CPP

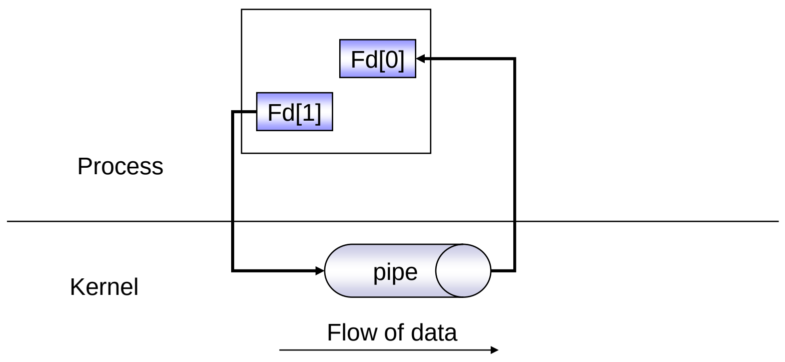
Interprocess communication

Pipe

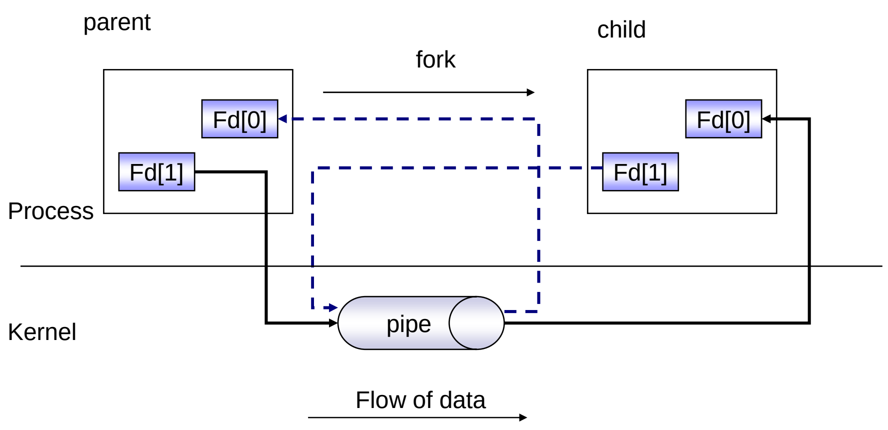
- Pipes are the original form of Unix IPC, although useful for many operations, their fundamental limitation is that they have no name, and can therefore be used only by related processes.
- a pipe is created by the pipe function and provides a one-way (unidirectional) flow of data.

```
#include <unistd.h>
int pipe (int fd[2]);
```

Pipe in single process



Pipe after fork



BBT

popen and pclose

- popen function creates a pipe and initiates another process that either reads from the pipe or write to the pipe.

  #include <stdio.h>

  FILE *popen (const char *command, const char *type);

  int pclose (FILE *stream)

  - If type is r, the calling process reads the standard output of the command.
  - If type is w, the calling process writes to the standard input of the command.

BBT

FIFO

- FIFO stands for first in, first out, and a Unix FIFO is similar to a pipe. It is a one-way (half-duplex) flow of data. But unlike pipes, a FIFO has a pathname associated with it, allowing unrelated processes to access a single FIFO. FIFOs are also called named pipes.

mkfifo

- A FIFO is created by the mkfifo function.

```
#include <sys/types.h>
#include <sys/stat.h>
Int mkfifo (const char *pathname, mode_t mode);
```

- It creates a new FIFO or returns an error of EEXIST if the named FIFO already exists.

FIFO operation

- Once a FIFO is created, it must be opened for reading or writing, using ether the open function, fopen.
- A FIFO must be opened either read-only or write-only. It must not be opened for read-write, because a FIFO is half-duplex.
- A write to a pipe or FIFO always appends the data, and a read always returns what is at the beginning of the pipe or FIFO, If lseek is called for a pipe of FIFO, the error ESPIPE is returned.

