ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Λειτουργικά Συστήματα

Ενότητα # 10: Προγραμματισμός UNIX

Διδάσκων: Γεώργιος Ξυλωμένος

Τμήμα: Πληροφορικής

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

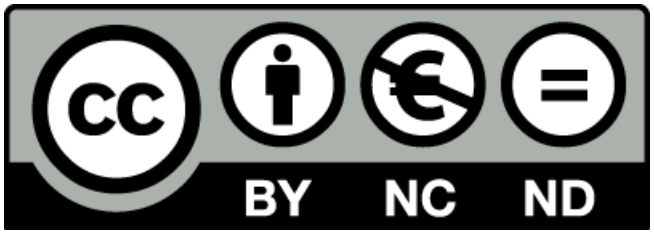
Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Οικονομικό Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Οι εικόνες προέρχονται από το βιβλίο «Σύγχρονα Λειτουργικά Συστήματα», A.S. Tanenbaum, 3η έκδοση, 2009, Εκδόσεις Κλειδάριθμος.



Σκοποί ενότητας

- Κατανόηση των διεργασιών και των νημάτων σε προγραμματιστικό επίπεδο.
- Εξοικείωση με τους μηχανισμούς διαχείρισης διεργασιών και νημάτων.
- Κατανόηση των βασικών μηχανισμών επικοινωνίας διεργασιών.
- Κατανόηση των βασικών μηχανισμών συγχρονισμού νημάτων.

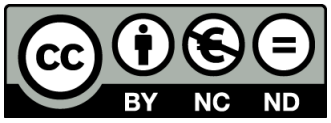
Περιεχόμενα ενότητας

- Εισαγωγή
- Διεργασίες
- Επικοινωνία διεργασιών
- Νήματα
- Νήματα POSIX

Εισαγωγή

Μάθημα: Λειτουργικά Συστήματα, **Ενότητα #10:** Προγραμματισμός UNIX

Διδάσκων: Γιώργος Ξυλωμένος, **Τμήμα:** Πληροφορικής



Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ
2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Προγραμματισμός UNIX (1 από 2)

- Έχουμε καλύψει διάφορες όψεις των ΛΣ
  - Και του Linux ως μελέτης περίπτωσης
  - Σε καθαρά θεωρητικό επίπεδο όμως
  - Ο προγραμματισμός είναι πιο πρακτικός
- Γιατί UNIX και όχι Linux;
  - Το Linux παρέχει πολλές επεκτάσεις στο UNIX
  - Οι κοινές κλήσεις όμως λειτουργούν παντού
    - Και σε παραλλαγές BSD και System V

Προγραμματισμός UNIX (2 από 2)

- Προγραμματισμός διεργασιών και νημάτων
  - Πολλές κλήσεις συστήματος αντικαθίστανται
    - Όλες οι γλώσσες προγραμματισμού χειρίζονται αρχεία
  - Ο χειρισμός διεργασιών και νημάτων όχι!
    - Ελάχιστες γλώσσες «βλέπουν» αυτές τις οντότητες
  - Ζητήματα επικοινωνίας και συγχρονισμού
    - Διεργασίες/νήματα εκτελούνται παράλληλα
    - Εισαγωγή στον ταυτόχρονο προγραμματισμό

Διεργασίες

Μάθημα: Λειτουργικά Συστήματα, **Ενότητα #10:** Προγραμματισμός UNIX

Διδάσκων: Γιώργος Ξυλωμένος, **Τμήμα:** Πληροφορικής

Έννοια της διεργασίας

- Διεργασία: εκτελούμενο πρόγραμμα
  - Αποτελείται από εικόνα μνήμης και κατάσταση
  - Μνήμη: κώδικας, σωρός, στοίβα
  - Κατάσταση: μετρητής, καταχωρητές
- Οι διεργασίες είναι ανεξάρτητες
  - Το ΛΣ απομονώνει τις διεργασίες
- Οι διεργασίες μπορεί να συνεργάζονται
  - Μηχανισμοί επικοινωνίας και συγχρονισμού

Διεργασίες (1 από 6)

- Δημιουργία: `pid_t fork(void);`
  - Ακριβές αντίγραφο της εκτελούμενης διεργασίας
    - Η εκτελούμενη διεργασία είναι ο γονέας
    - Η νέα διεργασία είναι το παιδί
  - Επιστρέφει 0 στο παιδί και pid παιδιού στο γονέα
  - Ο κώδικας παραμένει κοινός
  - Τα δεδομένα είναι τα ίδια μόνο αρχικά
    - Κάθε διεργασία έχει το δικό της αντίγραφο
  - Ίδια ανοιχτά αρχεία, σήματα και περιβάλλον

Διεργασίες (2 από 6)

