Αναφορά Τελικής Εργασίας Ενσωματωμένων Συστημάτων

Παπακωνσταντίνου Αντώνιος | tonypap@ece.auth.gr | Α.Ε.Μ: 8977

Πίνακας περιεχομένων

Λίστα Σχημάτων

1. Ανάλυση Λειτουργίας Προγράμματος

Το πρόγραμμα αποτελείται από τρία μέρη:

1.1 #FUNCTIONS:

Οι δηλώσεις των δομών *message_info* και του κυκλικού *buffer cir_tbuf_t* καθώς και οι βασικές λειτουργίες τους *(push/pull/checkIfExists)* και μία γεννήτρια ψευδοτυχαίων μηνυμάτων για τα μηνύματα που θα στέλνει το πρόγραμμα στις άλλες συσκευές.

1.2 #THREAD FUNCTIONS:

Τρεις λειτουργίες οι οποίες αντιστοιχούν στα τρία νήματα του προγράμματος. Όπως είναι ήδη γνωστό, η υλοποίηση πολλαπλών διαδικασιών στον ίδιο χρόνο επιτυγχάνεται μέσω νημάτων. Οι 3 λειτουργίες είναι:

1. -generateMsg(): σε αυτή τη λειτουργία δημιουργούνται τα τυχαία μηνύματα προς αποστολή και τοποθετούνται στον κυκλικό *buffer*, όπως ζητάει η άσκηση, κάθε 1 με 5 λεπτά.

2. -send_messages(): η λειτουργία του προγράμματος ως *client*, όπου στέλνει μηνύματα προς άλλους *server* που συναντάμε. Η αποστολή μηνυμάτων υλοποιείται μέσω *sockets* και για αυτό το λόγο απαιτείται από το σύστημα η *IP* και το *PORT* του αποστολέα. Από τα δεδομένα μας ξέρουμε ότι ο παραλήπτης έχει τη διεύθυνση της μορφής: *10.0.xx.yy.,* όπου *xx* τα πρώτα 2 ψηφία του αριθμού μητρώου και *yy* τα 2 τελευταία ψηφία. Μια χρήσιμη εντολή των συστημάτων UNIX είναι το *netstat*, το οποίο καλείται μέσω της *system*. Έτσι, τρέχοντας μία συγκεκριμένη εντολή *netstat ["netstat -an | grep 10.0. | awk -F \"[:]+\" '{print $5}' | tee netstat.log > /dev/null"]* μπορούμε όχι μόνο να αποσπάσουμε πιθανούς παραλήπτες της μορφής *10.0.xx.yy* που βρίσκονται στην εμβέλεια του δικτύου μας αλλά και να διαβάσουμε το *PORT* στο οποίο "ακούνε". Κάνοντάς το αυτό και αποθηκεύοντας τους πιθανούς παραλήπτες σε ένα *log file* μπορούμε στο επόμενο βήμα να

αναζητήσουμε αν υπάρχει διαθέσιμο μήνυμα στο *buffer* -που αφορά παραλήπτη εντός εύρος δικτύου- και να το στείλουμε μέσω *sockets*. Με αυτό τον τρόπο υλοποιήθηκε.

3. -Server(): τέλος, η λειτουργία του διακομιστή ακούει σταθερά σε ένα *Port (2020)* και αν δεχτεί κάποιο μήνυμα το κάνει *push* μέσα στον κυκλικό *buffer*.

1.3 #MAIN:

1. Εδώ εκτελούνται τα τρία νήματα (με τη χρήση του *pthread*). Κάθε νήμα αντιστοιχίζεται με μία λειτουργία *#THREAD FUNCTION*.

2. Όλα τα *threads* τρέχουν σε ξεχωριστά ατέρμονα *loops*.

3. Κατά τη διακοπή του προγράμματος μέσω *keyboard interruption* καθαρίζει η μνήμη και να κλείνουν όλα τα *sockets*.

2. Screenshots και Φωτογραφίες

- Παρακάτω παραθέτω τα τρία κύρια μέρη της εργασίας όπως τα έχω υλοποιήσει (δεν είναι όλο το πρόγραμμα):

```
28  // ########## FUNCTIONS ##########
29  // Error handling
30  void error(const char *msg)
31  {
32      perror(msg);
33      exit(0);
34  }
35
36  // Message Struct Declaration
37  struct message_info
38  {
39          uint32_t AEM_sender;
40          uint32_t AEM_receiver;
41          unsigned long long int timestamp;
42          char message[256];
43  };
44
45  // Buffer Struct Declaration
46  typedef struct {
47      struct message_info * const buffer;
48      int head;
49      int tail;
50      const int maxlen;
51  } circ_bbuf_t;
52
53  CIRC_BBUF_DEF(my_circ_buf, msg_buffer_size);
54
55  // Push data in buffer
56  int circ_bbuf_push(circ_bbuf_t *c, struct message_info data)
57  {
58      int next;
59
60      next = c->head + 1;  // next is where head will point to after this write.
61      if (next >= c->maxlen)
62          next = 0;
63
64      //if (next == c->tail)  // if the head + 1 == tail, circular buffer is full
65      //    return -1;
66
67      c->buffer[c->head] = data;  // Load data and then move
68      c->head = next;             // head to next data offset.
69      return 0;  // return success to indicate successful push.
70  }
```

