Κεφάλαιο 13 Σχεδίαση Λειτουργικών Συστημάτων

13.1.1 Στόχοι

4 Βασικοί στόχοι ΟΣ
1) Ορισμός αφαιρέσεων
2) Παροχή πρωτογενών λειτουργιών
3) Εξασφάλιση απομόνωσης
4) Διαχείριση του υλικού

13.1.2 Γιατί είναι δύσκολο να σχεδιαστεί ένα Ο.Σ

Συνήθως που κάνουν τη σχεδίαση ΟΣ δύσκολη
1) Τα ΟΣ είναι σχετικά προγράμματα
- Τα υποσυστήματα ενός ΟΣ αλληλεπιδρούν αρκετά με το σύστημα χωρίς με ακαθόριστες και απροβλέπτους τρόπους
2) Πρέπει να χειρίζονται τα ταυτοχρονισμό (concurrency) Υπάρχουν πολλοί users και I/O active ταυτόχρονα. Η διαχείριση ταυτοχρονισμού είναι δύσκολη. Race conditions & deadlock προβλήματα
3) Πρέπει να υπάρχει ανταπόκριση προς τους εξωτερικούς χρήστες Το ΟΣ πρέπει να λάβει μέτρα και να εγγυάται αυτό στους χρήστες
4) Συνεχώς να μοιράζονται πόροι μεταξύ των χρηστών με ασφάλεια
5) Προβλεψη τρόπων που θα θα αλλάξει το υλικό και οι εφαρμογές στο μακρινό μέλλον και να προετοιμαστούν κατάλληλα για αυτές
6) Οι ΟΣ designers δε χωρίζουν τα συστήματα τους λόγω αυξημένου κόστους

4) ΟΣ → φορτώσει → εκτέλεση σε πολλές πλατφόρμες υλικού. Διαφορετικές συσκευές I/O
5) backward compability μεταξύ ΟΣ του παρελθόντος

13.2.1 Καθοδηγητικές αρχές

Αρχή 1 Απλότητα

Οι απλές διασυνδέσεις είναι πιο εύκολα κατανοητές και υλοποιούνται χωρίς σφάλματα

Αρχή 2 Πληρότητα

Η διασύνδεση πρέπει να μπορεί να ικανοποιήσει όλες τις ανάγκες των χρηστών δηλαδή να είναι πλήρης. Το ΟΣ πρέπει να κάνει ακριβώς ότι χρειάζεται και τίποτα παραπάνω.

Αρχή 3 Αποδοτικότητα

Αν μια λειτουργία ή syscall δε μπορεί να υλοποιηθεί αποδοτικά, δεν αξίζει τον κόπο

13.2.2 Υποδείγματα

Συνήθη αρχιτεκτονικής σχεδίου δίνουν όλων των δυνατοτήτων τα συστήματα και παραδείγματα αυτού

Υποδείγματα διασυνδέσεων με το χρήστη

Όσο για διασυνδέσεις επιπέδου GUI όσο και στις διασυνδέσεις επιπέδου syscall, το πιο καλό είναι να υπάρχει ένα υπόδειγμα (ή περισσότερα) το οποίο να παρέχει τρόπο παρατήρησης της διασύνδεσης.
Οι ΟΣ design πρέπει να παρέχουν (πακέτα) βιβλιοθήκες και εργαλεία που θα τους δίνουν πρόσβαση σε διαδικασίες που δημιουργούν γραφικά στοιχεία και σύνθετα

Υποδείγματα εκτέλεσης

Αλγοριθμικό υπόδειγμα: το πρόγραμμα ξεκινάει για να εκτελέσει κάποια λειτουργία που γνωρίζει εκ των προτέρων ή μαθαίνει από τις παραμέτρους του.

```
main() {
int ...;
init();
do_something();
read(...);
do_something_else();
write(...);
keep_going();
exit(0);
}
```

Event-driven Paradigm

Το πρόγραμμα δίνει κάποιες αρχικές τιμές, εγκαθίσταται ένα παράθυρο για κάθε μια οθόνη και ύστερα περιμένει από το ΟΣ να ειδοποιήσει για το 1ο αυθόν. (Καλό για προγράμματα με υψηλή αλληλεπίδραση)

```
main() {
    mess-t msg;

    init();
    while (get_message(&msg)) {
        switch(msg.type) {
            case 1: ...;
            case 2: ...;
            case 3: ...;
        }}}
```

Υποδείγματα δεδομένων

Πού παρουσιάζεται οι δομές δεδομένων και οι I/O στον προγραμματισμό;

Fortran: όλα είναι πίνακες
UNIX: όλα ... είναι αρχεία
Windows: όλα είναι αντικείμενα
και όλα είναι εγγραφές

