

Εργαστήριο Λειτουργικών Συστημάτων

Μάθημα 6^ο Εξαμήνου,
Τομέας Λογικού και Υπολογιστών

Ιωάννης Βενέτης

Παρασκευή, 18 Μαρτίου, 2011
Αίθουσα ΒΑ



Σύνοψη 6^{ης} Διάλεξης

Λειτουργικό Σύστημα Minix

Οργάνωση του Λειτουργικού Συστήματος

Minix recompilation

To System Task

Σύνοψη Μαθήματος

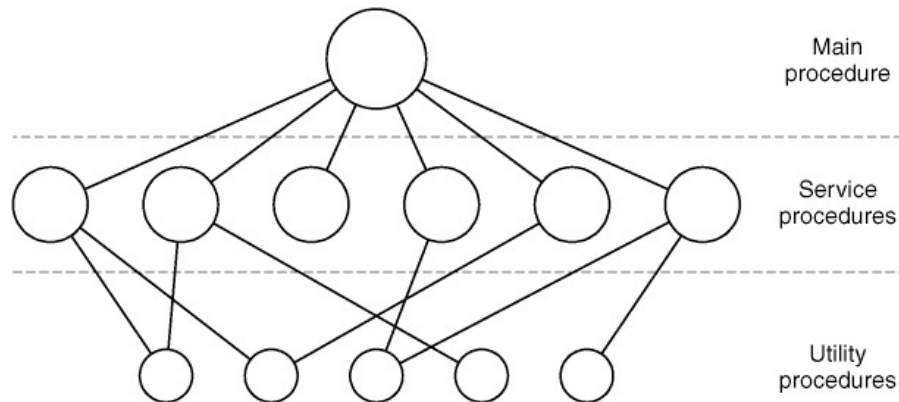
Σύνοψη Μαθήματος

3η Άσκηση

Επόμενη Διάλεξη



Ιεραρχική Δομή Συστήματος



- ▶ Τυπική δομή ενός μονολιθικού λειτουργικού συστήματος
- ▶ Ο πυρήνας προσφέρει ένα σύνολο από λειτουργίες
- ▶ Υλοποιεί διάφορες βοηθητικές συναρτήσεις



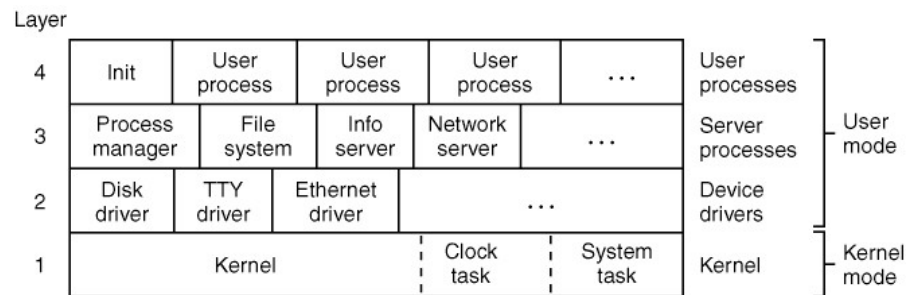
Η Εσωτερική Δομή του Συστήματος

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

- ▶ Μπορούμε να ορίσουμε μια εσωτερική δομή
- ▶ Μια συνάρτηση έχει πρόσβαση (μπορεί να χρησιμοποιήσει) στο αμέσως προηγούμενο επίπεδο (προς τα 'κάτω')

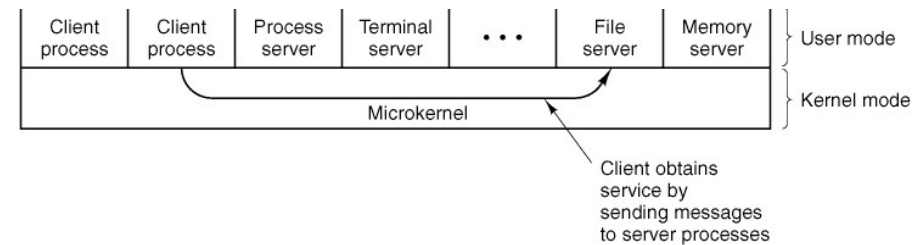


Η Εσωτερική Δομή του Minix 3



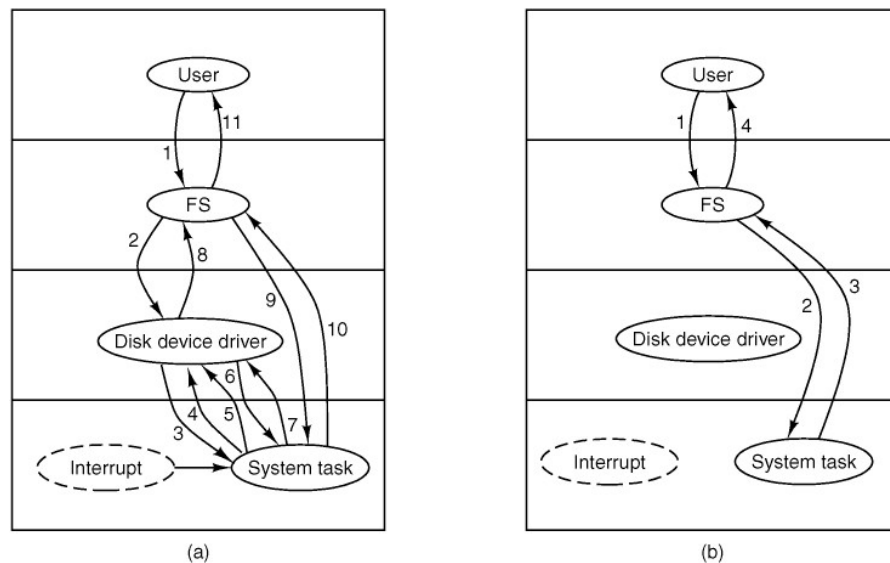
- ▶ Ορίζει 4 επίπεδα – στο ανώτερο επίπεδο εκτελούνται οι εντολές του χειριστή
- ▶ Οι ενδιάμεσες λειτουργίες υλοποιούνται από εξυπηρέτες (servers, managers . . .)
- ▶ Ο πυρήνας υλοποιεί μόνο το 1ο επίπεδο

