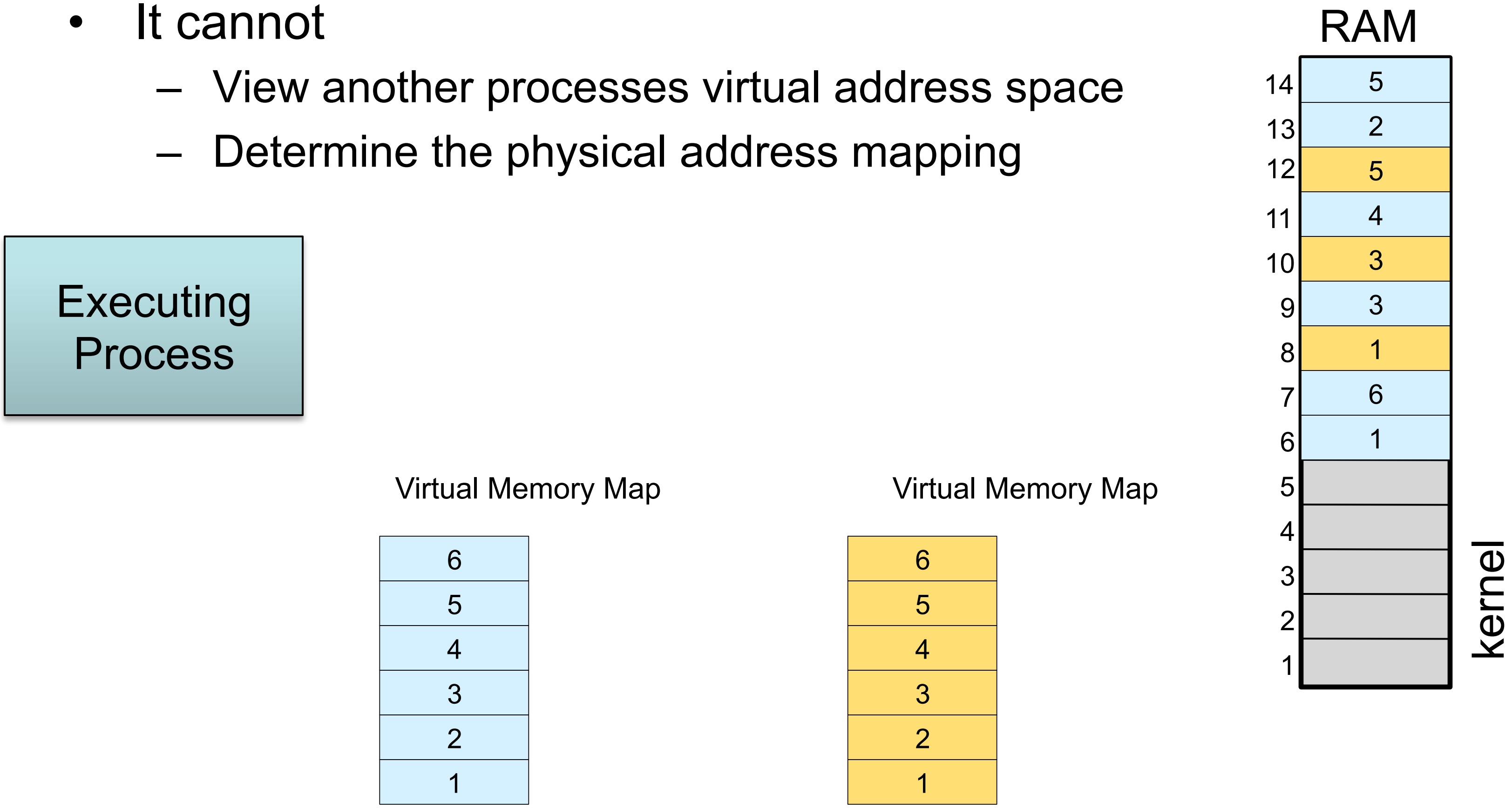
