

Εργαστήριο Λειτουργικών Συστημάτων: Περιγραφή της Εργασίας

Περίληψη

Στην εργασία αυτή θα υλοποιήσουμε κάποιες επεκτάσεις του πυρήνα ενός απλού λειτουργικού συστήματος, που θα το ονομάσουμε TINYOS-3. Σε αντίθεση με τα λειτουργικά συστήματα που χρησιμοποιούμε, το δικό μας σύστημα δε θα εκτελείται κατευθείαν στο υλικό του συστήματος, αλλά σαν διεργασία ενός άλλου λειτουργικού συστήματος, του Linux, θα χρησιμοποιεί προσομοιωμένους πόρους, δηλαδή θα εκτελείται σε μια (απλή) virtual machine. Αντί δηλαδή το TINYOS-3 να διαχειρίζεται τους υλικούς πόρους ενός υπολογιστή, θα διαχειρίζεται τους πόρους (χρόνο CPU, μνήμη, περιφερειακά κλπ.) που του δίνονται από το ξένο ΛΣ (host OS). Πέρα από αυτή τη διαφορά, το TINYOS-3 θα εκτελεί τις βασικές λειτουργίες ενός πραγματικού λειτουργικού συστήματος.

Περιεχόμενα

1	Εισαγωγικά	2
2	Γρήγορη αρχή	2
3	Εξοικείωση με τα αρχεία του TINYOS-3	3
3.1	Αρχεία για την εκτέλεση του TINYOS-3	3
3.2	Αρχεία που περιέχουν τον προσομοιωτή του υπολογιστή μας	4
3.3	Αρχεία με προγράμματα-εφαρμογές του TINYOS-3	4
3.4	Αρχεία συστήματος του TINYOS-3	5
3.5	Βοηθητικά αρχεία	5
4	Ο προσομοιωμένος υπολογιστής	6
5	Περιγραφή της εργασίας σας	8
6	Πρώτο μέρος	8
6.1	Πολυνηματικές διεργασίες	8
6.2	Δρομολόγηση νημάτων με τον αλγόριθμο Multilevel Feedback Queue	9
7	Δεύτερο μέρος	10
7.1	Pipes	10
7.2	Sockets	11
7.3	Παρουσίαση πληροφοριών του συστήματος	11
8	Εργαλεία ανάπτυξης που θα σας είναι χρήσιμα	11
8.1	Απαραίτητα εργαλεία	11
8.2	Χρήσιμα αλλά όχι απαραίτητα εργαλεία	12
8.3	Integrated Development Environments—IDE	12

10 Παραδοτέα της εργασίας

14

10.1 Παραδοτέο του κάθε μέρους	14
10.2 Καθυστερημένη παράδοση	14
10.3 Εξέταση του κάθε μέρους	14

1 Εισαγωγικά

Το TINYOS-3 υποστηρίζει ανεξάρτητες διεργασίες, τις οποίες δρομολογεί μοιράζοντας το χρόνο των πυρήνων του επεξεργαστή. Το TINYOS-3 έχει απόλυτο έλεγχο πάνω στις διεργασίες αυτές. Τις δημιουργεί, τις δρομολογεί όσο αυτές εκτελούνται, και τις τερματίζει όταν ολοκληρωθούν.

Το TINYOS-3 θα διαχειρίζεται τους πόρους ενός ιδεατού, προσομοιωμένου υπολογιστή. Θα είναι γραμμένο στη γλώσσα C. Για συμβατότητα με το Linux, θα χρησιμοποιήσουμε τον compiler του GNU Project, τον `gcc`. Επίσης, θα γράψουμε τον κώδικά μας σε μια σύγχρονη έκδοση της C, το λεγόμενο Standard C11. Το standard αυτό είναι απόλυτα συμβατό με την “κλασσική” C (το λεγόμενο standard C89) αλλά προσφέρει κάποιες χρήσιμες επεκτάσεις.¹

- Ο πυρήνας του TINYOS-3 θα είναι ένα απλό πρόγραμμα.
- Τα ((προγράμματα)) του TINYOS-3 δε θα είναι χωριστά αρχεία, αλλά εκτελέσιμες συναρτήσεις του προγράμματος του TINYOS-3. Με άλλα λόγια, στο TINYOS-3, ο κώδικας όλων των προγραμμάτων υπάρχει μόνιμα στην κύρια μνήμη.
Σημείωση: Η ύπαρξη του κώδικα όλων των προγραμμάτων στη μνήμη είναι συχνή σε *embedded* λειτουργικά συστήματα, όπως τα λειτουργικά συστήματα παιχνιδιομηχανών (πχ. SONY PlayStation), τα λειτουργικά συστήματα αυτομάτου ελέγχου (βιομηχανικών εργαλείων, οχημάτων, κλπ.), επικοινωνιών (κινητά τηλέφωνα, routers, δορυφόροι), κλπ.
- Ένας ακόμη περιορισμός του TINYOS-3 είναι η (μη-)διαχείριση της μνήμης. Για την ευκολία μας θα χρησιμοποιήσουμε τη βιβλιοθήκη συστήματος του Linux (`malloc`, `free`, κλπ.)
- Ένας τρίτος περιορισμός θα είναι στη διαχείριση E/E. Παρόλο που ο πυρήνας του TINYOS-3 υλοποιεί κλήσεις E/E, αυτές είναι χαμηλού επιπέδου. Μας λείπει μια βιβλιοθήκη συστήματος που θα μας έδινε τις γνωστές μας ευκολίες (`printf` κλπ), οπότε θα χρησιμοποιήσουμε και πάλι τη βιβλιοθήκη συστήματος του Linux για E/E (πχ. για `printf`), κάνοντάς την να συνεργαστεί με το TINYOS-3.

2 Γρήγορη αρχή

Στο URL `courses https://github.com/vsamtuc/tinyos3.git` θα βρείτε τον κώδικα της τωρινής έκδοσης του TINYOS-3. Μπορείτε να τον κατεβάσετε σε ένα directory και να τον τρέξετε με τις εξής εντολές

```
% git clone https://github.com/vsamtuc/tinyos3.git
% cd tinyos3
% touch .depend
% make
...
... (several lines of output)
```

¹Δείτε στο διαδίκτυο για λεπτομέρειες.

```
...
% ./mtask 1 0 1 1
FMIN = 34      FMAX = 44
*** Booting TinyOS
[T]      0 has arrived
[E]      0 is eating
[T]      0 is thinking
[.]      0 is leaving
*** TinyOS halted. Bye!
%
```

Αν πάνε όλα καλά, είναι ώρα να δημιουργηθεί το documentation.

```
% make doc
....
....
finished...
```

και να οδηγήσετε τον browser στο αρχείο `./doc/html/index.html`.

