

Εργαστήριο Λειτουργικών Συστημάτων

Μάθημα 6^{ου} Εξαμήνου,

Τομέας Λογικού και Υπολογιστών

Ιωάννης Χατζηγιαννάκης

Πέμπτη 6 Μαΐου 2010
Αίθουσα ΒΑ



Θέματα Σχεδιασμού / Αρχιτεκτονικής

- ▶ Η διαχείριση της μνήμης στο Minix 3 είναι ιδιαίτερα απλή
 - ▶ Δεν υλοποιεί μηχανισμό σελιδοποίησης (paging)
 - ▶ Υλοποιεί έναν απλό μηχανισμό swapping – απενεργοποιημένο
- ▶ Συγκεκριμένοι λόγοι για αυτή την στρατηγική
 1. Απλούστευση συστήματος – μείωση κώδικα
 2. Απόφαση προηγούμενων εκδόσεων
 3. Ευκολία στην μεταφορά σε ενσωματωμένα συστήματα
- ▶ Αν θελήσουμε να υλοποιήσουμε έναν μηχανισμό σελιδοποίησης ή κάποιον σύνθετο μηχανισμό swapping
 - ▶ Ο διαχωρισμός των λειτουργιών στον διαχειριστή διεργασιών και στον πυρήνα διευκολύνει σε μεγάλο βαθμό
 - ▶ Ο πυρήνας ασχολείται με θέματα ανάθεσης μνήμης, δημιουργίας διεργασιών κλπ. σε χαμηλό επίπεδο
 - ▶ Ο διαχειριστής διεργασιών ασχολείται με την υλοποίηση των μηχανισμών – δεν χρειάζεται να κάνουμε αλλαγές στον πυρήνα



Τμήματα Διεργασιών

- ▶ Οι διεργασίες στο Minix 3 χωρίζονται σε τρία τμήματα
 1. text – ο κώδικα ('εκτελέσιμο'), δεν μεταβάλλεται
 2. data -- τα δεδομένα
 3. stack – το stack εντολών
- ▶ Το μέγεθος του κάθε τμήματος μετριέται σε **clicks**
 - ▶ Κάθε click είναι 1024 bytes
- ▶ Στο αρχείο `/usr/src/include/minix/type.h` ορίζεται η δομή `mem_map` που περιγράφει το κάθε τμήμα
 - ▶ Εικονική θέση του τμήματος στην μνήμη (σε clicks)
 - ▶ Πραγματική θέση του τμήματος στην μνήμη (σε clicks)
 - ▶ Μέγεθος τμήματος(σε clicks)



Τμήματα Διεργασιών – Θέματα Υλοποίησης (1)

- ▶ Στο αρχείο `/usr/src/include/minix/type.h` ορίζεται η δομή `mem_map` που περιγράφει το κάθε τμήμα
- ▶ Η εικονική θέση και το μέγεθος μετριοίται σε *unsigned int*, η πραγματική θέση σε *unsigned long*

```
struct mem_map {  
    vir_clicks mem_vir;      /* virtual address */  
    phys_clicks mem_phys;   /* physical address */  
    vir_clicks mem_len;     /* length */  
};
```

- ▶ Η εικονική θέση και το μέγεθος μετριοίται σε *unsigned int*, η πραγματική θέση σε *unsigned long*



Τμήματα Διεργασιών – Θέματα Υλοποίησης (2)

- ▶ Η πληροφορία για τη θέση του κάθε τμήματος και το μέγεθος διατηρείται στον πυρήνα και στον διαχειριστή διεργασιών
- ▶ Ορίζεται ως ένας πίνακας
 - ▶ Η θέση 0 περιέχει το text (T)
 - ▶ Η θέση 1 περιέχει το data (D)
 - ▶ Η θέση 2 περιέχει το stack (S)

Πληροφορία στον Πυρήνα

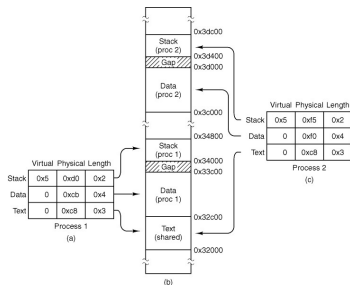
```
struct mem_map p_mmap[NR_LOCAL_SEGS];
```