Μοντέλο Επικοινωνίας Πελάτη – Εξυπηρέτη



- ▶ Η χρήση λειτουργιών χαμηλότερων επιπέδων γίνεται μέσω ανταλλαγής μηνυμάτων
- ▶ Η επικοινωνία ακολουθεί το μοντέλο Πελάτη – Εξυπηρέτη (Client - Server)
- ▶ Ο πυρήνας του συστήματος εξασφαλίζει την επικοινωνία
 - ▶ Διασφαλίζει την επικοινωνία μεταξύ διεργασιών 'γειτονικών' επιπέδων

Παράδειγμα Εκτέλεσης και Επικοινωνίας



Κλήσεις Συστήματος στο Minix

- ▶ Το Minix 3 ορίζει 53 βασικές κλήσεις
- ▶ Ακολουθούν το πρότυπο POSIX
 - ▶ Δεν ορίζει αντιστοιχία κλήσεων συστήματος με λειτουργίες
- ▶ Γενική Παρατήρηση
 - ▶ Σε ένα σύστημα με 1 επεξεργαστή, ένα πρόγραμμα που τρέχει στο επίπεδο των χειριστών μπορεί να εκτελέσει μόνο μια εντολή (instruction) ανά πάσα χρονική στιγμή
 - ▶ Μια κλήση στις λειτουργίες του συστήματος θα σταματήσει την ροή εκτέλεσης του προγράμματος και θα δώσει τον έλεγχο στο Λ.Σ.
- ▶ Οι κλήσεις του συστήματος στο Minix χωρίζονται ως εξής:
 - ▶ Διαχείριση διεργασιών
 - ▶ Σήματα
 - ▶ Διαχείριση αρχείων
 - ▶ Διαχείριση συστήματος αρχείων
 - ▶ Ασφάλεια / Προστασία
 - ▶ Διαχείριση ρολογιού

Διαχείριση διεργασιών

<code>pid = fork()</code>	Create a child process identical to the parent
<code>pid = waitpid(pid, &statloc, opts)</code>	Wait for a child to terminate
<code>s = wait(&status)</code>	Old version of waitpid
<code>s = execve(name, argv, envp)</code>	Replace a process core image
<code>exit(status)</code>	Terminate process execution and return status
<code>size = brk(addr)</code>	Set the size of the data segment
<code>pid = getpid()</code>	Return the caller's process id
<code>pid = getppid()</code>	Return the id of the caller's process group
<code>pid = setsid()</code>	Create a new session and return its process group id
<code>l = ptrace(req, pid, addr, data)</code>	Used for debugging



Σήματα

<code>s = sigaction(sig, &act, &oldact)</code>	Define action to take on signals
<code>s = sigreturn(&context)</code>	Return from a signal
<code>s = sigprocmask(how, &set, &old)</code>	Examine or change the signal mask
<code>s = sigpending(set)</code>	Get the set of blocked signals
<code>s = sigsuspend(sigmask)</code>	Replace the signal mask and suspend the process
<code>s = kill(pid, sig)</code>	Send a signal to a process
<code>residual = alarm(seconds)</code>	Set the alarm clock
<code>s = pause()</code>	Suspend the caller until the next signal



Διαχείριση αρχείων

<code>fd = creat(name, mode)</code>	Obsolete way to create a new file
<code>fd = mknod(name, mode, addr)</code>	Create a regular, special, or directory i-node
<code>fd = open(file, how, ...)</code>	Open a file for reading, writing or both
<code>s = close(fd)</code>	Close an open file
<code>n = read(fd, buffer, nbytes)</code>	Read data from a file into a buffer
<code>n = write(fd, buffer, nbytes)</code>	Write data from a buffer into a file
<code>pos = lseek(fd, offset, whence)</code>	Move the file pointer
<code>s = stat(name, &buf)</code>	Get a file's status information
<code>s = fstat(fd, &buf)</code>	Get a file's status information
<code>fd = dup(fd)</code>	Allocate a new file descriptor for an open file
<code>s = pipe(&fd[0])</code>	Create a pipe
<code>s = ioctl(fd, request, argp)</code>	Perform special operations on a file
<code>s = access(name, amode)</code>	Check a file's accessibility
<code>s = rename(old, new)</code>	Give a file a new name
<code>s = fcntl(fd, cmd, ...)</code>	File locking and other operations



Διαχείριση συστήματος αρχείων

<code>s = mkdir(name, mode)</code>	Create a new directory
<code>s = rmdir(name)</code>	Remove an empty directory
<code>s = link(name1, name2)</code>	Create a new entry, name2, pointing to name1
<code>s = unlink(name)</code>	Remove a directory entry
<code>s = mount(special, name, flag)</code>	Mount a file system
<code>s = umount(special)</code>	Unmount a file system
<code>s = sync()</code>	Flush all cached blocks to the disk
<code>s = chdir(dirname)</code>	Change the working directory
<code>s = chroot(dirname)</code>	Change the root directory



<code>s = chmod(name, mode)</code>	Change a file's protection bits
<code>uid = getuid()</code>	Get the caller's uid
<code>gid = getgid()</code>	Get the caller's gid
<code>s = setuid(uid)</code>	Set the caller's uid
<code>s = setgid(gid)</code>	Set the caller's gid
<code>s = chown(name, owner, group)</code>	Change a file's owner and group
<code>oldmask = umask(complmode)</code>	Change the mode mask

<code>seconds = time(&seconds)</code>	Get the elapsed time since Jan. 1, 1970
<code>s = stime(tp)</code>	Set the elapsed time since Jan. 1, 1970
<code>s = utime(file, timep)</code>	Set a file's "last access" time
<code>s = times(buffer)</code>	Get the user and system times used so far

Ο κατάλογος `/usr/src/`

- ▶ Το Minix αποτελείται από πολλαπλά επίπεδα που διαχωρίζουν τις επιμέρους λειτουργίες του.
- ▶ Η δομή `microkernel` είναι εμφανής και στην οργάνωση του κώδικα του
- ▶ Ο κώδικας του Minix βρίσκεται εξ' ολοκλήρου κάτω από τον κατάλογο `/usr/src/`
 - ▶ `/usr/src/kernel/` -- Ο κώδικας του πυρήνα του minix -- περιέχει πληροφορίες για τον μηχανισμό περάσματος μηνυμάτων, το μηχανισμό δρομολόγησης, το ρολόι του συστήματος, το μηχανισμό των system calls
 - ▶ `/usr/src/drivers/` -- Ο κώδικας για τη χρήση των συσκευών του συστήματος, όπως δίσκους, την κονσόλα, τον εκτυπωτή, κτλ
 - ▶ `/usr/src/servers/` -- Περιέχει τον κώδικα για τους βασικούς servers του συστήματος -- διαχείριση διεργασιών, σύστημα αρχείων και άλλοι μηχανισμοί
 - ▶ `/usr/src/include/` -- Ο κώδικας επικεφαλίδων του minix.

