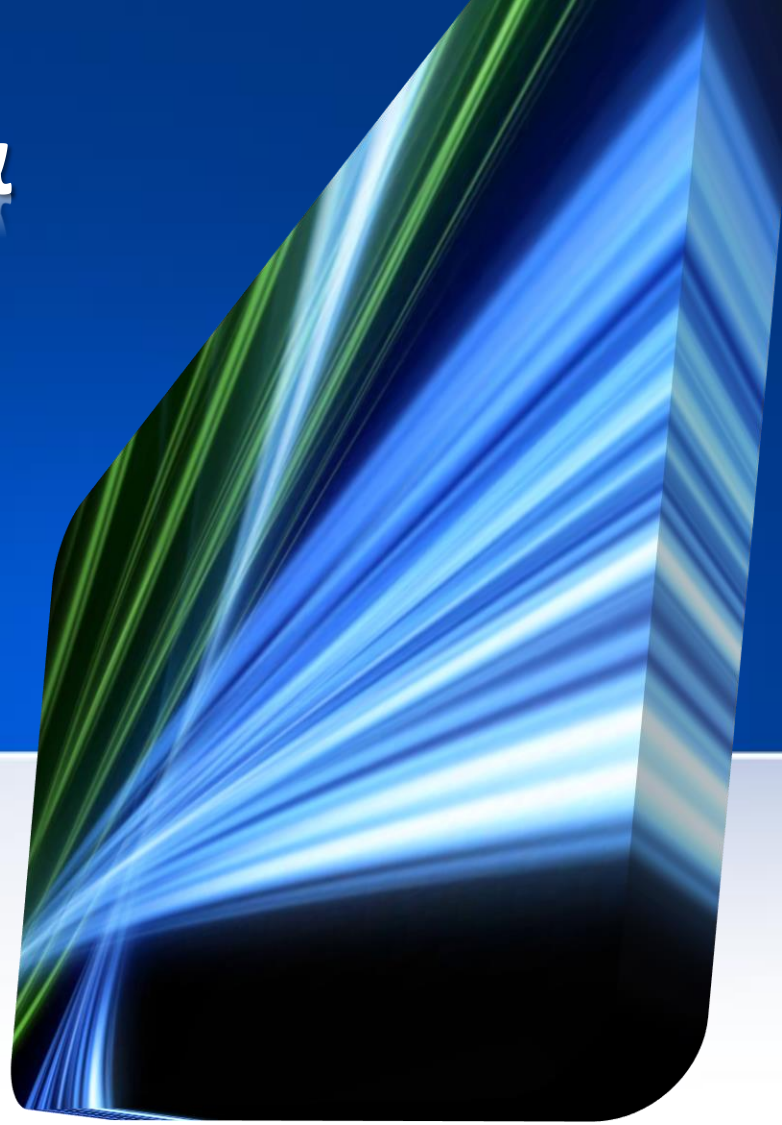
Λειτουργικά Συστήματα
6ο εξάμηνο ΣΗΜΜΥ
Ακ. έτος 2019-2020



Εργαστηριακή Άσκηση 1

Κανονισμός εργαστηριακών ασκήσεων

- Οι εργαστηριακές ασκήσεις διεξάγονται σε δύο Σειρές:
  Σειρά Α (Τρίτη 10.30 – 12.30) και
  Σειρά Β (Τρίτη 12.30 – 14.30)

- Οι φοιτητές είναι χωρισμένοι σε ομάδες των 2 ατόμων με αρίθμηση Σειρά Α1 – Σειρά Α40 και
  Σειρά Β1 – Σειρά Β40

Κανονισμός εργαστηριακών ασκήσεων

- Θα εκτελεστούν 4 εργαστηριακές ασκήσεις με βαρύτητα 10%, 30%, 30% και 30% επί του τελικού βαθμού εργαστηρίου

- Κάθε ομάδα πρέπει να προσέλθει για την προφορική εξέταση της κάθε άσκησης σε συγκεκριμένη ημερομηνία