Πληροφορία στον Διαχειριστή Διεργασιών

```
struct mem_map mp_seg[NR_LOCAL_SEGS];
```

- ▶ Η σταθερά NR_LOCAL_SEGS είναι 3 και ορίζεται στο αρχείο `/usr/src/include/minix/const.h`
 - ▶ Επίσης ορίζει τις 3 βοηθητικές σταθερές (T, D, S)

Κοινά Τμήματα Μνήμης – Παράδειγμα



Ελεύθερη Μνήμη – Οπές Μνήμης (Memory Holes)



- ▶ Όταν τερματίζει μια διεργασία ελευθερώνεται η μνήμη
 - ▶ Χρήση της κλήσης του συστήματος `exit`
- ▶ Αντίστοιχα όταν δημιουργείται μια νέα διεργασία πρέπει να εντοπίσουμε ένα αρκετά μεγάλο ελεύθερο τμήμα στην μνήμη
 - ▶ Καταγράφουμε τις ελεύθερες θέσεις μνήμης – οπές (holes)

Λίστα Οπών Μνήμης – Εσωτερικές Δομές

- ▶ Στο αρχείο `/usr/src/include/minix/type.h` ορίζεται η δομή `hole` που περιγράφει την κάθε οπή

```
struct hole {  
    struct hole *h_next; // pointer to next entry  
    phys_clicks h_base; // where does the hole be  
    phys_clicks h_len; // how big is the hole?  
};
```

- ▶ Στο αρχείο `/usr/src/servers/pm/alloc.c` γίνεται η διαχείριση των οπών και υλοποιούνται οι μηχανισμοί διαχείρισης της μνήμης
- ▶ **Εσωτερικά** διατηρούνται ορισμένες μεταβλητές

```
#define NIL_HOLE (struct hole *) 0  
PRIVATE struct hole *hole_head; // head of list  
PRIVATE struct hole hole[_NR_HOLES];
```

Μηχανισμοί Διαχείριση Μνήμης

Το αρχείο `/usr/src/servers/pm/alloc.c` προσφέρει τις εξής συναρτήσεις:

1. `alloc_mem` – Δέσμευση μνήμης συγκεκριμένου μεγέθους
2. `free_mem` – Αποδέσμευση μνήμης
3. `init_mem` – Αρχικοποίηση μνήμης κατά την εκκίνηση του διαχειριστή διεργασιών

```
PUBLIC phys_clicks alloc_mem(phys_clicks clicks);
```

```
PUBLIC void free_mem(base, clicks);  
phys_clicks base; // base address of block to free  
phys_clicks clicks; // number of clicks to free
```

```
PUBLIC void mem_init(chunks, free);  
struct memory *chunks; // list of free memory chunk  
phys_clicks *free; // memory size summaries
```

Δομή Πίνακα Διεργασιών

- ▶ Στο αρχείο `/usr/src/server/pm/mproc.h` ορίζεται η δομή του πίνακα των διεργασιών (για τον διαχειριστή διεργασιών)
- ▶ Συμπληρωματικές πληροφορίες για τις διεργασίες τηρούνται στον πυρήνα

```
EXTERN struct mproc {  
    // points to text, data, stack  
    struct mem_map mp_seg[NR_LOCAL_SEGS];  
    // process id  
    pid_t mp_pid;  
    ...  
    char mp_name[PROC_NAME_LEN]; // process name  
} mproc[NR_PROCS];
```

Ανάθεση Μνήμης

- ▶ Η ανάθεση μνήμης – χρησιμοποιείται αποκλειστικά τις κλήσεις του συστήματος `fork` και `exec`
 - ▶ Η μνήμη που έχει ανατεθεί δεν μπορεί να μεγαλώσει/μικρύνει κατά την διάρκεια ζωής της διεργασίας
- ▶ Εάν πρόκειται για μια διεργασία που 'μοιράζεται' το τμήμα `text` με κάποια άλλη, τότε η ανάθεση μνήμης γίνεται μόνο για τα τμήματα `data` και `stack`
- ▶ Η συνάρτηση `alloc_mem` ανατρέχει την λίστα οπών έως ότου βρει την πρώτη εγγραφή που είναι αρκετά μεγάλη για να 'χωρέσει' το μέγεθος της μνήμης που θέλουμε να αναθέσουμε
- ▶ Αν δεν βρεθεί καμία εγγραφή (αρκετά μεγάλη) τότε
 - ▶ Αν είναι ενεργοποιημένος ο μηχανισμός `swapping` -- προσπαθεί να κάνει `swap out` κάποια διεργασία και επαναλαμβάνει την αναζήτηση έως ότου βρεθεί μια αρκετά μεγάλη οπή
 - ▶ Αν δεν μπορεί να βρεθεί (συνεχόμενος) ελεύθερος χώρος τότε επιστρέφει `NO_MEM`

