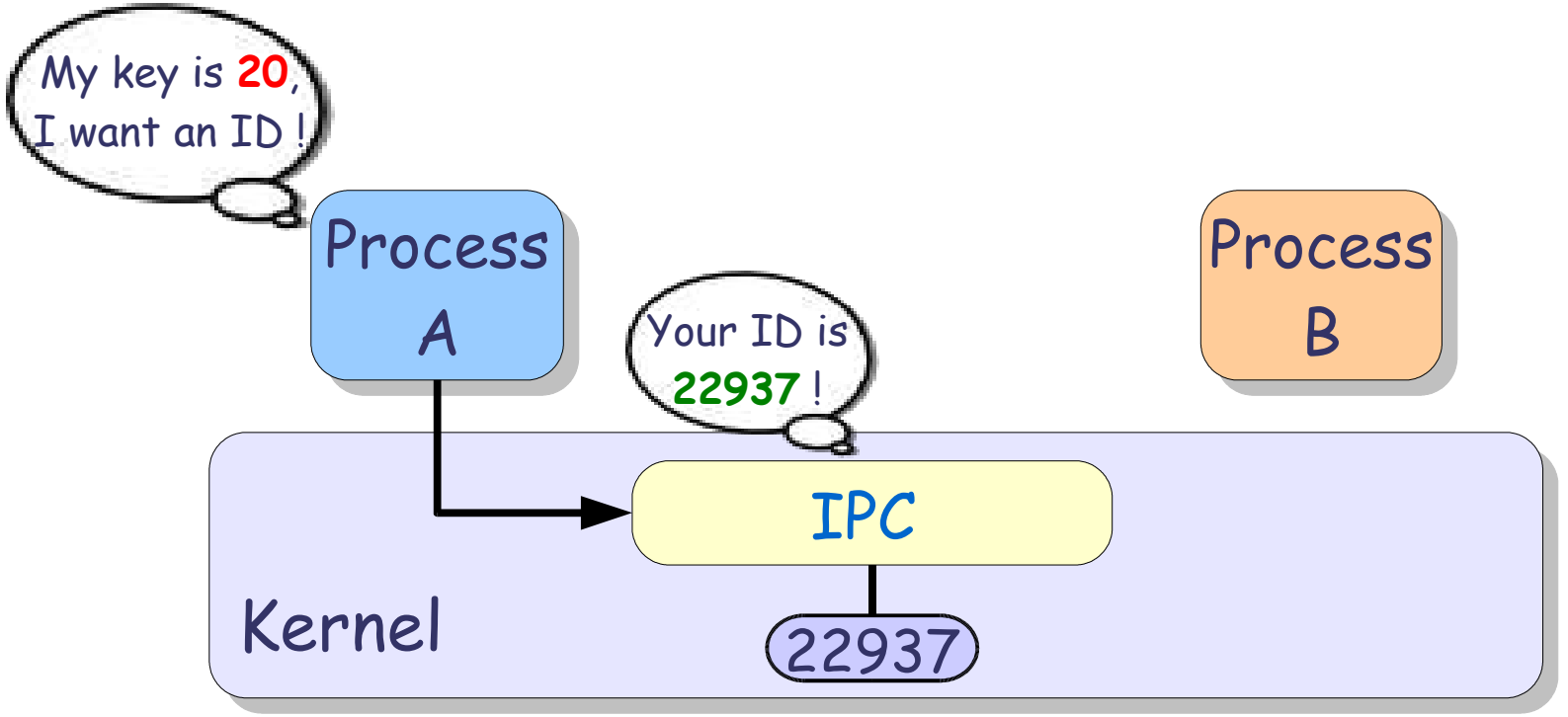


IPC Interface

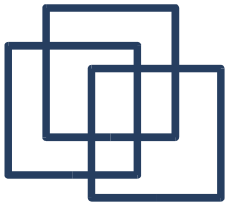
- Each IPC is identified by a unique key (`key_t`) and a data-structure:
  - Semaphore ID (`semid`, `semid_ds`),
  - Message Queues ID (`msqid`, `msqid_ds`),
  - Shared Memory Segment ID (`shmid`, `shmid_ds`)
- Creation through `XXXget()` functions:
  - Semaphore (`semget()`),
  - Message Queues (`msgget()`),
  - Shared Memory Segment (`shmget()`)
- Destruction through `XXXctl()` functions:
  - Semaphore (`semctl()`),
  - Message Queues (`msgctl()`),
  - Shared Memory Segment (`shmctl()`)



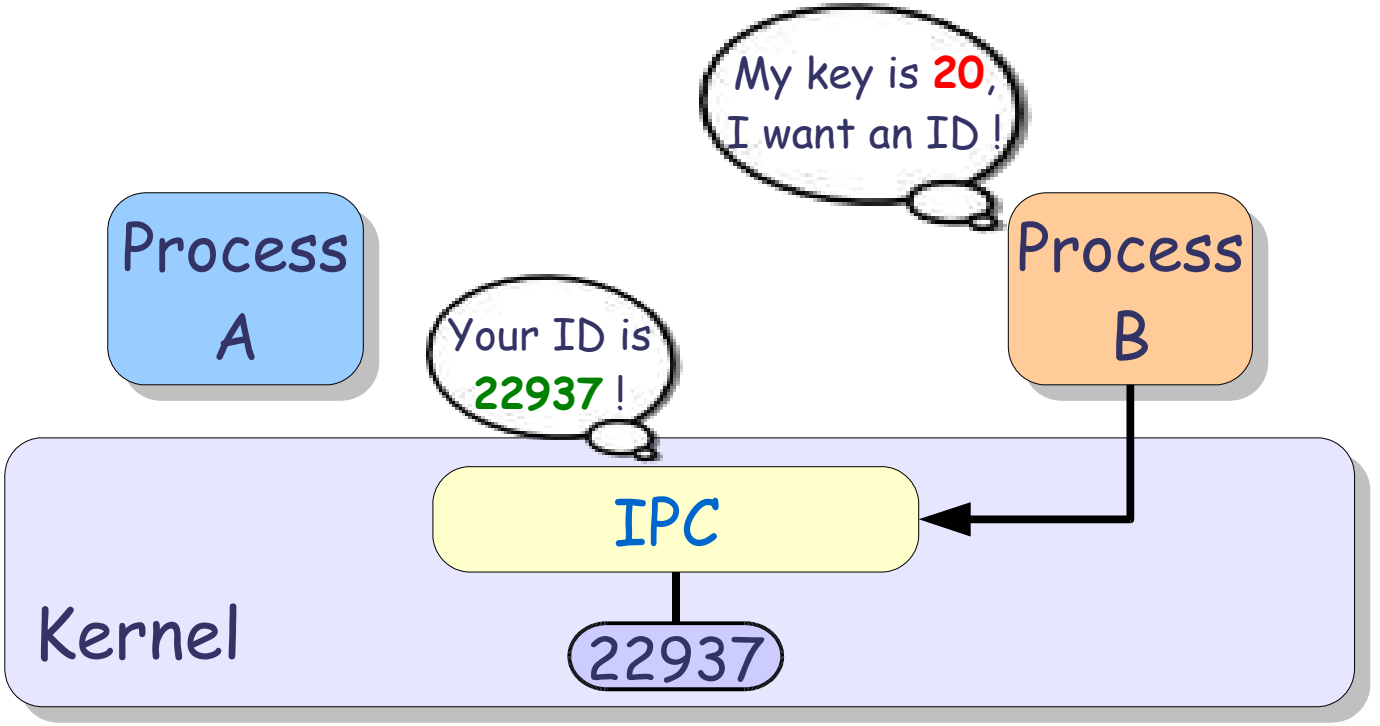
Creation of an IPC



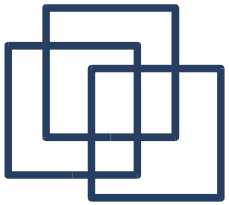
1. The user give a key
2. The kernel create an IPC object if necessary
3. The kernel return an ID