Βασικός κώδικας πυρήνα Minix

- ▶ `/usr/src/kernel/` -- Ο κώδικας του πρώτου επιπέδου του πυρήνα του minix, που περιέχει πληροφορίες για τον μηχανισμό περάσματος μηνυμάτων, το μηχανισμό δρομολόγησης, το ρολόι του συστήματος, το μηχανισμό των system calls
- ▶ `/usr/src/drivers/` -- Ο κώδικας για τη χρήση των συσκευών του συστήματος, όπως δίσκους, την κονσόλα, τον εκτυπωτή, κτλ
- ▶ `/usr/src/servers/` -- Περιέχει τον κώδικα για τους βασικούς servers του συστήματος. Αφού το Minix δεν είναι μονολιθικό, η διαχείριση διεργασιών, το σύστημα αρχείων και άλλοι μηχανισμοί, υλοποιούνται σε servers και βρίσκονται εδώ.
- ▶ `/usr/src/include/` -- Ο κώδικας επικεφαλίδων του minix.

Άλλοι κατάλογοι

- ▶ **/usr/src/lib/** Κώδικας για τις βασικές συναρτήσεις βιβλιοθηκών, όπως open, read κτλ
- ▶ **/usr/src/tools/** Αρχείο Makefile και scripts που χρησιμεύουν στο recompilation του πυρήνα
- ▶ **/usr/src/boot** Ο κώδικας για την εκίνηση του Minix
- ▶ **/usr/src/include** Ο κώδικας επικεφαλίδων του minix.
- ▶ **/usr/src/commands/** Κώδικας βασικών εντολών για τη χρήση του συστήματος, όπως cat, cp, ls, pwd, date κτλ
- ▶ **/usr/src/test/** Κώδικας από τεστ που επιβεβαιώνουν την ορθή λειτουργία του πυρήνα
- ▶ **/usr/src/etc/** Βασικό αντίγραφο scripts και config αρχείων του /etc



Χρήση κεφαλίδων στη C

`#include <filename>`

- ▶ Σημαίνει ``compiler αναζήτησε το filename στον standard φάκελο με τα header files`` (συνήθως /usr/include/ - έτσι είναι στο Minix)
- ▶ Χρησιμεύει στο να βρίσκονται τα αρχεία επικεφαλίδων για συναρτήσεις συστήματος και βιβλιοθηκών σε ένα γνωστό μέρος ώστε να μη δυσκολεύουν τον προγραμματιστή με την αναζήτηση τους

`#include ``filename```

- ▶ Σημαίνει ``compiler αναζήτησε το filename πρώτα στον τρέχον κατάλογο και μετά στον standard φάκελο με τα header files``.
- ▶ Χρησιμεύει για να μπορέσει ο προγραμματιστής να χρησιμοποιήσει δικές του βιβλιοθήκες ή βιβλιοθήκες εκτός συστήματος
- ▶ Μπορούν κάποιες βιβλιοθήκες συστήματος να παρακαμφθούν



/usr/src/include/

- ▶ Γίνεται copy στο /usr/include/μετά από κάθε recompilation
- ▶ **/usr/src/include/sys** extra header files για την υποστήριξη του POSIX
- ▶ **/usr/src/include/minix** κοινά header files όλων των επιπέδων του πυρήνα
- ▶ **/usr/src/include/ibm** header files μόνο για την πλατφόρμα των IBM PC-compatible συστημάτων
- ▶ Συμπεριλαμβάνονται extra αρχεία και κατάλογοι για την υποστήριξη extensions στο Minix (πχ δίκτυο)
- ▶ **Master Headers:** χρησιμοποιούνται από όλα τα .c αρχεία του καταλόγου όπου βρίσκονται
 - ▶ /usr/src/kernel/kernel.h
 - ▶ /usr/src/servers/mm/mm.h
 - ▶ /usr/src/servers/fs/fs.h



Σημαντικά header files

Master Headers: χρησιμοποιούνται από όλα τα .c αρχεία του καταλόγου όπου βρίσκονται

- ▶ /usr/src/kernel/kernel.h
- ▶ /usr/src/servers/mm/mm.h
- ▶ /usr/src/servers/fs/fs.h

/usr/include/unistd.h

περιέχει σταθερές και macros του POSIX καθώς και τις δηλώσεις των συναρτήσεων του POSIX και των συναρτήσεων πρόσβασης στα system calls.

/usr/include/minix/config.h

Το βασικό configuration αρχείο του kernel (customization parameters)



Συχνά συναντούμενα header files

- ▶ **type.h:** περιέχει ορισμούς δομών δεδομένων και τύπων
- ▶ **const.h:** περιέχει define statements σταθερών που συνήθως δεν αλλάζουν μεταξύ των compilations
- ▶ **glo.h:** περιέχει global δεδομένα που δηλώνονται ως extern
- ▶ **proto.h:** περιέχει τις δηλώσεις συναρτήσεων που ορίζονται σε άλλα αρχεία του ίδιου directory



Θέματα Κώδικα

- ▶ Κατά την ανάγνωση του κώδικα συναντάμε την εντολή `_PROTOTYPE`
 - ▶ Είναι ένα macro για τον ορισμό των συναρτήσεων αναλόγως την έκδοση της C που χρησιμοποιούμε για το compilation
 - ▶ Το `_PROTOTYPE(return-type function-name, (argument-type argument,...))`
 - ▶ μετατρέπεται σε `return-type function-name(argument-type argument, ...)`
- ▶ Μεταβλητές που θέλουμε να είναι καθολικές σε όλα τα αρχεία του κώδικα δηλώνονται με τη `EXTERN`
- ▶ Οι συναρτήσεις που έχουν το πρόθεμα `PRIVATE` μπορούν να χρησιμοποιηθούν μόνο από το αρχείο που τις ορίζουν
- ▶ Οι συναρτήσεις που έχουν το πρόθεμα `PUBLIC` μπορούν να χρησιμοποιηθούν από εξωτερικά αρχεία που κάνουν `include` τα αρχεία που τις ορίζουν