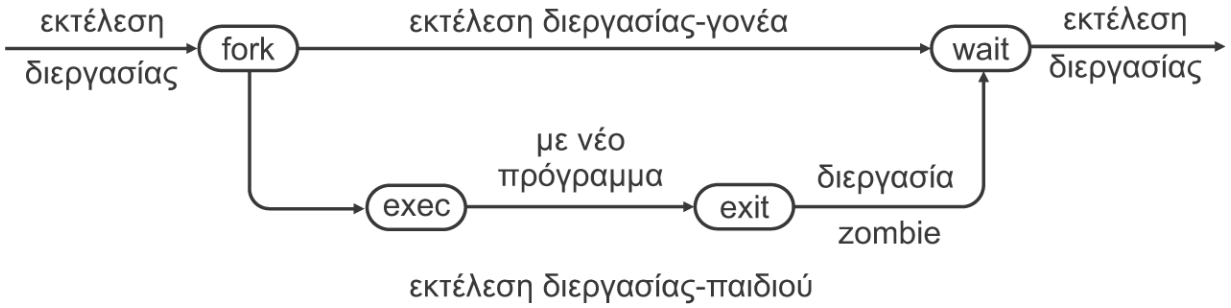
- Τερματισμός: `void exit(int status);`
  - Επιστρέφει κωδικό εξόδου στον πατέρα
  - Ο πατέρας λαμβάνει και σήμα `SIGCHLD`
- Αναμονή τερματισμού παιδιού
  - `pid_t wait(int *status);`
  - Επιστρέφει αμέσως αν έχει τερματίσει παιδί
  - Επιστρέφει το pid του παιδιού και έναν κωδικό
    - Περιλαμβάνει τρόπο εξόδου και κωδικό εξόδου

Διεργασίες (3 από 6)

- Σύνθετη αναμονή τερματισμού παιδιού
  - `pid_t waitpid(pid_t pid, int *status, int options);`
  - Συμπεριφέρεται ανάλογα με pid
    - < -1: περιμένει παιδί από συγκεκριμένη ομάδα
    - -1: περιμένει οποιοδήποτε παιδί
    - 0: περιμένει παιδί που ανήκει στην τρέχουσα ομάδα
    - > 0: περιμένει παιδί με συγκεκριμένο pid
  - Οι επιλογές επιτρέπουν και άμεση επιστροφή

Διεργασίες (4 από 6)

- Αντικατάσταση κώδικα: κλήσεις `exec`
  - Φόρτωση νέου κώδικα και δεδομένων
  - Δεν αλλάζει το PID της διεργασίας
  - Τα ανοιχτά αρχεία παραμένουν
  - Τυπικός κύκλος ζωής διεργασίας



Διεργασίες (5 από 6)

- Παραλλαγές της exec με λίστα
  - `int execl(const char *path, const char *arg, ...);`
  - `int execlp(const char *file, const char *arg, ...);`
  - `int execle(const char *path, const char *arg , ..., char * const envp[]);`
  - Όνομα αρχείου, παράμετροι εισόδου, περιβάλλον

Διεργασίες (6 από 6)

- Παραλλαγές της exec με πίνακα
  - `int execv(const char *path, char *const argv[]);`
  - `int execvp(const char *file, char *const argv[]);`
  - `int execve(const char *file, char *const argv [], char *const envp[]);`
  - Οι παράμετροι εισόδου είναι σε πίνακα

Επικοινωνία διεργασιών

Μάθημα: Λειτουργικά Συστήματα, **Ενότητα #10:** Προγραμματισμός UNIX

Διδάσκων: Γιώργος Ξυλωμένος, **Τμήμα:** Πληροφορικής

Μηχανισμοί επικοινωνίας (1 από 2)

- Οι διεργασίες είναι απομονωμένες
  - Μπορούν να επικοινωνούν ελεγχόμενα
  - Μηχανισμοί επικοινωνίας διεργασιών (IPC)
- Σήματα
  - Κατάλληλα για πολύ απλή επικοινωνία
- Ανώνυμοι και επώνυμοι αγωγοί
  - Απλά κανάλια παρόμοια με σειριακά αρχεία
  - Ανώνυμοι: επικοινωνία διεργασιών ίδιας ομάδας
  - Επώνυμοι: επικοινωνία τυχαίων διεργασιών

Μηχανισμοί επικοινωνίας (2 από 2)

- Ουρές μηνυμάτων
  - Επικοινωνία μέσω γραμματοκιβωτίων
  - Παρέχουν αρκετές πρόσθετες δυνατότητες
- Κοινή μνήμη και σηματοφορείς
  - Δεν παρεμβαίνει ο πυρήνας
  - Σηματοφορείς για συγχρονισμό
- Υποδοχές
  - Χρησιμοποιούνται κυρίως στο δίκτυο
  - Κατάλληλες και για τοπική επικοινωνία

Σήματα (1 από 5)