Απελευθέρωση Μνήμης

- ▶ Η απελευθέρωση μνήμης
 - ▶ Χρησιμοποιείται από την κλήση του συστήματος `exit`
 - ▶ Όταν είναι ενεργοποιημένος ο μηχανισμός `swapping`
- ▶ Προσπαθεί να εντοπίσει κάποια υπάρχουσα οπή που είναι 'δίπλα' στην θέση μνήμης που πρόκειται να απελευθερωθεί
 - ▶ Συγκώνευση της θέσης μνήμης με την υπάρχουσα οπή
- ▶ Αλλιώς εισάγει μια νέα εγγραφή στην λίστα οπών με την θέση μνήμης που απελευθερώθηκε
- ▶ Ο πίνακας των οπών έχει συγκεκριμένο μέγεθος
 - ▶ Το μέγεθος ορίζεται από την σταθερά `_NR_HOLES` ορίζεται σε $(2 * \text{NR_PROCS} + 4)$ -- αρχείο `/usr/src/include/minix/sys_config.h`
 - ▶ Αν και είναι ικανοποιητικά μεγάλος (για τις απαιτήσεις του Minix 3) -- μπορεί να γεμίσει
 - ▶ Σε αυτή την περίπτωση προκύπτει ένα `kernel panic`

Προηγούμενο Μάθημα

Διαχείριση Μνήμης
Υλοποίηση Διαχείρισης Μνήμης
Δομές Διαχειριστή Διεργασιών

Λειτουργικό Σύστημα Minix

Σήματα
Λειτουργίες Σημάτων
Υλοποίηση Διαχείρισης Σημάτων

Σύνοψη Μαθήματος

Σύνοψη Μαθήματος
Βιβλιογραφία
Επόμενη Διάλεξη



- ▶ Ένας βασικός τρόπος επικοινωνίας των διεργασιών είναι η χρήση σημάτων
 - ▶ Μεταφέρουν πληροφορία σε διεργασίες που δεν περιμένουν κάποια είσοδο
 - ▶ Χαρακτηρίζονται ως τα Interrupt λογισμικού
- ▶ Μελετήσαμε τα θέματα προγραμματισμού στο προηγούμενο εξάμηνο
- ▶ Το MINIX 3 ακολουθεί το πρότυπο POSIX
 - ▶ Ο κώδικας που αναπτύσσεται σε άλλο λειτουργικό UNIX μπορεί να μεταφερθεί στο MINIX 3 με 'αχετική' ευκολία
 - ▶ Όπως και με τα υπόλοιπα ζητήματα, η υλοποίηση των σημάτων είναι μιμησιατική



Ορισμός Σημάτων -- signal.h (1)

```

/* hangup */
#define SIGHUP          1
/* interrupt (DEL) */
#define SIGINT          2
/* quit (ASCII FS) */
#define SIGQUIT         3
/* illegal instruction */
#define SIGILL          4
/* trace trap (not reset when caught) */
#define SIGTRAP        5
/* IOT instruction */
#define SIGABRT         6
/* bus error */
#define SIGBUS          7

```



Ορισμός Σημάτων -- signal.h (2)

```

/* floating point exception */
#define SIGFPE          8
/* kill (cannot be caught or ignored) */
#define SIGKILL         9
/* user defined signal # 1 */
#define SIGUSR1         10
/* segmentation violation */
#define SIGSEGV         11
/* user defined signal # 2 */
#define SIGUSR2         12
/* write on a pipe with no one to read it */
#define SIGPIPE         13
/* alarm clock */
#define SIGALRM         14