13ε.3 Η διασύνδεση κλήσεων συστήματος

- Αν τα αρχεία, οι διεργασίες, I/O δε βολεύουν με αρχεία/αντικείμενα θα πρέπει να έχουμε ειδικές κλήσεις για τη διαχείριση του καθενός
- Επιλογή ανάμεσα στις συνδεδεμένες και στις ασύνδετες κλήσεις
→ Οι πρώτες κλήσεις συστήματος για ανάμνηση είναι συνδεδεμένες
→ Κάποια πρωτόκολλα απομακρυσμένης πρόσβασης είναι ασύνδετα
- Visibility των syscall. Όλα τα syscall του UNIX είναι ορατά (POSIX) αν όχι: WINDOWS

13.3.1 Δομή του συστήματος

Πολυεπίπεδα συστήματα

Για ένα σύστημα, επιλογή λειτουργιών του
1) Το επίπεδο bare μηχανής πρέπει να κρύψει τις χαμηλότερες λειτουργίες του υλικού
2) Το επόμενο να διαχειρίζεται τις διακοπές, context switches και MMU
3) Διαχείριση νημάτων, scheduling και συγχρονισμός τους
4) Device driver καθένας εκτελείται ως ξεχωριστό νήμα, με δικό του state, pc, regs κτλ
5) VM
6) ...
7) Syscall handler

Εξαιρέσεις

Επικοινωνία ακρών. Αν κάτι πρέπει να γίνει από το ίδιο πρόγραμμα χρήστη η δυνατότητα να γίνεται σε χαμηλότερο επίπεδο είναι αχρείαστη

πχ Μεταφορά αρχείου δεν είναι απαραίτητο να γίνει μέσω πρωτοκόλλου αφού μπορεί να γίνει στο user level

Συστήματα πελάτη-διακομιστή βασισμένα σε μικροπυρήνα

Το ΟΣ κάνει λίγα πράγματα.
Microkernel, το μεγαλύτερο φύλλο του ΟΣ εκτελείται ως διεργασία server σε επίπεδο χρήστη. Κάθε device driver → user program → προστασία του πυρήνα
Κάθε I/O διαβάζει memory map για να βρει τα pages του
Έτσι δεν επηρεάζονται/παραβιάζονται οι δομές του πυρήνα από τα I/O
Αν δεν υπάρχει hardware support → syscall για I/O processes η οποία μαζεύει λίγα bytes (θύρα, αριθ). Ο kernel ελέγχει αν τα process κατέχει όλες τις θύρες της λίστας. Αν ναι, συνεχίζει αν αρνεί για να βεβαιωθεί η I/O

Πρόβλημα context switch → χαμηλή απόδοση CPU

Επικοινωνία συστήματος

Modern provider σαν πυρήνα με προστατευμένο τρόπο
Ξεκινάει από ελάχιστο σύστημα με μηχανισμό προστασίας και όσα προθέτουμε προστατευμένες υπηρεσίες μέχρι να φτάσουμε στην επιθυμητή κατάσταση

Νήματα πυρήνα

Ο kernel μπορεί να δημιουργεί από τέλεια νήματα ώστε, ο χρήστης όταν κάνει μια syscall, αυτό το user thread να εκτελείται σε kernel, blockάρεται και μεταβιβάζει τον έλεγχο σε kernel thread

13.3.2 Μηχανισμός και πολιτική

Διαχωρισμός των δύο.

πχ Πυρήνας scheduler κ προτεραιότητες. Ο scheduler έχουν τη μηχανισμό
Η πολιτική καθορίζει τις προτεραιότητες αυτές

πχ2 Paging. Ο μηχανισμός περιλαμβάνει τη διαχείριση MMU διατήρηση λιστών και κώδικα για μετακίνηση σελίδων. Η πολιτική καθορίζει τι θα γίνει σε page fault

πχ3 Δυνατότητα φορτωμένη υπολογιστών στον πυρήνα. Ο μηχανισμός αφορά τα τρόπο εισαγωγής και αυτών των, τα system calls, που καίνουν και δέχονται. Η πολιτική είναι το ποιος επιτρέπεται να φορτώνει υπολογιστές στον πυρήνα

13.3.3 Ορθογωνικότητα

Δυνατότητα να συνδιάζονται ανεξάρτητα διαφορετικές έννοιες
Άρα συνήθως τις απλότητες και πολυπλοκότητες.

πχ syscall clone. Δημιουργεί νέο νήμα. Δέχεται ως παράμετρο ένα bitmap ο οποίος επιτρέπει την ανεξάρτητη κοινή χρήση ή αντιγραφή του Address space, του working dir, file descriptor και σήματα. Αν σφάλμα → new process (fork like) αλλιώς νέο νήμα

πχ2 διεργασία != νήμα
Διεργασία = αποδέκτης πόρων
Νήμα = time scheduling entity

πχ3 διεργασία: fork → exec (αλλαγή μνήμης image)
νέοι χώροι διευθύνσεων

Διαφορετικά αρχεία που ανοίγονται

13.3.5 Χρόνος δέσμευσης

early binding : απλή αλλά όχι ευέλικτη
late binding : πολύπλοκη αλλά πιο ευέλικτη