BBT

Additional Properties of Pipes and FIFOS

- A descriptor can be set nonblocking in two ways.
  1. The O_NONBLOCK flag can be specified when open is called.
  2. If a descriptor is already open, fcntl can be called to enable the O_NONBLOCK flags.

BBT

Effect of NON_BLOCK on Pipe and FIFO

Current operation	Existing opens of pipe or FIFO	return	
		Blocking (default)	O_NONBLOCK set
Open FIFO read-only	FIFO open for writing	Return OK	Return OK
	FIFO not open for writing	Block until FIFO is opened for writing	Returns OK
Open FIFO write-only	FIFO open for reading	Returns OK	Returns OK
	FIFO not open for reading	Blocks until FIFO is opened for reading	Returns an error of ENXIO

BBT

Effect of NON_BLOCK on Pipe and FIFO –cont.

Current operation	Existing opens of pipe or FIFO	return	
		Blocking (default)	O_NONBLOCK set
Read empty pipe or empty FIFO	Pipe or FIFO open for writing	Block until data is in the pipe or FIFO, or until the pipe or FIFO is no longer open for writing	Return an error of EAGAIN
	Pipe or FIFO not open for writing	Read returns 0(end-of-file	Return 0 (end-of-file)
Write to pipe or FIFO	Pipe or FIFO open for reading	(See next slice)	(See next slice)
	FIFO not open for reading	SIGPIPE generated for thread	SIGPIPE generated

BBT

Write to pipe

- If the number of bytes to write is less than or equal the PIPE_BUF, the write is guaranteed to be atomic. if the number of bytes to write is greater than PIPE_BUF, there is no guarantee that the write operation is atomic.
- The setting of the O_NONBLOCK flag has no effect on the atomicity of writes to a pipe or FIFO. But when a pipe or FIFO is set non-blocking, the return value from write depends on the number of bytes to write and the amount of space currently available in the pipe or FIFO.

BBT

Cont.

- If the number of bytes is less than or equal to PIPE_BUF:
  - If there is room in the pipe or FIFO for the requested number of bytes, all bytes are transferred.
  - If there is not enough room in the pipe or FIFO for the request number of bytes, return is make immediately with an error of EAGAIN.
- If the number of bytes to write is greater than PIPE_BUF:
  - If there is room for at least 1 byte in the pipe or FIFO, the kernel transfers whaterver the pipe or FIFO can hold, and return the number.
  - If the pipe or FIFO is full, return is an error of EAGAIN.

BBT

semaphore

- A semaphore is a primitive used to provide synchronization between various processes or between the various threads in a given process.

BBT

Posix named semaphores

- Named semaphores can be used to synchronize processes or threads.

```
#include <semapthore.h>
sem_t *sem_open (const char *name, int oflag,…
    /* mode_t mode, unsigned int value */ )
int sem_unlink (const char *name)
int sem_close (sem_t *sem)
```

BBT

Memory-based semaphores

- Memory-based semaphores are stored in shared memory and can be used to synchronize processes or threads.

```
int sem_init (sem_t *sem, int pshared, unsigned int value)
int sem_destroy (sem_t *sem)
```

BBT

Semaphore operations

int sem_wait (sem_t *sem)
int sem_trywait (sem_t *sem)
int sem_post (sem_t *sem)
int sem_getvalue (sem_t *sem, int *sval)

