Libraries to be included = unistd.h, stdio.h, stdlib.h, string.h, fcntl.h, sys/shm.h, sys/stat.h, sys/mman.h → shm 'ifo'

```
int shm_fd = shm_open ( name, O_CREATE | O_RDWR, 0666)
                        const char*
ftruncate (shm_fd, size)
                   int
void* ptr = mmap (0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0)
if (ptr == MAP_FAILED) ..
```

memcpy (ptr, fpo_str, strlen(fpo_str)+1)
const char*

```
char* str = (char*) malloc (8)
memcpy (str, ptr, 8)
```

```
const char* fifo_w = "/tmp/wfifo"
mkfifo (fifo_w, 0666)
int fifo_write = open (fifo_w, OWRONLY)
write ( fifo_write, &i, sizeof (int))
                    i = int
read ( fifo_read, &j, sizeof (int))
close (fifo_write)
```

```
# include : stdio.h, stdlib.h, string.h, unistd.h, sys/types.h, sys/socket.h, sys/un.h
# define SOCK_PATH "unix_socket_example"
struct sockaddr_un serv_addr, cli_addr;
int sock_fd = socket (AF_UNIX, SOCK_DGRAM, 0)
bzero ((char*) & serv_addr, sizeof (serv_addr));
serv_addr.sun_family = AF_UNIX
strcpy (serv_addr.sun_path, SOCK_PATH);
bind (sockfd, (struct sockaddr *)& serv_addr, sizeof (serv_addr))

int file_size;
int n = recvfrom (sockfd, buffer, 256, 0, (struct socketaddr *)& cli_addr, & len);
    bytes received                                                  sizeof (cli_addr)

fwrite (buffer, 1, n, stdout) → byte_received number of bytes to stdout
unlink (SOCK_PATH)
```

```
FILE *fp;
fp = fopen ("test.txt", "r");
fseek (fp, 0, SEEK_END);
int file_size = ftell (fp)
fclose (fp)
sendto (sockfd, & file_size, sizeof (file_size), 0,
                    struct
fp = fopen ("test.txt", "r")
while ( fgets (buffer, 256, fp) != NULL)
fclose (fp)
close (sockfd)
```

defs.h → include any struct at the top
→ define functions from proc.c
→ int getnumprocs ();
→ int getmaxPID();
→ int getprocinfo (int pid, struct processinfo * procinfo);

proc.c → has all functions like allocproc, fork ...
→ if some new variables for a process, do this in allocproc function
→ acquire and release lock.

```
if ( curproc → walk_addr ) :
    np → tf → eip = (uint) (curproc → walk_addr)
                  ↳ start on a function on returning
```

```
int getnumprocs () {
    struct proc* p;
    int count = 0
    acquire (& ptable.lock);
    for (p = ptable.proc ; p < & ptable.proc[NPROC]; p++) {
        if (p → state != UNUSED ) count++;
    }
    release (& ptable.lock);
    return count;
}
int getprocinfo (int pid, struct processinfo * procinfo)
```

user.h :

```
# include "processInfo.h"
int hello ();
int getProcInfo (int pid, struct processInfo* procinfo);
:
```

usys.S

```
SYSCALL (hello)
SYSCALL (getprio)
:
```

proc.h
Include all new variables of a process in proc.h

syscall.c

```
extern int sys_hello (void);
extern int sys_helloYou (void);

[SYS_hello]   sys_hello,
[SYS_helloYou]  sys_helloYou
```

syscall.h

```
# define SYS_hello   22
# define SYS_helloYou  23
```
} → syscall numbers

sysproc.c

```
int sys_hello(void) {
    int ptr
    if (argint (0, &ptr) < 0) return -1;
    char* name = (char*) ptr;
    cprintf ("%s\n", name);
    return 0;
}
int sys_welcomeDone () {
    struct proc* curproc = myproc();
    if (curproc → ret_addr) {
        curproc → tf → eip = (uint)(curproc → ret_addr);
        return 0;
    }
    return -1;
}
```

```
int pipe_in [2]
int pipe_out [2]
pipe(pipe_in)
pipe(pipe_out)
pid = fork()
if (pid == 0):
    close(pipe_in[1])
    dup2(pipe_in[0], 0)  -> stdin
    close(pipe_in[0])
else:
    close()
    write(pipe_in[1], "This is the data \n", 14)
    read(pipe_out[0], buffer, sizeof(buffer));
    waitpid(pid, NULL, 0)
```

```
#include stdlib.h, stdio.h, string.h, unistd.h,
  sys/types.h, sys/ipc.h, sys/msg.h
struct msg_mys { long int msg_type;
                 char some_text[512]; };
msgid = msgget((key_t)14534, 0666 | IPC_CREAT);
msgsnd (msgid, (void*) & some_data, 512, 0);

same in the receiver
msgrcv (msgid, (void*) & some_data, BUF SIZ,
        msg_to_rec, 0);
```

```
#include <netinet/in.h>   stdio.h   stdlib.h
  string.h, sys/socket.h,   unistd.h
#define PORT 800          int opt = 1
struct sockaddr_in address
socklen_t addrlen = sizeof(address)
if server_fd = socket (AF_INET, SOCK_STREAM,0)
setsockopt (server_fd, SOL_SOCKET,
    SO_REUSEADDR | SO_REUSEPORT,
    & opt, sizeof(opt))

address.sin_family = AF_INET
address.sin_addr.s_addr = INADDR_ANY
address.sin_port = htons(PORT)
bind (server_fd, (struct sockaddr*) & address,
    sizeof(address))

listen (server_fd, 3)
new_socket = accept (server_fd,
    (struct sockaddr*) & address,
    & addrlen))

valread = read (new_socket, buffer, 1024-1)
send (new_socket, hello, strlen(hello), 0)
close (new_socket)
close (server_fd)

client => connect (client_fd, (struct sockaddr*)
    & serv_addr,
    sizeof(serv_addr));
```

Remove this in client