Creation of an IPC



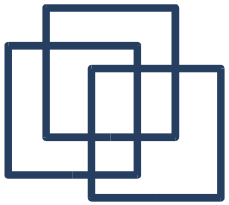
1. The user give a key
2. The kernel create an IPC object if necessary
3. The kernel return an ID



ipcs (IPC Status)

ipcs is used to get the status of all the IPC objects of your system

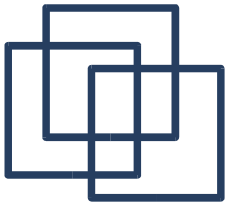
```
[fleury@hermes]$ ipcs
------ Shared Memory Segments --------
key        shmid      owner     perms      bytes      nattch     status
0x00000000 98305      fleury    600        393216     2          dest
------ Semaphore Arrays --------
key        semid      owner     perms      nsems
------ Message Queues --------
key        msqid      owner     perms      used-bytes   messages

[fleury@hermes]$ ipcs -m
------ Shared Memory Segments --------
key        shmid      owner     perms      bytes      nattch     status
0x00000000 98305      fleury    600        393216     2          dest
[fleury@hermes]$ ipcs -s
------ Semaphore Arrays --------
key        semid      owner     perms      nsems

[fleury@hermes]$ ipcs -p -m
------ Shared Memory Creator/Last-op --------
shmid      owner      cpid      lpid
98305      fleury     4463      5294
```



ipcrm (IPC Remove)

ipcrm is used to remove IPC objects from your system

```
[fleury@hermes]$ ipcs
------ Shared Memory Segments --------
key        shmid      owner     perms      bytes      nattch     status
0x00000000 98305      fleury    600        393216     2          dest
------ Semaphore Arrays --------
key        semid      owner     perms      nsems

------ Message Queues --------
key        msqid      owner     perms      used-bytes   messages

[fleury@hermes]$ ipcrm -m 98305

[fleury@hermes]$ ipcs
------ Shared Memory Segments --------
key        shmid      owner     perms      bytes      nattch     status

------ Semaphore Arrays --------
key        semid      owner     perms      nsems

------ Message Queues --------
key        msqid      owner     perms      used-bytes   messages
```

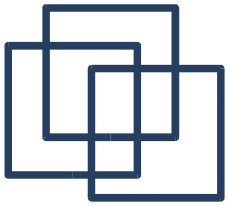


Creation of an IPC

```
#include <sys/types.h>
#include <sys/ipc.h>
int XXXget(key_t key, int flags);
```

- key:
  - An integer
  - IPC_PRIVATE:
    Create a new key and a new IPC
- flags:
  - IPC_CREAT:
    Create entry if key does not exist
  - IPC_EXCL:
    Fail if key exists
  - IPC_NOWAIT:
    Fail if request must wait

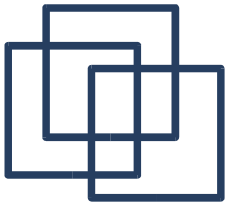
Note: The choice of IPC_PRIVATE was unfortunate. IPC_NEW would better fit to this keyword.



IPC control operations

```
#include <sys/types.h>
#include <sys/ipc.h>
int XXXctl(int ipcid, int cmd,
           struct ipcid_ds *buf);
```

- ipcid: IPC ID
- cmd:
  - IPC_STAT:
    Copy information from the kernel data structure associated with key into the ipcid_ds structure pointed to by buf
  - IPC_SET:
    Write the values of the ipcid_ds structure pointed to by buffer to the kernel data structure associated with this IPC
  - IPC_RMID:
    Destroy the IPC

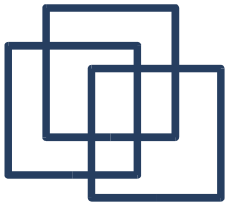


Creating an IPC Object

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

int main() {
  key_t key = IPC_PRIVATE;
  int msqid;

  if ((msqid = msgget(key, 0)) == -1) {
    perror("myipc");
    exit(1);
  }

  printf("The key is %i\n", key);
  printf("The identifier is %i\n", msqid);

  exit(0);
}
```

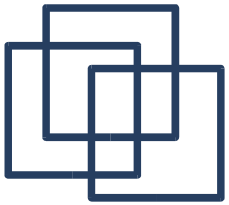
```
[fleury@hermes]$ ./myipc
The key is 0
The identifier is 262144
[fleury@hermes]$ ipcs -q
------ Message Queues --------
key        msqid   owner   perms  used-bytes  messages
[fleury@hermes]$ su -c 'ipcs -q'
------ Message Queues --------
key        msqid   owner   perms  used-bytes  messages
0x00000000 262144  fleury  0      0           0
```

Note: The permissions are not set properly !!!



Ownership & Access Policy

- Each IPC has an ownership and access data (`ipc_perm`):
  - `uid_t uid`: Owner's user ID
  - `gid_t gid`: Owner's group ID
  - `uid_t cuid`: Creator's user ID
  - `gid_t cgid`: Creator's group ID
  - `mode_t mode`: Read/write permissions
- At creation time, the values are:
  - `uid_t uid`: Effective user ID of the creating process
  - `gid_t gid`: Effective group ID of the creating process
  - `uid_t cuid`: Effective user ID of the creating process
  - `gid_t cgid`: Effective group ID of the creating process
  - `mode_t mode`: Read/write permissions from the umask of the creating process

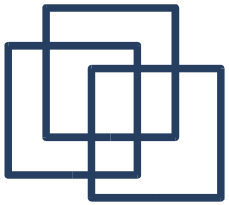


Creating an IPC (take two)

```
[fleury@hermes]$ ./myipc
The key is 0
The identifier is 262144
[fleury@hermes]$ ipcs -q

------ Message Queues --------
key        msqid   owner   perms used-bytes  messages
0x00000000 262144  fleury  0     0           0
```

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/types.h>

int main() {
  key_t key = IPC_PRIVATE;
  int msqid;

  if ((msqid = msgget(key, 0666)) == -1) {
    perror("myipc");
    exit(1);
  }

  printf("The key is %i\n", key);
  printf("The identifier is %i\n", msqid);

  exit(0);
}
```