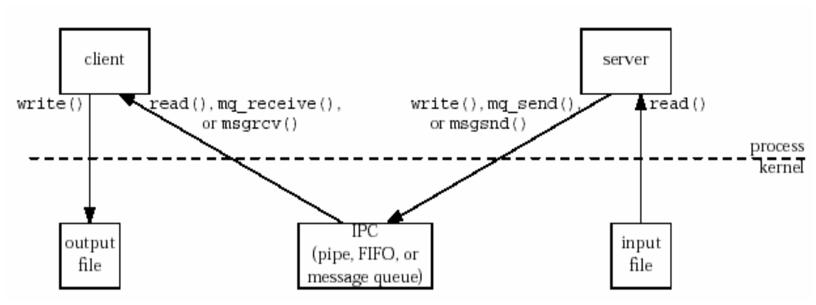
BBT

Share Memory

- Shared memory is the fastest form of IPC available. Once the memory is mapped into the address space of the processes that are sharing the memory region. No kernel involvement occurs in passing data between the processes.
  - Memory –mapped files: a file is opened by open, and the resulting descriptor is mapped into the address space of the process by mmap.
  - Shared memory objects: the function shm_open opens a Posix IPC name, returning a descriptor that is then mapped into the address space of the process by mmap.

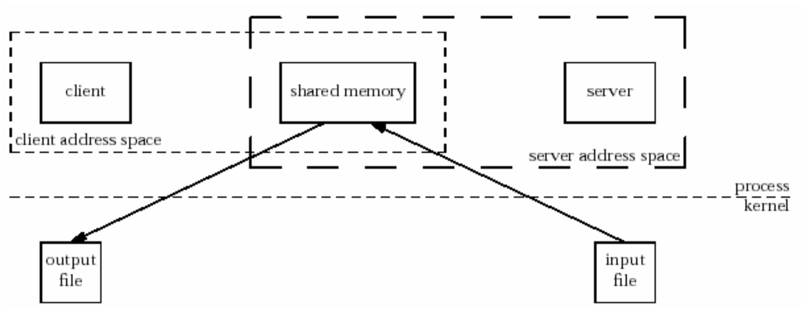
BBT

Flow of file data from server to client.


client
server
write()
read(), mq_receive(), or msgrcv()
write(), mq_send(), or msgsnd()
read()
process
kernel
output file
IPC (pipe, FIFO, or message queue)
input file

BBT

Copying file data from server to client using shared memory.



BBT

mmap

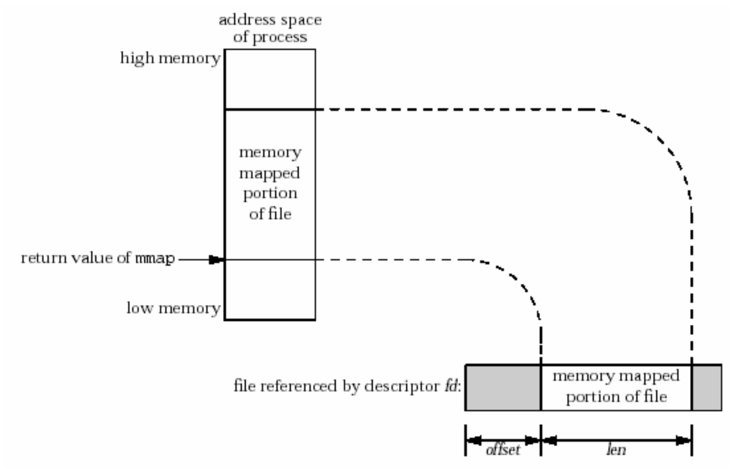
- The mmap function maps either a file or a Posix shareed memory object into the address space of a process. We use this function for three purposes
  - With a regular file to provide memory-mapped I/O
  - With special files to provide anonymous memory mapping
  - With shm_open to provide Posix shared memory between unrelated processes

  #include <sys/mman.h>

  void *mmap (void *addr, size_t len, int prot, int flags, int fd, off_t offset);

BBT

Example of memory-mapped file.



address space of process
high memory
memory mapped portion of file
return value of mmap
low memory
file referenced by descriptor fd:
memory mapped portion of file
offset
len

BBT

Protection of memory

- The protection of the memory-mapped region is specified by the prot argument using the constants below

PROT_READ	Data can be read.
PROT_WRITE	Data can be written.
PROT_EXEC	Data can be executed.
PROT_NONE	Data cannot be accessed

BBT

flags

MAP_SHARED	Changes are shared.
MAP_PRIVATE	Changes are private.
MAP_FIXED	Interpret addr exactly.

