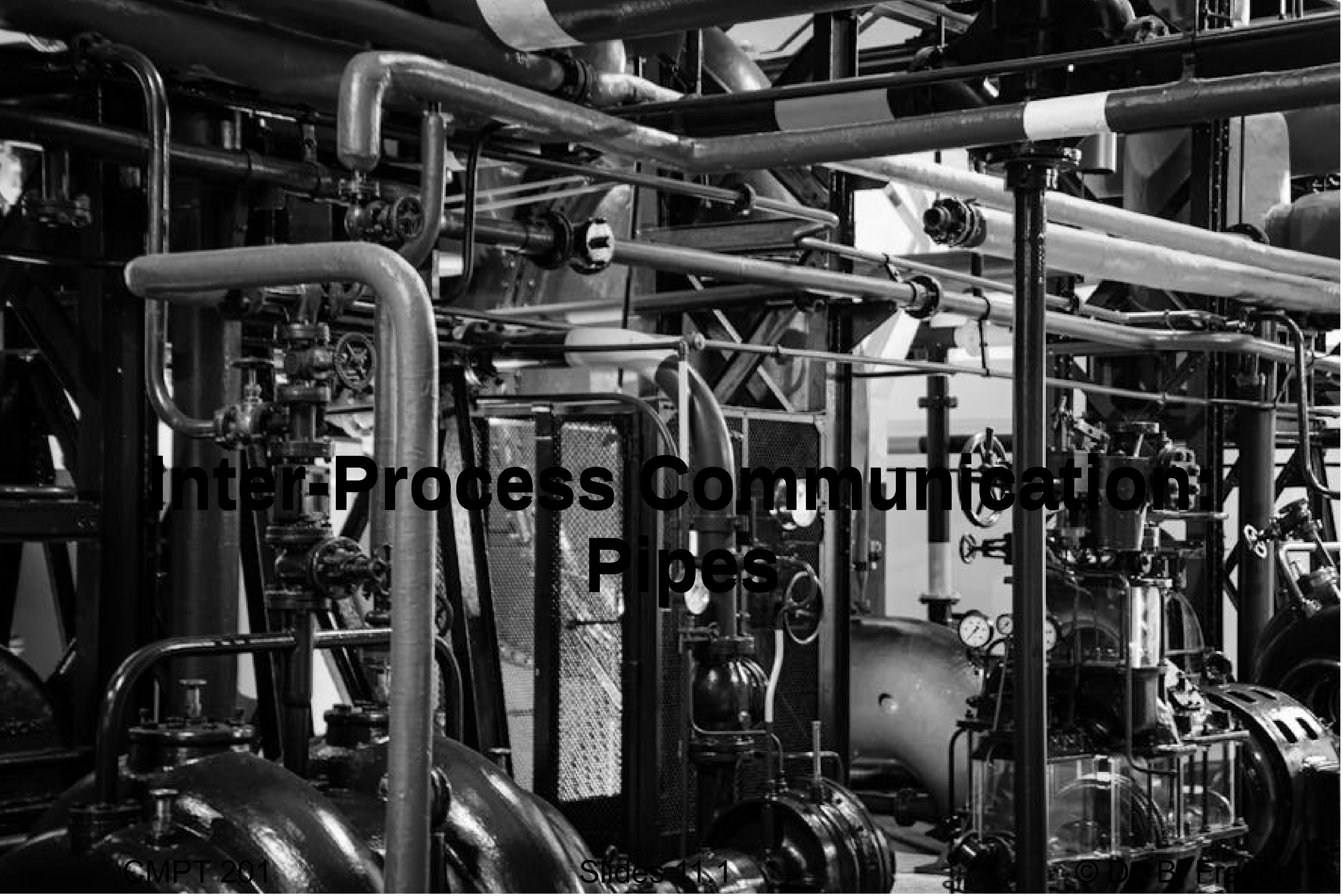

Inter-Process Communication: Pipes

Topics

- How can two processes send data between themselves?
  - What if they are parent-child?
  - What if they are unrelated?
  - What if we want to send full messages, not just bytes?

IPC

- Inter-process communication (IPC)

  ..

  - E.g., UNIX domain socket is an example of this,
- Other facilities:
  - pipes,
  - FIFOs,
  - message queues,
  - memory mapping, and shared memory.

Pipes

Pipe Usage

- We've used shell pipes:
  ```
  ps aux | grep bash
  ```
  - | is a pipe.
  - The output of the first becomes input to the second.
- Can use pipes programmatically:
  ```
  int pipe(int filedes[2])
  ```
    - man 7 pipe
  - ..
    - filedes[0] gives us the..
    - filedes[1] gives us the..