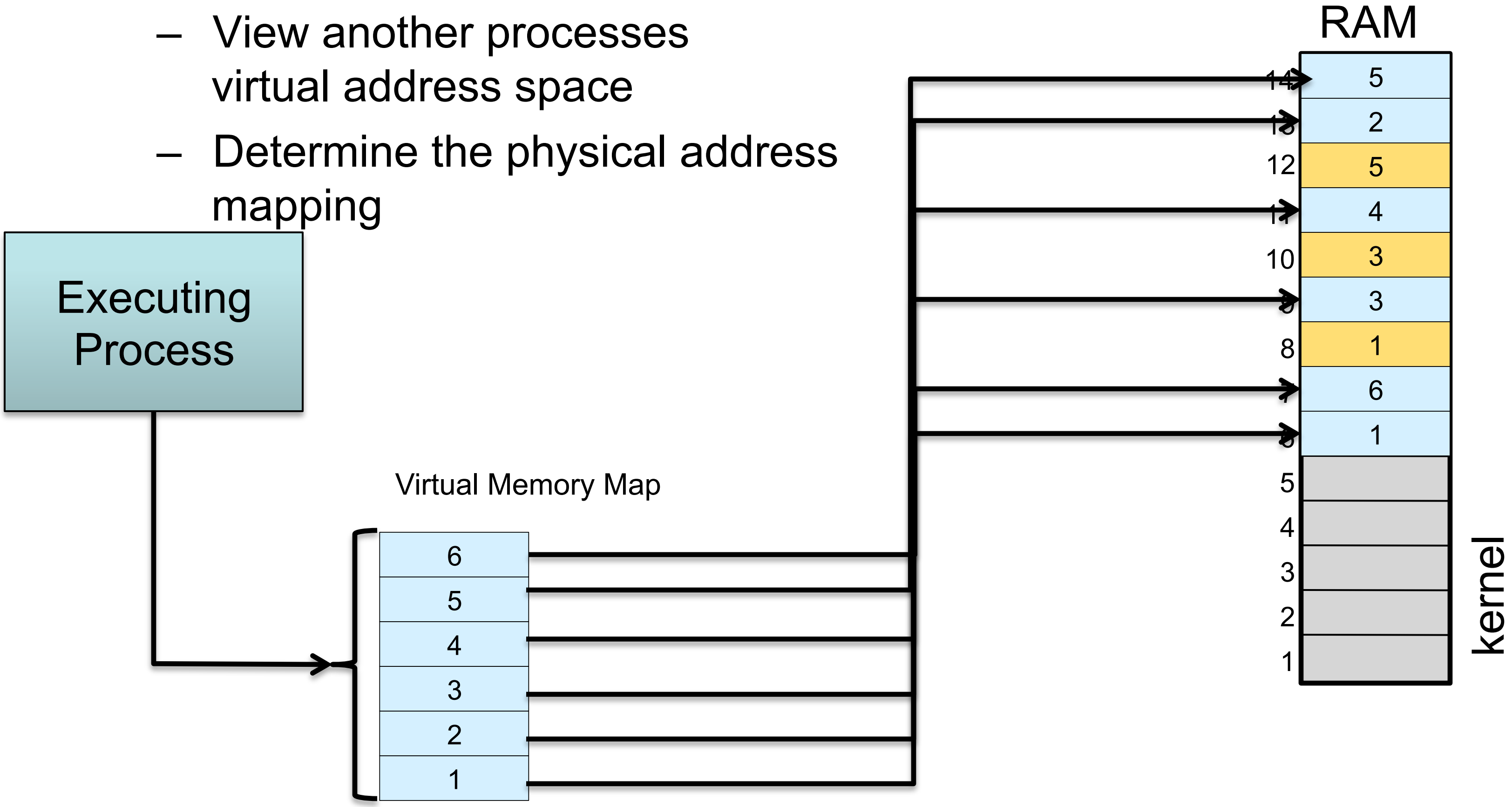
Virtual Memory View