Εικόνα 1

```
72  // Pull data from buffer
73  int circ_bbuf_pop(circ_bbuf_t *c, struct message_info *data)
74  {
75      int next;
76
77      if (c->head == c->tail)  // if the head == tail, we don't have any data
78          return -1;
79
80      next = c->tail + 1;  // next is where tail will point to after this read.
81      if(next >= c->maxlen)
82          next = 0;
83
84      *data = c->buffer[c->tail];  // Read data and then move
85      c->tail = next;              // tail to next offset.
86      return 0;  // return success to indicate successful push.
87  }
88
89  // Random Message Generator
90  static char *rand_string(char *str, size_t size)
91  {
92          srand(time(NULL));
93      const char charset[] = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJK .";
94      if (size) {
95          --size;
96          for (size_t n = 0; n < size; n++) {
97              int key = rand() % (int) (sizeof charset - 1);
98              str[n] = charset[key];
99          }
100         str[size] = '\0';
101     }
102     return str;
103 }
104
105 // Convert message to String
106 char* rand_string_alloc(size_t size)
107 {
108      char *s = malloc(size + 1);
109      if (s) {
110          rand_string(s, size);
111      }
112      return s;
113 }
```

Εικόνα 2

```
115  // Check for Redundancies
116  int checkIfExists (struct message_info k)
117  {
118          int b;
119          for(b=0; b<msg_buffer_size; b++){
120                  if(messages_bucket[b].AEM_sender == k.AEM_sender
121                          && messages_bucket[b].AEM_receiver == k.AEM_receiver
122                          && strcmp(messages_bucket[b].message, k.message) == 0
123                          && messages_bucket[b].timestamp == k.timestamp){
124                          return 1;
125                  }
126          }
127          return 0;
128  }
```