Pipe Details

- A pipe has the following characteristics:
  - ..
  - It is unidirectional:

    ..
  - ..
- A pipe creates file descriptors, so use regular file I/O:
  - non-buffered I/O:
    - read(), write()
  - buffered I/O:
    - fprintf(), fscanf()).

Parent-Child Communication

- A typical use case:
  - ..
- Fork copies file descriptors
  - Both file descriptors (filedes[0] and filedes[1]) available in both parent and child because..
  - Parent parent and child can use pipe to communicate.
- Question: How could we encapsulate this in a module?

Point 1: Different Ends

- Important point 1:
  - ..

    (So each process closes end they don't use)
- E.g., child could write to pipe and parent read from pipe.
  - Parent closes write end: close(filedes[1])
  - Child closes read end: close(filedes[0])
  - Child writes into pipe and parent reads from it.
- Take a look at the example from man pipe.

Point 2: Buffer Size

- Important Point 2: Pipe buffer size
  - ..
- When calling write() with n bytes:

  if n <= PIPE_BUF, ..

  if n > PIPE_BUF, ..

  (other writes maybe interleaved between parts of this write).
  - Details depend on if it's a non-blocking pipe; see man 7 pipe
  - PIPE_BUF == 4096 on Linux.

Point 3: Close all write()

- Important Point 3:

  ..

  - This can be used as a signaling mechanism.
- An example scenario:
  - A parent creates pipe and calls fork()
  - Parent process closes write FD and read()s.
  - Child process closes read FD and write()s its data.
  - Data is exchanged via the pipe
  - ..
  - Once parent has read all data in the pipe's buffer, read() returns 0.
    - Parent then knows child has closed write end.

Duplicating File Pipes

int dup2(int oldfd, int newfd)

- Can redirect another program's input/output to pipes.
  - dup2() system call

    ..

- E.g., Redirect standard output to the write end of the pipe:

  dup2(filedes[1], STDOUT_FILENO);
  - ..

- E.g., Redirect a pipe to the standard input.

  dup2(filedes[0], STDIN_FILENO);
  - Any reads from STDIN are instead read from the read end of the pipe.

Running a Program with Pipes

FILE *popen(const char *command, const char *mode)

It does three things to conveniently run a command:

- ..
- if mode == "r":
  returns a file stream which is connected to the STDOUT of the command.
- if mode == "w":
  returns a file stream which is connected to the STDIN of the command

- Use pclose() to close.

Activity: Pipe to child and back

- Activity:
  modify the example in man pipe as follows:
  - The parent should send a string to the child.
  - The child should send the string back to the parent in upper-case
  - The parent should print out the received string.

FIFOs

FIFO between unrelated processes

- Two or more..
  (parent, child, grandchild)
  - However, unrelated processes can't share a pipe.
  - Instead, they can share a FIFO to communicate with each other.
- ..
  int mkfifo(const char *pathname, mode_t mode)
    - pathname is the name of the FIFO to be created.
    - mode is the permission, same as open().
  - Similar to UNIX domain sockets as it creates a file.
  - Use unlink() to remove a FIFO, just like a file.

Opening a FIFO

- Process only needs to know the FIFO's pathname:
  unrelated processes can share a FIFO.
  - One process creates FIFO with mkfifo()
  - Any processes can use open(), read(), write(), etc. to access.
- A FIFO is still unidirectional and typically for two processes:
  - One process should open it for read and other for write.
  - open() blocks until the other process calls open() as well.

FIFO Activity

- Activity: write two programs:
  - One program should create a FIFO and read a string from it and print it out
  - The other program should write a string to the FIFO and print it out.

POSIX Message Queues

Message Queue

- Message Queue
  - similar to a FIFO, but

    ..

    a message is..
  - man 7 mq_overview
- 5 important functions.
  - mq_open()
  - mq_send()
  - mq_receive()
  - mq_close(), and
  - mq_unlink()

Message Queue: mq_send()

```
int mq_send( mqd_t mqdes,
     const char *msg_ptr, size_t msg_len,
     unsigned int msg_prio);
```

  - Message queue sends structured data using a pointer (msg_ptr) to the structured data.
  - msg_prio determines a priority of the message.
  - The queue is a priority queue,
    i.e.,..
    (and FIFO for the same priority).

- mq_receive() retrieves the oldest highest priority message
  - Gets the whole message at once,
    ..

Summary

- Inter-process communication (IPC):
  - Pipes: Send data between two related processes
  - FIFO: Send data between unrelated processes
  - Message Queue: Send full messages