- During execution, each process can only view its virtual addresses,
- It cannot
  - View another processes virtual address space
  - Determine the physical address mapping



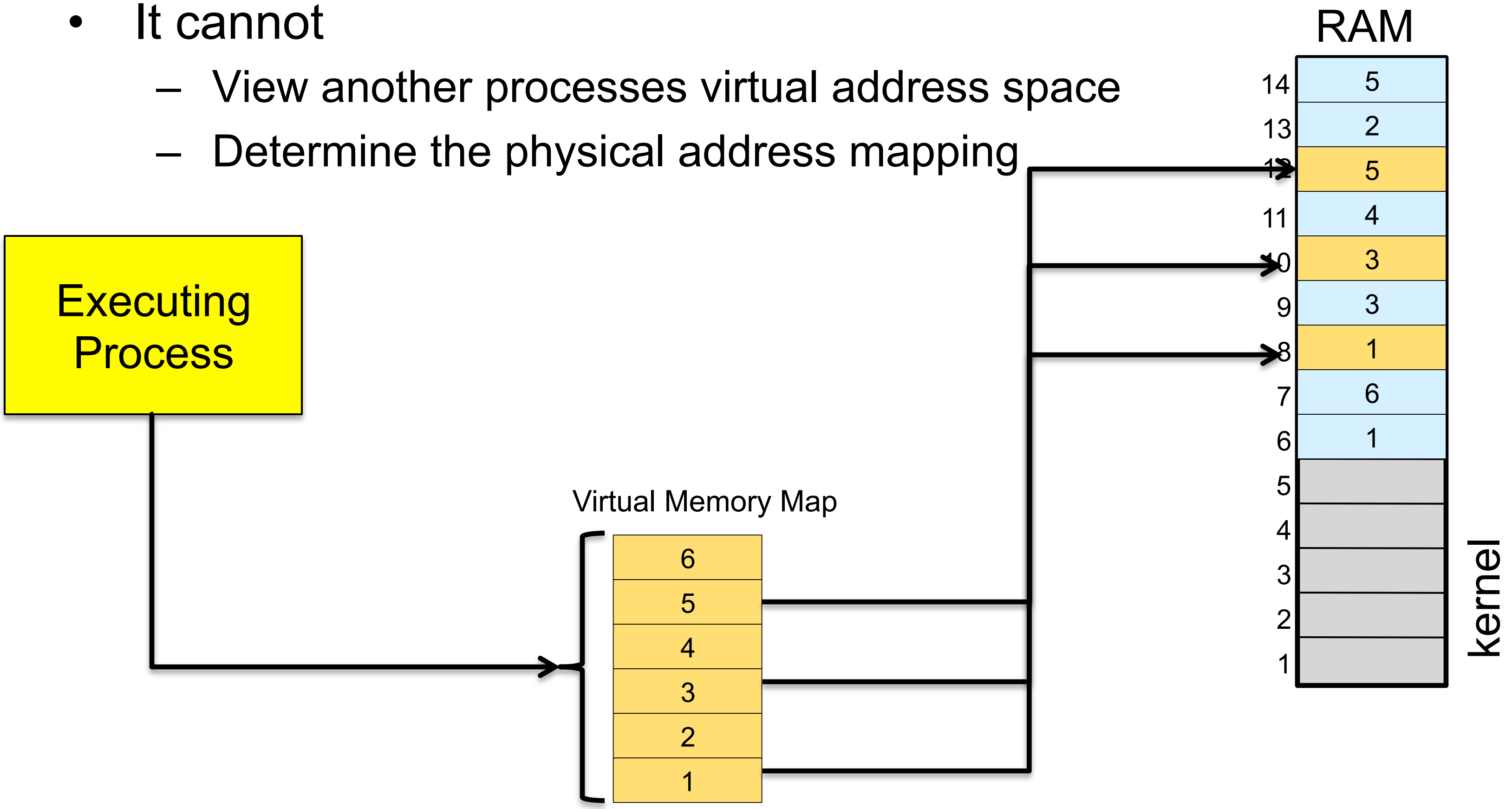
Virtual Memory View

- During execution, each process can only view its virtual addresses,
- It cannot
  - View another processes virtual address space
  - Determine the physical address mapping