Κανονισμός εργαστηριακών ασκήσεων

- Δεν υπάρχουν υποχρεωτικές παρουσίες στο εργαστήριο εκτός από τις ημερομηνίες εξέτασης των ασκήσεων
- Δεν απαιτείται παράδοση γραπτών αναφορών των εργαστηριακών ασκήσεων

Κανονισμός εργαστηριακών ασκήσεων

- Κάθε ομάδα μπορεί προαιρετικά να προσέλθει στο εργαστήριο κάθε εβδομάδα την ώρα του Τμήματος της για διατύπωση απορίων ή διευκρινήσεων, πρέπει όμως να εξετάζεται στις προκαθορισμένες ημερομηνίες

Εργαστηριακή Άσκηση 1

- Εισαγωγή στις εντολές φλοιού σε περιβάλλον Linux
- Εισαγωγή στο περιβάλλον προγραμματισμού
- Διαχείριση αρχείων
- Δημιουργία διεργασιών

Εργαστηριακή Άσκηση 1

Να γραφτεί πρόγραμμα σε γλώσσα προγραμματισμού C και περιβάλλον Linux στο οποίο η διεργασία πατέρας (F) δημιουργεί 2 διεργασίες (C1,C2) οι οποίες ανταλλάσσουν ένα κρυπτογραφημένο μήνυμα μέσω ενός αρχείου.

Το κλειδί κρυπτογράφησης και το όνομα του αρχείου που υπάρχει το μήνυμα δίνονται από τον χρήστη κατά την εκτέλεση του προγράμματος.

Εργαστηριακή Άσκηση 1

Αρχικά η διεργασία πατέρας δημιουργεί την διεργασία C1, η οποία γράφει ένα κρυπτογραφημένο μήνυμα σε ένα αρχείο.

Στην συνέχεια η διεργασία πατέρας δημιουργεί την διεργασία C2 η οποία πρέπει να αποκρυπτογραφήσει το μήνυμα.

Η κρυπτογράφηση – αποκρυπτογράφηση γίνεται με χρήση του Caesar encryption algorithm

Εργαστηριακή Άσκηση 1

Προσδοκώμενο αποτέλεσμα:

Η διεργασία C2 να καταφέρει να διαβάσει το μήνυμα που έχει γράψει στο αρχείο η διεργασία C1

Θεωρία Εργ. Άσκησης 1

Διεργασία είναι ένα πρόγραμμα που εκτελείται

Είναι μια μονάδα εργασίας **μέσα** στο σύστημα. Το πρόγραμμα είναι μια παθητική οντότητα, η διεργασία είναι μια **ενεργή οντότητα**.

Η διεργασία χρειάζεται

- **πόρους** (CPU, μνήμη, μονάδες Ε/Ε, αρχεία) για την εκπλήρωση των καθηκόντων της
- **δεδομένα** αρχικοποίησης