Πήγαν όλα καλά? Συγχαρητήρια! Τώρα είστε έτοιμοι να αρχίσετε την εργασία. Αν κάτι δεν πήγε καλά, συμβουλευτείτε τους βοηθούς ή τον διδάσκοντα.

3 Εξοικείωση με τα αρχεία του TINYOS-3

Αν δείτε τα περιεχόμενα του directory μετά την εκτέλεση της εντολής `make`, θα βρείτε μέσα του ένα μεγάλο αριθμό από αρχεία. Για να μπορέσετε να ξεκινήσετε τη δουλειά, θα σας βοηθήσει η παρακάτω περιγραφή του τί είναι το κάθε αρχείο. Τα αρχεία ανήκουν στις παρακάτω κατηγορίες.

3.1 Αρχεία για την εκτέλεση του TINYOS-3

Τα τερματικά του TINYOS-3 υλοποιούνται από το πρόγραμμα `terminal`. Για να υποστηρίξετε π.χ. 2 τερματικά, πρέπει να ανοίξετε 2 διαφορετικά παράθυρα τερματικών στον υπολογιστή σας και μέσα σε κάθε παράθυρο να εκτελέσετε το πρόγραμμα `terminal`, με όρισμα τον αριθμό του τερματικού. Για παράδειγμα, αν χρησιμοποιείτε Linux Ubuntu θα μπορείτε να κάνετε το εξής:

```
% gnome-terminal -e "./terminal 0"
% gnome-terminal -e "./terminal 1"
```

Οι παραπάνω εντολές θα δημιουργήσουν 2 νέα παράθυρα. Το κάθε παράθυρο αντιστοιχεί σε ένα τερματικό. Μέσα στο κάθε παράθυρο θα είναι τυπωμένο με κόκκινο φόντο το μήνυμα “DISCONNECTED”, που σημαίνει ότι δεν υπάρχει κάποιο πρόγραμμα που να είναι συνδεδεμένο στα τερματικά αυτά.

Τώρα, μπορείτε να γράψετε (στο αρχικό σας παράθυρο) το παρακάτω:

```
% ./bios_example5
```

Θα δείτε αμέσως και τα δύο τερματικά να συνδέονται, και να εμφανίζουν το ίδιο μήνυμα:

Type a line:

Μπορείτε να γράψετε κάτι και το κάθε τερματικό θα το τυπώσει στην οθόνη και αμέσως θα σας ζητήσει να τυπώσετε άλλη μια γραμμή (αλλά μετά από 3 γραμμές το πρόγραμμα τερματίζει).

Απαραίτητα για τη λειτουργία των τερματικών είναι 8 “ειδικά αρχεία” (πρόκειται για named pipes και όχι για κανονικά αρχεία, που θα τα περιγράψουμε στο μάθημα), τα `connn`, `kbdnn`, όπου $n = 0 \dots 3$. Τα αρχεία αυτά τα δημιούργησε η εντολή `make` και είναι απαραίτητα για τη λειτουργία των terminals.

Σημείωση: Όσοι από εσάς προτιμάτε, όπως εγώ, να δουλεύετε σε τερματικό αντί σε γραφικά παράθυρα, μπορείτε να χρησιμοποιήσετε το πρόγραμμα `tmux` του Linux

3.2 Αρχεία που περιέχουν τον προσομοιωτή του υπολογιστή μας

Πρόκειται για 2 αρχεία το `bios.c` και το `bios.h` εκ των οποίων το δεύτερο θα πρέπει να το διαβάσετε οπωσδήποτε (περιγράφεται και στη συνέχεια της εκφώνησης). Τα αρχεία αυτά συνδέονται στα εκτελέσιμα του `tinyos-3` κατά το `compilation`.

Επίσης υπάρχουν τα 5 παραδείγματα `bios_example1` ως και `bios_example5` (και τα αντίστοιχα `.c` αρχεία) που περιέχουν κάποια πολύ απλά δοκιμαστικά για τον προσομοιωτή.

3.3 Αρχεία με προγράμματα-εφαρμογές του `TINYOS-3`

Τα αρχεία αυτά συνδέονται κατά το `compilation` με το `tinyos-3` και τον προσομοιωτή και δημιουργούν δύο εκτελέσιμα: το `mtask` και το `tinyos_shell`. Αυτά είναι και τα αρχεία που θα εκτελείτε πιο συχνά απ' όλα, για να δοκιμάζετε την υλοποίησή σας.

`tinyos_shell`. Αυτό το πρόγραμμα είναι μια πολύ απλοϊκή υλοποίηση ενός `shell` από το οποίο ο χρήστης μπορεί να εκτελέσει διάφορα απλά προγράμματα που τυπώνει στη γραμμή εντολών.

Το πρόγραμμα αυτό δέχεται δύο ορίσματα στη γραμμή εντολών, τον αριθμό των πυρήνων της προσομοίωσης και τον αριθμό των τερματικών. Για παράδειγμα, η εντολή

```
./tinyos_shell 4 2
```

Ξεκινά μια προσομοιωμένη μηχανή με 4 πυρήνες και 2 τερματικά.

