Hypothetical Part Stamper Machine Design Handouts

- Messaging
  - Linux SYSV IPC MSG
  - PS MSG Message Structure
  - PS Message Protocol
- Task Diagram
- I/O (Hardware Interface)
  - Monitor commands
  - Bit List

Part Stamper Messaging

- From the start, our hero's design has revolved around command messages for communication between the system tasks
- This is a methodology that is supported by all commercial embedded RTOS products
- We will uncover the advantages and disadvantages of this approach as we dig into our hero's design and implement it

Messaging in Linux

- One of the benefits of choosing Linux as a development environment is the multitude of ways to solve a problem
- Our hero chose to stick with the SYSV IPC mechanisms because these more closely map to traditional RTOS features
- The following is an excerpt from the msgop man page…

msgop man page

- To send or receive a message, the calling process allocates a structure that looks like the following
  - struct msgbuf {
    - long mtype; /* message type, must be > 0 */
    - char mtext[1]; /* message data */
  - };
  - but with an array mtext of size msgsz, a non-negative integer value. The structure member mtype must have a strictly positive integer value that can be used by the receiving process for message selection

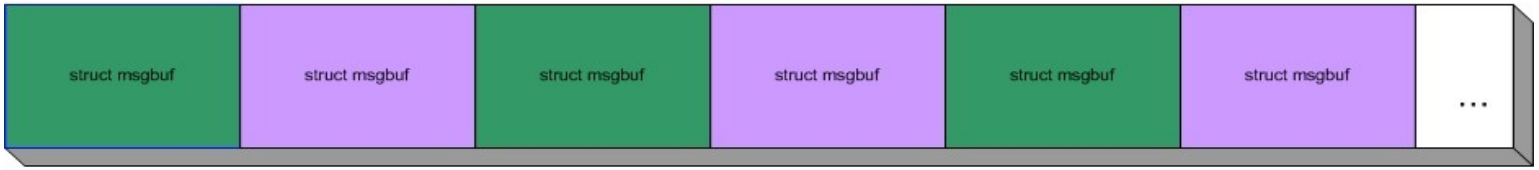
message.h

- #define MSGBUF_SIZE 256
- #define MSGBUF_PAYLOAD_SIZE (MSGBUF_SIZE-sizeof(long))
- Typedef struct {
  - MSG_LIST message;
  - PROC_LIST sender;
  - PROC_LIST receiver;
  - MP_ERROR error;
- } MESSAGE_PACKET;
- typedef struct {
  - MESSAGE_PACKET mp;
  - char data[MSGBUF_PAYLOAD_SIZE-sizeof(MESSAGE_PACKET)];
- } GENERIC_MSG;
- typedef struct {
  - long mtype;
  - GENERIC_MSG gm;

  } msgbuf;

SYSV IPC MESSAGE QUEUE

Msgget returns a handle to a kernel resource, a message queue




```
typedef struct {
   mtype   (long)
   gm      (GENERIC_MSG)
} msgbuf;
```

```
typedef struct {
  mtype          (long)
  gm.mp          (MESSAGE_PACKET)
  gm.data[236]   (char)
} msgbuf
```

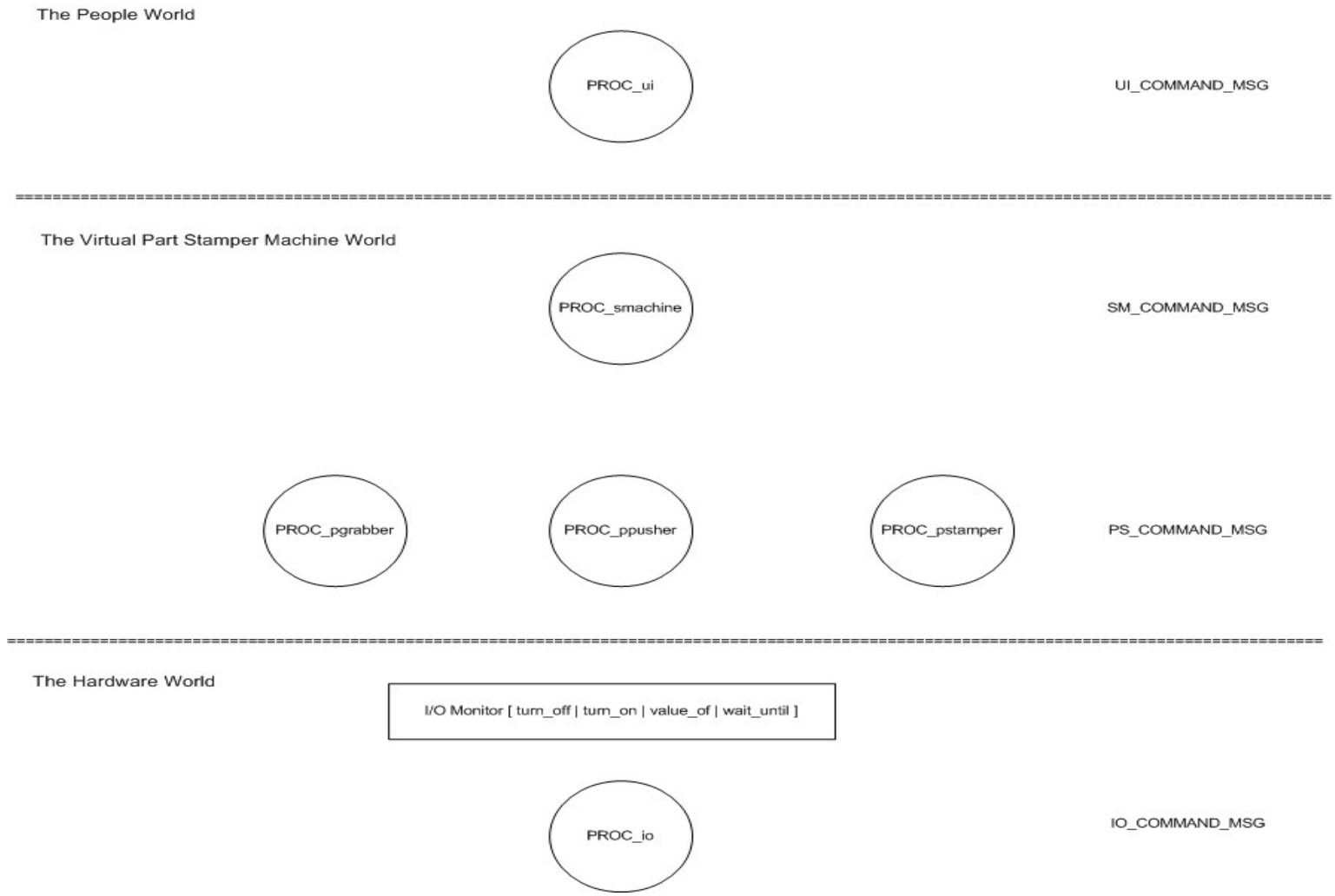
```
typedef struct {
  mtype              (long)              4
  gm.mp.message      (MSG_LIST)          4
  gm.mp.sender;      (PROC_LIST)         4
  gm.mp.receiver;    (PROC_LIST)         4
  gm.mp.error;       (MP_ERROR)          4
  gm.data[2366];     (char)            236
} msgbuf                               (256)
```

