Real-time Systems SMD138

Lecture 8:
Examples of reactive objects
Object graphs

interrupt  →  SYNC  →  ASYNC  →  AFTER  →  write to port

read from port  →  SYNC  →  ASYNC  →  AFTER  →  write to port

interrupt  →  SYNC  →  ASYNC  →  AFTER  →  read from port

interrupt  →  SYNC  →  ASYNC  →  AFTER  →  read from port
A simple counter

typedef struct {
    Object super;
    int val;
} Counter;

#define initCounter(n)    { initObject(), n }

int inc( Counter *self, int arg ){
    self->val = self->val + arg;
}

int reset( Counter *self, int arg ){
    self->val = arg;
}

int sample( Counter *self, int arg ){
    return self->val;
}
An event filter

typedef int (*Predicate)(int);

typedef struct {
    Object super;
    Predicate predicate;
    Object *next;
    Method output;
} Filter;

#define initFilter(predicate,next,output)  
{ initObject(), predicate, (Object*)next, (Method)output }  

int input( Filter *self, int arg ){
    if (self->predicate(arg))
        ASYNC( self->next, self->output, arg );
}
An event filter

```c
int isSmall( int n ) {
    return (n >= 0 && n <= 10);
}

Obj2 obj2 = initObj2( ...);
Filter filter = initFilter( isSmall, &obj2, obj2In );
Obj1 obj1 = initObj1( &filter, input );
```
typedef struct {
    Object super;
    int old1, old2;
    Object *next;
    Method output;
    int k0, k1, k2;
} SignalProc;

#define initSignalProc(next,output,k0,k1,k2) 
    { initObject(), 0, 0, (Object*)next, (Method)output, k0, k1, k2 }

int input( SignalProc *self, int arg ) {
    int val = self->k0*arg + self->k1*self->old1 + self->k2*self->old2;
    ASYNC( self->next, self->output, val );
    self->old2 = self->old1;
    self->old1 = arg;
}
A clock

typedef struct {
    Object super;
    int sec, min, hour;
} Clock;

#define initClock()    { initObject(), 0, 0, 0 }

int tick( Clock *self, int arg ) {
    self->sec++;
    if (self->sec == 60) { self->sec = 0; self->min++; }
    if (self->min == 60) { self->min = 0; self->hour++; }
    AFTER( SEC(1), self, tick, 0 )
}

int sample( Clock *self, CalendarTime *arg ) {
    arg->sec = self->sec; arg->min = self->min; arg->hour = self->hour;
}
Q: Will the clock start oscillating by itself?
A: No...

Clock clock = initClock();

STARTUP( ASYNC(&clock, tick, 0) );

Ignition event!
An on-off clock

typedef struct {
    Object super;
    int sec;
    int enabled;
} OnOffClock;

#define initOnOffClock()    { initObject(), 0, 1 }

int tick( OnOffClock *self, int arg ) {
    if (self->enabled)
        self->sec = self->sec + 1;
    AFTER( SEC(1), self, tick, 0 )
}

int sample( OnOffClock *self, int arg ) { return self->sec; }  

int enable( OnOffClock *self, int en ) { self->enabled = en; }
An on-off clock

No increments!

tick(...)  enable(..., 0)  enable(..., 1)
A different on-off clock

typedef struct {
   Object super;
   int sec, enabled;
} OnOffClock2;

#define initOnOffClock2()    { initObject(), 0, 1 }

int tick( OnOffClock2 *self, int arg ) {
   if (self->enabled) {
      self->val = self->val + 1;
      AFTER( SEC(1), self, tick, 0 )
   }
}

int sample( OnOffClock2 *self, int arg ) {   return self->sec;   }

int enable( OnOffClock2 *self, int en ) {
   self->enabled = en;
   if (en)   ASYNC( self, tick, 0 );
}
A different on-off clock

tick(...)

enable(..., 0)

enable(..., 1)

Ghost tick

Non-existent
A buffer repository

```c
#define SIZE ...

typedef struct {
    Object super;
    int head, tail, size;
    int buf[SIZE];
} Buffer;

#define initBuffer()    { initObject(), 0, 0, 0 }

int bufIn( Buffer *self, int arg ) {
    if (self->size < SIZE) {
        self->buf[self->head] = arg;
        self->head = (self->head + 1) % SIZE;
        self->size++;
    }
    return 0;
}

int bufOut( Buffer *self, int arg ) {
    int val = 0;
    if (self->size > 0) {
        val = self->buf[self->tail];
        self->tail = (self->tail + 1) % SIZE;
        self->size--;
    }
    return val;
}
```
A reactive buffer

```c
#define SIZE ...
typedef struct {
  Object super;
  Object *client;
  Method m;
  int open;
  int head, tail, size;
  int buf[SIZE];
} Buffer;

#define initBuffer(obj,m)  
  { initObject(), obj, m, 0, 0, 0 }

int bufIn( Buffer *self, int arg ) {
  if (self->open)
    ASYNC( self->client, self->m, arg );
  else if (self->size < SIZE) {
    self->buf[self->head] = arg;
    self->head = (self->head+1)%SIZE;
    self->size++;
  }
}

int bufCtrl( Buffer *self, int open ) {
  self->open = open;
  while (open && self->size > 0) {
    ASYNC(self->client,self->m,self->buf[self->tail]);
    self->tail = (self->tail+1)%SIZE;
    self->size--;
  }
}
```
Port objects

Task A:

```
... PORTx |= 0x01;
...
```

Task B:

```
... PORTx &~ 0x02;
...
```

Diagram:
- PORTx
  - Input: read
  - Output: write
Port objects

Task A:

```
... PORTx = PORTx | 0x01;
...
```

Task B:

```
... PORTx = PORTx & 0xfd;
...
```
Port objects

Task A:

```c
... unsigned char tmp1 = PORTx;
PORTx = tmp1 | 0x01;
...
```

Task B:

```c
... unsigned char tmp2 = PORTx;
PORTx = tmp2 & 0xfd;
...
```

Critical sections!
Port objects

Solution: access port via a proxy object!

typedef struct {
    Object super;
    unsigned char *port;
} Proxy;

#define initProxy(port)   { initObject(), port }

int set( Proxy *self, unsigned char bits ) {
    *self->port = *self->port | bits;
}

int clr( Proxy *self, unsigned char bits ) {
    *self->port = *self->port & ~bits;
}
Port objects

Proxy proxy = initProxy(&PORTx);

Task A:
...  
SYNC(&proxy, set, 0x01);
...

Task B:
...  
SYNC(&proxy, clr, 0x02);
...
Contact bounces

Handling = ignoring (if timestamps are too dense)

Unwanted events

Desired events

Application-dependent handling
Filtering interrupts

#define DELTA  MSEC(X)

Time

Recall: global non-object variable
= local state of top-level object

int deBounce( void ) {
    Time now = BASELINE();
    if (now - last_valid >= DELTA) {
        last_valid = now;
        actualHandler();
    } else /* ignore */ ;
}

int actualHandler( void ) { ... }

INTERRUPT( SIG XXX, deBounce() );