IPC Overview

- Method to send data between processes
- Multi-language
  - C, C++, Java, etc.
- Multi-platform
  - Linux, Windows
Central server maintains all publishers, subscribers, and message types

Some process publishes message

Central forwards it to all subscribed processes
Connecting to Central

- IPC_connectModule("module name", "central host IP address");

- IPC_disconnect();
```cpp
#include <stdio.h>
#include "ipc/ipc.h"

int main() {
    IPC_connectModule ("testCommander", "localhost");
    printf ("Connected to IPC central\n");
    IPC_disconnect();
    printf ("Disconnected from IPC central\n");
    return 1;
}
```
Connecting to Central

- IPC_connectModule("module name", "central host IP address");

- IPC_connect ("module name");
  - Uses CENTRALHOST environment variable
  - If CENTRALHOST not set, defaults to localhost
```cpp
#include <stdio.h>
#include "ipc/ipc.h"

int main() {

    IPC_connect("testCommander");
    printf("Connected to IPC central\n");
    IPC_disconnect();
    printf("Disconnected from IPC central\n");

    return 1;
}
```
Each message has a name and a format
Format indicates data type to be passed
- “int”
- “double”
- structs: “{int, double}”

IPC_defineMsg ("message name", IPC_VARIABLE_LENGTH,
               "format string");
Publishing a Message

- IPC_publishData ("message name", pointerToData);
```c
#include <stdio.h>
#include "ipc/ipc.h"

int main()
{
    IPC_connectModule ("testCommander", 'localhost');
    printf ("Connected to IPC central\n");
    IPC_defineMsg ("go forward command", IPC_VARIABLE_LENGTH, "double");
    double forwardCommand = 2.0;
    IPC_publishData ("go forward command", &forwardCommand);
    printf ("Sent the command: %f\n", forwardCommand);
    IPC_disconnect();
    printf ("Disconnected from IPC central\n");
    return 1;
}
```
Subscribing to a Message

- Give a handler function which processes the message
- Function has parameters for message data and your own data
- `IPC_subscribeData ("message name", handlerFunction, yourData);`
- `void handlerFunction (MSG_INSTANCE msgInstance, BYTE_ARRAY callData, void* userData);`
Listening for a Message

- `IPC_listen(numMilliseconds);`
  - Blocks until message received or timeout
  - Can handle at most one message

- `IPC_listenClear (numMilliseconds);`
  - Blocks until messages received or timeout
  - Handles multiple messages

- `IPC_listenClear (0);`
  - Handles all pending messages
```c
#include <stdio.h>
#include "ipc/ipc.h"

int main() {
    IPC_connectModule ("testCommander", "localhost");
    printf ("Connected to IPC central\n");
    IPC_defineMsg ("go forward command", IPC_VARIABLE_LENGTH, "double");
    double forwardCommand = 2.0;
    IPC_publishData ("go forward command", &forwardCommand);
    printf ("Sent the command: %f\n", forwardCommand);
    IPC_disconnect();
    printf ("Disconnected from IPC central\n");
    return 1;
}

void forwardCommandHandler (MSG_INSTANCE ref, void* callData,
                            void* userData) {
    double incomingCommand = -1;
    IPCSubscribeData ("go forward command", forwardCommandHandler,
                      &incomingCommand);
    printf ("Subscribed and listening for command\n");
    //listen for maximum 1000 seconds,
    //return when 1 message has been processed.
    IPC_listen (1000000);
    printf ("Received the robot command: %f\n", incomingCommand);
    IPC_disconnect();
    printf ("Disconnected from IPC central\n");
    return 1;
}
```
More Useful Things

- IPC_setMsgQueueLength("message name", queueLength);
  - All incoming messages are queued
  - Upon IPC_listen, oldest message is handled
  - This function sets the maximum number of messages to queue

- Available query/response system
  - See the manual
Practice

- Write two programs using IPC
- Commander sends command as x-y-heading struct
- Controller receives command and replies with int (success or not)
- Commander receives success notification