Virtual Memory View

- During execution, each process can only view its virtual addresses,
- It cannot
  - View another processes virtual address space
  - Determine the physical address mapping

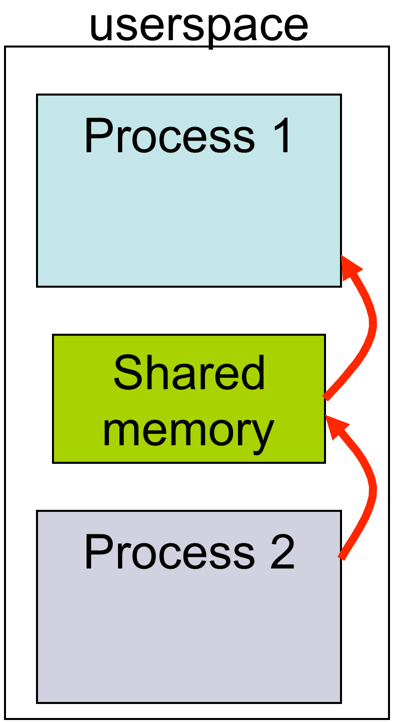


Inter Process Communication

- Advantages of Inter Process Communication (IPC)
  - Information sharing
  - Modularity/Convenience
- 3 ways
  - Shared memory
  - Message Passing
  - Signals

Shared Memory

- One process will create an area in RAM which the other process can access
- Both processes can access shared memory like a regular working memory
  - Reading/writing is like regular reading/writing
  - Fast
- Limitation : Error prone. Needs synchronization between processes