Note: By definition IPC_PRIVATE = 0

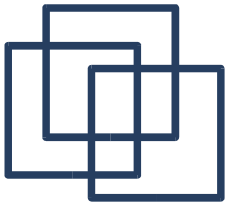


Creating an IPC (given key)

```
[fleury@hermes]$ ./myipc
myipc: No such file or directory
The identifier is -1
```

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/types.h>

int main() {
  int msqid;

  if ((msqid = msgget(20, 0666)) == -1) {
    perror("myipc");
    exit(1);
  }

  printf("The identifier is %i\n", msqid);

  exit(0);
}
```

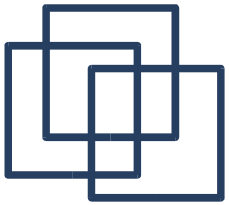
Note: By default, a new key is not created, IPC_CREAT must be specified



Creating an IPC (given key)

```
[fleury@hermes]$ ./myipc
The identifier is 425984
[fleury@hermes]$ ipcs -q

------ Message Queues --------
key        msqid   owner   perms  used-bytes  messages
0x00000014 425984  fleury  666    0           0
```

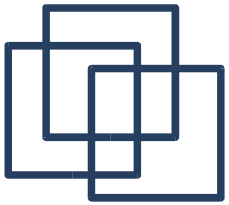
```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/types.h>

int main() {
  int msqd;

  if ((msqid = msgget(20, 0666 | IPC_CREAT)) == -1) {
    perror("myipc");
    exit(1);
  }

  printf("The identifier is %i\n", msqid);

  exit(0);
}
```



Creating an IPC (given key)

```
[fleury@hermes]$ ./myipc
The identifier is 425984
The identifier is 425984
[fleury@hermes]$ ipcs -q

------ Message Queues --------
key        msqid   owner   perms  used-bytes  messages
0x00000014 425984  fleury  666    0           0
```

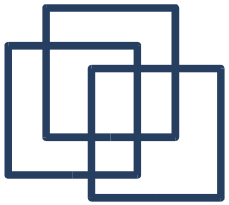
```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/types.h>

int main() {
  int msqid;

  fork();

  if ((msqid = msgget(20, 0666 | IPC_CREAT)) == -1) {
    perror("myipc");
    exit(1);
  }

  printf("The identifier is %i\n", msqid);

  exit(0);
}
```



Deleting an IPC (given key)

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

int main() {
  int msqid;

  if ((msqid = msgget(key, 0666 | IPC_CREAT)) == -1) {
    perror("myipc");
    exit(1);
  }

  printf("The identifier is %i\n", msqid);

  if  ((msgctl(msqid, IPC_RMID, NULL) == -1)) {
    perror("myipc");
    exit(1);
  }

  exit(0);
}
```

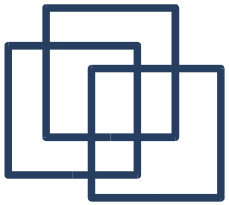
```
[fleury@hermes]$ ./myipc
The identifier is 688128
[fleury@hermes]$ ipcs -q

------ Message Queues --------
key        msqid   owner   perms  used-bytes  messages

[fleury@hermes]$
```



Getting an IPC status

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

int main() {
  int msqid;
  struct msqid_ds *status;

  if ((msqid = msgget(key, 0666 | IPC_CREAT)) == -1) {
    perror("myipc");
    exit(1);
  }

  printf("The identifier is %i\n", msqid);

  if  ((msgctl(msqid, IPC_STAT, NULL) == -1)) {
    perror("myipc");
    exit(1);
  }

  printf("Messages in queue: %i\n", (int) status.msg_qnum);
  printf("Bytes in queue: %i\n", (int) status.msg_qbytes);
  printf("Last process sending: %i\n", (int) status.msg_lspid);
  printf("Last process receiving: %i\n", (int) status.msg_lrpid);

  exit(0);
}
```

```
[fleury@hermes]$ ./myipc
The identifier is 65536
Messages in queue: 0
Bytes in queue: 0
Last process sending: 0
Last process receiving: 0
[fleury@hermes] ipcs -q
------ Message Queues --------
key        msqid  owner   perms  used-bytes  messages
0x00000014 65536  fleury  666    0           0
```

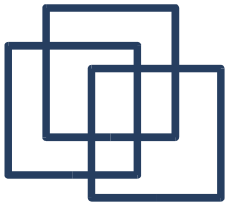


Relate an IPC and a File

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

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

The function `ftok()` uses the identity of the file `pathname` (an already existing and accessible file) and the least significant 8 bits of `proj_id` (non zero) to generate an IPC key, suitable for use with `msgget()`, `semget()`, or `shmget()`.

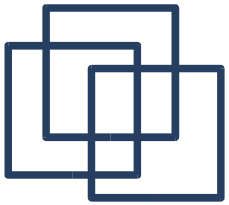
Note: Previously `proj_id` was a char, that's why only the least significant 8 bits are taken into account.



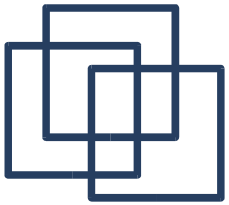
ipc() (Linux specific)

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <sys/msg.h>
int ipc(unsigned int call,
        int first, int second,
        int third, void *ptr,
        long fifth);
```

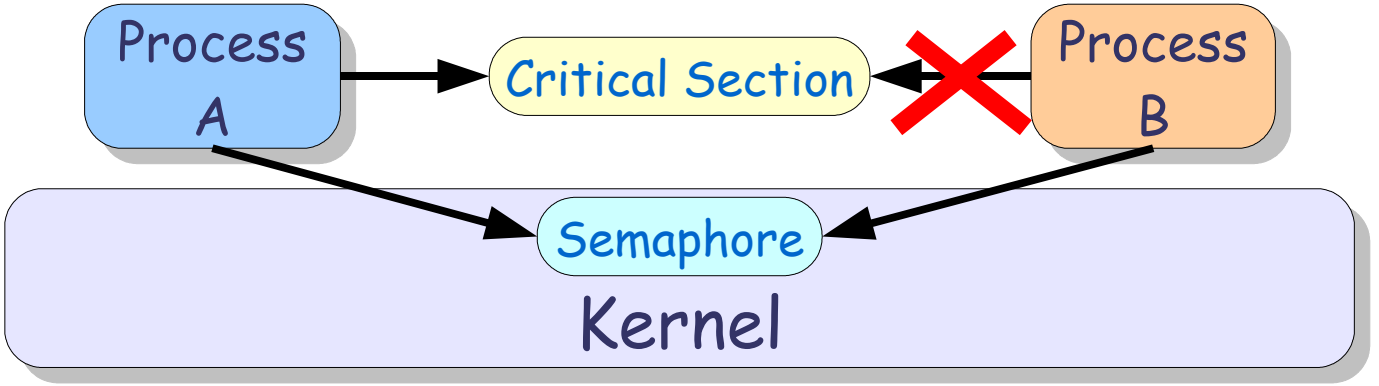
- Implements any call to an IPC function
- Parameters are depending on which function you are calling (`call` tell what function is called)
- Don't use this function if portability is required