Compiling Minix 3

```
bash-3.00# cd /usr/src/tools
bash-3.00# make hdboot
```

- ▶ Αντικατάσταση του `/usr/include/` με το `/usr/src/include/`
- ▶ Recompile των αρχείων κάτω από τους καταλόγους `/usr/src/kernel/`, `/usr/src/servers/` και `/usr/src/drivers/` σε object files.
- ▶ Linking των object files κάθε καταλόγου σε ενιαίο εκτελέσιμο πρόγραμμα
- ▶ Ομοίως για τα υπόλοιπα χρήσιμα (εκτός πυρήνα) sources (driver συσκευής που θα φιλοξενήσει τον πυρήνα, άλλοι servers, log, tty κτλ)
- ▶ Το σύνολο των παραπάνω προγραμμάτων θα αποτελέσει το image του πυρήνα
- ▶ Θα γίνει εγκατάσταση του νέου image



Ένα απλό παράδειγμα (1)

- ▶ Θέλουμε να εξετάσουμε την εντολή `mkdir`
- ▶ Αναζητούμε τον κώδικα της εντολής

```
bash-3.00# find /usr/src -name mkdir*.c
/usr/src/commands/simple/mkdir.c
```

- ▶ Αρχικά ελέγχουμε την `main`

```
/* Main Module */
int main(argc, argv) {
...
    error=0
    while (optind < argc)
        error |= mkdir(argv[optind++]);
    return error;
}
```

- ▶ ... Βρήκαμε την εντολή σε user level και ο κώδικας της



Ένα απλό παράδειγμα (2)

- ▶ Πρέπει να εξετάσουμε την συνάρτηση `makedir()`

```
int makedir(dirname) {
    if(mkdir(dirname, DEFAULT_NODE)) ...
```

- ▶ Καλεί τη συνάρτηση `mkdir()` -- δεν ορίζεται ορίζεται στο αρχείο ...
- ▶ Θα πρέπει να δηλώνεται σε κάποιο included header file
 - ▶ Βρίσκουμε τη δήλωση της συνάρτησης στο συγκεκριμένο header file `/usr/src/include/sys/stat.h`
 - ▶ Βρίσκεται μέσα στο `include/sys/` ... -- πρόκειται για POSIX συνάρτηση
 - ▶ ... άρα θα πρέπει να ψάξουμε στην POSIX lib
- ▶ Στο `/usr/src/lib/posix` εντοπίζουμε το `_mkdir.c` που περιγράφει πως η `mkdir` κάνει το system call MKDIR στο File System (FS)
- ▶ ... Βρήκαμε ποια βασική (POSIX) library call χρησιμοποιεί



Ένα απλό παράδειγμα (3)

`/usr/src/lib/posix/_mkdir.c`

```
PUBLIC int mkdir(name, mode) {
    ...
    return (_syscall(FS, MKDIR, &m));
}
```

- ▶ ... Βρήκαμε ποιο system call καλεί αυτή η συνάρτηση βιβλιοθήκης
- ▶ Επόμενο βήμα: πρέπει να κοιτάσουμε στον πυρήνα για το syscall που στέλνεται στον FS και λέγεται MKDIR
- ▶ Ελέγχουμε το αρχείο `/usr/include/minix/callnr.h`

```
#define MKDIR          39
```

- ▶ Αναζητούμε αυτόν τον αριθμό στο αντίστοιχο `/usr/src/servers/fs/table.c`
- ▶ Διαπιστώνουμε ότι υπάρχει ένας πίνακας όπου ο αριθμός 39 αντιστοιχεί σε μια συνάρτηση `do mkdir()` που χειρίζεται το syscall



Ένα απλό παράδειγμα (4)

```
PUBLIC _PROTOTYPE (int (*call_vec[]), (void)) = {
    do_mkdir, /* 39 = mkdir */
```

- ▶ Επόμενος στόχος ... που είναι η `do_mkdir()` ? στο αρχείο `open.c`

`/usr/src/servers/fs/open.c`

```
PUBLIC int do_mkdir() {
    /* Perform the mkdir(name, mode) system call. */
    ...
}
```

- ▶ ... Βρήκαμε τη συνάρτηση που χειρίζεται το system call
- ▶ Ας πειραματιστούμε! – ΕΔΩ θα προσθέτουμε μια απλή `printf`
- ▶ Recompile (`make hdboot`) -- και επανεκκίνηση



Ένα απλό παράδειγμα (5)

```
.....
PUBLIC int do_mkdir()
{
    /* Perform the mkdir(name, mode) system call. */

    int r1, r2; /* status codes */
    ino_t dot, dotdot; /* inode numbers for . and .. */
    mode_t bits; /* mode bits for the new inode */
    char string[NAME_MAX]; /* last component of the new dir's path name */
    struct inode *rip, *ldirp;

    printf("Hello!");

    if (fetch_name(m_in.name1, m_in.name1_length, M1) != OK) return(err_code);

    /* Next make the inode. If that fails, return error code. */
    bits = I_DIRECTORY | (m_in.mode & RWX_MODES & fp->fp_umask);
    rip = new_inode(&ldirp, user_path, bits, (zone_t) 0, TRUE, string);
    if (rip == NIL_INODE ;; err_code == EEXIST) {
        put_inode(rip); /* can't make dir: it already exists */
        put_inode(ldirp);
        return(err_code);
    }
}

"open.c" 565L, 18823C 344,16 61%
```



Ένα απλό παράδειγμα (6)

```
installboot -image image ../kernel/kernel \  
  ../servers/pm/pm \  
  ../servers/fs/fs \  
  ../servers/rs/rs \  
  ../servers/ds/ds \  
  ../drivers/tty/tty \  
  ../drivers/memory/memory \  
  ../drivers/log/log \  
  ../servers/init/init  
text  data  bss  size  
24176  3384  44384  71944  ../kernel/kernel  
21296  3108  93940  118344  ../servers/pm/pm  
41536  5232  5019704  5066472  ../servers/fs/fs  
6848   840   20388   28076   ../servers/rs/rs  
3280   464   1808    5552   ../servers/ds/ds  
27072  5696  48104  80872  ../drivers/tty/tty  
6144  287784  3068   296996  ../drivers/memory/memory  
5968   572   63280  69820  ../drivers/log/log  
7056   2412  1356   10824  ../servers/init/init  
-----  
143376  309492  5296032  5748900  total  
exec sh mkboot hdboot  
install image /dev/c0dd0p0s0:/boot/image/3.1.2ar0  
Done.  
/usr/src/tools# _
```