```



```

/* software termination signal from kill */
#define SIGTERM          15
/* EMT instruction */
#define SIGEMT          16
/* child process terminated or stopped */
#define SIGCHLD          17
/* window size has changed */
#define SIGWINCH         21
/* continue if stopped */
#define SIGCONT          18
/* stop signal */
#define SIGSTOP          19
/* interactive stop signal */
#define SIGTSTP          20

```



```

/* background process wants to read */
#define SIGTTIN          22
/* background process wants to write */
#define SIGTTOU          23

/* new kernel message */
#define SIGKMSG          29
/* kernel signal pending */
#define SIGKSIG          30
/* kernel shutting down */
#define SIGKSTOP         31

```



Βασική Λειτουργία Σημάτων

- ▶ Όταν μια διεργασία λάβει ένα σήμα, το σύστημα ορίζει κάποιες προκαθορισμένες αντιδράσεις
- ▶ Μια διεργασία μπορεί να ζητήσει κάποια σήματα να αγνοηθούν
- ▶ Μια διεργασία μπορεί να αλλάξει την προκαθορισμένη αντίδραση για ένα σήμα ορίζοντας έναν signal handler
- ▶ Επομένως υπάρχουν τρεις βασικές φάσεις για την διαχείριση των σημάτων
 1. Προετοιμασία: Ρυθμίσεις σημάτων και αντιστοίχιση signal handler
 2. Απάντηση: Χρήση signal handler όταν ληφθεί ένα σήμα
 3. Επαφορά: επιστροφή της διεργασίας στην 'κανονική' ροή εκτέλεσης
- ▶ Το Λ.Σ. αναλαμβάνει να εκτελέσει τον signal handler και να επαναφέρει την διεργασία στην προηγούμενη ροή εκτέλεσης



Προετοιμασία Σημάτων (1)

- ▶ Υπάρχουν διάφορες κλήσεις του συστήματος για την ρύθμιση της αντίδρασης μιας διεργασίας, σε εισερχόμενα σήματα
 - ▶ sigaction -- ορισμός signal handler, επαναφορά προκαθορισμένης αντίδρασης, 'απενεργοποίηση' σήματος
 - ▶ sigprocmask -- μπλοκάρωμα σήματος
- ▶ Για κάθε διεργασία αντιστοιχούν διάφορες μεταβλητές τύπου sigset_t (στον πίνακα των διεργασιών)
- ▶ Ορίζεται στο αρχείο /usr/src/include/signal.h ως typedef unsigned long sigset_t;
- ▶ Χρησιμοποιείται ως bitmask όπου κάθε bit αντιστοιχεί σε ένα σήμα
 - ▶ π.χ., η μεταβλητή `mp_catch` ορίζει κατά πόσο το συγκεκριμένο σήμα είναι 'ενεργοποιημένο' για την διεργασία
 - ▶ π.χ., η μεταβλητή `mp_ignore` ορίζει κατά πόσο το συγκεκριμένο σήμα είναι 'απενεργοποιημένο' για την διεργασία



Προετοιμασία Σημάτων (2)

- ▶ Επίσης, για κάθε διεργασία, για κάθε σήμα, αντιστοιχεί μια μεταβλητή τύπου `sigaction` (στον πίνακα των διεργασιών)

```
struct sigaction {
    /* SIG_DFL, SIG_IGN, or pointer to function */
    __sighandler_t sa_handler;
    /* signals to be blocked during handler */
    sigset_t sa_mask;
    /* special flags */
    int sa_flags;
};
```

- ▶ Ο `signal handler` ορίζεται ως
`typedef void _PROTOTYPE((*__sighandler_t),
(int));`



Προετοιμασία Σημάτων – Παράδειγμα

Αρχείο `testsignal.c`

```
void catch_int(int sig_num) {
    printf("Don't do that...");
}

int main() {
    struct sigaction sig;
    sigset_t sset;
    sigemptyset(&sset);
    sig.sa_flags = 0;
    sig.sa_handler = catch_int;
    sig.sa_mask = sset;
    sigaddset(&sig.sa_mask, SIGINT);
    sigaction(SIGINT, &sig, NULL);
}
```



Αποστολή Σημάτων (1)