userspace

Process 1

Shared memory

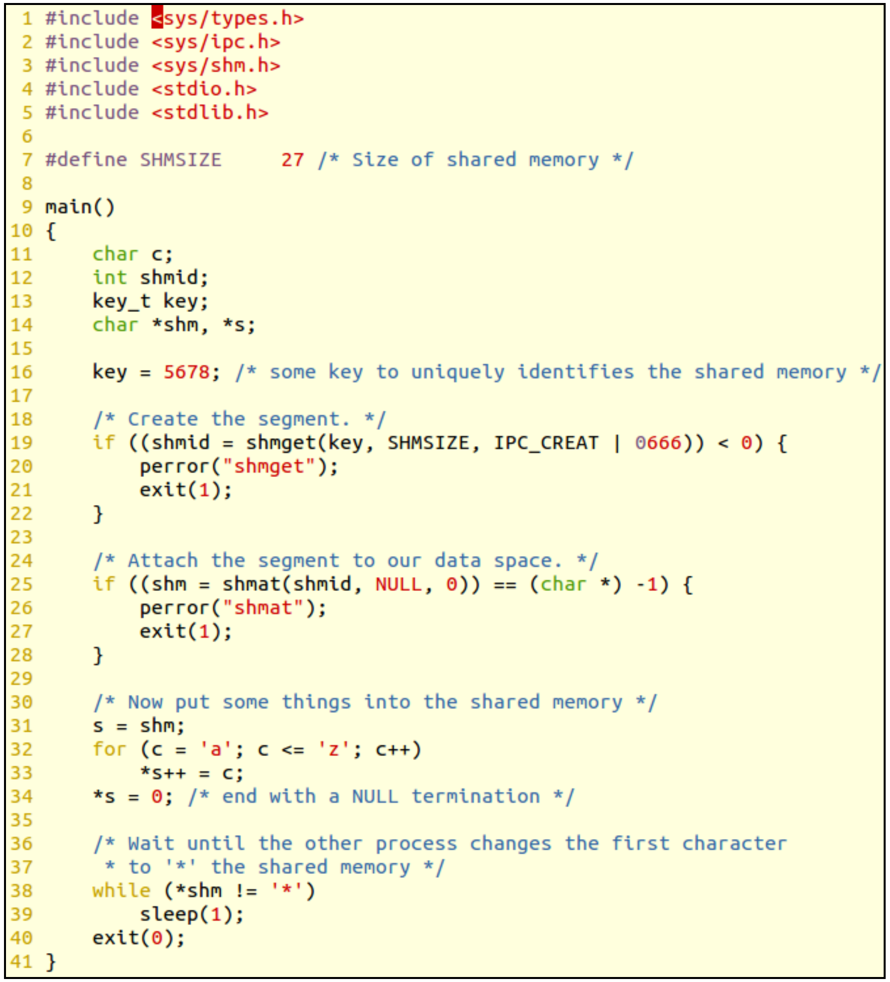
Process 2

Shared Memory in Linux

- **int shmget (key, size, flags)**
  - Create a shared memory segment;
  - Returns ID of segment : shmid
  - key : unique identifier of the shared memory segment
  - size : size of the shared memory (rounded up to the PAGE_SIZE)
- **int shmat(shmid, addr, flags)**
  - **At**tach shmid shared memory to address space of the calling process
  - addr : pointer to the shared memory address space
- **int shmdt(shmid)**
  - **Det**ach shared memory

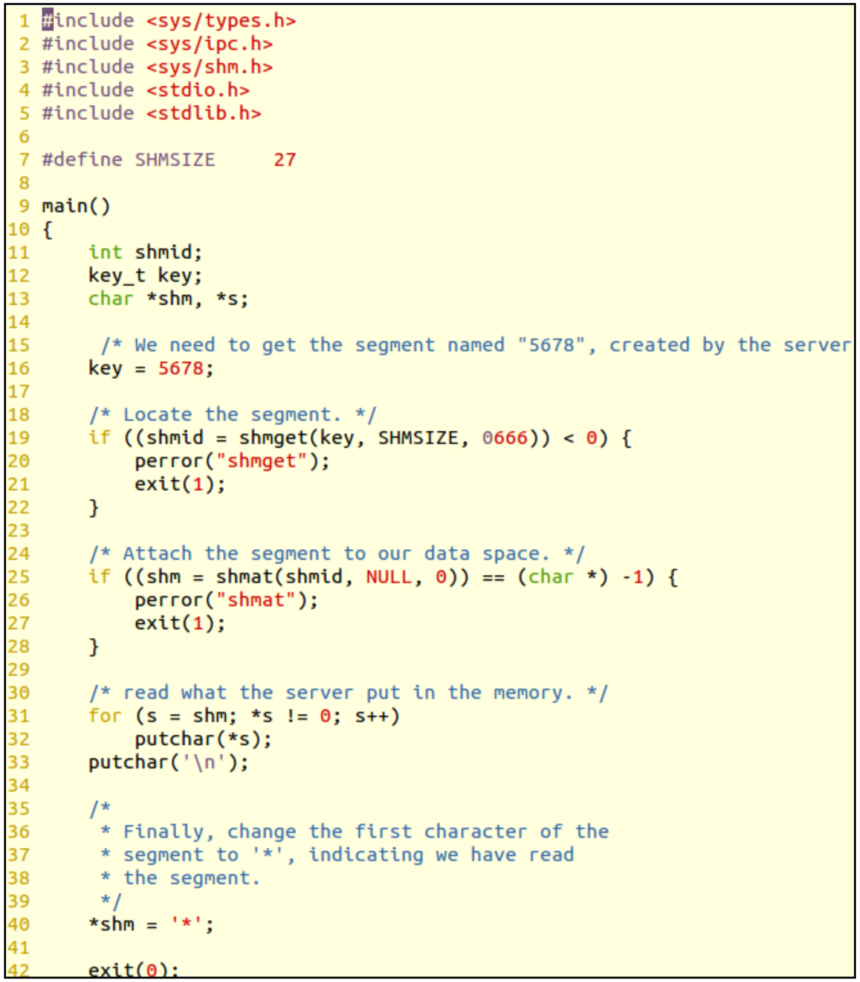
Example

server.c



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

#define SHMSIZE     27 /* Size of shared memory */

main()
{
    char c;
    int shmid;
    key_t key;
    char *shm, *s;

    key = 5678; /* some key to uniquely identifies the shared memory */

    /* Create the segment. */
    if ((shmid = shmget(key, SHMSIZE, IPC_CREAT | 0666)) < 0) {
        perror("shmget");
        exit(1);
    }