Για να το δοκιμάσετε, πριν ακόμη υλοποιήσετε τίποτε, δοκιμάστε το παρακάτω παράδειγμα.

```
linux% ./tinyos_shell 4 2
*** Booting TinyOS with 1 cores and 0 terminals
Starting tinyos shell
Type 'help' for help, 'exit' to quit.
% help
This is a simple shell for tinyos.

You can run some simple commands. Every command takes a
only **integer** arguments. The list of commands and the
number of arguments for each command is shown by
typing 'ls'.
```

When you are tired of playing, type 'exit' to quit.

```
%
```

`mtask`. Όταν εκτελείται το πρόγραμμα αυτό, το `tinyos-3` ξεκινά και μια προσομοίωση του προβλήματος των Φιλοσόφων που Γευματίζουν, όπως το περιγράψαμε στο μάθημα, με πολλαπλές διεργασίες. Το πρόγραμμα δέχεται 4 ορίσματα: τον αριθμό των πυρήνων του επεξεργαστή, τον αριθμό των τερματικών, τον αριθμό των φιλοσόφων και τον αριθμό των “γύρων” που κάνουν.

```

% mtask 1 0 5 1
FMIN = 31      FMAX = 41
*** Booting TinyOS
[T] . . . .      0 has arrived
[E] . . . .      0 is eating
[T] . . . .      0 is thinking
[.] . . . .      0 is leaving
. [T] . . . .    1 has arrived
. [E] . . . .    1 is eating
. [T] . . . .    1 is thinking
. [.] . . . .    1 is leaving
. . [T] . . .    2 has arrived
. . [E] . . .    2 is eating
. . [T] . . .    2 is thinking
. . [.] . . .    2 is leaving
. . . [T] . .    3 has arrived
. . . [E] . .    3 is eating
. . . [T] . .    3 is thinking
. . . [.] . .    3 is leaving
. . . . [T] .    4 has arrived
. . . . [E] .    4 is eating
. . . . [T] .    4 is thinking
. . . . [.] .    4 is leaving
*** TinyOS halted. Bye!

```

Αν αναρωτιέστε πώς είναι δυνατόν να εκτελούνται τα προγράμματα αυτά, αφού ακόμη δεν έχετε υλοποιήσει την εργασία σας, διαβάστε παρακάτω.

3.4 Αρχεία συστήματος του TINYOS-3

Ερχόμαστε τώρα στα αρχεία που αφορούν το ίδιο το TINYOS-3. Αυτά θα τα περιγράψουμε εδώ κάπως πιο αναλυτικά, αλλά σίγουρα θα πρέπει να διαβάσετε και τον κώδικά τους με πολλή προσοχή.

tinyos.h Το αρχείο αυτό το συζητήσαμε ήδη παραπάνω. Θα πρέπει να διαβάσετε προσεκτικά τα σχόλιά του, για να μπορέσετε να υλοποιήσετε την εργασία.

tinyoslib.h, tinyoslib.c Τα δύο αυτά αρχεία περιλαμβάνουν τη σύνδεση του TINYOS-3 με τη βιβλιοθήκη της C στο Linux. Συγκεκριμένα, περιλαμβάνουν τον κώδικα που μας επιτρέπει να χρησιμοποιούμε τις `printf`, `scanf`, `getline` και τις άλλες βιβλιοθήκες για τύπων στην οθόνη και διάβασμα από το πληκτρολόγιο, μέσα από τις διεργασίες του TINYOS-3. Δεν είναι απαραίτητο να διαβάσετε τον κώδικά τους, αλλά αν το κάνετε, ψάξτε στο διαδίκτυο την τεκμηρίωση των ρουτινών που καλούν, και ειδικά των `fopencookie`, `setbuf`, `__fsetlocking`.

kernel_*.c Αυτά είναι τα αρχεία που περιέχουν την τρέχουσα υλοποίηση του TINYOS-3 και τα οποία καλείστε να επεκτείνετε.

3.5 Βοηθητικά αρχεία

Τέλος, 4 πολύ σημαντικά βοηθητικά αρχεία:

Makefile Το αρχείο αυτό περιέχει όλη την πληροφορία που χρειάζεστε για να μεταφράσετε το πρόγραμμά σας. Βεβαιωθείτε ότι στο περιβάλλον ανάπτυξης που χρησιμοποιείτε θα χρησιμοποιήσετε αυτό το αρχείο ακριβώς (εκτός αν είστε έμπειροι και ξέρετε τι κάνετε).

Ειδικότερα, βεβαιωθείτε ότι κατά το compiling χρησιμοποιείτε τα παρακάτω flags του compiler

-pthread Ο προσομοιωμένος υπολογιστής βασίζεται στα pthreads.

-D_GNU_SOURCE Κάποιες συναρτήσεις βιβλιοθήκης αλλάζουν συμπεριφορά χωρίς αυτή τη σημαία.

-std=C11 Χρησιμοποιήστε απαραίτητα τη σύγχρονη έκδοση της C.

-fno-builtin-printf Δε θέλουμε να χρησιμοποιήσουμε την ενσωματωμένη (και βελτιστοποιημένη) υλοποίηση της printf.

-lpthread -lrt -lm κατά τη σύνδεση (linking) χρειάζονται αυτές οι βιβλιοθήκες.

util.h, util.c Μέσα στα δύο αυτά αρχεία περιέχεται κάποιος πολύ χρήσιμος κώδικας που σας συμβουλεύω ανεπιφύλακτα να χρησιμοποιήσετε:

- Μια υλοποίηση για διπλά-συνδεδεμένες λίστες που είναι εξαιρετικά απλή στη χρήση και πολύ γρήγορη.
-

validate_api.c Το αρχείο αυτό περιέχει μια σειρά από unit tests για όλα τα system calls που σας ζητείται να υλοποιήσετε. Για παράδειγμα, μπορείτε να δοκιμάσετε τον κώδικα που κατεβάσατε, τρέχοντας

```
% ./validate_api -c 1,2,4,16 -t 0,1,2 basic_tests
```

και να δείτε να εκτελούνται μια σειρά από tests. Μπορείτε εύκολα να προσθέσετε τα δικά σας tests στο αρχείο αυτό, ή να δημιουργήσετε ένα δικό σας αρχείο με tests.