- Σήματα (signals): διακοπές λογισμικού
  - Ο παραλήπτης τα αγνοεί ή τα αντιμετωπίζει
    - Ορισμένα (`SIGKILL/SIGSTOP`) δεν αγνοούνται
  - Αλλιώς σκοτώνουν τον παραλήπτη
- Αντιμετώπιση σημάτων
  - Η διεργασία δηλώνει συνάρτηση χειρισμού
  - Όταν συμβεί το σήμα, διακόπτεται η εκτέλεση
  - Εκτελείται η συνάρτηση χειρισμού (handler)
  - Επιστροφή στο σημείο που διακόπηκε η εκτέλεση

Σήματα (2 από 5)

- Άμεση αποστολή σημάτων
  - `int kill(pid_t pid, int sig);`
  - Στέλνει το σήμα sig στη διεργασία pid
  - Πρέπει να ανήκουν στην ίδια ομάδα διεργασιών
- Έμμεση παραγωγή σημάτων
  - `SIGALRM`: λήξη χρονομέτρου alarm()
- Παραγωγή σημάτων από το σύστημα
  - Σήματα εξαιρέσεων (διαίρεση με το μηδέν)
  - Συνδυασμοί πλήκτρων (CTRL-C)

Σήματα (3 από 5)

- Ορισμός χρονομέτρων
  - `unsigned int alarm(unsigned int sec);`
  - Το χρονόμετρο θα σημάνει σε `sec` δεύτερα
  - Ακυρώνει τυχόν προηγούμενο χρονόμετρο
    - Με παράμετρο μηδέν απλά το ακυρώνει
    - Επιστρέφει χρόνο που είχε μείνει
  - Στην εκπνοή στέλνει `SIGALRM` στη διεργασία

Σήματα (4 από 5)

- Χειρισμός σημάτων
  - `int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);`
  - Το `signum` προσδιορίζει ένα σήμα
  - Το `act` δείχνει τον τρόπο αντιμετώπισης
    - Δομή με πολλά πεδία (και χειριστή)
  - Το `oact` δείχνει τον παλιό τρόπο αντιμετώπισης

Σήματα (5 από 5)

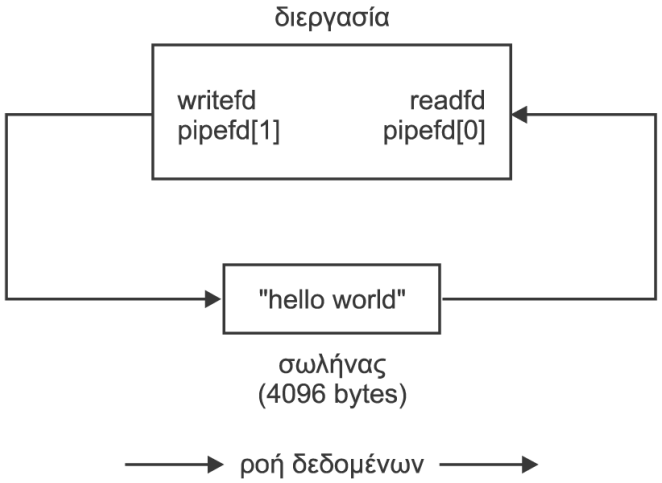
- Χειριστές σημάτων
  - Ο χειριστής δίνεται στο πεδίο `sa_handler`
  - Δείκτης στη συνάρτηση χειρισμού
  - Ο χειριστής είναι `void handler(int sig);`
  - Όταν καλείται βλέπει και το σήμα
  - Επιτρέπεται και `SIG_DFL` ή `SIG_IGN`
- Τα σήματα δεν έχουν μνήμη
  - Αν δεν τα αντιμετωπίσουμε, απλά χάνονται!

Αγωγοί (1 από 9)

- (Ανώνυμοι) Αγωγοί (pipes)
  - Μονόδρομη ροή byte μεταξύ διεργασιών
  - Μοιάζουν με ακολουθιακά αρχεία
    - Λειτουργούν σαν ενταμιευτές
    - Συνήθως μικρή χωρητικότητα (π.χ. 4096 bytes)
  - Αυτόματος έλεγχος ροής
    - Όταν γεμίσουν εμποδίζεται ο συγγραφέας
    - Όταν αδειάσουν εμποδίζεται ο αναγνώστης

Αγωγοί (2 από 9)

- Δημιουργία αγωγού
  - `int pipe(int filedes[2]);`
  - Επιστρέφει δύο περιγραφητές αρχείου
    - `filedes[0]` για ανάγνωση, `filedes[1]` για εγγραφή