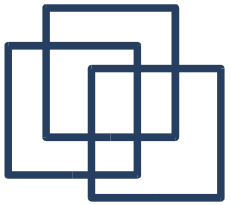
Semaphores



Semaphores



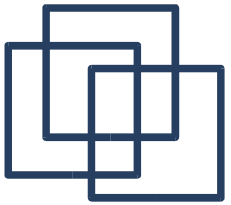
- The IPC semaphore object is a set of semaphores (set of values).
- Each semaphore set is identified by `semid` (id of the set) and each semaphore within the set by `semnum`.
- Each operation performed through `semop()` is atomic.
- The semaphore structure is composed of the following members:
  - unsigned short semval: Semaphore value
  - unsigned short semncnt: Number of processes waiting for semval to increase.
  - unsigned short semzcnt: Number of processes waiting for semval to become 0.
  - pid_t sempid: Process ID of last operation.



System Wide Limitations

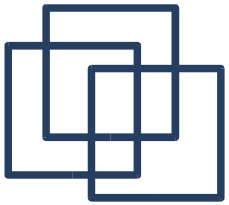
System wide limits on semaphores (/proc/sys/kernel/sem):

- SEMMSL:
  Maximum number of semaphores per set
- SEMMNS:
  Maximum number of semaphores in all sets
- SEMOPM:
  Maximum number of operations specified in semop()
- SEMMNI:
  Maximum number of semaphore identifiers



Semaphores API

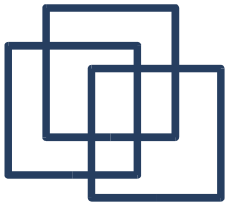
- `semget()`:
  Get a semaphore set's identifier
- `semctl()`:
  Control of semaphores informations
- `semop()`:
  Semaphore operations
- `semtimedop()`:
  Semaphore timed operations



semget()

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semget(key_t semid, int nsems, int semflg);
```

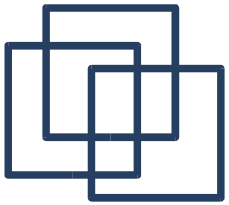
- Get an ID from a key
- Same behaviour as others get() functions with `semid` and `semflg`
- `nsems`: Number of semaphores for this set



semget()

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/sem.h>

int main() {
  int semid;

  /* Creation of the set of semaphores */
  if ((semid = semget(20, 5, 0666 | IPC_CREAT)) == -1) {
    perror("mysems");
    exit(1);
  }

  printf("The ID of the semaphore set is: %i\n", semid);

  exit(0);
}
```

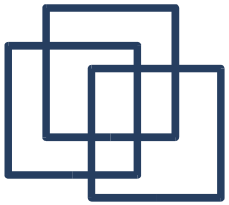
```
[fleury@hermes]$ ./mysems
The ID of the semaphore set is: 65536
[fleury@hermes]$ ipcs -s

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0x00000014 65536      fleury     666        5
```



semctl()

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semctl(int semid, int semnum, int cmd, ...);
```

- semid: IPC ID
- semnu: ID of the semaphore in the set
- cmd: Usual IPC_STAT, IPC_SET, IPC_RMID, and also:
  - GETVAL: Get the current value of the semaphore
  - GETALL: Get the current values for all semaphores
  - SETVAL: Set the current value of the semaphore
  - SETALL: Set the current value for all semaphores
  - GETZCNT: Get the number of processes waiting the value to be 0
  - GETNCNT: Get the number of processes waiting the value to increase
  - GETPID: Get the PID of the last process that accessed the semaphore

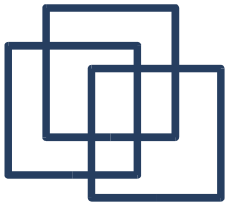


semctl(IPC_RMID)

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/sem.h>

int main() {
  int semid;

  /* Creation of the set of semaphores */
  if ((semid = semget(20, 5, 0666 | IPC_CREAT)) == -1) {
    perror("mysems");
    exit(1);
  }

  printf("The ID of the semaphore set is: %i\n", semid);

  /* Deletion of the set of semaphores */
  if (semctl(semid, 0, IPC_RMID) == -1) {
    perror("mysemaphores");
    exit(1);
    }

  exit(0);
}
```

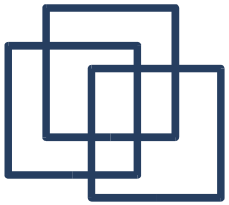
```
[fleury@hermes]$ ./mysems
The ID of the semaphore set is: 65536
[fleury@hermes]$ ipcs -s

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
```

Note: The semaphores in the set are numbered starting at 0.



semctl(SETVAL/GETVAL)

```
int main() {
  union semun semunion;
  int semid;

  if ((semid = semget(20, 5, 0666 | IPC_CREAT)) == -1) {
    perror("mysems");
    exit(1);
  }

  semunion.val = 10;
  if (semctl(semid, 0, SETVAL, semunion) == -1) {
    perror("mysems");
    exit(1);
  }

  if (semctl(semid, 0, GETVAL, semunion) == -1) {
    perror("mysems");
    exit(1);
    }
  printf("The value of the semaphore is %i\n", semunion.val);

  exit(0);
}
```

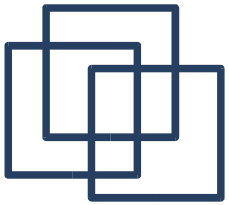


semctl(SETVAL/GETVAL)

```
int main() {
  union semun semunion;
  int semid;

  if ((semid = semget(20, 5, 0666 | IPC_CREAT)) == -1) {
    perror("mysems");
    }
  s
  i
    perror("mysems");
    exit(1);
    }
  printf("The value of the semaphore is %i\n", semunion.val);

  exit(0);
}
```

Note: The definition of semun is sometimes missing in the header files. You may need to add it in the program:

```
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 only) */
};
```

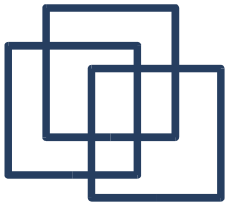


semctl(SETALL/GETALL)

```
[fleury@hermes]$ ./mysems
The value of the semaphore 0 is 1
The value of the semaphore 1 is 1
The value of the semaphore 2 is 2
The value of the semaphore 3 is 1
The value of the semaphore 4 is 3
```

```c
int main() {
  union semun semunion;
  unsigned short array[5] = {1, 1, 2, 1, 3};
  int semid, i;

  if ((semid=semget(20, 5, 0666 | IPC_CREAT)) == -1) {
    perror("mysems");
    exit(1);
  }

  semunion.array = array;
  if (semctl(semid, 0, SETALL, semunion) == -1) {
    perror("mysems");
    exit(1);
  }

  if (semctl(semid, 0, GETVAL, semunion) == -1) {
    perror("mysems");
    exit(1);
  }

  for (i=0; i<5; i++) {
    printf("The value of the semaphore %i is %i\n", i, semunion.array[i]);
  }
  exit(0);
}
```