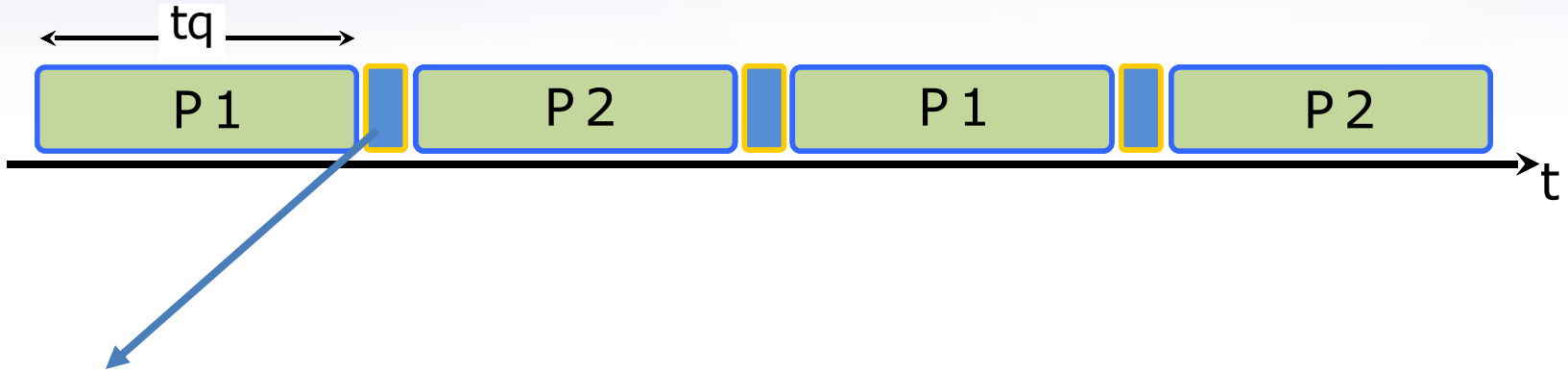
Θεωρία Εργ. Άσκησης 1

Μοντέλο Διαμοιρασμού Χρόνου

- Πολλαπλές διεργασίες Θέλουν να εκτελεστούν ταυτόχρονα.
- Το Λειτουργικό Σύστημα μοιράζει τον χρόνο του επεξεργαστή και αναλαμβάνει να τις χρονοδρομολογίσει.
- Οι διεργασίες έχουν την (ψευδ)αίσθηση ότι χρησιμοποιούν αποκλειστικά τον επεξεργαστή
- Ο χρονοδρομολογητής αναλαμβάνει:
    - Την επιλογή της διεργασίας που θα χρησιμοποιήσει τον επεξεργαστή
    - Την αλλαγή της διεργασίας που εκτελείται στον επεξεργαστή (context switch)

Θεωρία Εργ. Άσκησης 1

Μοντέλο Διαμοιρασμού Χρόνου



- Επιλογή επόμενης διεργασίας (scheduling)
- Αλλαγή περιβάλλοντος λειτουργίας (Context Switch)

Χρήσιμες συναρτήσεις

Κάθε διεργασία έχει συσχετισμένο μαζί της έναν εγγυημένα μοναδικό αριθμό ταυτότητας διεργασίας(process-id, pid) που παρέχεται δυναμικά από το Λειτουργικό Σύστημα. Ο αριθμός αυτός χρησιμοποιείται για να αναφερθούμε σε κάποια διεργασία.

Μια διεργασία μπορεί να μάθει το pid της εκτελώντας την κλήση:

```
pid_t  getpid(void)
```

Το pid μιας διεργασίας μπορεί να αποθηκευτεί σε μια μεταβλητή τύπου **pid_t**

Χρήσιμες συναρτήσεις

Παράδειγμα 1: Μία διεργασία ενημερώνεται για το pid της και στη συνέχεια το εκτυπώνει.

```
#include<stdio.h>
#include <stdlib.h>
int main()
{
        pid_t mypid;
        mypid = getpid() ;
        printf(" My id: %d\n", mypid);
        return(0);
}
```

Χρήσιμες συναρτήσεις

Κάθε διεργασία μπορεί να μάθει το αριθμό ταυτότητας (pid) της γονικής διεργασίας (δηλαδή της διεργασίας που τη δημιούργησε) χρησιμοποιώντας την εντολή getppid() εκτελώντας την κλήση:

pid_t getppid(void)

Χρήσιμες συναρτήσεις

Παράδειγμα 2: Μία διεργασία ενημερώνεται για το pid της γονικής διεργασίας και στη συνέχεια το εκτυπώνει.

```
pid_t parent_pid;
parent_pid = getppid() ;
printf(“ My parent’s id: %d\n”, parent_pid);
```

Χρήσιμες συναρτήσεις

Μια διεργασία μπορεί να δημιουργήσει μια νέα διεργασία-παιδί, πιστό αντίγραφο του εαυτού της με χρήση της κλήσης fork()

Η κλήση fork() επιστρέφει την τιμή 0 στην διεργασία παιδί και το pid του παιδιού στην διεργασία πατέρα.

Με τον τρόπο αυτό η διεργασία-παιδί που προέκυψε μπορεί να αντικαθιστά το πρόγραμμα που εκτελεί (αρχικά ίδιο με του πατέρα) με νέο πρόγραμμα.

Χρήσιμες συναρτήσεις

Προαιρετικά, ο γονέας μπορεί να περιμένει μέχρι τον τερματισμό κάποιας διεργασίας-παιδιού του με την κλήση wait() .