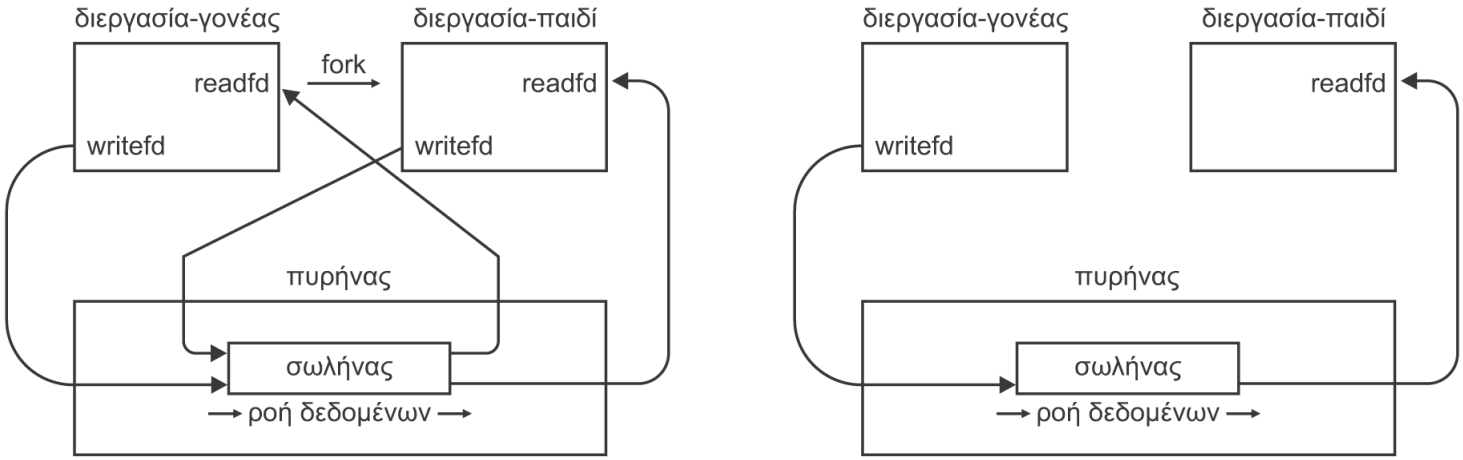


Αγωγοί (3 από 9)

- Επικοινωνία διεργασιών με αγωγούς
  - Οι αγωγοί είναι «ανώνυμοι»
    - Η `pipe()` δεν παίρνει κάποιο όνομα
  - Δεν «ανοίγουν» από πολλές διεργασίες
  - Μπορούν μόνο να «κληρονομηθούν»
  - Με `fork()` το παιδί κληρονομεί τους σωλήνες
  - Κλείσιμο άκρων που δεν χρειάζονται
  - Μονόδρομη ή αμφίδρομη επικοινωνία

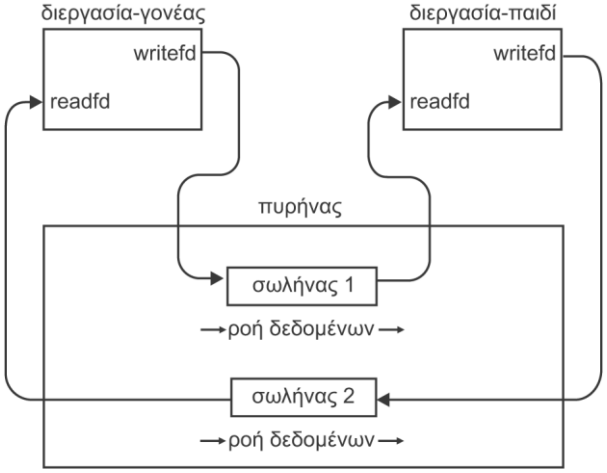
Αγωγοί (4 από 9)

- Μονόδρομη επικοινωνία
  - Ο γονέας δημιουργεί το σωλήνα με `pipe()`
  - Ο γονέας κλωνοποιείται με την κλήση `fork()`
  - Ο γονέας κλείνει το άκρο ανάγνωσης
  - Το παιδί κλείνει το άκρο εγγραφής



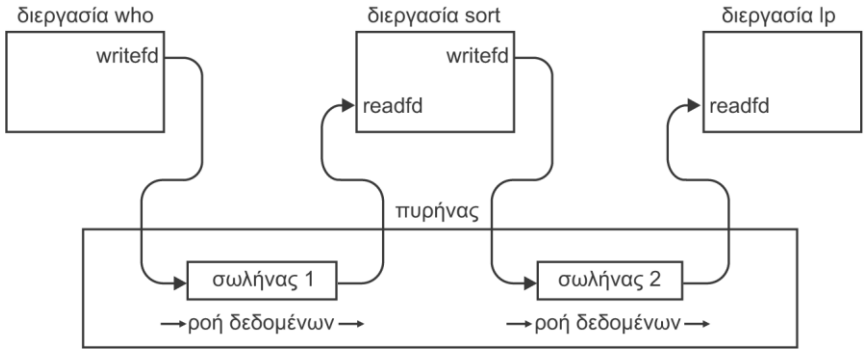
Αγωγοί (5 από 9)