Ένα απλό παράδειγμα (7)

```
Starting services: random lance inet printer.  
Starting daemons: update cron syslogd.  
Starting networking: dhcpd nonamed.  
Alarm call  
Unable to obtain an IP address.  
Local packages (start): sshd done.  
/dev/rescue is read-write mounted on /boot/rescue  
Minix Release 3 Version 1.2a (console)  
145-116-229-112.uilenstede.casema.nl login: root  
  
To install X Windows, run 'packman' with the install CD still in the  
drive. To start X Windows after you have installed it, login as root  
and type: 'xdm'. For more information about configuring X Windows, see  
www.minix3.org.  
  
If you do not have sufficient memory to run X Windows, standard MINIX 3  
supports multiple virtual terminals. Just use ALT+F1, F2, F3 and F4 to  
navigate among them.  
  
To get rid of this message, edit /etc/motd.  
  
~# mkdir tmp  
Hello!~# _
```



Βασική Δομή του Λειτουργικού Συστήματος

- ▶ Είδαμε πως μπορούμε να κάνουμε αλλαγές στον πυρήνα του Λ.Σ.
- ▶ Εξετάσαμε την λειτουργία μιας χαρακτηριστικής εντολής
 1. Βρήκαμε τον κώδικα της εντολής (εκτελέσιμο πρόγραμμα)
 2. Διαπιστώσαμε ότι χρησιμοποιεί μια συνάρτηση του συστήματος αρχείων
 3. Ανατρέξαμε στον κώδικα του συστήματος αρχείων
 4. Εντοπίσαμε την συνάρτηση που χειρίζεται η εντολή
- ▶ Γενικότερα
 - ▶ Ένα πρόγραμμα (εκτελέσιμο) χρησιμοποιεί λειτουργίες του συστήματος
 - ▶ Οι λειτουργίες υλοποιούνται από συγκεκριμένες συναρτήσεις
 - ▶ Οι συναρτήσεις βασίζονται σε ένα σύνολο από βοηθητικές συνιστώσες
- ▶ Υπάρχει μια ιεραρχική δομή στην εκτέλεση των λειτουργιών ...



Γενικά περί System Task

- ▶ Οι βασικές συνιστώσες του Minix είναι ανεξάρτητες διεργασίες
 - ▶ Εκτελούνται εκτός του πυρήνα
 - ▶ Δεν έχουν άμεση πρόσβαση στους πίνακες του πυρήνα
 - ▶ Δεν επιτρέπεται να χειριστούν τις μονάδες εισόδου/εξόδου
- ▶ Παράδειγμα: Η κλήση του συστήματος *fork* υλοποιείται από τον διαχειριστή διεργασιών
 - ▶ Όταν δημιουργείται μια νέα διεργασία, πρέπει να ενημερωθεί ο πυρήνας – για λόγους χρονοπρογραμματισμού
 - ▶ Πώς μπορεί να επικοινωνήσει ο διαχειριστής διεργασιών με τον πυρήνα ;
- ▶ Λύση: Υλοποίηση του System Task
 - ▶ Ο πυρήνας προσφέρει μια ομάδα υπηρεσιών προς τους οδηγούς και τους διαχειριστές για την πρόσβαση στους πίνακες του πυρήνα, χειρισμό εισόδων/εξόδων κλπ.
 - ▶ Αυτές οι υπηρεσίες δεν είναι προσβάσιμες από απλές διεργασίες



Σκοπός του System Task (1)

- ▶ Το System Task είναι compiled μαζί με τον πυρήνα του συστήματος (δηλ. ίδιο εκτελέσιμο)
- ▶ Στην πραγματικότητα είναι διαφορετική διεργασία και χρονοπρογραμματίζεται διαφορετικά
- ▶ Ο σκοπός του System Task είναι η εξυπηρέτηση αιτήσεων για πρόσβαση στις ειδικές υπηρεσίες του πυρήνα
 - ▶ Αποκλειστικά προς τους οδηγούς και τους διαχειριστές
- ▶ Στα Λ.Σ. με μονολιθικό πυρήνα, ο όρος 'Κλήση Συστήματος' αναφέρεται σε όλες τις υπηρεσίες που προσφέρει ο πυρήνας
- ▶ Στα σύγχρονα Λ.Σ., το πρότυπο POSIX περιγράφει τις ελάχιστες κλήσεις συστήματος που προσφέρει ο πυρήνας
 - ▶ Μπορεί να προσφέρει επεκτάσεις στο πρότυπο, υλοποιώντας μια συλλογή συναρτήσεων που είναι πιο εύχρηστες και πιο κατανοητές



Σκοπός του System Task (2)

- ▶ Επίσης, στα μονολιθικά Λ.Σ., όλες οι κλήσεις του συστήματος και οι επεκτάσεις
 - ▶ Φαίνονται σαν διαφορετικές βιβλιοθήκες
 - ▶ Είναι μέρος του πυρήνα
- ▶ Στα Λ.Σ. με μικρο-πυρήνα -- Minix
 - ▶ Συνιστώσες του Λ.Σ. που υλοποιούν βασικές κλήσεις του συστήματος εκτελούνται εκτός πυρήνα
 - ▶ Εξακολουθούμε να χρησιμοποιούμε τον όρο 'κλήση συστήματος POSIX'
 - ▶ Αυτό όμως από μόνο του δεν αρκεί για να καταλάβουμε το σημείο που υλοποιείται η κλήση
 - ▶ Και εφόσον ορισμένες υλοποιούνται εκτός πυρήνα – δεν αρκεί για να αποκτήσουν άμεση πρόσβαση στις πληροφορίες του πυρήνα