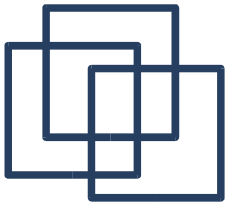
Note: The array must have the exact same size as the semaphores set.



semun & semid_ds

```
struct semid_ds {
   struct ipc_perm sem_perm;  /* Ownership and permissions */
   time_t          sem_otime; /* Last semop time           */
   time_t          sem_ctime; /* Last change time          */
   unsigned short  sem_nsems; /* No. of semaphores in set  */
};
```

```
union semun {
   int               semval; /* 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 only) */
};
```

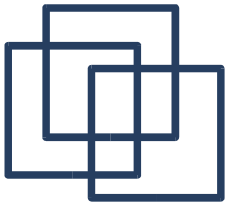


semop()

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semop(int semid, struct sembuf *sops,
          unsigned nsops);
```

Atomic operations on semaphores are performed through the `semop()` system call

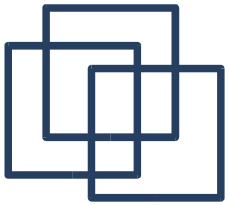
- `semid`: IPC ID of the set of semaphores
- `sops`: Array of operation(s) to perform atomically !
- `nsops`: Number of elements in the array `sops`.



sembuf

```
struct sembuf {
  ushort sem_num; /* semaphore ID          */
  short sem_op;   /* semaphore operation */
  short sem_flg;  /* operation flags       */
}
```

- sem_op: Adds this value to the semaphore value
- sem_flg:
  - IPC_NOWAIT:
    Operation is performed if it can be done instantly
  - SEM_UNDO:
    Automatically undo when the process terminates



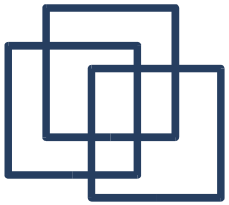
semop()

```c
int lock(int semid, int semnum) {
  struct sembuf semb;

  semb.sem_num = semnum;
  semb.sem_op = -1;
  semb.sem_flg = SEM_UNDO;

  if (semop(semid, &semb, 1) == -1) {
    perror("lock");
    return 1;
  }

  return 0;
}
```

```
[fleury@hermes]$ ./lock_sem0
^C
[fleury@hermes]$ ./lock_sem1
[fleury@hermes]$
```

Note: The value of a semaphore can't be less than 0. If a process try to decrease it when the value is 0, then it will hang until the value increase again.

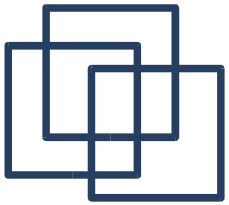
```c
int unlock(int semid, int semnum) {
  struct sembuf semb;

  semb.sem_num = semnum;
  semb.sem_op = 1;
  semb.sem_flg = SEM_UNDO;

  if (semop(semid, &semb, 1) == -1) {
    perror("unlock");
    return 1;
  }

  return 0;
}
```



semtimedop()

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semtimedop(int semid, struct sembuf *sops,
               unsigned nsops, struct timespec *timeout);
```

- Look at the manual page... :-)

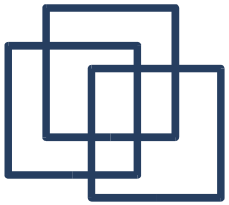


Full Example

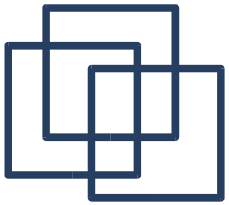
```
int main() {
  union semun semunion;
  unsigned short array[5] = {1, 1, 1, 1, 1};
  int semid;
  pid_t id;

  /* Creation of the IPC */
  if ((semid = semget(20, 5, 0666 | IPC_CREAT)) == -1) {
    perror("mysems");
    exit(1);
  }

  /* Initialization of the semaphores */
  semunion.array = array;
  if (semctl(semid, 0, SETALL, semunion) == -1) {
    perror("mysems");
    exit(1);
  }

  id = fork(); /* Forking */

  if (lock(semid, 0)) /* Locking */
    exit(1);

  /* Critical section */
  printf("I'm %i and I'm in critical section !\n", id);
  sleep(1);

  if (unlock(semid, 0)) /* Unlocking */
    exit(1);

  exit(0);
}
```

```
[fleury@hermes]$ ./mysems
I'm 0 and I'm in critical section !
I'm 4994 and I'm in critical section !
[fleury@hermes]$
```

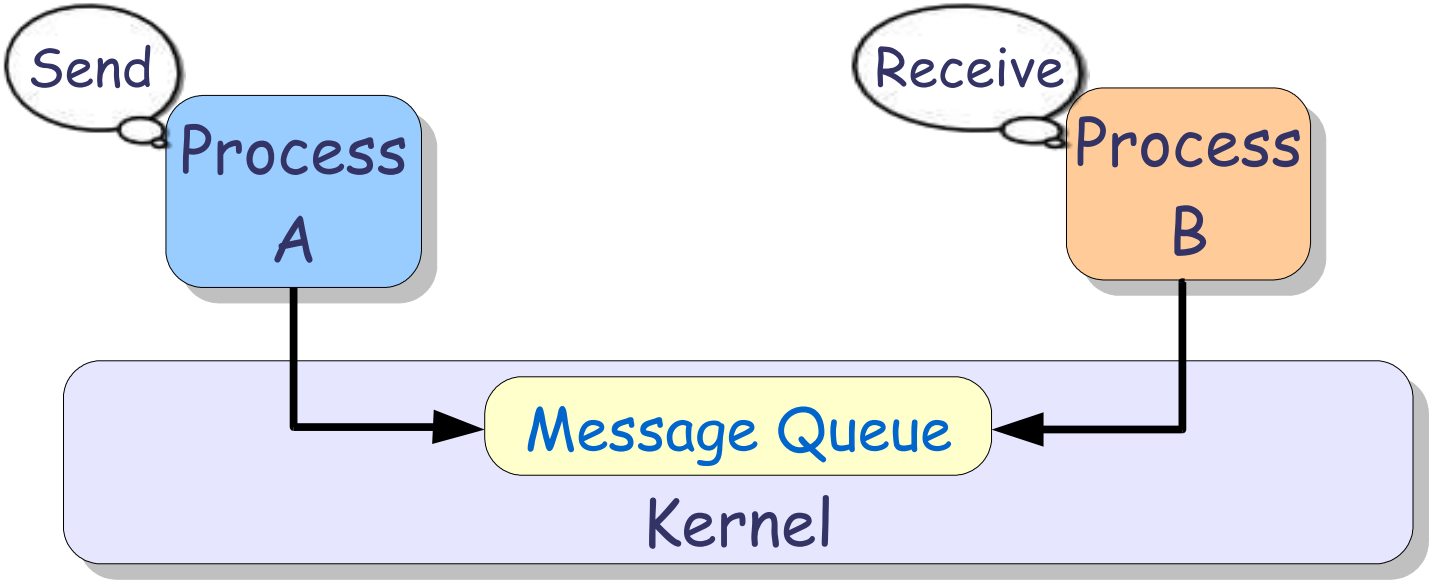
Note: The set of semaphores is still here. Think to clean after you.