    /* Attach the segment to our data space. */
    if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
        perror("shmat");
        exit(1);
    }

    /* Now put some things into the shared memory */
    s = shm;
    for (c = 'a'; c <= 'z'; c++)
        *s++ = c;
    *s = 0; /* end with a NULL termination */

    /* Wait until the other process changes the first character
     * to '*' the shared memory */
    while (*shm != '*')
        sleep(1);
    exit(0);
}
```

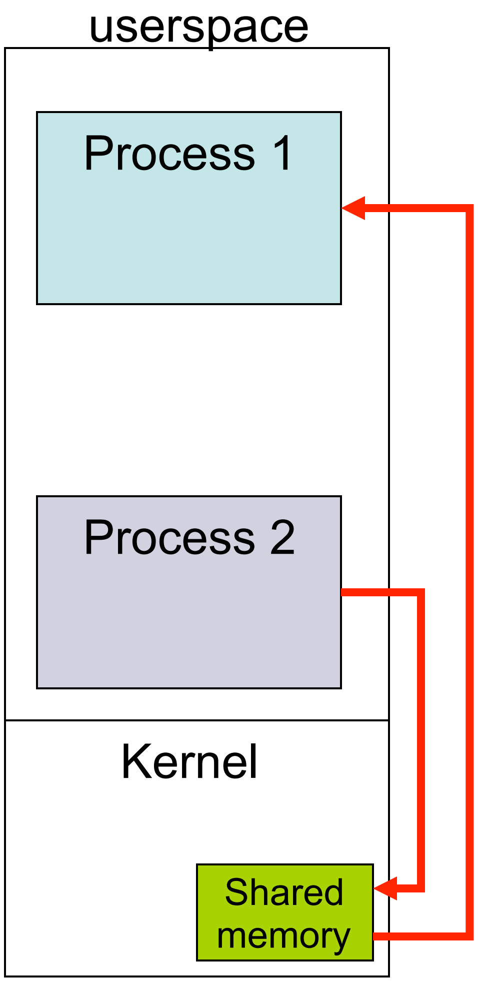
client.c



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

#define SHMSIZE     27

main()
{
    int shmid;
    key_t key;
    char *shm, *s;

     /* We need to get the segment named "5678", created by the server
    key = 5678;

    /* Locate the segment. */
    if ((shmid = shmget(key, SHMSIZE, 0666)) < 0) {
        perror("shmget");
        exit(1);
    }

    /* Attach the segment to our data space. */
    if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
        perror("shmat");
        exit(1);
    }

    /* read what the server put in the memory. */
    for (s = shm; *s != 0; s++)
        putchar(*s);
    putchar('\n');

    /*
     * Finally, change the first character of the
     * segment to '*', indicating we have read
     * the segment.
     */
    *shm = '*';

    exit(0);
```

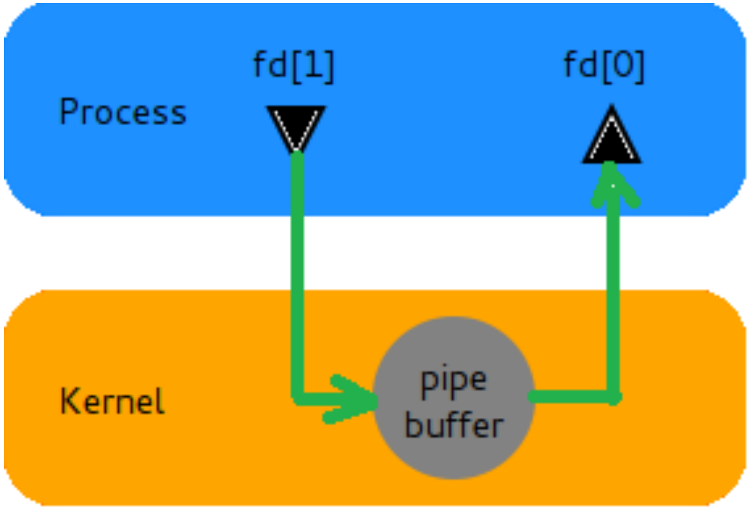
Message Passing

- Shared memory created in the kernel
- System calls such as send and receive used for communication
  - Cooperating : each send must have a receive
- Advantage : Explicit sharing, less error prone
- Limitation : Slow. Each call involves marshalling / demarshalling of information

