Εργαστήριο Λειτουργικών Συστημάτων
Ακ. Έτος 2012-13
Άσκηση 3

Να τροποποιηθεί ο αλγόριθμος χρονοπρογραμματισμού του Minix ώστε σε κάθε σειρά προτεραιότητας, ώστε να «ευνοούνται» οι διεργασίες που χρησιμοποιούν κατά κύριο λόγο το CPU (CPU bound) έναντι των διεργασιών που προκαλούν πολλά I/O interrupts (I/O bound).

Ο αλγόριθμός σας θα δουλεύει ως εξής:

1. Κάθε διεργασία λαμβάνει ένα quantum όπως έχουμε δει. Αν μια διεργασία καταναλώνει το quantum πλήρως, σημαίνει ότι όλη η δουλειά που έχει να κάνει γίνεται στον επεξεργαστή. Αντίθετα, όταν η διεργασία χρειάζεται πόρους από το σκληρό δίσκο, τότε δεν προλαβαίνει να καταναλώσει όλο το quantum, καθώς μπλοκάρει περιμένοντας την ανάγνωση του σχετικού αρχείου.
2. Εκμεταλλευόμενοι αυτό το φαινόμενο, θα τροποποιήσετε τον αλγόριθμο χρονοπρογραμματισμού στο minix (user-space scheduling) ώστε αν μια διεργασία δεν καταναλώσει όλο της το quantum, να μετατίθεται στην τελευταία ουρά προτεραιότητας. Οι διεργασίες που καταναλώνουν όλο τους το quantum θα παραμένουν στην ουρά που βρίσκονται.

Διευκρινίσεις:
Διαβάστε πολύ καλά την εισαγωγή στο user-space scheduling εδώ
http://wiki.minix3.org/en/Userspace_scheduling
Επίσης χρήσιμη θα σας φανεί η αναφορά του Swift για το πως δουλεύει το user-space scheduling στο Minix που θα βρείτε εδώ
http://www.minix3.org/docs/scheduling/report.pdf

Για να δοκιμάσετε τον κώδικά σας, θα πρέπει να δείτε πως συμπεριφέρεται το λειτουργικό με και χωρίς τον τροποποιημένο σας αλγόριθμο, παρατηρώντας τον χρόνο εκτέλεσης δύο προγραμμάτων:

- Γράψτε ένα απλό πρόγραμμα (io_bound.c) το οποίο λαμβάνει σαν παράμετρο εκτέλεσης το όνομα ενός αρχείου (χρησιμοποιήστε π.χ. το clients.txt από την 1η άσκηση) και κατόπιν το ανοίγει, διαβάζει από αυτό 100 bytes και το κλείνει. Βάλτε τον κώδικα σε ένα loop ώστε να εκτελείται αυτή η διαδικασία κάποιες χιλιάδες (π.χ. 3500) φορές.

- Γράψτε ένα απλό πρόγραμμα (cpu_bound.c) το οποίο λαμβάνει σαν παράμετρο εκτέλεσης μια λέξη και επαναλαμβάνει κάποιες χιλιάδες φορές έναν έλεγχο για να δει αν η λέξη είναι παλίνδρομο (δηλαδή διαβάζεται το ίδιο και από την ανάποδη, π.χ. το racecar).

Και για τα δύο προγράμματα φροντίστε να κρατάτε τον χρόνο εκτέλεσης. Δίνεται ένας βασικός σκελετός για δομή των προγραμμάτων πιο κάτω. Φτιάξτε επίσης ένα bash script το οποίο με την κλήση του, εκτελεί ταυτόχρονα από τρία στιγμιότυπα και των δύο δοκιμαστικών προγραμμάτων, ώστε να μπορείτε να διεξάγετε πειράματα.

Οδηγίες υποβολής

Τα παραδοτέα σας είναι:

1. Οι τροποποιήσεις στον κώδικα του Minix (γραπτά) και τα σχετικά τροποποιημένα αρχεία
2. Ο κώδικας των αρχείων των δοκιμαστικών προγραμμάτων (io_bound.c και cpu_bound.c) και του bash script
3. Screenshots που δείχνουν τον χρόνο εκτέλεσης των δοκιμαστικών προγραμμάτων με τον αλγόριθμό σας και με τον default αλγόριθμο του Minix
4. Πίνακες αποτελεσμάτων εκτέλεσης των προγραμμάτων 20 φορές με και χωρίς τον αλγόριθμό σας με σχετικά στατιστικά (μέσο όρο χρόνου εκτέλεσης, τυπικές αποκλίσεις)
5. Μισή το πολύ σελίδα Α4 με τα σχόλιά σας για την παρατηρούμενη συμπεριφορά του αλγορίθμου.