Εικόνα 3

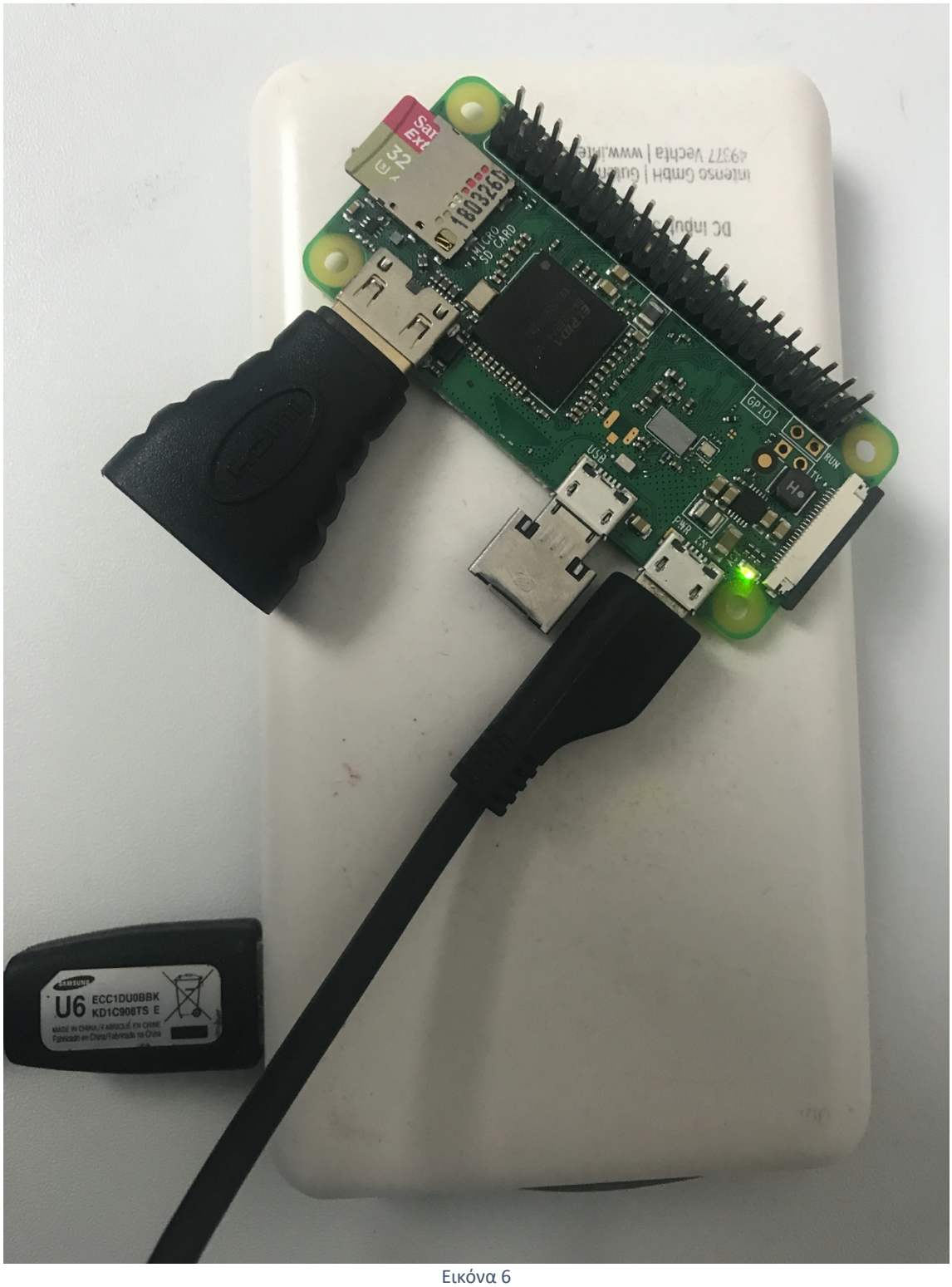
```
130  // ######## THREAD FUNCTIONS ########
131  // Create a random message based on the criteria given
132  void *generateMsg()
133  {
134          for(;;){
135                  const int msg_size = 10; // define msg size
136
137                  char to_send[500] = ""; // temp msg variable to send
138
139                  char* msg = rand_string_alloc(msg_size); // create random msg string
140
141                  struct message_info a_message1;
142
143                  // Initialize the a_message value for test purposes
144                  strcpy(a_message1.message, msg);
145                  a_message1.AEM_sender = 8977;
146                  int r = (rand() % 500) + 8501;
147                  a_message1.AEM_receiver = r; //make it random 8500-9000
148                  a_message1.timestamp = (unsigned long long int)time(NULL);
149
150                  // Generate temp message
151                  char str[1000];
152                  sprintf(str, "%d", a_message1.AEM_sender);
153
154                  strcat(to_send, (char*) str);
155                  strcat(to_send, "_");
156
157                  sprintf(str, "%d", a_message1.AEM_receiver);
158                  strcat(to_send, (char*) str);
159
160                  strcat(to_send, "_");
161
162                  sprintf(str, "%llu", a_message1.timestamp);
163                  strcat(to_send, str);
164
165                  strcat(to_send, "_");
166                  strcat(to_send, (char*) a_message1.message);
167                  printf("%s", to_send);
168
169                  // Push newly created message in the bucket
170                  circ_bbuf_push(&my_circ_buf, a_message1);
171                  isEmpty=0;
172                  printf("\nNEW MESSAGE CREATED\n");
```

Εικόνα 4

```
427  // ########## MAIN ##########
428  int main(int argc, char *argv[])
429  {
430          int NTHREADS=3;
431          pthread_t threads[NTHREADS];
432          int thread_args[NTHREADS];
433          int rc, i;
434
435          //Run server as a Thread
436          pthread_create(&threads[0], NULL, Server, NULL);
437
438          //Run generate messages as a Thread
439          pthread_create(&threads[1], NULL, generateMsg, NULL);
440
441          //Run send messages from client to other server as a Thread
442          pthread_create(&threads[2], NULL, send_messages, NULL);
443
444          pthread_join(threads[0], NULL);
445          pthread_join(threads[1], NULL);
446          pthread_join(threads[2], NULL);
447
448          //Free the variables here??
449          return 0;
450  }
```

Εικόνα 5

- Τέλος παραθέτω το Raspberry Pie σε λειτουργία:



Εικόνα 6

3. Σχόλια

- Η εργασία με όλα τα απαραίτητα αρχεία υπάρχει και στο GitHub σε προσωπικό Repository στην παρακάτω διεύθυνση:

https://github.com/tonypapakon/Embedded-Real-Time-Systems

- Εάν υπάρχουν απορίες για την εργασία μπορείτε να μου στείλετε μήνυμα στο e-mail μου, το παραθέτω στην πρώτη σελίδα.