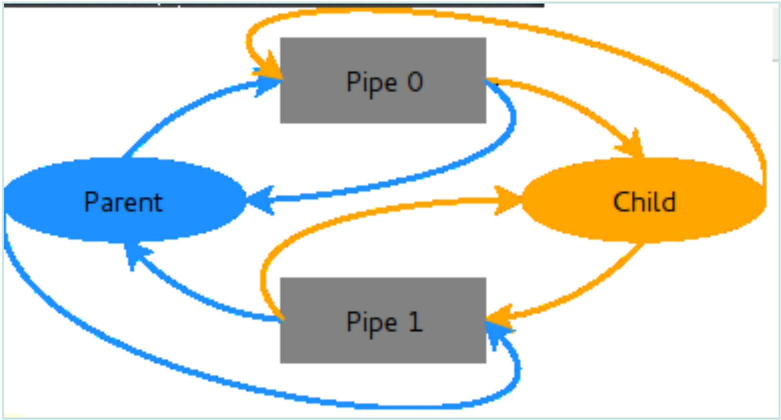


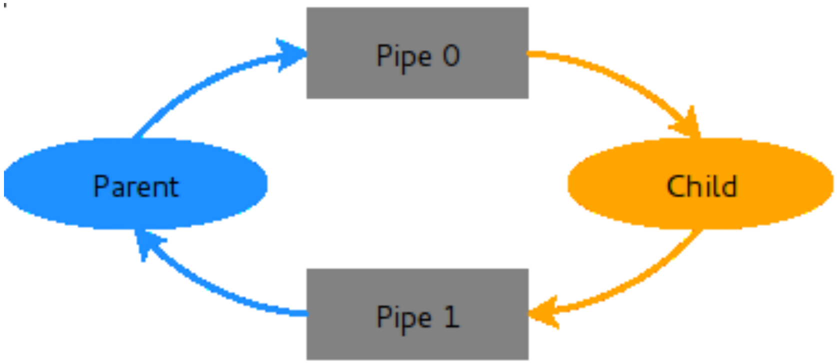
Pipes

- Always between parent and child
- Always unidirectional
- Accessed by two associated file descriptors:
  - fd[0] for reading from pipe
  - fd[1] for writing to the pipe