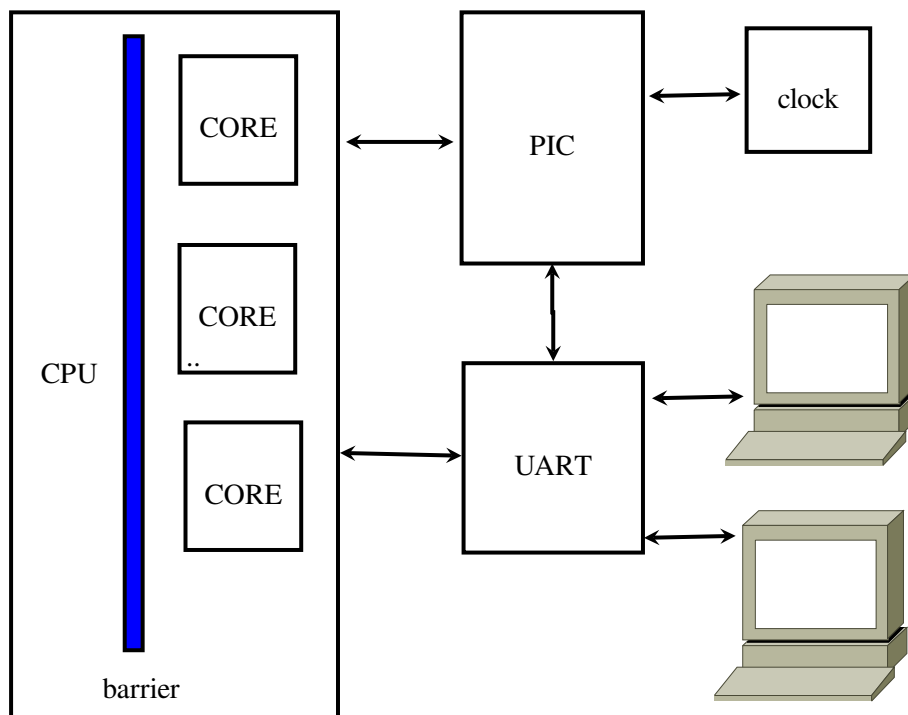
4 Ο προσομοιωμένος υπολογιστής

Το TINYOS-3 διαχειρίζεται τους πόρους ενός προσομοιωμένου υπολογιστή, που υλοποιείται στο αρχείο `bios.c`. Ο υπολογιστής αυτός προσφέρει ένα πολύ απλό API για πολύ χαμηλού επιπέδου πρόσβαση στους πόρους του. Το API αυτό υλοποιείται από firmware εκτός του tinyos, που ονομάζεται BIOS—Basic Input Output System.

Τα μέρη αυτά περιγράφονται παρακάτω.

CPU Η CPU μπορεί να έχει από 1 ως 128 πυρήνες. Μέσα σε κάθε πυρήνα, η συνάρτηση `cpu_core_id()` επιστρέφει τον αύξοντα αριθμό του πυρήνα. Η συνάρτηση `cpu_cores()` επιστρέφει τον αριθμό των πυρήνων στη CPU. Κάθε πυρήνας διαθέτει ξεχωριστό interrupt vector και μπορεί να δεχθεί interrupts κάθε είδους. Κάθε πυρήνας μπορεί να απενεργοποιήσει τα interrupts ανεξάρτητα από τους άλλους, με τη ρουτίνα `cpu_disable_interrupts()` και να τα επανενεργοποιήσει με την εντολή `cpu_enable_interrupts()`.

Interrupts Κάθε πυρήνας μπορεί να ορίσει ρουτίνες που θα εκτελεστούν κατά τη λήψη interrupt, με κλήση της συνάρτησης `cpu_interrupt_handler(...)`. Όταν έρθει κάποιο και πριν κληθεί ο αντίστοιχος interrupt handler, η προηγούμενη κατάσταση των interrupts σώζεται και τα interrupts απενεργοποιούνται (αν ήταν ενεργά), οπότε ο interrupt handler μπορεί να εκτελεστεί χωρίς διακοπή. Ο interrupt handler μπορεί να επανενεργοποιήσει τα interrupts αν το επιθυμεί. Ωστόσο, κατά την επιστροφή του interrupt handler, αποκαθίσταται η κατάσταση των interrupts που σώθηκε κατά τη λήψη της διακοπής.



Σχήμα 1: Σχηματικό διάγραμμα του προσομοιωμένου υπολογιστή που εκτελεί το TINYOS-3

barrier Οι πυρήνες μπορούν να συγχρονιστούν μέσω ενός hardware barrier. Κάθε πυρήνας καλεί τη συνάρτηση `cpu_core_barrier_sync()` και σταματά ώσπου να καλέσουν όλοι οι πυρήνες. Όταν καλέσουν τη συνάρτηση όλοι οι πυρήνες, ξεκινούν όλοι και πάλι.

Inter-core interrupt Υπάρχει ένα ειδικό interrupt το οποίο επιτρέπει σε έναν πυρήνα να στείλει interrupt σε έναν άλλο—ή στον εαυτό του. Αυτό γίνεται με κλήση της συνάρτησης `cpu_ici(...)`.

Programmable Interrupt Controller (PIC) Αυτό είναι το σύστημα που κατευθύνει εξωτερικά interrupts (από τα περιφερειακά) στους πυρήνες.

clock Το PIC χρησιμοποιεί το clock για να υλοποιήσει timers, έναν για κάθε πυρήνα. Ο κάθε πυρήνας μπορεί να ορίσει ένα διάστημα (σε msec) το οποίο, όταν εξαντληθεί θα προκαλέσει το interrupt ALARM. Η συνάρτηση `bios_set_timer()` ενεργοποιεί έναν timer με μια νέα τιμή (καταργώντας ενδεχόμενη ήδη μέτρηση χρόνου) ενώ η συνάρτηση `bios_cancel_timer()` απενεργοποιεί τον timer.

Universal Asynchronous Receiver-Transmitter (UART) Το UART ελέγχει μέχρι 4 σειριακές θύρες και επιτρέπει στους πυρήνες να διαβάζουν και να γράφουν δεδομένα σε αυτές, με τις `bios_read_serial()` και `bios_write_serial()` αντίστοιχα, ένα byte τη φορά. Η συνάρτηση `bios_serial_ports()` επιστρέφει τον αριθμό των ενεργών serial ports. Για κάθε serial port, το UART διαθέτει δύο εσωτερικούς buffer, έναν για δεδομένα που έρχονται από τα περιφερειακά και έναν για δεδομένα που κατευθύνονται στα περιφερειακά.

Όταν μια κλήση της `bios_read_serial` αποτύχει επειδή δεν υπάρχουν δεδομένα στον buffer του UART, η `bios_read_serial` επιστρέφει 0. Στην περίπτωση αυτή, το UART μπαίνει σε κατάσταση ειδοποίησης: θα στείλει ένα interrupt `SERIAL_RX_READY` μόλις εμφανιστούν δεδομένα στον αντίστοιχο buffer.

Αντίστοιχα, αν στον buffer εξόδου δεν υπάρχει χώρος, η συνάρτηση `bios_write_serial` επιστρέφει 0 και το UART μπαίνει σε κατάσταση ειδοποίησης: θα στείλει ένα `SERIAL_TX_READY` μόλις δημιουργηθεί κενό στον αντίστοιχο buffer.

