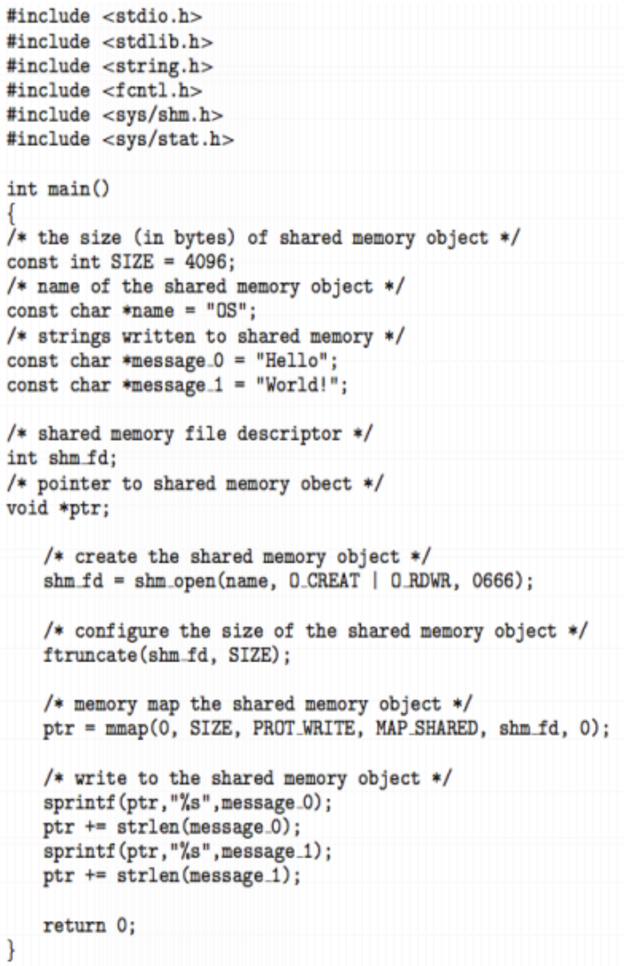
IPC

- Shared memory
- Message passing
  - Direct
    - Connection
  - Indirect
    - No connection
  - Synchronous
    - Wait for completion
      - Synchronous send and receive rendez vous
  - Asynchronous
    - Do not wait
  - Buffering
    - 0 memory
    - Unbounded
    - Bounded

Examples of IPC Systems - POSIX

- POSIX Shared Memory
  - Process first creates shared memory segment
  - shm_fd = sh_open(name, o creat I o RDWR, 0666);
  - Also used to open an existing segment to share it
  - Set the size of the object
    - ftruncate((shm fd, 4096);
  - Now the process could write to the shared memory
  - sprintf(shared memory, “Writing to shared memory”);

IPC POSIX Producer



```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main()
{
/* the size (in bytes) of shared memory object */
const int SIZE = 4096;
/* name of the shared memory object */
const char *name = "OS";
/* strings written to shared memory */
const char *message_0 = "Hello";
const char *message_1 = "World!";

/* shared memory file descriptor */
int shm_fd;
/* pointer to shared memory obect */
void *ptr;

    /* create the shared memory object */
    shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);

    /* configure the size of the shared memory object */
    ftruncate(shm_fd, SIZE);

    /* memory map the shared memory object */
    ptr = mmap(0, SIZE, PROT_WRITE, MAP_SHARED, shm_fd, 0);

    /* write to the shared memory object */
    sprintf(ptr,"%s",message_0);
    ptr += strlen(message_0);
    sprintf(ptr,"%s",message_1);
    ptr += strlen(message_1);

    return 0;
}
```

- C i/0
    - Inx = 6;
    - Printf(“the answer =%d\n”,x)
    - The answer = 6

    - Float y=6.3;

    - Printf(“the answer = %f\n”, y)
    - The answer = 6.3

- Examples of IPC Systems - Mach
    - Mach communication is message based
        - Even system calls are messages
        - Each task get two mailboxes at creation - kernel and notify
        - Only three systems calls needed for message transfer
            - Msg_send(), msg_receive(), msg_rcp()
        - Mailboxes needed for communication, created via
            - Port_allocate

- Examples of IPC systems - windows
    - Message-passing via advanced local procedure call (lpc) facility
        - Only works between processes on the same system
        - Uses ports (like mailboxes) to establishes and maintain communication channels
        - Communication works follows
            - The client opens a handle to the subsystems connection port object
            - The client sends a connection request
            - The server creates two private communication ports and returns the handle to one of them to the client
            - The server creates two private communication ports and returns the handle

- Communications in client server systems
    - Sockets
        - A socket is defined as an endpoint for communication
        - Concatenation of an ip address and port
            - A number included at start of message packet to differentiate network services on a host
        - The socket 161.25.19.8:1625 refers to port 1625 on host 161.25.19.8
        - Communication consists between a pair of sockets
        - All ports below 1024 are well known used for standard services
        - Ip address 127.0.0.1loopback to self
    - Remote procedure calls
        - Remote procedure call (RPC) abstracts procedure calls between processes on networked systems
            - Again uses ports for service differentiation
        - Stubs - client side proxy for the actual procedure on the server
        - The client side stub locates the server and marshalls the parameters
        - The side stub receives this message unpacks the marshalled parameters and performs the procedure on the server
        - On windows stub code compile from specification written in Microsoft interface definition language (MIDL)

    - Data representation handled via external data representation (XDL) to account for different architecture
        - Big-endian and little-endian
    - Remote communication has more failure scenarios than local
        - Messages can be delivers exactly once rather than at most once
    - OS typically provides a rendezvous (or matchmaker)service to connect client and server

- Pipes
    - Acts as a conduit allowing two processes to communicate
    - Issues
        - Is communication unidirectional or bidirectional?
        - In the case of two what communication is it half or full duplex
        - Must there exist a relationship (parent child) between the communicating processes
        - Man the pipes be used over a network
    - Ordinary pipes
        - Cannot be accessed from outside the process that created it. Typically a parent process creates a pipe and uses it to communicate with a child process that it created
    - Named pipes
        - Can be accessed without a parent child relationship

- Ordinary pipes
    - Ordinary pipes allow communication in standard producer consumer style
    - Producer writes to one end (the write end of the pipe)
    - Consumer read from the other end (the read end of the pipe)
    - Ordinary pipes are unidirectional
    - Require parent child relationship between communicating processes
    - windows calls these anonymous pipes
    - See unix and windows code