Η κλήση wait() αναστέλλει την εκτέλεση του καλούντος προγράμματος μέχρις ότου τερματισθεί η εκτέλεση κάποιας από τις διεργασίες παιδιά του. Η συνάρτηση wait() επιστρέφει το pid της θυγατρικής διεργασίας ή -1 για σφάλμα. Η κατάσταση εξόδου της θυγατρικής διεργασίας βρίσκεται στη μεταβλητή status. Επίσης, αν κάποια διεργασία παιδί έχει ήδη τερματιστεί, τότε η κλήση επιστρέφει αμέσως -1.

Ο οικειοθελής τερματισμός μιας διεργασίας μπορεί να γίνει με τη κλήση exit()

Χρήσιμες συναρτήσεις

Παράδειγμα 3.1: Μία διεργασία δημιουργεί μια νέα διεργασία

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
        int status;
        pid_t child ;
        child = fork();
        if(child<0){
                //error
                ...
        }
        if(child==0){
                //child's code
                ...
                exit(0);
        }
        else {
                //father's code
                ...
                wait(&status);
                exit(0);
        }
    return 0;
}
```

Χρήσιμες συναρτήσεις

Παράδειγμα 3.2: Μία διεργασία δημιουργεί μια νέα διεργασία

```
pid_t pid = fork();
if (pid == -1)
{
        perror("fork");
}
else if (pid == 0)
{
        printf("CHILD: My pid is %d, my father is %d\n", getpid(), getppid());
}
else
{
        printf("PARENT: My pid is %d, my father is %d\n", getpid(), getppid());
        wait(&status);
 }
```

Χρήσιμες συναρτήσεις

Η εντολή sleep() εισάγει μια αναμονή στο σύστημα, για όσα δευτερόλεπτα της δώσουμε ως παράμετρο.

Παράδειγμα 4. Αναμονή 5 δευτερολέπτων

```
sleep(5)
```

Παράρτημα

Ακολουθούν χρήσιμες πληροφορίες χρήσης περιβάλλοντος προγραμματισμού

Χρήσιμες πληροφορίες χρήσης περιβάλλοντος προγραμματισμού

Μονοπάτι (path):
Συμβολοσειρα από αναγνωριστικά χωρισμένα από τον χαρακτήρα **/**
πχ: `/home/christos/first.c`

Το μονοπάτι είναι

- *απόλυτο* αν ξεκινάει με **/** → αφετηρία είναι η αρχή της ιεραρχίας
- *σχετικό* → αφετηρία είναι ο τρέχων κατάλογος (ΤΚ)

Το αναγνωριστικό:

. σηματοδοτεί τον ΤΚ
.. σηματοδοτεί τον πατέρα του ΤΚ

Διαχείριση καταλόγων

Εντολές

cd: Αλλαγή τρέχοντος καταλόγου

mkdir: Δημιουργία καταλόγου

rmdir: Διαγραφή καταλόγου

Διαχείριση Αρχείων

Εντολές

cat: Εκτύπωση

cp: Αντιγραφή

mv: Μετακίνηση

rm: Διαγραφή

Χρήσιμές εντολές

Εντολές

echo: Εκτύπωση μηνύματος

ls: Εκτύπωση λίστας περιεχομένων ενός καταλόγου

mv: Μετακίνηση

rm: Διαγραφή

Compiling & linking

- **Compile** (Μεταγλώττιση):

first.c ⇒ first.o

second.c ⇒ second.o

- **Link** (Σύνδεση):

first.o + second.o ⇒ executable

Παράδειγμα compiling & linking ενός αρχείου

```
$ gcc  -Wall  -c   first.c
$ gcc  first.o     -o     first
$ ./first
```

ή

```
$ gcc  -Wall    first.c –o first
$ ./first
```

Παράδειγμα compiling & linking πολλαπλών αρχείων

```
$ gcc -Wall -c first.c
$ gcc -Wall -c second.c

$ gcc first.o second.o –o allinone
$ ./allinone
```

Χρήσιμα Links

https://help.ubuntu.com/community/UsingTheTerminal

https://files.fosswire.com/2007/08/fwunixref.pdf

http://www.gnu.org/software/libc/manual/html_node/Processes.html#Processes