- Αμφίδρομη επικοινωνία
  - Ο πατέρας ανοίγει δύο σωλήνες
  - Ο πατέρας κλείνει το άκρο ανάγνωσης 1
  - Ο πατέρας κλείνει το άκρο εγγραφής 2
  - Το παιδί κάνει το ανάποδο



Αγωγοί (6 από 9)

- Πολλαπλοί αγωγοί
  - Ο φλοιός διαβάζει το `who | sort | lpr`
  - Δημιουργεί όσους σωλήνες χρειάζονται
  - Κλωνοποιείται για κάθε διεργασία
  - Κλείνει τα περιττά άκρα των σωλήνων
  - Εκτελεί τα προγράμματα των διεργασιών



Αγωγοί (7 από 9)

- Σύνδεση αγωγών με stdin / stdout
  - Στο φλοιό έχουμε ανακατεύθυνση
    - Η είσοδος / έξοδος συνδέεται με αγωγό
  - Αρχικά κλείνουμε το stdin / stdout
  - Μετά αντιγράφουμε εκεί τον αγωγό
  - `int dup(int oldfd);`
    - Ο `oldfd` αντιγράφεται στον πρώτο διαθέσιμο
  - `int dup2(int oldfd, int newfd);`
    - Ο `oldfd` αντιγράφεται στον `newfd`

Αγωγοί (8 από 9)

- Οι αγωγοί είναι ροές byte χωρίς όρια
  - Μπορούμε να γράψουμε 10 byte και μετά άλλα 5
  - Στην ανάγνωση φαίνεται ότι γράφτηκαν 15 byte
- Οι σωλήνες δεν έχουν «τέλος»
  - Όσο ο σωλήνας είναι ανοικτός, μπορεί να γραφτεί
  - Οι αναγνώσεις επιστρέφουν ή εμποδίζονται
  - Η ροή κλείνει όταν κλείσει το άκρο εγγραφής
    - Αρχικά διαβάζονται όσα δεδομένα εκκρεμούν
    - Στην επόμενη ανάγνωση θα επιστραφεί 0

Αγωγοί (9 από 9)

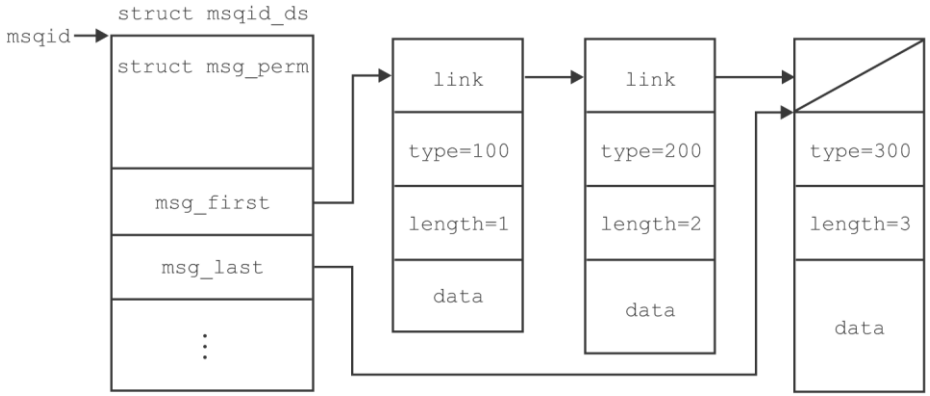
- (Επώνυμοι) Αγωγοί (FIFO)
  - `int mkfifo(const char *pathname, mode_t mode);`
  - Λειτουργούν όπως οι απλοί αγωγοί
    - Δεν περνάνε δεδομένα μέσω αρχείων
  - Έχουν όνομα στο σύστημα αρχείων
    - Και δικαιώματα προσπέλασης
  - Μπορούν να ανοίξουν από πολλές διεργασίες
    - Κάθε άκρο ανοίγει χωριστά

Ουρές μηνυμάτων (1 από 10)

- Ουρές μηνυμάτων (message queues)
  - Ροή μηνυμάτων αυθαίρετης δομής
    - Μόνο η επικεφαλίδα είναι τυποποιημένη
    - Το υπόλοιπο ορίζεται από την εφαρμογή
  - Επιτρέπονται πολλοί αναγνώστες / συγγραφείς
    - Οι ουρές έχουν «όνομα» στο σύστημα
  - Επιτρέπουν πλήρως ασύγχρονη επικοινωνία
    - Τα μηνύματα αποθηκεύονται από το ΛΣ

Ουρές μηνυμάτων (2 από 10)

- Δομή `msqid` με βασικά στοιχεία ουράς
  - Κεφαλή/ ουρά λίστας, μηνύματα / max bytes
  - Pid/χρόνοι τελευταίου αναγνώστη / συγγραφέα
- Ουρά μηνυμάτων με σύνδεση μίας κατεύθυνσης
  - Επόμενο μήνυμα, τύπος, μήκος, δεδομένα



