BasicOS

Communication between Processes

Ludovic Apvrille ludovic.apvrille@telecom-paris.fr
Eurecom, office 470

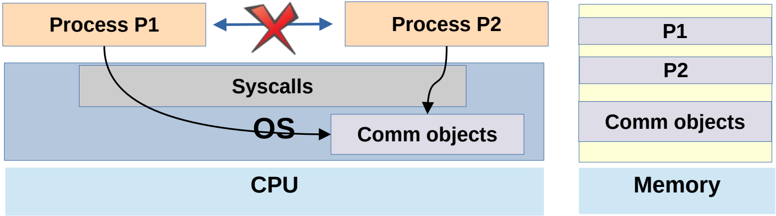
https://perso.telecom-paris.fr/apvrille/BasicOS/

Communication between Processes

Definition

Exchange of data between at least two processes

- Processes cannot communicate without a feature provided by the OS
  - Indeed: default behaviour = isolation between processes (**protection**)



Basic Communication (See Previous Lecture)

Data streams between processes with |

cmd1 | cmd2: ouput stream of *cmd*1 is forwarded to the input stream of *cmd*2

Data streams using intermediate files

```
$ ls > /tmp/listoffiles

$ grep bi /tmp/listoffiles
bin
```

Others: sending a signal, error codes, ...

```
$ kill -9 1017
$ cmd1&&cmd2
```

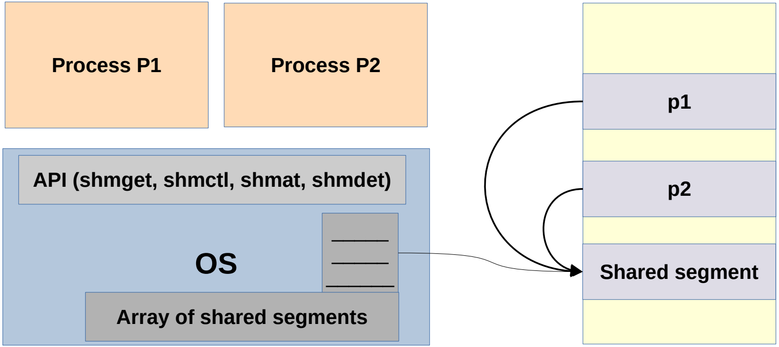
Advanced Communication

- Shared memory
- Message queues
- Signals
- Networking
- Semaphores
- ...

Shared Memory

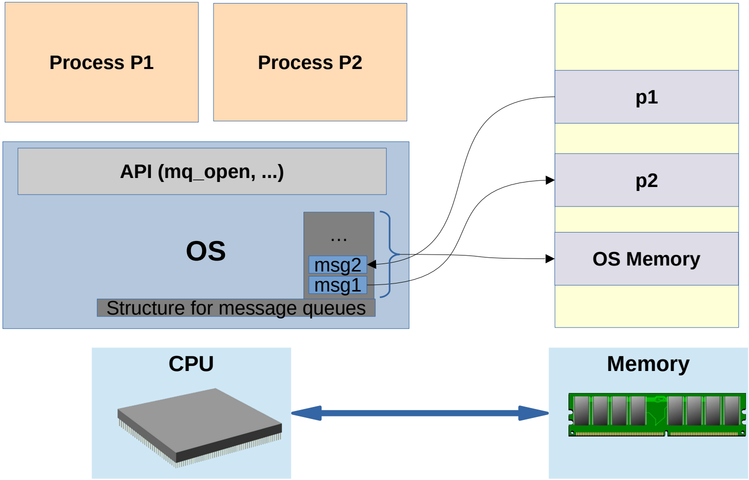
Basics

- Two or more processes may share a given region of memory
- Most efficient way to exchange data because there is no copy of data between one process address space to another