Τέλος, το UART διαθέτει και ένα μηχανισμό timeout: αν σε κάποια θύρα δεν υπάρχει δραστηριότητα από κάποιο πυρήνα για ένα χρονικό διάστημα (περίπου 300 msec), το UART στέλνει το αντίστοιχο interrupt (ή και τα δύο) στη CPU

Με τη συνάρτηση `bios_serial_interrupt_core(...)` η CPU μπορεί να καθορίσει τον πυρήνα που θα λάβει τα interrupts, χωριστά για κάθε σειριακή θύρα. Αρχικά, τα interrupts για όλες τις σειριακές θύρες κατευθύνονται στον πυρήνα 0.

5 Περιγραφή της εργασίας σας

Ο σκοπός σας σε αυτή την εργασία είναι να υλοποιήσετε μια σειρά από επεκτάσεις στον πυρήνα του TINYOS-3. Οι επεκτάσεις αυτές είναι οι παρακάτω:

1. Θα πρέπει να μετατρέψετε τις διεργασίες του TINYOS-3 σε πολυνηματικές. Για το σκοπό αυτό θα πρέπει να υλοποιήσετε τα system calls που υπάρχουν στο `tinysos.h`.
2. Θα πρέπει να υλοποιήσετε έναν scheduler προτεραιοτήτων, με βάση τον αλγόριθμο Multilevel Feedback Queues που είδαμε στην τάξη. Θα βρείτε έναν πιο λεπτομερή ορισμό του στη Wikipedia και θα καλυφθεί και στη θεωρία.
3. Θα πρέπει να υλοποιήσετε δύο μηχανισμούς επικοινωνίας διεργασιών, pipes και sockets.
4. Τέλος, θα πρέπει να υλοποιήσετε ένα system call που θα επιτρέπει σε ένα πρόγραμμα χρήστη να τυπώνει στο πληροφόριες για τις παρούσες διεργασίες.

Η παράδοση των παραπάνω θα γίνει σε 2 μέρη και θα ακολουθήσει προφορική εξέταση της κάθε ομάδας μετά από κάθε παράδοση. Οι ημερομηνίες παράδοσης αναφέρονται στην 1η σελίδα.

Για να εκτελέσετε την εργασία σας θα χωριστείτε σε ομάδες των 3 ατόμων. Οι ομάδες αυτές θα δηλωθούν στο courses μέχρι την προθεσμία που θα ανακοινωθεί. Μετά την πάροδο της προθεσμίας, καμία προσθήκη στις ομάδες δε θα γίνει δεκτή. Εργασίες από μικρότερες ομάδες θα βαθμολογηθούν το ίδιο με αυτές των 3 ατόμων.

Οι ομάδες σας θα πρέπει να αρχίσουν να συνεργάζονται άμεσα, διότι οι προθεσμίες της παράδοσης είναι σχετικά στενές.

6 Πρώτο μέρος

Στο πρώτο μέρος της εργασίας θα ασχοληθούμε κυρίως με τη CPU και τα νήματα.

Αρχικά, θα υλοποιήσουμε κάποια system calls έτσι ώστε οι διεργασίες μας να μπορούν να είναι πολυνηματικές—στον κώδικα που σας δόθηκε είναι μονονηματικές.

Επίσης, θα υλοποιήσουμε έναν καλύτερο αλγόριθμο δρομολόγησης από αυτόν που υπάρχει ήδη (η τωρινή υλοποίηση είναι βασισμένη σε Round-Robin) ο οποίος ονομάζεται Multilevel Feedback Queue.

6.1 Πολυνηματικές διεργασίες

Θέλουμε επίσης να δώσουμε τη δυνατότητα στις διεργασίες να έχουν πολλά νήματα. Για το σκοπό αυτό, θα πρέπει να υλοποιήσετε τα system calls που αναφέρονται στο `tinysos.h`. Οι βασικές λειτουργίες των νημάτων είναι οι

- `CreateThread`
- `ThreadSelf`
- `ThreadJoin`
- `ThreadExit`
- `ThreadDetach`

Για να υλοποιήσετε τις παραπάνω λειτουργίες θα πρέπει:

- Να καταλάβετε καλά την υλοποίηση των διεργασιών (PCB και system calls `Exec`, `Exit`, `WaitChild` κλπ). Ιδιαίτερα, να καταλάβετε πώς τα system calls είναι υλοποιημένα ως λειτουργίες ενός monitor.
- Επίσης, να καταλάβετε τη δημιουργία νέων threads και τον τερματισμό threads από τον scheduler.
- Να διαβάσετε προσεκτικά τις προδιαγραφές των παραπάνω thread system calls που έχετε να υλοποιήσετε. Οι περιγραφές βρίσκονται στο αρχείο `tinyos.h`. Δώστε βαρύτητα στη διαφορά της `Exit` και της `ThreadExit`.
- Να δημιουργήσετε ένα καινούριο control block, ας το ονομάσουμε `process_thread_control_block` (PTCB), και για κάθε νήμα διεργασίας να έχετε ένα τέτοιο αντικείμενο, όπου θα αποθηκεύονται οι απαραίτητες πληροφορίες για αυτό τον πόρο-νήμα.
- Το κάθε PTCB πρέπει να σχετίζεται με κάποιο TCB καθώς και με κάποιο PCB, αλλά είναι χωριστό αντικείμενο, καθώς οι πληροφορίες του και η διάρκεια ζωής του είναι ανεξάρτητη από τα άλλα δύο control blocks.
- Σχεδιάστε από πριν τις βοηθητικές ρουτίνες που θα χρειαστείτε. Γράφετε unit tests για να εκτελείτε τον κώδικά σας και να τον δοκιμάζετε σε απομόνωση, χωρίς να τρέχει όλο το `TINYOS-3`.
- Μπορείτε να χρησιμοποιείτε τη σουίτα `thread_tests` του `validate_api` για να τρέξετε unit tests πάνω στον κώδικά σας.

Μετά τη σωστή υλοποίηση των process threads, μπορείτε να εκτελέσετε το πρόγραμμα `symp_thr` του `tinyos_shell` που είναι υλοποιημένο με πολυνηματική διεργασία.