Linux :	1.6 % data (memory) overhead	(4 / 256 = 0.0157)
PS Message Protocol :	6.3 % data (memory) overhead	(16/256 = 0.0625)
Total :	7.9 % data (memory) overhead	(20 / 256 = 0.078125)

Semantics of the PS Message Protocol

- Each task in the system has an input command message queue
- All message traffic to a task is routed to the input command message queue
- Every outbound command message generates an inbound reply packet, with fields from original message copied/filled in, indicating the command has completed, either successfully or in error

Task & Message Diagram



The Message Library

- void generic_msg_free(GENERIC_MSG *);
- GENERIC_MSG * generic_msg_init(MSG_LIST);

- GENERIC_MSG * msg_receive(MSG_LIST);
- void msg_send(GENERIC_MSG *);
- void msg_reply(GENERIC_MSG *);

sendcmd Cheat Sheet

```
typedef enum {
        PROC_NONE = -1,
        PROC_pgrabber = 0,
        PROC_ppusher = 1,
        PROC_pstamper = 2,
        PROC_smachine = 3,
        PROC_ui = 4,
        PROC_tcl = 5,
        PROC_main = 6,
        LAST_PROC
} PROC_LIST;
```

```
typedef enum {
        MP_NOERROR,             // 00
        MP_NOMESSAGE,           // 01
        MP_NOSENDER,            // 02
        MP_NORECEIVER,          // 03
        MP_BADCOMMAND,          // 04
        MP_NULLRETURN,          // 05
        MP_UNHANDLEDMSG,        // 06
        MP_BADSTATE,            // 07
        MP_LAST
} MP_ERROR;
```

```
typedef enum {
        PS_COMMAND_MSG   // 00
        SM_COMMAND_MSG   // 01
        UI_COMMAND_MSG   // 02
        PS_GENERIC_MSG   // 03
        LAST_MSG
} MSG_LIST;
```

StateMachine Cheat Sheet

```
typedef enum {
        sm_start,                // 00
        sm_stop,                 // 01
        sm_return_state,         // 02
        sm_reset,                // 03
        sm_io_value_change,      // 04
        sm_cmd_last
} SM_CMD_LIST;

typedef struct {
        SM_CMD_LIST  command;
        int          continuous;
        int          state;
        IO_BIT_LIST  bit;
        int          value;
} SMCM;
```

```
typedef enum {
        state_quiet,             // 00
        state_ehardware,         // 01
        state_grab,              // 02
        state_push,              // 03
        state_stamp,             // 04
        state_stamp_grab,        // 05
        state_last,
        state_nostate = -1
} SM_STATE;
```

Part-Stamper Cheat Sheet

```
typedef enum {
        ps_grab_part,     // 00
        ps_push_part,     // 01
        ps_stamp_part,    // 02
        ps_return_state,  // 03
        ps_reset,         // 04
        ps_cmd_last
} PS_CMD_LIST;

 typedef struct {
        PS_CMD_LIST  command;
        int                  state;
 } PSCM;
```

I/O Monitor Cheat Sheet

```
typedef enum {
        io_turn_off,    // 00
        io_turn_on,     // 01
        io_value_of,    // 02
        io_report,      // 03
        io_reset,       // 04
        io_cmd_last
} IO_CMD_LIST;
```

```
typedef enum {
        cylAretracted,    // 00
        cylAextended,     // 01
        cylAsolenoid,     // 02
        cylBretracted,    // 03
        cylBextended,     // 04
        cylBsolenoid,     // 05
        cylCretracted,    // 06
        cylCextended,     // 07
        cylCsolenoid,     // 08
        airpressure,      // 09
        Estop,            // 10
        IO_LAST
} IO_BIT_LIST;
```