Η υποβολή σας θα ακολουθεί απαρέγκλιτα τη δομής της φόρμας υποβολής που ακολουθεί και θα βρίσκεται ένα ενιαίο αρχείο Word / PDF. Μαζί με αυτό θα υποβάλλετε τα αρχεία του τροποποιημένου κώδικα και των δοκιμαστικών προγραμμάτων. Συμπιέστε όλα τα σχετικά αρχεία της υποβολής και υποβάλλετέ τα σύμφωνα με τις εξής οδηγίες:

- Να ονομάσετε το συμπιεσμένο αρχείο σε [AM]_ex3.[zip|rar] (π.χ. 3345_ex3.rar ή 3345_ex3.zip)
- Το συμπιεσμένο αρχείο θα πρέπει να υποβληθεί ηλεκτρονικά στην ιστοσελίδα http://diogenis.ceid.upatras.gr/~akomninos/oslab

Σημειώνεται ότι η εργασία είναι ομαδική με επιθυμητό μέγεθος ομάδας το πολύ 3 άτομα.

Φόρμα υποβολής 3ης Άσκησης

Ονόματα φοιτητών
ΑΜ φοιτητών

Άσκηση 3

1. Τροποποιήσεις Minix

Λίστα τροποποιηθέντων αρχείων Minix

/usr/src/somedir/file1.c
/usr/src/somedir2/file2.c
….

Τροποποιήσεις αρχείων Minix

/usr/src/somedir/file1.c

Line	Code
50	if (var1>0) {
51	var2=var1 + 56;
…	…

/usr/src/somedir2/file2.c
….

2. Κώδικες δοκιμαστικών προγραμμάτων

cpu_bound.c

```
//cpu_bound.c dokimastiko programma
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
…
```

io_bound.c

```
//io_bound.c dokimastiko programma
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
…
```

Bash script

```
#!usr/pkg/bin/bash
for i in 1 2 3
do
./cpu_bound racecar&
./io_bound clients.txt&
done
echo “All done!”
```

3. Screenshot εκτέλεσης προγραμμάτων

[σχετικά screenshots τα οποία δείχνουν το output από τα δοκιμαστικά προγράμματα]

4. Πίνακες αποτελεσμάτων εκτελέσεων

Πρόγραμμα	**cpu_bound.c**					
# πειράματος	Με default αλγόριθμο			Με δικό μου αλγόριθμο		
1	Στιγμ. 1	Στιγμ. 2	Στιγμ. 3	Στιγμ. 1	Στιγμ. 2	Στιγμ. 3
2						
..						
20						
Μέσος όρος						
Τυπική απόκλιση						
Συνολικός Μ.Ο						

Πρόγραμμα	**io_bound.c**					
# πειράματος	Με default αλγόριθμο			Με δικό μου αλγόριθμο		
1	Στιγμ. 1	Στιγμ. 2	Στιγμ. 3	Στιγμ. 1	Στιγμ. 2	Στιγμ. 3
2						
..						
20						
Μέσος όρος						
Τυπική απόκλιση						
Συνολικός Μ.Ο						

5. Σχόλια για τη συμπεριφορά του αλγορίθμου
[μέχρι μισή σελίδα Α4]

```
//skeletos dokimastikou programmatos
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/times.h>

static time_t real_start;
static time_t real_end;

static struct tms start_sys;
static struct tms end_sys;
static clock_t start;
static clock_t end;

void start_clock() {
  time(&real_start);
  start = times(&start_sys);
}

void end_clock() {
  time(&real_end);
  end = times(&end_sys);
}

void print_clock_results() {
  // real,system,user
  printf("%i,%jd,%jd\n",
         (int)(real_end - real_start),
         (intmax_t)(end_sys.tms_stime - start_sys.tms_stime),
         (intmax_t)(end_sys.tms_utime - start_sys.tms_utime));
}

int main(int arg, char **argv) {
  int i = 0;
  start_clock();

  while(i < 3500) {
// do some stuff
  }

  end_clock();
  print_clock_results();
  return 0;
}
```