6.2 Δρομολόγηση νημάτων με τον αλγόριθμο Multilevel Feedback Queue

Η τρέχουσα υλοποίηση του scheduler χρησιμοποιεί τον αλγόριθμο Round-Robin: υπάρχει μια FIFO ουρά στο τέλος της στην οποία προστίθενται τα νήματα που γίνονται `READY` και από την αρχή της οποίας αφαιρούνται τα νήματα τα οποία θα δρομολογηθούν σε κάποιο core.

Για να βελτιωθεί η διαδραστικότητα του λειτουργικού συστήματος `TINYOS-3`, θα πρέπει να υλοποιηθεί ο αλγόριθμος των multilevel feedback queues. Ο αλγόριθμος αυτός χρησιμοποιεί ένα array από ουρές, μια ουρά για κάθε προτεραιότητα. Κάθε νήμα έχει στο TCB του ένα ακέραιο πεδίο, ας το πούμε `priority`. Όταν το quantum ενός νήματος τελειώσει, ο scheduler υπολογίζει μια νέα τιμή για το πεδίο αυτό, που είναι στην πραγματικότητα η ουρά στην οποία θα προστεθεί το νήμα (όταν γίνει `READY` και `clean`). Τέλος, το πρώτο στοιχείο της πρώτης κατά προτεραιότητα μη-κενής ουράς επιλέγεται ως το επόμενο νήμα.

Πώς μπορεί να προσαρμοσθεί η τιμή της προτεραιότητας ενός νήματος:

- Στην πρώτη του εκτέλεση, ένα νήμα εισάγεται σε μια μέση θέση (είτε στην ουρά με την υψηλότερη προτεραιότητα).

- Αν ένα νήμα εξάντλησε το quantum του, η προτεραιότητά του μειώνεται κατά 1.
- Αν ένα νήμα δεν εξάντλησε το quantum του επειδή είναι διαδραστικό (εκτελεί κάποιες μορφές I/O) τότε η προτεραιότητά του αυξάνει κατά 1.
- Αν ένα νήμα δεν εξάντλησε το quantum του για κάποιο άλλο λόγο η προτεραιότητά του παραμένει αμετάβλητη.
- Νήματα τα οποία περιμένουν για πολλή ώρα στην ουρά τους (((πολλή ώρα))) σημαίνει ότι έχουν βγει από την ουρά ((πολλά)) νήματα), τότε αυξάνεται η προτεραιότητά τους κατά 1.

Επίσης, θα πρέπει να λάβετε υπόψη την περίπτωση του priority inversion όπου ένα νήμα περιμένει σε κάποιο mutex για κάποιο νήμα χαμηλότερης προτεραιότητας, και να προσαρμόσετε τις προτεραιότητες κατάλληλα.

Στον παραπάνω αλγόριθμο, κάποιες λειτουργίες δεν είναι προσδιορισμένες με σαφήνεια. Εναπόκειται σε εσάς να διαλέξετε τον τρόπο υλοποίησης και να δοκιμάσετε την απόδοση του συστήματός σας.

7 Δεύτερο μέρος

Στο μέρος αυτό θα ασχοληθούμε με είσοδο/έξοδο. Θα πρέπει να υλοποιήσετε τα system calls:

- Pipe για να δημιουργείτε νήματα καθώς και αντίστοιχες λειτουργίες για read/write.
- Τα system calls για τα sockets:
 1. Socket
 2. Connect
 3. Listen
 4. Accept
 5. Shutdown

καθώς και αντίστοιχες λειτουργίες για read/write.

- Τέλος, το system call OpenInfo καθώς και αντίστοιχες λειτουργίες για read/write.

7.1 Pipes

Τα pipes είναι ένας μηχανισμός επικοινωνίας διεργασιών που θα περιγραφεί και στη θεωρία. Για να τον υλοποιήσετε,

- Θα πρέπει να καταλάβετε τη λειτουργία των File Control Block και των κοινών τους λειτουργιών (Read, Write, Close κλπ.)
- Θα πρέπει να υλοποιήσετε ένα νέο τύπο Control Block (ας πούμε pipe_control_block).
- Θα πρέπει να υλοποιήσετε τα stream system calls (Read, Write, κλπ) για τα pipes.
- Διαβάστε τις προδιαγραφές στα docs (αρχείο tinyos.h).
- Χρησιμοποιήστε τη σουίτα pipe_tests του προγράμματος validate_api, για να τεστάρετε την υλοποίησή σας.

Αφού υλοποιήσετε τα pipes, μπορείτε να τα δοκιμάσετε στο tinyos_shell, για να τρέξετε μικρά pipelines.

7.2 Sockets

Τα sockets είναι ένας μηχανισμός επικοινωνίας διεργασιών που είναι κατάλληλος για επικοινωνία μέσω δικτύου. Εμείς θα υλοποιήσουμε sockets που θα επιτρέπουν την επικοινωνία τοπικών διεργασιών μόνο.

Για να δημιουργηθεί σύνδεση μεταξύ δύο socket θα υλοποιηθούν τα system calls που περιγράφονται στα docs με βάση όσα θα πούμε στη θεωρία.

- Θα πρέπει να καταλάβετε τη λειτουργία των File Control Block και των κοινών τους λειτουργιών (Read, Write, Close κλπ.)
- Είναι καλό να έχετε ήδη υλοποιήσει τα pipes, καθώς τα sockets θα έχουν κάποια κοινά σημεία, αλλά με μεγαλύτερη πολυπλοκότητα.
- Θα πρέπει να υλοποιήσετε ένα νέο τύπο Control Block (ας πούμε socket_control_block).
- Θα πρέπει να υλοποιήσετε τα stream system calls (Read, Write, κλπ) όπως και για τα pipes.
- Επιπλέον θα πρέπει να υλοποιήσετε τα system calls δημιουργίας σύνδεσης, δηλ. Accept, Connect, Listen.
- Διαβάστε τις προδιαγραφές στα docs (αρχείο tinyos.h).
- Χρησιμοποιήστε τη σουίτα socket_tests του προγράμματος validate_api, για να τεστάρετε την υλοποίησή σας.

7.3 Παρουσίαση πληροφοριών του συστήματος

Με αυτό το system call OpenInfo η ιδέα είναι να επιστρέφετε στο χρήστη πληροφορίες σχετικά με τις διεργασίες που εκτελούνται εκείνη τη στιγμή. Αυτό γίνεται ως εξής: η OpenInfo δημιουργεί ένα stream, το οποίο επιστρέφει μια ακολουθία από bytes, που αντιστοιχεί σε ένα array από αντικείμενα **struct procinfo**.