Ουρές μηνυμάτων (3 από 10)

- Δομή `ipc_perm` με δικαιώματα ουράς
  - Περιλαμβάνεται στη δομή `msqid`
  - UID/GID δημιουργού
  - UID/GID ιδιοκτήτη
  - Χάρτης bit ανάγνωσης/εγγραφής
- Αναγνωριστικό τύπου `key_t`
  - Προσδιορίζει μία ουρά στο ΛΣ
  - Γνωστό στις διεργασίες που επικοινωνούν

Ουρές μηνυμάτων (4 από 10)

- Άνοιγμα ουράς
  - `int msgget(key_t key, int msgflag);`
  - Επιστρέφει τοπικό αναγνωριστικό ουράς
  - Οι σημαίες ορίζουν πώς θα ανοίξει η ουρά
    - Χωρίς σημαία: άνοιγμα ουράς ή αποτυχία
    - `IPC_CREAT`: άνοιγμα ουράς ή δημιουργία της
    - `IPC_CREAT|IPC_EXCL`: δημιουργία ουράς ή αποτυχία
  - Οι σημαίες δίνουν και τα δικαιώματα πρόσβασης
    - Ίδια κωδικοποίηση με κλήσεις `open()` και `creat()`

Ουρές μηνυμάτων (5 από 10)

- Έλεγχος της ουράς
  - `int msgctl(int msqid, int cmd, struct msqid_ds *buf);`
  - Η εντολή `cmd` ορίζει τι θέλουμε να κάνουμε
  - `IPC_RMID`: διαγραφή ουράς
  - `IPC_SET`: αλλαγές σε δομή `msqid`
    - Οι επιθυμητές τιμές είναι στη δομή `buf`
  - `IPC_STAT`: επιστροφή δομής `msqid` στην `buf`

Ουρές μηνυμάτων (6 από 10)

- Τύπος μηνυμάτων: `struct msgbuf`
  - Περιέχει δύο πεδία: τύπο και περιεχόμενο
  - Ο τύπος (`mtype`) είναι long και > 0
    - 0 και αρνητικές τιμές έχουν ειδική σημασία
  - Τα δεδομένα (`mtext`) είναι πίνακας χαρακτήρων
    - Η διεργασία τα κωδικοποιεί όπως θέλει
  - Ο πυρήνας προσθέτει μήκος και άλλα στοιχεία
    - Χρήση μόνο για εσωτερική οργάνωση της ουράς

Ουρές μηνυμάτων (7 από 10)

- Αποστολή μηνυμάτων σε ουρά
  - `int msgsnd(int msqid, struct msgbuf *msgp, size_t msgsz, int msgflg);`
  - Το μήκος (`msgsz`) δεν περιλαμβάνει τον τύπο
    - Περιλαμβάνονται μόνο τα δεδομένα
  - Σημαία `IPC_NOWAIT`
    - Επιστροφή αν δεν υπάρχει χώρος
    - Αλλιώς εμποδισμός μέχρι να αδειάσει χώρος

Ουρές μηνυμάτων (8 από 10)

- Λήψη μηνυμάτων από ουρά
  - `ssize_t msgrcv(int msqid, struct msgbuf *msgp, size_t msgsz, long msgtyp, int msgflg);`
  - Επιστρέφεται μήκος μηνύματος (χωρίς τον τύπο)
  - Σημαία `IPC_NOWAIT`
    - Επιστροφή αν δεν υπάρχει μήνυμα
    - Αλλιώς εμποδισμός μέχρι να εμφανιστεί μήνυμα
  - Σημαία `MSG_NOERROR`
    - Αποκοπή δεδομένων αν είναι > `msgsz`

Ουρές μηνυμάτων (9 από 10)

- Επιλογή μηνυμάτων μέσω `msgtype`
  - 0: πρώτο μήνυμα της ουράς
  - >0: πρώτο μήνυμα με τύπο msgtype
  - <0: πρώτο μήνυμα με μικρότερο τύπο
    - Πρέπει να είναι <= |`msgtype`|
  - Οι αρνητικοί τύποι επιβάλλουν προτεραιότητες
    - Ζητάμε μηνύματα με ελάχιστη προτεραιότητα
    - Από αυτά, επιλέγουμε το πιο σημαντικό

Ουρές μηνυμάτων (10 από 10)

- Οι ουρές δεν έχουν «τέλος»
  - Η ουρά δεν κλείνει, μόνο διαγράφεται
    - Αν γίνει αυτό, χάνονται τα εκκρεμή μηνύματα
  - Όσο υπάρχει, μπορεί να την ανοίξει κάποιος
    - Δεν μπορούμε να δείξουμε «τέλος αρχείου»
  - Χρειαζόμαστε μηνύματα τερματισμού
    - Με χωριστό τύπο ή με μηδενικό μήκος
  - Η τελευταία διεργασία διαγράφει την ουρά