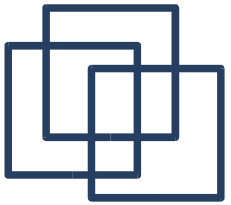
Message Queues



Message Queues



- Queues are Linked-list of messages (with a maximum number of cells)
- Message size must be known (unlike pipes which are exchanging streams)
- Messages are typed (to avoid confusion when fetching one message)
- Mailbox Mechanism (`send()/receive()`)

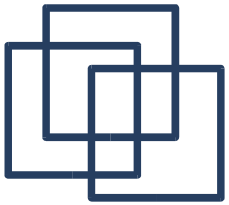


System Wide Limitations

- `MSGMNI`:
  Maximum number of message queues
  (`/proc/sys/kernel/msgmni`)

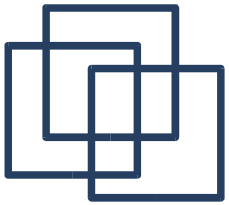
- `MSGMAX`:
  Maximum number of messages per queue
  (`/proc/sys/kernel/msgmax`)

- `MSGMNB`:
  Maximum number of overall messages
  (`/proc/sys/kernel/msgmnb`)



Message Queues API

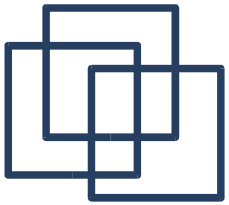
- `msgget()`:
  Create or open a message queue
- `msgctl()`:
  Control message queue informations
- `msgsnd()`:
  Send a message to a message queue
- `msgrcv()`:
  Receive a message from a message queue



msgget()

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgget(key_t key, int flags);
```

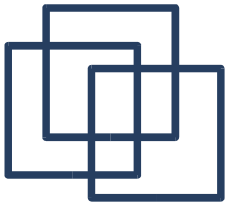
- key:
  - An integer
  - IPC_PRIVATE:
    Create a new key and a new IPC
- flags:
  - IPC_CREAT:
    Create entry if key does not exist
  - IPC_EXCL:
    Fail if key exists
  - IPC_NOWAIT:
    Fail if request must wait



msgctl()

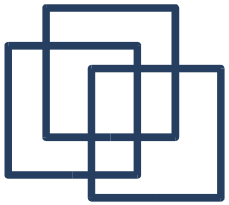
```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgctl(int msgid, int cmd,
           struct msqid_ds *buffer);
```

- msgid: IPC ID
- cmd:
  - IPC_STAT:
    Copy information from the kernel data structure associated with msqid into the msqid_ds structure pointed to by buffer
  - IPC_SET:
    Write the values of some members of the msqid_ds structure pointed to by buffer to the kernel data structure associated with this message queue, updating also its msg_ctime member
  - IPC_RMID:
    Remove the message queue, awake all waiting reader and writer processes



msqid_ds

```
struct msqid_ds {
 ipc_perm msg_perm;      /* Ownership and permissions */
 time_t   msg_stime;     /* Time of last msgsnd()     */
 time_t   msg_rtime;     /* Time of last msgrcv()     */
 time_t   msg_ctime;     /* Time of last change       */
 ulong    __msg_cbytes; /* No. of bytes in queue     *
                          * (Linux specific)         */
 msgqnum_tmsg_qnum;      /* No. of messages in queue  */
 msglen_t msg_qbytes;    /* Maximum number of bytes   *
                          * allowed in queue         */
 pid_t    msg_lspid;     /* PID of last msgsnd()      */
 pid_t    msg_lrpid;     /* PID of last msgrcv()      */
};
```



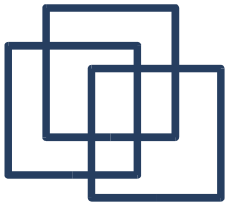
msgsnd()

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgsnd(int msqid,
           void *msgp,
           size_t msgsz,
           int msgflg);
```

- msqid: IPC ID
- msgp: Pointer to the message data (can be anything):

```
struct msgbuf {
 long  mtype;  /* message type, must be > 0 */
 char *mtext;  /* message data of size msgsz */
};
```

- msgsz: Size of the message (bytes)
- msgflg:
  - IPC_NOWAIT: Immediate return if no message of the type is in queue
  - MSG_EXCEPT: If (msgtyp > 0) read the first message in the queue.
  - MSG_NOERROR: Truncate the message text if longer than msgsz bytes.



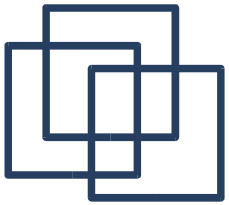
msgrcv()

- msqid: IPC ID
- msgp: Pointer to the message data (can be anything):

```
struct msgbuf {
 long  mtype;  /* message type, must be > 0 */
 char *mtext;  /* message data of size msgsz */
};
```

- msgsz: Size of the message (bytes)
- msgtype: Type of the message
- msgflg:
  - IPC_NOWAIT: Immediate return if no message of the type is in queue
  - MSG_EXCEPT: If (msgtyp > 0) read the first message in the queue.
  - MSG_NOERROR: Truncate the message text if longer than msgsz bytes.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgget(int msqid,
           void *msgp,
           size_t msgsz,
           long msgtype,
           int msgflg);
```

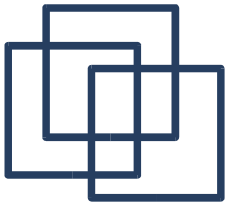


Full Example

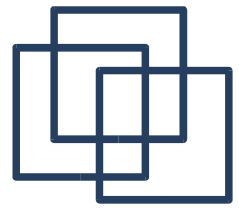
```
struct mymsg {
  long mtype;
  char *mtext;
};

int main() {
  int msqid;
  struct mymsg msg;
  char buffer[10] = “abcdefghi\0”;

  msg.mtype = 1;
  msg.mtext = buffer;

  /* Creation of the IPC */
  if ((msqid = msgget(20, 0666 | IPC_CREAT)) == -1) {
    perror("mymsg");
    exit(1);
  }

  /* Sending a message */
  msgsnd(msqid, &msg, 3, 0);

  sleep(5);

