16/10/2017

[redacted]

- Ο κώδικας του task μπορεί να διακοπεί από event (σε αντίθεση με το command)
- Όταν γίνει post ένα task ο κώδικας μπαίνει στην ουρά.

- Μου ήρθε ένα μήνυμα event το αντιγράφω σε buffer κάνω post το task τρέχω το μήνυμα. Όταν τελειώσει το μήνυμα ξανατρέχω το task.
- Τα task που έχω γίνει post εκτελούνται σειριακά.
- Αν έχω ένα task που διαβάζει μεταβλητές και έρθει και ένα event που πειράζει αυτές τις μεταβλητές ⇒ Τότε προστατεύω αυτές τις μεταβλητές με την εντολή atomic block.
- Το task πάντα πρέπει να επιστρέψει void.

→ Sensor Reading

- interface Read<val_t>
- nx_... //ανεξάρτητο αρχιτεκτονική

- data (16 bit)

data & 0x07 ⇒ κρατάμε τα 3 τελευταία bits.

→ message:

- Δεν πειράζω απευθείας ~~τα~~ τα αρχεία των μηνυμάτων.

→ Interf. AMSend.

getPayload //Επιστρέφει ένα δείκτη στα δεδομένα αυτού του μηνύμ

→ Interf. Packet

→ " AMPacket

source //Σου λέει ποιος σου έστειλε ένα μήνυμα

SOS ⊙ Πρέπει να ελαχιστοποιήσουμε την πληροφορία που στέλνουμε. (Sending a Message σελ

SOS ⊙ ∀ διαφορετικό απο μήνυμ. θα έχω μια διαφορ. μεταβλ.)

- AMSend.send (unicast/broadcast, ..
- σελ 13 λάθος θέλει sensor_reading_t * data = ...

Scanned by CamScanner

1) .h αρχεία
2) AppC αρχεία

→ Εκκίνηση του Αλγορίθμου

18/10/2017

Τι να κάνω στον κώδικα
προσθέτω
αλλάζω
(π) σβήνω εντολές

- AM_... = active message
- Στα struct θα είναι καλό να χρησ. nx_...
- ~~Θα~~ Θα πρέπει να ελαχιστοποιήσω την πληροφ. όσο μπορώ.
- Αν έχω ένα generic component καλό θα είναι να το μετονομάζω.

AppC.nc

·) Για κάθε τύπο μηνύματος χρειάζομαι
→ AMSenderC(...)
→ AMReceiverC(...)
→ PacketQueueC(SENDER...)...
→ Po >> (RECEIVER...)...

Μέσα στο struct δηλώνω τα στοιχεία του μηνυμ.

- και στα δυο αρχεία .nc βάζω ένα interface
- Το wiring είναι σωστό

~~· Για ουρά προσθέτω στο τέλος:~~

→ 4 χρειάζ.
SRTree.RoutingPacket → ...
>> → ...
>> → ...
>> → ...
>> → ...

→ 2 για ουρά
SRTree.RoutingSendQueue → ...
>> ... → ...

→ SRTreeC.nc

·) ∀ τύπο μηνυμ. → χρειάζομαι ένα Interface
→ interface για τις ουρές

- Να δω πού γίνεται το event → Όνομα.fired() (όνομα του Timer)
→ Όνομα.start
(δηλ. ξεκινάει ο timer).

- implementation{ ...

→ Δεν υπάρχουν global μεταβλητές γιατί κάθε αισθητήρας τρέχει τον ίδιο κώδικα.

~~· Πριν να στείλω~~ ένα

- Εγώ ξέρω ότι στάλθηκε το μήνυμα μόνο όταν μου έρθει το signal ότι στάλθηκε (με το που πάρω αυτό το μήνυμα κάνω το ραδιόφωνο μη απασχολημένο.) π.χ. → RoutingSendBusy / NotifySendBusy } boolean.

↳ True προσπαθώ να στείλω μήνυμα

18/10/2017

- Οι ουρές είναι FIFO.
- curdepth // βάθος κόμβου
- ID == 0 είναι η ρίζα.
- dbg("Boot", "- - - - ...", ...);
  τύπος μηνυμ. ίδιο με printf.

1] .h αρχείο
2] AppC αρχείο
3] SRTreeC.nc
↳ interfaces
↳ μεταβλητές
↳ Booted
signal → ↳ startDone
↳ RoutingMsgTimer.fired
→ sendRoutingTask();
↳ ReceiveRouting Msg
ένα
↳ sendDone
→ receiveRoutingTask
↳ Notify: Task που στέλνει
↳ Notify: Receive

- Η ρίζα όταν ξυπνήσει δεν στέλνει κατευθείαν το μήνυμα (αλλά L207) γιατί οι άλλοι κόμβοι μπορεί να κοιμούνται.
  line
  ↓ για περισσότερους αισθητήρες αυξάνω του timer

- mrpkt → ... // γεμίζω τα πεδία του μηνύμ.
  ← για "global"
- atomic { ... } // Με αυτό τον τρόπο δεν μπορεί να διακοπή ο κώδικας
- call Routing... .setDestination(...) // σε ποιον το στέλνω
  call >> .setPayloadLength(...) // μέγεθος του μηνυμ.
  enqueueDone = call ... // βάζω το μήνυμα ~~στην~~ ουρά

- post sendRoutingTask(); // Τραβάω το μήνυμα από την ουρά και το στέλνω
- radioRoutingSendPkt = call Routing... .dequeue(); // παίρνω το μήνυμα από την ουρά
  μεταβλητή
- sendDone = call ... ≠ if(sendDone == SUCCESS)
  signal

- Ο κώδικας στους event handler πρέπει να είναι σύντομος.
- Προσοχή να χρεισ. memcpy(...).
- ReceiveRoutingTask() ← Ίσως αλλάζω πατέρα
  ++ unicast μηνυμ. στον ~~[illegible]~~ (νέο) πατέρα
- Αν αλλάξω τον πατέρα στέλνω μήνυμα.
- Περιοδικά πράγματα με περιοδικούς timer ++ για πρώτη φορά συναρτήσει με Ho).
- Θα πρέπει να αγνοήσω πολλά πράγματα. Θα πρέπει να είναι σύμφωνα με το TAG.