Pipes for two way communication



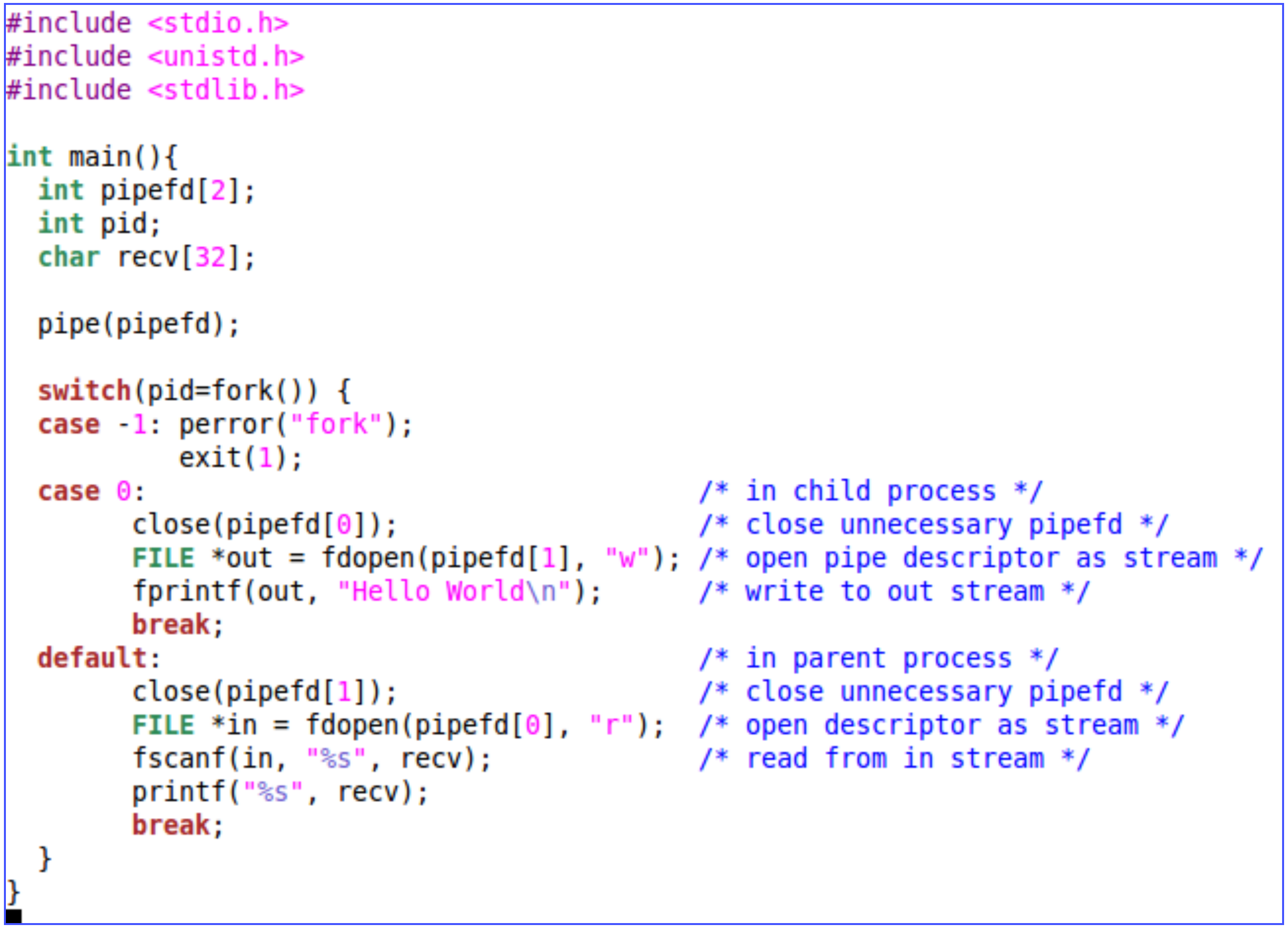


- Two pipes opened

  pipe0 and pipe1
- Note the unnecessary pipes

- Close the unnecessary pipes

Example
(child process sending a string to parent)



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

int main(){
  int pipefd[2];
  int pid;
  char recv[32];

  pipe(pipefd);

  switch(pid=fork()) {
  case -1: perror("fork");
           exit(1);
  case 0:                                   /* in child process */
        close(pipefd[0]);                   /* close unnecessary pipefd */
        FILE *out = fdopen(pipefd[1], "w"); /* open pipe descriptor as stream */
        fprintf(out, "Hello World\n");      /* write to out stream */
        break;
  default:                                  /* in parent process */
        close(pipefd[1]);                   /* close unnecessary pipefd */
        FILE *in = fdopen(pipefd[0], "r");  /* open descriptor as stream */
        fscanf(in, "%s", recv);             /* read from in stream */
        printf("%s", recv);
        break;
  }
}
```

Signals

- Asynchronous unidirectional communication between processes
- Signals are a small integer
  - eg. 9: kill, 11: segmentation fault
- Send a signal to a process
  - kill(pid, signum)
- Process handler for a signal
  - sighandler_t signal(signum, handler);
  - Default if no handler defined

ref : http://www.comptechdoc.org/os/linux/programming/linux_pgsignals.html