  /* Receiving a message */
  msgrcv(msqid, &msg, 3, 1, 0);

  printf("The message is: %s", msg.mtext);
  exit(0);
}
```

```
[fleury@hermes]$ ipcs -q
------ Message Queues --------
key        msqid    owner   perms used-bytes  messages
0x00000014 2850816  fleury  666   0           0
[fleury@hermes]$ ipcs -q
------ Message Queues --------
key        msqid    owner   perms used-bytes  messages
0x00000014 2850816  fleury  666   3           1
[fleury@hermes]$ ipcs -q
------ Message Queues --------
key        msqid    owner   perms used-bytes  messages
0x00000014 2850816  fleury  666   0           0
```

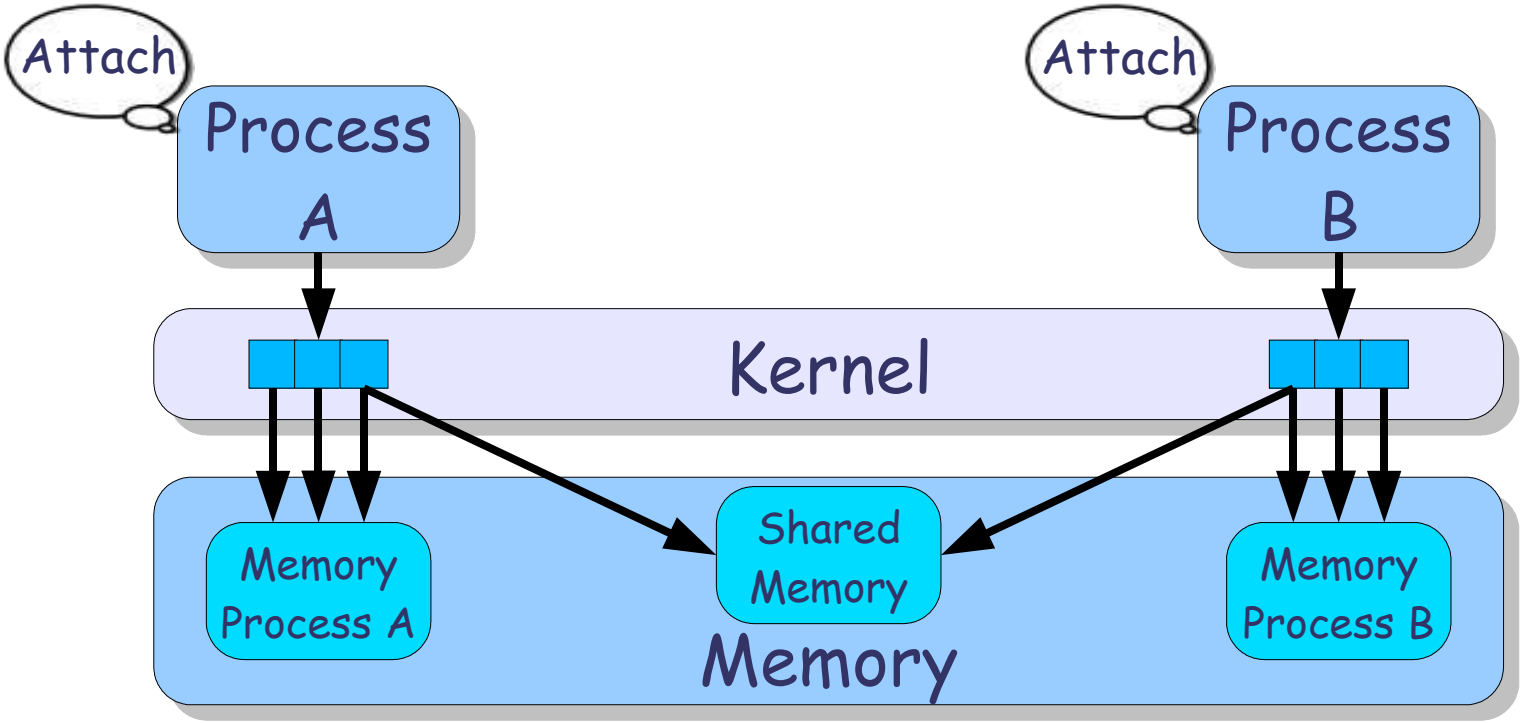
```
[fleury@hermes]$ ./mysmsg
The message is: abcdefghi
[fleury@hermes]$
```

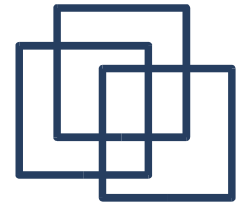


Shared Memory

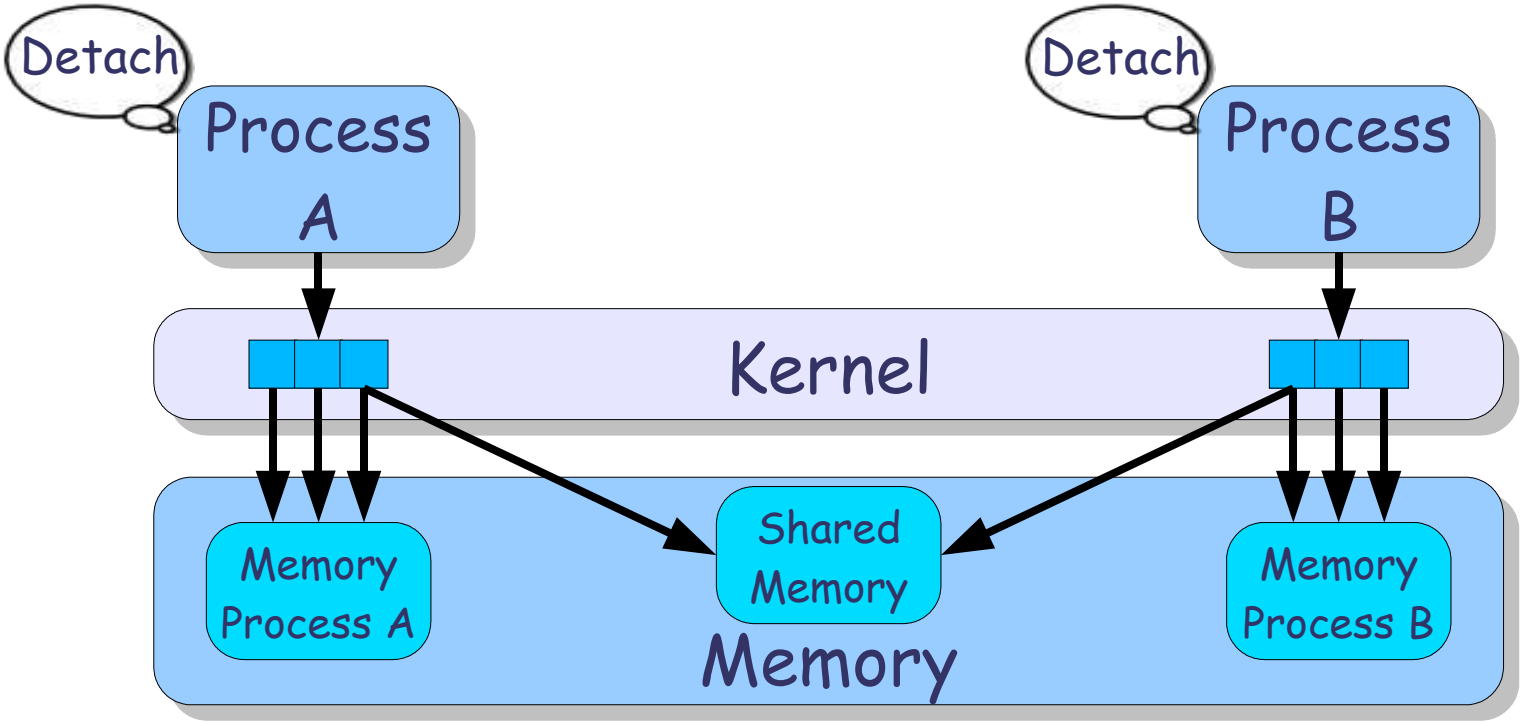


Shared Memory

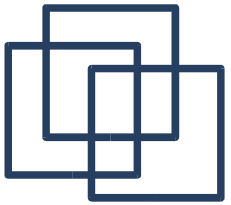




Shared Memory



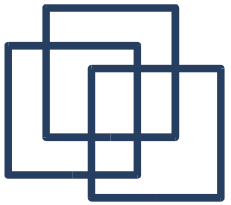
- Allow unrelated processes to share the same logical memory.
- Warning !
  - No mechanism preventing race conditions or read/write problems. Accessing this memory should be protected via semaphores. Remember also to clean after you !
  - This does not enlarge the logical memory of a process, it only replaces a part of it by a shared memory.



System Wide Limitations

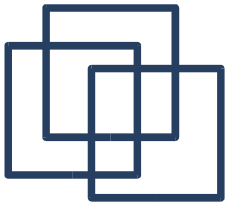
System wide limits on shared Memory (/proc/sys/kernel/shm*):

- SHMALL: Maximum number of shared memory pages (/proc/sys/kernel/shmall)
- SHMMAX: Maximum size (bytes) of shared segments (/proc/sys/kernel/shmmax)
- SHMMIN: Minimum size (bytes) of a shared segment (/proc/sys/kernel/shmmin)
- SHMMNI: Maximum number of shared segments (/proc/sys/kernel/shmmni)



Shared Memory API

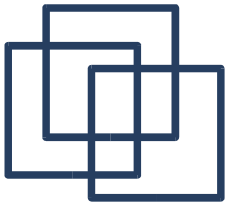
- `shmget()`:
  Allocate a memory area and return an identifier
- `shmctl()`:
  Control shared memory informations
- `shmat()`:
  Attach an IPC shared memory area to the process
- `shmdt()`:
  Detach an IPC shared memory area from the process



shmget()

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key, int size, int flags);
```