Τα ΟΣ χρησιμοποιούν early binding για τις περισσότερες δομές δεδομένων αλλά χρησιμοποιούν late binding για δομές ευελιξίας

13.3.6 Στατικές ή δυναμικές δομές

στατικές δομές : καταναλώνει εύκολότερα, προγραμματίζεται εύκολότερα, πιο γρήγορες
δυναμικές δομές : ευέλικτες

13.3.7 Αναλυτική ή εμπειρική υλοποίηση

Πρώτα από κάτω προς τα πάνω έπειτα ΟΣ. Πρώτα γράφεται ο κώδικας που κρύβει το υλικό χαμηλού επιπέδου
Μετά περνάμε στον πολυπρογραμματισμό. Αν όλα τρέχουν συνεχώς, ορίζουμε πίνακες και δομές που χρειάζονται σε όλο το σύστημα και ειδικές αυτών που αναφέρονται στη διαχείριση εργασιών νημάτων και → ΜΚΔ
Τελικά το σύστημα.

13.3.4 Αφαίρεση

2 επίπεδα : εσωτερικό και εξωτερικό
Unix: filename → i-node name
Windows: filename → index στον MFT

Καταλόγοι χρησιμοποιούνται για την αντιστοίχιση εξωτερικών ονομάτων σε εσωτερικά

13.3.8 Χρήσιμες τεχνικές

Απόκρυψη του υλικού

Χωρισμός διακοπών. Οι διακοπές κάνουν δυσκολότερο τον προγραμματισμό αλλά είναι οι πόροι να τις χωρίσετε

Λύση 1 Μεταφορά διακοπών σε κάτι άλλο πχ βαρύ ίνσεκτ

Λύση 2 Μεταφορά σε μια λειτουργία unlock που αναφέρεται σε ένα mutex για το οποίο περιμένει ο αντίστοιχος οδηγός

Λύση 3 Μεταφορά σε μήνυμα σε κάποιο νήμα

Τα OS είναι σχεδιασμένα για να τρέχουν σε διαφορετικό υλικό. Επιλέγεις να υπάρχει ένα σύνολο πυρήνων εργασιών που χρησιμοποιούνται για την παραγωγή όλων των εκδόσεων

Ευελιξία

Η ευελιξία χρησιμοποιείται σαν είσοδο (button pressed) και έξοδο

πχ Χρήση μεγάλου αριθμού διακοπών. Pressed → open file, το σύστημα εξαρτάται από το ποιο και τον μαύρο και μικρό αριθμό διακοπών από τον ίνσεκτ και τον χρησιμοποιούν ως index για να βρει device driver

πχ debugger

Δυνατότητα επαναχρησιμοποίησης

Reuse ίδιο κώδικα → σύνολο των bindings και debugging μηχανών

Δυνατότητα επαναεκκίνησης

Ικανότητα του κώδικα να εκτελείται ή περισσότερες φορές ταυτόχρονα. Σε πολυεπεξεργαστές, όταν 1 CPU εκτελεί μια διαδικασία κάποια άλλη μπορεί να αρχίσει να την εκτελεί πριν η πρώτη τελειώσει. Σε αυτή την περίπτωση η περισσότερα νήματα σε διαφορετικές CPU μπορεί να εκτελούν ίδιο κώδικα ταυτόχρονα (mutex protection)

Ίδιο το πρόβλημα και στους πολυεπεξεργαστές (διακοπές)

Ουρά

Κάθε ΟΣ διαθέτει πολλές διαδικασίες που καλούνται σπάνια ή λειτουργούν με λίγα δεδομένα, ώστε η βελτιστοποίηση δεν αξίζει.
Συναρτήσεις που βρίσκονται σε κρίσιμη διαδρομή (context switch), πρέπει να γίνει προσπάθεια (assembly) για να είναι γρήγορες

Πρώτα έλεγχοι για σφάλματα

Τα πολύπλοκα ελέγχουν στην κλήση συστήματος για την εύρεση λαθών
Αν κάποιοι έλεγχοι αποτύχει → free resources → αν όχι leak πρόβλ.

13.4.1 Γιατί είναι αργά τα ΟΣ

Βραδύτητα ΟΣ → εσωτερικό πρόβλημα
Επιδιώκουμε στην προσθήκη νέων features
Plug and play → full scan at boot → όλα

13.4.2 Τι πρέπει να βελτιστοποιηθεί

Οι βελτιστοποιήσεις πρέπει να γίνονται σε σημεία που θα προκύψουν προβλήματα.
Αν η απόδοση φτάσει σε ικανοποιητικό επίπεδο, δεν αξίζει παραπάνω βελτίωση.

13.4.3 Συμβιβασμοί χώρου-χρόνου

Λιγοκατάσταση διαδικασιών με macro

Διεργασίες 1) κλήση, καταναλώνει χώρο στη στοίβα και νέων
2) ... loops → σπάνε

macro: 1) πιο αποδοτικό αφού δε χρειάζεται να πάει στη στοίβα
2) [illegible] μία φορά εκφωνείται