Νήματα

Μάθημα: Λειτουργικά Συστήματα, **Ενότητα #10:** Προγραμματισμός UNIX

Διδάσκων: Γιώργος Ξυλωμένος, **Τμήμα:** Πληροφορικής

Έννοια του νήματος

- Ανεξάρτητη ροή ελέγχου σε μία διεργασία
  - Μέρος μίας διεργασίας
    - Κοινή μνήμη, αρχεία, δικαιώματα
  - Ανεξάρτητη κατάσταση εκτέλεσης
    - Μετρητής προγράμματος, καταχωρητές
  - Ανεξάρτητος χρονοπρογραμματισμός
    - Πολλά νήματα μπορεί να εκτελούνται παράλληλα
  - Οι κλασικές διεργασίες έχουν ένα νήμα

Νήματα και διεργασίες

- Πλεονεκτήματα νημάτων
  - Γρήγορη δημιουργία / τερματισμός / εναλλαγή
    - Δεν χρειάζεται αλλαγή χάρτη μνήμης
  - Πολύ γρήγορη επικοινωνία
    - Μέσω της κοινής μνήμης
- Εφαρμογές νημάτων
  - Επικάλυψη εισόδου / εξόδου / επεξεργασίας
  - Μετατροπή ασύγχρονου κώδικα σε σύγχρονο
  - Αξιοποίηση παραλληλισμού σε πολυπεξεργαστές

Συγχρονισμός νημάτων

- Τα νήματα μοιράζονται (σχεδόν) τα πάντα
  - Επικοινωνούν μέσω κοινών δομών
    - Δεν χρησιμοποιούν IPC για να αποφύγουν τον πυρήνα
  - Η παράλληλη λειτουργία απαιτεί συγχρονισμό
  - Αμοιβαίος αποκλεισμός
    - Αποκλειστική χρήση κοινών δομών
  - Μεταβλητές συνθήκης
    - Για προβλήματα τύπου παραγωγού / καταναλωτή

Νήματα POSIX

Μάθημα: Λειτουργικά Συστήματα, **Ενότητα #10:** Προγραμματισμός UNIX

Διδάσκων: Γιώργος Ξυλωμένος, **Τμήμα:** Πληροφορικής

Pthreads API

- Τυποποιημένο API νημάτων για C
  - Μέρος των προτύπων POSIX (POSIX threads)
  - Υλοποιείται στα περισσότερα συστήματα UNIX
    - Πιθανόν παράλληλα με άλλα API νημάτων
  - Διαχείριση νημάτων
  - Mutexes για αμοιβαίο αποκλεισμό
  - Μεταβλητές συνθήκης για συγχρονισμό
  - Κλειδώματα και φράγματα

Διαχείριση νημάτων (1 από 4)

- Δημιουργία νήματος
  - `int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void*), void *arg);`
  - Αρχικά η διεργασία έχει ένα μόνο νήμα
  - Με την `pthread create` δημιουργείται ένα νέο
  - `thread`: αναγνωριστικό νήματος
  - `attr`: αντικείμενο ιδιοτήτων νήματος
  - `start_routine`: συνάρτηση εκκίνησης
  - `arg`: όρισμα για τη συνάρτηση εκκίνησης

Διαχείριση νημάτων (2 από 4)

- Δημιουργία / καταστροφή ιδιοτήτων
  - `int pthread_attr_destroy(pthread_attr_t *attr);`
  - `int pthread_attr_init(pthread_attr_t *attr);`
  - Μεταβιβάζονται κατά τη δημιουργία του νήματος
  - Μπορούν να τροποποιηθούν / διαβαστούν αργότερα
    - Μπορούμε να την περιμένουμε με `join()` ή όχι
    - Πολιτική / παράμετροι χρονοπρογραμματισμού
    - Μέγεθος / διεύθυνση / υπερχείλιση στοίβας

Διαχείριση νημάτων (3 από 4)

- Τερματισμός νήματος: διάφοροι τρόποι
  - `void pthread_exit(void *value_ptr);`
    - Επιστρέφει κατάσταση τερματισμού
  - Επιστροφή αρχικής συνάρτησης νήματος
  - Κλήση της `exit()` για τερματισμό της διεργασίας
  - Επιστροφή συνάρτησης `main()` στο βασικό νήμα
    - Τερματίζει όλα τα νήματα άμεσα
    - Εκτός αν καλέσει την `pthread_exit()` πριν τερματίσει

Διαχείριση νημάτων (4 από 4)

