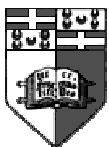
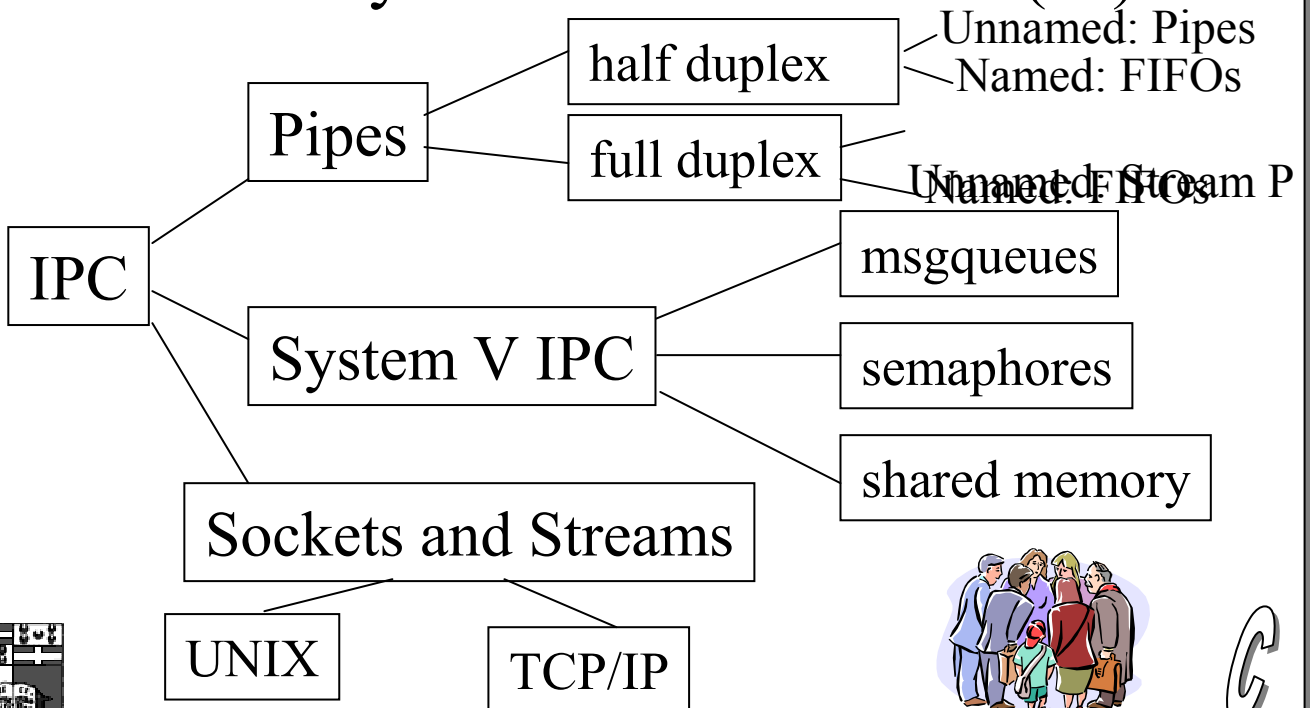




IPC

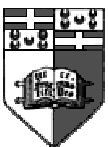
- Inter Process Communication allows different processes to communicate between themselves.
- So far processes could communicate using fork/child inheritance, passing arguments in *exec()* calls, through the file system and using signals.
- There are further structures which allow processes to communicate more efficiently and with more ease (??).





IPC (cont)

- IPC structures provide different flavors of communication.
- Some can only be used between related processes (fork/child). These are the unnamed structures.
- Named structures can be used by anyone having access rights.
- System V IPC structures follow the same access protocol, but some extra form of initial communication is necessary for the processes to use them.
- Different implementations might support one type of IPC and not another. Also some structures are handled differently between implementations.

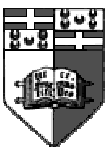


Unix $\frac{1}{2}$ Duplex Unnamed Pipes

- These are the oldest and most widely implemented version of IPC.
- Data can only flow in one direction.
- Pipes must be used in related processes since their identifier is the file descriptor.

```
#include <unistd.h>
int pipe(int fd[2]);
    return -1 on error
```

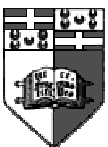
- *pipe()* creates a pipe and places 2 file descriptors inside *fd*. *fd[0]* is opened for reading and *fd[1]* is opened for writing.
- Pipes work in a FIFO fashion thus the output to *fd[1]* is the input to *fd[0]*.