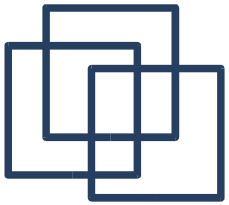
- key:
  - An integer
  - IPC_PRIVATE:
    Create a new key and a new IPC
- size: Number of memory pages allocated to the new memory segment
- flags:
  - IPC_CREAT: Create entry if key does not exist
  - IPC_EXCL: Fail if key exists
  - IPC_NOWAIT: Fail if request must wait



shmctl()

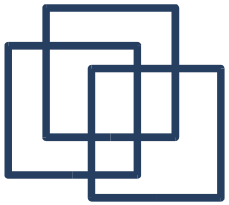
- shmid: IPC ID
- cmd:
  - IPC_STAT:
    Copy information from the kernel data structure associated with key into the ipcid_ds structure pointed to by buf
  - IPC_SET:
    Write the values of the ipcid_ds structure pointed to by buffer to the kernel data structure associated with this IPC
  - IPC_RMID:
    Destroy the IPC
  - SHM_LOCK/SHM_UNLOCK (Linux specific):
    Prevent/Allow the memory to be swapped

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmctl(int shmid, int cmd,
           struct shmid_ds *buf);
```



shmid_ds

```
struct shmid_ds {
  struct ipc_perm shm_perm;    /* Ownership and permissions    */
  size_t          shm_segsz;   /* Size of segment (bytes)      */
  time_t          shm_atime;   /* Last attach time             */
  time_t          shm_dtime;   /* Last detach time             */
  time_t          shm_ctime;   /* Last change time             */
  pid_t           shm_cpid;    /* PID of creator               */
  pid_t           shm_lpid;    /* PID of last stmat()/shmdt()  */
  shmatt_t        shm_nattch;  /* No. of current attaches      */
  ...
};
```

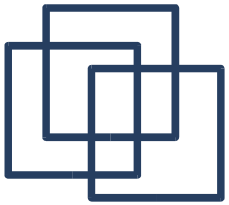


shmat()

```
#include <sys/types.h>
#include <sys/shm.h>
void *shmat(int shmid,
            const void *shmaddr,
            int shmflg);
```

- shmid: IPC ID
- shmaddr:
  - If NULL, the system is choosing a suitable address.
  - If not NULL, the given address is taken (if free).
- shmflg:
  - 0: Read/write access.
  - SHM_RDONLY: Read only access.
  - SHM_RND: If (shmaddr != NULL) attach is at shmaddr rounded down to the nearest multiple of SHMLBA (Segment low boundary address multiple).
  - SHM_REMAP (Linux specific): The segment replaces any existing mapping in the range starting at shmaddr and continuing for the size of the segment.

Note: On success shmat() returns the address of the memory segment and -1 in case of failure.



shmdt()

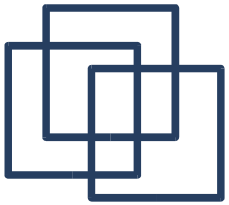
```c
#include <sys/types.h>
#include <sys/shm.h>
int shmat(const void *shmaddr);
```

- shmaddr: Address of the shared memory segment to detach from the process.
- Return:
  - 0 on success
  - -1 on fail

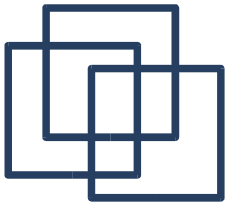
Full Example

```c
#define MEMSIZE 1
#define BUFSIZE 1000

int main()
{
  int shmid, i;
  char *buffer;

  /* Creation of the IPC */
  if ((shmid = shmget(20, MEMSIZE, IPC_CREAT | 0666)) == -1) {
    perror("shared_memory");
    exit(1);
  }

  /* Attach to the IPC */
  if ((int) (buffer = shmat(shmid, NULL, 0)) == -1) {
    perror("myshm");
    exit(1);
  }

  /* Use the shared memory */
  for (i=0; i<BUFSIZE; i++)
    buffer[i] = 'a';
  buffer[BUFSIZE]='\0';
  puts(buffer);

  /* Detach to the IPC */
  if (shmdt(buffer) == -1) {
    perror("myshm");
    exit(1);
  }

  /* Destroy the IPC */
  if (shmctl(shmid, IPC_RMID, NULL) == -1) {
    perror("myshm");
    exit(1);
  }

  exit(0);
}
```

```
[fleury@hermes]$ ./myshm
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```



Questions ?



Next Weeks

- Threads
  - Creation/Termination
  - Synchronizations Mechanisms
- Programming CORBA (ORBit2)