Οργάνωση του System Task

- ▶ Το System Task αναγνωρίζει 28 τύπους μηνυμάτων
 - ▶ κάθε ένα από αυτά μπορεί να θεωρηθεί μια κλήση του πυρήνα
- ▶ Σε ορισμένες περιπτώσεις ο ίδιος τύπος μηνύματος εμφανίζεται με διαφορετικά ονόματα με την χρήση macros
- ▶ Σε άλλες περιπτώσεις μια συγκεκριμένη συνάρτηση επεξεργάζεται διαφορετικούς τύπους μηνυμάτων
- ▶ Η βασική δομή του System Task είναι απλή (αρχείο *kernel/system.c*)
- ▶ Αρχικοποιεί τις εσωτερικές δομές (συνάρτηση *initialize*)
- ▶ Εκτελεί ένα ατέρμονο βρόγχο (συνάρτηση *sys_task*)
 - ▶ Δέχεται νέα μηνύματα
 - ▶ Εκτελεί τις αντίστοιχες συναρτήσεις
 - ▶ Απαντάει με τα αποτελέσματα
- ▶ Περιέχει ορισμένες βοηθητικές συναρτήσεις



Κλήσεις Πυρήνα (1)

- ▶ Οι κλήσεις του πυρήνα μπορούν να κατηγοριοποιηθούν σύμφωνα με τον διαχειριστή που τις καλεί
 - ▶ από τις ακόλουθες κλήσεις, οι *sys_fork*, *sys_exec*, *sys_exit*, *sys_trace* αντιστοιχούν σε κλήσεις συστήματος που ορίζει το POSIX
 - ▶ η *sys_nice* δεν ορίζεται από το POSIX
 - ▶ Δ.Δ. = Διαχειριστής Διεργασιών

Μήνυμα	Αποστολέας	Περιγραφή
<i>sys_fork</i>	Δ.Δ.	Μια διεργασία έγινε fork
<i>sys_exec</i>	Δ.Δ.	Ορισμός του δείκτη του stack μετά από μια κλήση exec
<i>sys_exit</i>	Δ.Δ.	Τερματισμός διεργασίας
<i>sys_nice</i>	Δ.Δ.	Αλλαγή προτεραιότητας χρονοπρογραμματισμού
<i>sys_trace</i>	Δ.Δ.	Εκτέλεση λειτουργίας για την κλήση ptrace



Κλήσεις Πυρήνα (2)

- ▶ Η κλήση `sys_privctl` χρησιμοποιείται από τον Reincarnation Server
 - ▶ είναι ο διαχειριστής που είναι υπεύθυνος για την μετατροπή 'απλών' διεργασιών σε διεργασίες τους συστήματος
- ▶ Η `sys_privctl` αλλάζει τα δικαιώματα μιας διεργασίας για να μπορεί να κάνει κλήσεις στον πυρήνα
- ▶ Χρησιμοποιείται όταν εκτελεστεί ένας οδηγός εισόδου / εξόδου και κάποιος διαχειριστής
 - ▶ δεν βρίσκεται στο image του συστήματος – δηλ. δεν φορτώνεται απευθείας κατά την εκκίνηση του συστήματος
 - ▶ για δεύτερη φορά (λόγω προβλήματος)

Μήνυμα	Αποστολέας	Περιγραφή
<code>sys_privctl</code>	R.S.	Ορισμός / Αλλαγή δικαιωμάτων



Κλήσεις Πυρήνα (3)

- ▶ Οι ακόλουθες κλήσεις του πυρήνα έχουν σχέση με τα σήματα
 - ▶ Σ.Α. = Σύστημα Αρχείων / Διαχειριστής Αρχείων
 - ▶ ΠΥ – Κονσόλα / Τερματικό

Μήνυμα	Αποστολέας	Περιγραφή
<code>sys_kill</code>	Δ.Δ., Σ.Α., ΠΥ	Αποστολή σήματος σε διεργασία μετά από μια κλήση kill
<code>sys_getksig</code>	Δ.Δ.	Ο Δ.Δ. ελέγχει για σήματα που εκκρεμούν
<code>sys_endsig</code>	Δ.Δ.	Ο Δ.Δ. ολοκλήρωσε τον έλεγχο για εκκρεμή σήματα
<code>sys_sigsend</code>	Δ.Δ.	Αποστολή σήματος σε διεργασία
<code>sys_sigreturn</code>	Δ.Δ.	Αποκατάσταση διεργασίας μετά από την παραλαβή σήματος



Κλήσεις Πυρήνα (4)

- ▶ Αυτή η κατηγορία κλήσεων του πυρήνα είναι ιδιαιτερότητα του Minix
 - ▶ Προσφέρουν πρόσβαση στους οδηγούς εισόδου/εξόδου

Μήνυμα	Αποστολέας	Περιγραφή
<code>sys_irqctl</code>	Οδηγοί	Ενεργοποίηση, απενεργοποίηση και ρύθμιση interrupt
<code>sys_devio</code>	Οδηγοί	Ανάγνωση/Εγγραφή σε μια θύρα εισόδου/εξόδου
<code>sys_sdevio</code>	Οδηγοί	Ανάγνωση/Εγγραφή string σε μια θύρα εισόδου/εξόδου
<code>sys_vdevio</code>	Οδηγοί	Ανάγνωση/Εγγραφή διάνυσματος από αιτήσεις εισόδου/εξόδου
<code>sys_int86</code>	Οδηγοί	Πρόσβαση στις κλήσεις του BIOS



Κλήσεις Πυρήνα (5)

- ▶ Οι ακόλουθες κλήσεις του πυρήνα έχουν σχέση με την διαχείριση της μνήμης

Μήνυμα	Αποστολέας	Περιγραφή
<code>sys_newmap</code>	Δ.Δ.	Αρχειοποίηση πίνακα μνήμης μιας διεργασίας
<code>sys_segctl</code>	Οδηγοί	Προσθήκη τμήματος και επιστροφή επιλογή (για ανάγνωση)
<code>sys_memset</code>	Δ.Δ.	Εγγραφή χαρακτήρα σε θέση μνήμης
<code>sys_umap</code>	Οδηγοί	Μετατροπή εικονικής διεύθυνσης μνήμης σε πραγματική
<code>sys_vircopy</code>	Σ.Α., Οδηγοί	Αντιγραφή με την χρήση εικονικών διευθύνσεων
<code>sys_physcopy</code>	Οδηγοί	Αντιγραφή με την χρήση πραγματικών διευθύνσεων μνήμης
<code>sys_vircopy</code>	Όλοι	Διάνυσμα από αιτήσεις vircopy
<code>sys_physcopy</code>	Όλοι	Διάνυσμα από αιτήσεις physcopy