Κάθε αντικείμενο procinfo περιέχει πληροφορίες για μια διεργασία που υπάρχει *αυτή τη στιγμή* στο process table.

Δεν υπάρχουν υλοποιημένα unit tests για το OpenInfo, οπότε είναι σημαντικό να γράψετε τα δικά σας.

Μετά τη σωστή υλοποίηση, θα μπορείτε να εκτελείτε στο tinyos_shell την εντολή sysinfo, και να βλέπετε τη λίστα με τις τρέχουσες διεργασίες.

8 Εργαλεία ανάπτυξης που θα σας είναι χρήσιμα

Ένας βασικός στόχος του μαθήματος και ειδικά του project σας, είναι να εξοικειωθείτε με τα επαγγελματικά εργαλεία προγραμματισμού σε C. Ακολουθεί μια σύντομη αναφορά στα εργαλεία αυτά.

8.1 Απαραίτητα εργαλεία

gcc Ο compiler της C, είναι ίσως το πιο απαραίτητο εργαλείο. Προσφέρει μια πλειάδα επιλογών για την παραγωγή κώδικα και όχι μόνο.

text editor Ένας καλός editor για προγραμματισμό μπορεί να κάνει τη ζωή του προγραμματιστή πολύ πιο εύκολη. Σας συνιστώ να διαλέξετε είτε τον Sublime Text είτε τον VSCode. Για τους φίλους του ρετρό, υπάρχει πάντα και ο αγαπημένος μου Emacs.

make Με το πρόγραμμα αυτό το compiling αυτοματοποιείται σε μεγάλο βαθμό, ώστε να μη χρειάζεται να γράφετε μεγάλες και περίπλοκες εντολές στην κονσόλα. Στο συγκεκριμένο project, το make είναι απαραίτητο διότι το compiling που πρέπει να κάνετε είναι κάπως περίπλοκο.

gdb Ο debugger είναι ένα εργαλείο με το οποίο μπορούμε να παγώσουμε την εκτέλεση ενός προγράμματος και να παρατηρήσουμε τις τιμές των μεταβλητών σε μια συγκεκριμένη στιγμή της εκτέλεσης.

8.2 Χρήσιμα αλλά όχι απαραίτητα εργαλεία

valgrind Το πρόγραμμα αυτό είναι πολύ χρήσιμο στην αναζήτηση σφαλμάτων που έχουν να κάνουν με διαχείριση μνήμης.

git Εξαιρετικά χρήσιμο πρόγραμμα για κάθε συνεργασία περισσότερων από 1 προγραμματιστών! Σας επιτρέπει να μοιράζεστε κώδικα μεταξύ σας και να κρατάτε παλιές εκδόσεις του κώδικά σας.

doxygen Το εργαλείο αυτό παράγει εύκολα documentation σε HTML και άλλα formats, από τα σχόλια που βρίσκονται μέσα στον κώδικα.

8.3 Integrated Development Environments—IDE

Υπάρχουν κάποια εργαλεία που ενσωματώνουν όλα τα παραπάνω. Τα δύο πιο γνωστά είναι το Eclipse και το NetBeans, χωρίς να παραγνωρίζονται και κάποια άλλα, όπως πχ. το KDevelop. Γενικά, τα εργαλεία αυτά έχουν αρκετά πολύπλοκο configuration, οπότε το πιο πιθανό είναι να σας δυσκολέψει η χρήση τους παρά να κερδίσετε σε χρόνο.

9 Testing του κώδικα

Είναι αρκετά δύσκολο να αναπτύξετε σωστό concurrent κώδικα χωρίς μια συστηματική μέθοδο για testing. Για το σκοπό αυτό, στον κώδικα που σας δίνεται έχει προστεθεί εκτεταμένη υποστήριξη για tests.

Υπάρχουν (γενικά) δύο ειδών tests που μπορείτε (και πρέπει) να χρησιμοποιήσετε.

1. Unit tests, και
2. API tests

Και για τα δύο αυτά είδη tests μπορείτε να χρησιμοποιήσετε τη βιβλιοθήκη **unit_testing** που έχετε στον κώδικα που κατεβάσατε. Μπορείτε να δείτε το αρχείο `test_example.c` για να δείτε ένα πολύ απλό test program. Εκτελέστε το αρχείο αυτό για να δείτε πώς τρέχει.

validate_api: Οι κλήσεις συστήματος που υποστηρίζει το `tinyos` έχουν συγκεκριμένες προδιαγραφές, όπως αυτές περιγράφονται στον κώδικα του **tinyos.h**. Για να ελέγξετε την υλοποίησή σας, μπορείτε να χρησιμοποιήσετε το πρόγραμμα **validate_api**.

Το πρόγραμμα αυτό περιέχει έναν μεγάλο αριθμό από tests, όπου το κάθε ένα test ελέγχει κάποια συγκεκριμένη συμπεριφορά ενός ή περισσότερων system calls. Για παράδειγμα, τα βασικά system calls που είναι ήδη υλοποιημένα, ελέγχονται από μια ομάδα tests που ονομάζονται `basic_tests`. Για να τα τρέξετε δώστε την παρακάτω εντολή:

```
$ ./validate_api -t 2 basic_tests
running suite: basic_tests
test_boot                                     : ok
```

```

test_pid_of_init_is_one [cores= 1,term=2]: ok
test_waitchild_error_on_nonchild [cores= 1,term=2]: ok
test_waitchild_error_on_invalid_pid [cores= 1,term=2]: ok
test_exec_getpid_wait [cores= 1,term=2]: ok
test_exec_copies_arguments [cores= 1,term=2]: ok
test_exit_returns_status [cores= 1,term=2]: ok
test_main_return_returns_status [cores= 1,term=2]: ok
test_wait_for_any_child [cores= 1,term=2]: ok
test_orphans_adopted_by_init [cores= 1,term=2]: ok
test_cond_timedwait_timeout [cores= 1,term=2]: ok
test_cond_timedwait_signal [cores= 1,term=2]: ok
test_cond_timedwait_broadcast [cores= 1,term=2]: ok
test_null_device [cores= 1,term=2]: ok
test_get_terminals [cores= 1,term=2]: ok
test_open_terminals [cores= 1,term=2]: ok
test_dup2_error_on_nonfile [cores= 1,term=2]: ok
test_dup2_error_on_invalid_fid [cores= 1,term=2]: ok
test_dup2_copies_file [cores= 1,term=2]: ok
test_close_error_on_invalid_fid [cores= 1,term=2]: ok
test_close_success_on_valid_nonfile_fid [cores= 1,term=2]: ok
test_close_terminals [cores= 1,term=2]: ok
test_read_kbd [cores= 1,term=2]: ok
test_read_kbd_big [cores= 1,term=2]: ok
test_read_error_on_bad_fid [cores= 1,term=2]: ok
test_read_from_many_terminals [cores= 1,term=2]: ok
test_write_con [cores= 1,term=2]: ok
test_write_con_big [cores= 1,term=2]: ok
test_write_error_on_bad_fid [cores= 1,term=2]: ok
test_write_to_many_terminals [cores= 1,term=2]: ok
test_child_inherits_files [cores= 1,term=2]: ok
suite basic_tests completed [tests=31, failed=0]
basic_tests : ok
$