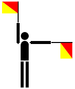


Message Queues

- Way to exchange **messages** between processes



Signals



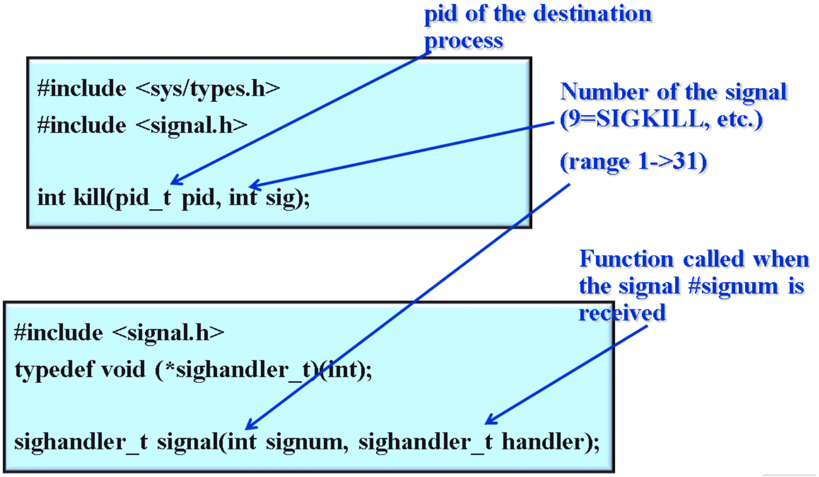
Signals

- Asynchronous events
- For example, signal to stop a process (CTRL-C)
- The OS forwards signals to the destination process

Signals under Linux

- 31 signals
- Name = SIG*: SIGKILL, SIGSTOP, SIGTRAP, etc.
- Three ways to manage signals that are received
  1. Ignore signals (except SIGKILL, SIGSTOP)
  2. Catch signals
  3. Let the default action apply
- If the signal is not ignored, the process is awoken (if necessary)

UNIX System Calls for Sending and Receiving Signals



pid of the destination process

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

int kill(pid_t pid, int sig);
```

Number of the signal (9=SIGKILL, etc.)

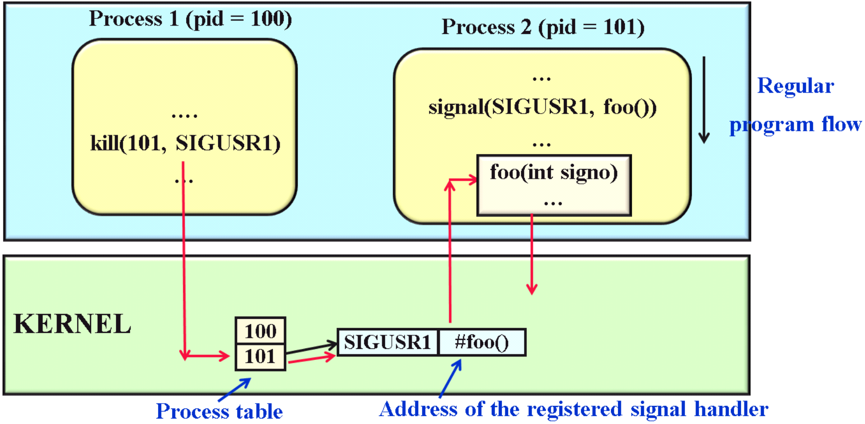
(range 1->31)

Function called when the signal #signum is received

```
#include <signal.h>
typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);
```

Sending and Catching a Signal



Code at Receiver's Side

```c
#include <signal.h>

void getSignal(int signo) {
  if (signo == SIGUSR1) {
    printf("Received SIGUSR1\n");
  } else {
    printf("Received%d\n", signo);
  }
  return;
}

int main(void) {
  printf("Registering #SIGUSR1=%d\n", SIGUSR1);
  signal(SIGUSR1, getSignal);
  sleep(30);
  printf("End of sleep\n");
}
```

(Simplified code... ALWAYS test the return value of all functions!)

Code at Sender's Side

```c
#include <sys/types.h>
#include <signal.h>

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

  if (argc <2) {
    printf("Usage: sender <destination process pid>\n");
    exit(-1);
  }

  pid = atoi(argv[1]);
  printf("Sending SIGURG to %d\n", pid);
  if (kill(pid, SIGURG) == -1) return;
  printf("Sending SIGUSR1 to %d\n", pid);
  if (kill(pid, SIGUSR1) == -1) return;
}
```

Let’s Execute this Code!

Shell 1

```
$ gcc -o receiver receiver.c
$ receiver
Registering #SIGUSR1=10


Received SIGUSR1
End of sleep
$
```

Shell 2

```
$ gcc -o sender sender .c


$ ps
 PID TTY          TIME CMD
23930 pts/1    00:00:00 bash
2241 pts/1    00:00:00 receiver
2242 pts/1    00:00:00 ps
$ sender 2241
Sending SIGURG to 2241
Sending SIGUSR1 to 2241
$
```