Κλήσεις Πυρήνα (6)

- ▶ Η κλήση `sys_times` αντιστοιχεί στην κλήση του συστήματος `times` που ορίζει το POSIX
- ▶ Η κλήση `sys_alarm` έχει 'κάποια' σχέση με την κλήση του συστήματος `alarm` που ορίζει το POSIX -- δεν έχει σχέση με τη λειτουργία κουδουνιού που προσφέρεται στις απλές διεργασίες
- ▶ Η κλήση `sys_alarm` μπορεί να κληθεί από τον Δ.Δ. όταν αντιμετωπιστεί σοβαρό πρόβλημα, ή από τον χειριστή όταν πατήσει CTRL-ALT-DEL

Μήνυμα	Αποστολέας	Περιγραφή
<code>sys_times</code>	Δ.Δ.	Επιστροφή χρόνων uptime & proctime
<code>sys_setalarm</code>	Δ.Δ., Σ.Α., Οδηγοί	Χρόνοπρογραμματισμός σύγχρονου κουδουνιού
<code>sys_abort</code>	Δ.Δ., ΠΥ	Panic: Το Λ.Σ. δεν μπορεί να συνεχίσει
<code>sys_getinfo</code>	Όλοι	Επιστροφή πληροφοριών συστήματος

Δομή κώδικα του System Task

- ▶ Το System Task υλοποιείται από το `system.h` και `system.c`
 - ▶ Τα αρχεία είναι στον φάκελο `/usr/src/kernel`
- ▶ Το Minix δεν είναι **μόνο** για γενική χρήση
 - ▶ Στοχεύει και σε ενσωματωμένα συστήματα
- ▶ Επιτρέπει την ενεργοποίηση/απενεργοποίηση των κλήσεων του πυρήνα – μείωση μεγέθους εκτελέσιμου
- ▶ Το αρχείο `kernel/config.h` ορίζει ποιες κλήσεις είναι ενεργές

```
#define USE_FORK    1 // fork a new process
#define USE_NEWMAP 1 // set a new memory map
#define USE_EXEC    1 // update process after execute
#define USE_EXIT    1 // clean up after process exit
#define USE_NICE    1 // change scheduling priority
```

- ▶ Αν αφαιρέσουμε μια δήλωση, όλα τα τμήματα του κώδικα που σχετίζονται με την συγκεκριμένη κλήση δεν θα γίνουν compile

Ορισμός Κλήσεων Πυρήνα

- ▶ Όλες οι κλήσεις του πυρήνα ορίζονται στο αρχείο `com.h`
 - ▶ Βρίσκεται στον φάκελο `/usr/src/include/minix`

```
#define KERNEL_CALL 0x600 // base for kernel calls
                          // to SYSTEM

#define SYS_FORK      (KERNEL_CALL + 0) // sys_fork()
#define SYS_EXEC      (KERNEL_CALL + 1) // sys_exec()
#define SYS_EXIT      (KERNEL_CALL + 2) // sys_exit()
#define SYS_NICE      (KERNEL_CALL + 3) // sys_nice()

#define NR_SYS_CALLS 28 // number of system calls
```

- ▶ Αν θέλουμε να προσθέσουμε μια νέα κλήση, πρέπει να την ορίσουμε εδώ
- ▶ Ανεξάρτητα από το αν θα την απενεργοποιήσουμε ή όχι

Αρχείο Επικεφαλίδας System Task

- ▶ Το αρχείο `system.h` ορίζει τις συναρτήσεις που χειρίζονται κάθε κλήση του πυρήνα
 - ▶ Για την κλήση `sys_xxx` αντιστοιχεί την συνάρτηση `do_xxx`

```
/* Default handler for unused kernel calls. */
_PROTOTYPE( int do_unused, (message *m_ptr) );
_PROTOTYPE( int do_exec, (message *m_ptr) );
_PROTOTYPE( int do_fork, (message *m_ptr) );
_PROTOTYPE( int do_trace, (message *m_ptr) );
_PROTOTYPE( int do_nice, (message *m_ptr) );
_PROTOTYPE( int do_copy, (message *m_ptr) );
#define do_vircopy do_copy
#define do_physcopy do_copy
```

- ▶ Η συνάρτηση `do_unused` είναι μια 'κενή' συνάρτηση
- ▶ Η συνάρτηση `do_copy` αντιστοιχεί επίσης στις κλήσεις `sys_vircopy` `sys_physcopy`

Αντιστοίχιση Συναρτήσεων – Κλήσεις Πυρήνα

- ▶ Για την αντιστοίχιση συναρτήσεων με κλήσεις συστήματος, στο *system.c* ορίζεται ο πίνακας *call_vec()*

```
PUBLIC int (*call_vec[NR_SYS_CALLS])(message *m_ptr)
```

- ▶ Ανατρέχουμε στον πίνακα με βάση τον αριθμό της κλήσης (όπως ορίζεται στο μήνυμα) – αυτή η τεχνική χρησιμοποιείται και σε άλλα σημεία του Minix
- ▶ Προσθέτουμε στοιχεία στον πίνακα με την χρήση του macro *map*

```
#define map(call_nr, handler) \
{extern int dummy[NR_SYS_CALLS>(unsigned)(call_nr-KERNEL_CALL) ? 1:-1]; \
 call_vec[(call_nr-KERNEL_CALL)] = (handler)
```

- ▶ Αν προσπαθήσουμε να αντιστοιχίσουμε μια συνάρτηση σε μια κλήση που δεν υπάρχει, το macro επιστρέφει ένα πίνακα με αρνητικό μέγεθος ... ο compiler θα βγάλει λάθος



Αρχικοποίηση System Task

- ▶ Η αρχικοποίηση γίνεται με την συνάρτηση *initialize* (αρχείο *system.c*)

```
PRIVATE void initialize(void){
    for (i=0; i<NR_SYS_CALLS; i++)
        call_vec[i] = do_unused;

    /* Process management. */
    map(SYS_FORK, do_fork);
    map(SYS_EXEC, do_exec);
    map(SYS_EXIT, do_exit);
    map(SYS_NICE, do_nice);
    ...
}
```