```

Εκτός από τα `basic_tests` υπάρχουν επίσης ήδη υλοποιημένες οι παρακάτω ομάδες από tests

thread_tests Ελέγχουν την υλοποίηση των process threads που σας ζητούνται στο 1ο μέρος.

pipe_tests Ελέγχουν την υλοποίηση των pipes που σας ζητούνται στο 2ο μέρος.

socket_tests Ελέγχουν την υλοποίηση των sockets που σας ζητούνται στο 2ο μέρος.

Η εκτέλεση αυτών των tests θα σας βοηθήσει πολύ στο να μειώσετε τα bugs στον κώδικά σας, καθώς και να καταλάβετε καλύτερα τυχόν λογικά λάθη που μπορεί να έχει η υλοποίησή σας.

Είναι **πολύ σημαντικό** να γράψετε και δικά σας tests που να εξετάζουν τη συμπεριφορά του δικού σας κώδικα με λεπτομερή τρόπο. Τα tests που είναι υλοποιημένα στο `validate_api` δεν μπορούν να τεστάρουν λεπτομέρειες της υλοποίησής σας, καθώς δεν μπορούν να τις γνωρίζουν.

Το να γράψετε ένα test είναι κάτι πολύ εύκολο και θα σας γλυτώσει από πολύ κόπο το να το κάνετε.

10 Παραδοτέα της εργασίας

Η παράδοση της εργασίας σας θα γίνει σε δύο μέρη:

Πρώτο μέρος: Το πρώτο μέρος αφορά την υλοποίηση των ζητουμένων που περιγράφονται στην Παρ. 6.

Δεύτερο μέρος: Το δεύτερο μέρος αφορά την υλοποίηση των ζητουμένων που περιγράφονται στην Παρ. 7.

10.1 Παραδοτέο του κάθε μέρους

Θα πρέπει να υλοποιήσετε και να παραδώσετε τον πυρήνα του `tinyos-3` που να υλοποιεί τις νέες λειτουργίες και κλήσεις συστήματος του αρχείου `tinyos.h` όπως αυτές προσδιορίζονται εκεί.

Θα πρέπει να παραδώσετε τον κώδικα του πυρήνα σας μαζί με όλα τα απαραίτητα αρχεία για να μπορέσει ο βαθμολογητής να μεταφράσει (compile) και να εκτελέσει τον κώδικά σας με μια εντολή. Μπορείτε να χρησιμοποιήσετε για το σκοπό αυτό το πρόγραμμα `make` με το αρχείο `Makefile` που σας δίνεται, ή να γράψετε το δικό σας `Makefile`.

Η εργασία σας θα γίνει σε ομάδες των τριών φοιτητών. Μπορείτε να συζητάτε τον κώδικά σας με άλλους, πχ. συμφοιτητές σας, εμένα, τους βοηθούς, κλπ. αλλά δεν μπορείτε να παραδώσετε κώδικα που δεν γράψατε οι ίδιοι. Οι βοηθοί του μαθήματος, πέρα από την εξέταση που θα κάνουν, θα χρησιμοποιήσουν αυτόματα προγράμματα εντοπισμού αντιγραφής, τα οποία είναι πολύ δύσκολο να ξεγελάσετε. Η αντιγραφή θα αντιμετωπιστεί *πολύ αυστηρά*.

Η παράδοση της εργασίας θα γίνει ηλεκτρονικά μέσα από το `courses`.

10.2 Καθυστερημένη παράδοση

Δε θα δοθεί καμιά παράταση στην παράδοση της εργασίας. Όσες εργασίες παραδοθούν μετά την προθεσμία παράδοσης θα έχουν μείωση του βαθμού τους κατά μία μονάδα (με άριστα το 10) *για κάθε ημερολογιακή ημέρα καθυστέρησης*.

10.3 Εξέταση του κάθε μέρους

Καλή δουλειά

και να θυμάστε

KISS : Keep It Simple Stupid!

και

RTFM : Read The Fine Manual!

Πηγές πληροφόρησης

1. Ο κώδικας που σας δίνεται περιέχει εκτεταμένη τεκμηρίωση, που μπορείτε να διαβάσετε μέσω ενός web browser.
2. Οι διαλέξεις του μαθήματος περιέχουν εκτεταμένη αναφορά στη θεωρία που χρειάζεται για να υλοποιήσετε την εργασία σας
3. Οι διαφάνειες των φροντιστηρίων περιέχουν πολλές χρήσιμες λεπτομέρειες.
4. Read the Source Luke
5. Η τεκμηρίωση του Linux στις σελίδες της εντολής `man`, πχ. `man sigaction`, `man makecontext`, `man sex` κλπ.
6. Το βιβλίο του μαθήματος.
7. Η βιβλιοθήκη.
8. Το internet.
9. Εγώ και οι βοηθοί, στο μάθημα ή στις ώρες γραφείου μας ή μέσω email.
10. Guru-συμφοιτητές σας!
11. Προσοχή: Ο κώδικας του linux σίγουρα θα σας μπερδέψει παρά θα σας βοηθήσει στην εργασία. Αλλά, πέρα από αυτό, είναι εξαιρετικό ανάγνωσμα για να μάθετε σωστό προγραμματισμό.