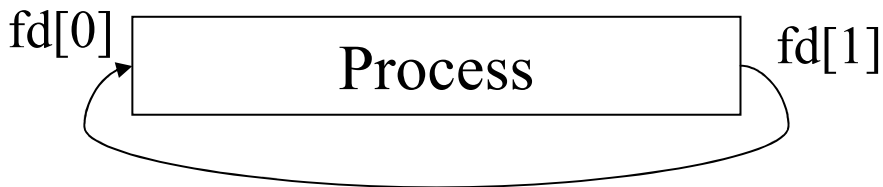
Pipes (cont)

- *fstat()* returns *st_mode* of **FIFO** which can be tested by the **S_ISFIFO(mode_t st_mode)** macro.
- We use normal *read()*, *write()* and *close()* operations to access pipes.
- If we read from a pipe whose write end has been closed, *read()* returns 0, showing an end of file.
- If we write to a pipe whose read end has been closed, *write()* returns -1 with *errno* set to EPIPE. Also before write returns, the signal SIGPIPE is generated.
- The constant PIPE_BUF gives the maximum amount of bytes that can be written in one go without interleaving between different writers.





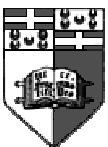
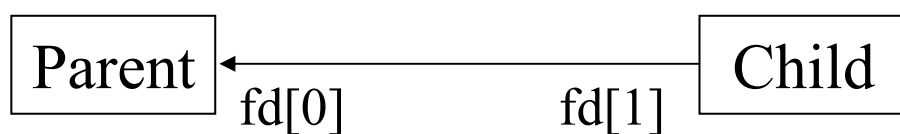
Pipes (cont)



After calling *pipe()*

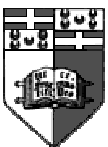
- Within one process, a pipe is useless.
- Yet we normally *fork()* a child process and then close one side in the parent and close the other side in the child.

```
Ex: int fd[2];
    pipe(fd);
    if (fork()==0) // child
        close(fd[0]);
    else // parent
        close(fd[1]);
```



Unix FIFO's: named Pipes

- FIFO's are just like normal files but they behave like pipes with a pathname.
- FIFO's are **full duplex** and more than one process can open a FIFO.
- Other unrelated processes can access FIFO's using the normal *open*, *read*, *write* and *close* system calls.
- To remove a FIFO file, call *unlink()*.
- Normal access file permissions apply. The *stat()* function returns the type as FIFO like pipes. In 'ls' they are marked with 'p'.
- If we write to a FIFO which no process has opened for reading, SIGPIPE is generated.
- When no writer exists, a *read()* returns a 0 for end of file.

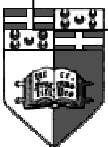




FIFO's (cont)

- *mode* in *mkfifo()* is the same as *mode* in *open()* for file access permissions.
- The constant `O_NONBLOCK` can be passed to *open()* in the *oflag* argument.
- If `O_NONBLOCK` is not specified, the open blocks until there exists a two way connection.
- If `O_NONBLOCK` is specified, an open for read-only returns immediately. An open for write-only returns with error `-1` and *errno* `ENXIO` unless another process has opened the FIFO for reading.

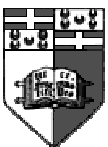
```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char *pathname, mode_t mode);
           returns -1 on error
```





Stream pipes

- Some implementations actually implement standard pipes as bi-directional, others implement them as uni-directional.
 - See pages 478-479 of Stevens to know how to implement unnamed, bi-directional pipes (aka. stream pipes) for specific implementations (`s_pipe`). Just note the differences not the meaning of `socketpair()`.



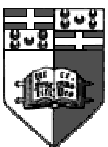


Pipes and FIFO's

Addendum

- When issuing a *read()* call to a pipe or a FIFO in which no data is currently present, the process will block until such data could be read.
- To avoid the process from blocking, we can set the `O_NONBLOCK` flag and then *read()* will immediately return `-1` with `errno` set to `EAGAIN`, whenever no data is available on the pipe or FIFO.
- For FIFO files, we can specify `O_NONBLOCK` in the *open()* command and for pipes we can set it using *fcntl()*.

```
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
int fcntl(fd,F_SETFL,O_NONBLOCK);
return -1 on error
```





Exercises

- Implement a consumer and producer using pipes.
- Now implement the consumer and producer using a FIFO.
- Make a process which passes numbers to another process terminated by the -1 value and let the child return back the results using two pipes. You have made a client server program !!