- Αναμονή τερματισμού νήματος
  - `int pthread_join(pthread_t thread, void **value_ptr);`
  - Μπλοκάρισμα μέχρι να τερματίσει ένα νήμα
  - Μπορούμε να δούμε την κατάσταση τερματισμού
- Αποσύνδεση νήματος (δεν γίνεται `join()`)
  - `int pthread_detach(pthread_t thread);`
- Θανάτωση άλλου νήματος
  - `int pthread_cancel(pthread_t thread);`

Αμοιβαίος αποκλεισμός (1 από 3)

- Χρήση μεταβλητών mutex
  - Τύπος `pthread_mutex_t`
  - Μόνο ένα νήμα μπορεί να κλειδώσει το mutex
    - Τα υπόλοιπα μπλοκάρουν μέχρι να ξεκλειδωθεί
    - Μπορούμε να έχουμε και δοκιμαστικό κλείδωμα
  - Το mutex πρέπει να είναι ορατό σε όλους
    - Ορισμός σε καθολικό επίπεδο

Αμοιβαίος αποκλεισμός (2 από 3)

- Δημιουργία mutex: δήλωση μεταβλητής
- Αρχικοποίηση mutex
  - `int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);`
  - Αρχικά το mutex είναι ξεκλείδωτο
- Καταστροφή mutex
  - `int pthread_mutex_destroy(pthread_mutex_t *mutex);`

Αμοιβαίος αποκλεισμός (3 από 3)

- Κλείδωμα και ξεκλείδωμα mutex
  - `int pthread_mutex_lock(pthread_mutex_t *mutex);`
    - Εμποδισμός μέχρι να είναι διαθέσιμο
  - `int pthread_mutex_trylock(pthread_mutex_t *mutex);`
    - Επιστροφή αν δεν είναι διαθέσιμο
  - `int pthread_mutex_unlock(pthread_mutex_t *mutex);`
    - Ξεκλείδωμα του mutex

Μεταβλητές συνθήκης (1 από 5)

- Συγχρονισμός νημάτων
  - Τα mutex επιτρέπουν αμοιβαίο αποκλεισμό
  - Πρόβλημα παραγωγού / καταναλωτή
    - Έστω ότι έχουμε έναν κυκλικό χώρο αποθήκευσης
    - Ο παραγωγός εμποδίζεται όταν ο χώρος γεμίσει
    - Ο καταναλωτής εμποδίζεται όταν ο χώρος αδειάσει
    - Πρέπει να περιμένουμε για μία συνθήκη
    - Πρέπει να απελευθερώσουμε το mutex

Μεταβλητές συνθήκης (2 από 5)

- Μεταβλητές συνθήκης
  - Τύπος `p_thread_t`
  - Χρησιμοποιείται μόνο σε συνδυασμό με mutex
    - Πρέπει να έχουμε κλειδώσει πρώτα το mutex
  - Ένα νήμα μπορεί να εμποδιστεί σε μία συνθήκη
    - Αυτόματα ξεκλειδώνει το mutex
  - Ένα άλλο νήμα μπορεί να το αφυπνίσει
    - Θα συνεχίσει όταν κλειδώσει και το mutex

Μεταβλητές συνθήκης (3 από 5)

- Δημιουργία μεταβλητής συνθήκης
  - Δήλωση μεταβλητής `p_thread_t`
- Αρχικοποίηση μεταβλητής συνθήκης
  - `int pthread_cond_init(pthread_cond_t *cond, const pthread_condattr_t *attr);`
- Καταστροφή μεταβλητής συνθήκης
  - `int pthread_cond_destroy(pthread_cond_t *cond);`

Μεταβλητές συνθήκης (4 από 5)

- Αναμονή σε μεταβλητή συνθήκης
  - `int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);`
  - Μπλοκάρει το νήμα μέχρι να έρθει σήμα
    - Πρέπει να έχει ήδη κλειδωθεί το mutex
    - Απελευθερώνει αυτόματα το mutex στον εμποδισμό
  - Όταν ξεκινήσει κλειδώνει ξανά το mutex
    - Θα πρέπει τελικά να ξεκλειδωθεί

Μεταβλητές συνθήκης (5 από 5)

- Σήμα σε μεταβλητή συνθήκης
  - `int pthread_cond_broadcast(pthread_cond_t *cond);`
    - Ξεμπλοκάρει όλα τα νήματα της συνθήκης
  - `int pthread_cond_signal(pthread_cond_t *cond);`
    - Ξεμπλοκάρει ένα (τυχαίο) νήμα της συνθήκης
  - Πρέπει το νήμα να έχει κλειδώσει ήδη το mutex
  - Τελικά το νήμα πρέπει να ξεκλειδώσει το mutex

Τέλος Ενότητας # 10

Μάθημα: Λειτουργικά Συστήματα, **Ενότητα #10:** Προγραμματισμός UNIX

Διδάσκων: Γιώργος Ξυλωμένος, **Τμήμα:** Πληροφορικής