  - Portable code should specify addr as a null pointer and should not specify MAP_FIXED.
  - After mmap returns success, the fd argument can be closed. This has no effect on the mapping that was established by mmap.

BBT

munmap

```
#include <sys/mman.h>
int munmap (void *addr, size_t len);
```

BBT

msync

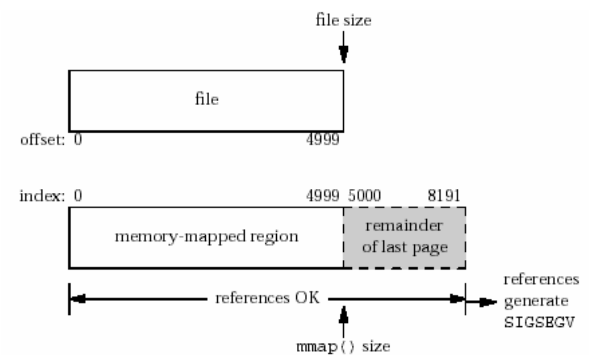
- The kernel’s virtual memory algorithm keeps the memory-mapped file (typically on disk) synchronized with the memory-mapped region in memory, assuming a MAP_SHARED segment. If we modify a location in memory that is memory-mapped to a file, then at some time later the kernel will update the file accordingly. But sometimes, we want to make certain that the file on disk corresponds to what is in the memory-mapped region, and we call msync to perform this synchronization.

```
#include <sys/mman.h>
int msync (void *addr, size_t len, int flags);
```

flags

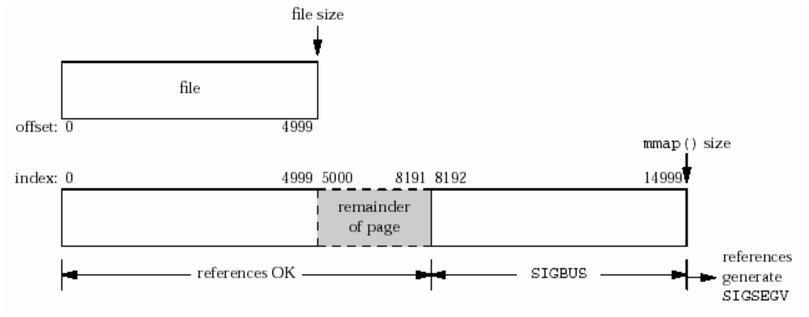
MS_ASYNC	perform asynchronous writes
MS_SYNC	perform synchronous writes
MS_INVALIDATE	invalidate mappings

Memory mapping when mmap size equals file size.



BBT

Memory mapping when mmap size exceeds file size.



BBT

Posix Share memory

- #include <sys/mman.h>
- int shm_open (const char *name, int oflag, mode_t mode)
- int shm_unlink (const char *name)
    - Oflag must contain either O_RDONLY or O_RDWR, and the following flags can also be specified: O_CREAT, O_EXCL, or O_TRUNC.

BBT

ftruncate

- We call ftruncate to specify the size of a newly created shared memory object or to change the size of an existing object.

```
#include <unistd.h>
int ftruncate (int fildes, off_t length)
```

  - For a regular file: If the size of the file was larger than length, the extra data is discarded. If the size of the file was smaller then length, whether the file is changed or its size is increased is unspecified.
  - For a shared memory object: ftruncate sets the size of the object to length.

BBT

fstat

- We can call fstat to obtain information about the object when we open an existing shared memory object.

```
#include <sys/types.h>
#include <sys/stat.h>
int fstat (int fildes, struct stat *buf)
```

stat

```
struct stat{
    mode_t st_mode;
    uid_t st_uid;
    gid_t st_gid;
    off_t st_size;
    …
}
```