- ▶ Αρχικά αντιστοιχεί σε όλες τις κλήσεις του πυρήνα την συνάρτηση *do_unused* (κενή συνάρτηση)
- ▶ Στην συνέχεια αντιστοιχεί μια προς μια τις συναρτήσεις



Βασική Λειτουργία του System Task

- ▶ Η βασική λειτουργία ορίζεται στην συνάρτηση *sys_task*
 - ▶ Ατέρμονος βρόγχος

```
PUBLIC void sys_task(){
    static message m;
    register int result;
    register struct proc *caller_ptr;
    unsigned int call_nr;
    int s;

    /* Initialize the system task. */
    initialize();

    while (TRUE) {
        ...
    }
}
```



Βασικός Βρόγχος System Task

```
while (TRUE) {
    // Get work. Block until a message arrives
    receive(ANY, &m);
    call_nr = (unsigned) m.m_type - KERNEL_CALL;
    caller_ptr = proc_addr(m.m_source);

    /* See if the caller made a valid request and try to handle it. */
    if (!(priv(caller_ptr)->s_call_mask & (1<<call_nr))) {
        kprintf("SYSTEM: request %d from %d denied.\n", call_nr,m.m_source);
        result = ECALDENIED; /* illegal message type */
    } else if (call_nr >= NR_SYS_CALLS) { /* check call number */
        kprintf("SYSTEM: illegal request %d from %d.\n", call_nr,m.m_source);
        result = EBADREQUEST; /* illegal message type */
    } else
        result = (*call_vec[call_nr])(&m); /* handle the kernel call */

    if (result != EDONTREPLY) {
        m.m_type = result; /* report status of call */
        if (OK != (s=lock_send(m.m_source, &m)))
            kprintf("SYSTEM, reply to %d failed: %d\n", m.m_source, s);
    }
}
```

- ▶ Η συνάρτηση *priv* ελέγχει αν η διεργασία αποστολέας έχει πρόσβαση στην συγκεκριμένη κλήση του πυρήνα



Σύνοψη 6^{ης} Διάλεξης

Λειτουργικό Σύστημα Minix

Οργάνωση του Λειτουργικού Συστήματος

Minix recompilation

To System Task

Σύνοψη Μαθήματος

Σύνοψη Μαθήματος

3η Άσκηση

Επόμενη Διάλεξη

Στο σημερινό μάθημα είδαμε

- ▶ Δομή κώδικα του Minix με έμφαση στα header files
- ▶ Πως κάνουμε Minix kernel recompilation
- ▶ Ένα σύντομο παράδειγμα της χρησιμότητας γνώσης της δομής των sources
- ▶ To System Task στο Minix 3
 - ▶ Επισκόπηση
 - ▶ Σκοπός
 - ▶ Δομή – Οργάνωση
 - ▶ Υλοποίηση

Βιβλιογραφία

- ▶ Βιβλίο “Operating Systems Design and Implementation, Third Edition” (Andrew S. Tanenbaum)
 1. Κεφάλαιο 1: Εισαγωγή
 - ▶ Παράγραφος 1.4 System Calls
 - ▶ Παράγραφος 1.5 Operating System Structure
 2. Κεφάλαιο 2: Processes
- ▶ Βιβλίο ‘Σύγχρονα Λειτουργικά Συστήματα’ (A.Tanenbaum)
 1. Κεφάλαιο 1: Εισαγωγή
 - ▶ Παράγραφος 1.6 Κλήσεις Συστήματος
 - ▶ Παράγραφος 1.7 Η Δομή των Λειτουργικών Συστημάτων
 2. Κεφάλαιο 2: Διεργασίες
 3. Κεφάλαιο 10: Μελέτη Περίπτωσης 1: Unix και Linux
 - ▶ Παράγραφος 10.3 Οι Διεργασίες στο UNIX

Τρίτη Άσκηση

- ▶ Αφορά το λειτουργικό σύστημα MINIX
- ▶ Οι απαντήσεις πρέπει να ακολουθούν συγκεκριμένη μορφή
- ▶ Ατομική άσκηση
- ▶ Παράδοση γίνεται με την χρήση του εργαλείου *submit* μέσω του συστήματος **zenon**
- ▶ Μέχρι **Δευτέρα 16 Απριλίου, ώρα 23:59**
- ▶ Αν παρατηρηθεί αντιγραφή, τότε όλοι όσοι συνεργάστηκαν και εμπλέκονται στην αντιγραφή, θα μηδενίζονται στο μάθημα

1ο Πρόβλημα

Περιγράψτε τη διαδικασία που πρέπει να ακολουθήσουμε για να δημιουργήσουμε ένα νέο image του minix όπου η κλήση πυρήνα `SYS_TRACE` να είναι απενεργοποιημένη.

2ο Πρόβλημα

Αναπτύξτε μια νέα κλήση του πυρήνα (kernel call) που να επιστρέφει την έκδοση του πυρήνα (`OS_VERSION`). Υλοποιήστε μια απλή εντολή που χρησιμοποιεί την νέα κλήση του πυρήνα που αναπτύξατε. Τα αρχεία που θα παραδώσετε θα πρέπει να είναι καλά δομημένα (με την χρήση tab, κλπ.) και καλά σχολιασμένα.

3ο Πρόβλημα

Αναπτύξτε μια νέα κλήση του πυρήνα (kernel call) που να δέχεται έναν ακέραιο ως παράμετρο και να επιστρέφει το AM σας πολλαπλασιασμένο με τον ακέραιο που δώθηκε. Υλοποιήστε μια απλή εντολή που χρησιμοποιεί την νέα κλήση του πυρήνα που αναπτύξατε. Τα αρχεία που θα παραδώσετε θα πρέπει να είναι καλά δομημένα (με την χρήση tab, κλπ.) και καλά σχολιασμένα.

Επόμενη Διάλεξη

- ▶ Ο Πυρήνας του MINIX 3 και το System Task
- ▶ Επανάληψη από μάθημα 'Λειτουργικά Συστήματα Ι'
 1. Κεφάλαιο 2: Διεργασίες
- ▶ Βιβλίο 'Σύγχρονα Λειτουργικά Συστήματα' (A.Tanenbaum)
 1. Κεφάλαιο 2: Διεργασίες
- ▶ Βιβλίο 'Operating Systems Design and Implementation, Third Edition' (Andrew S. Tanenbaum)
 1. Κεφάλαιο 2: Processes