- ▶ Όταν δημιουργείται ένα νέο σήμα διάφορα τμήματα του MINIX 4 ενεργοποιούνται για την διαχείριση και αποστολή του σήματος στην διεργασία
- ▶ Η διαδικασία ξεκινάει από τον διαχειριστή διεργασιών
 - ▶ Εντοπίζει την διεργασία που πρέπει να παραλάβει το σήμα
 - ▶ Ελέγχει κατά πόσο η διεργασία έχει 'ενεργοποιηθεί' το σήμα
- ▶ Εφόσον το σήμα είναι ενεργοποιημένο ξεκινάει η διαδικασία παράδοσης του σήματος
 - ▶ Η κανονική ροή εκτέλεσης πρέπει να διακοπεί
 - ▶ Πληροφορίες σχετικά με την ροή εκτέλεσης αποθηκεύονται στο `stack` (εφόσον υπάρχει χώρος)
 - ▶ Αυτά τα βήματα γίνονται από τον διαχειριστή διεργασιών



Αποστολή Σημάτων (2)

- ▶ Η τελική παράδοση του σήματος γίνεται από το `system task`
 - ▶ Διακόπεται η κανονική ροή εκτέλεσης
 - ▶ Αλλάζει ο `program counter` έτσι ώστε ροή εκτέλεσης να συνεχίσει με τον `signal handler`
 - ▶ Όταν ολοκληρωθεί η εκτέλεση του `signal handler` γίνεται μια κλήση συστήματος `SIGRETURN`
- ▶ Η κλήση του συστήματος `SIGRETURN`
 - ▶ Ενεργοποιεί τον διαχειριστή διεργασιών για την επαναφορά του `stack`
 - ▶ Ενεργοποιεί τον πυρήνα για την αλλαγή του `program counter` έτσι ώστε η ροή εκτέλεσης να επανέλθει στην προηγούμενη κατάσταση
- ▶ Η παραπάνω διαδικασία μπορεί να χρειαστεί να εκτελεστεί πολλαπλές φορές αν ένα σήμα αφορά μια ομάδα/ιεραρχία διεργασιών



Λίστα Σημάτων (1)

Signal	Description	Generated by
SIGHUP	Hangup	KILL system call
SIGINT	Interrupt	TTY
SIGQUIT	Quit	TTY
SIGILL	Illegal instruction	Kernel (*)
SIGTRAP	Trace trap	Kernel (M)
SIGABRT	Abnormal termination	TTY
SIGFPE	Floating point exception	Kernel (*)
SIGKILL	Kill (cannot be caught or ignored)	KILL system call

* Βασίζονται στην υποστήριξη του hardware

M Δεν ορίζονται από το POSIX αλλά υποστηρίζονται για λόγους συμβατότητας



Λίστα Σημάτων (2)

Signal	Description	Generated by
SIGUSR1	User-defined signal #1	Not supported
SIGUSR2	User defined signal #2	Not supported
SIGSEGV	Segmentation violation	Kernel (*)
SIGPIPE	Write on a pipe with no one to read it	FS
SIGALRM	Alarm clock, timeout	PM
SIGTERM	Software termination signal from kill	KILL system call
SIGCHLD	Child process terminated or stopped	PM

* Βασίζονται στην υποστήριξη του hardware

M Δεν ορίζονται από το POSIX αλλά υποστηρίζονται για λόγους συμβατότητας



Λίστα Σημάτων (3)

Signal	Description	Generated by
SIGCONT	Continue if stopped	Not supported
SIGSTOP	Stop signal	Not supported
SIGTSTP	Interactive stop signal	Not supported
SIGTTIN	Background process wants to read	Not supported
SIGTTOU	Background process wants to write	Not supported
SIGKMESS	Kernel message	Kernel
SIGKSIG	Kernel signal pending	Kernel
SIGKSTOP	Kernel shutting down	Kernel

▶ Τα σήματα που δημιουργεί ο πυρήνας είναι ειδικά σήματα που ενημερώνουν τις διεργασίες του συστήματος για ειδικά γεγονότα



Ορισμός Βασικών signal handler -- signal.h

```
/* Macros used as function pointers. */

/* default signal handling */
#define SIG_DFL    ((__sighandler_t) 0)
/* ignore signal */
#define SIG_IGN    ((__sighandler_t) 1)
/* block signal */
#define SIG_HOLD   ((__sighandler_t) 2)
/* catch signal */
#define SIG_CATCH  ((__sighandler_t) 3)
```



```

/* deliver signal on alternate stack */
#define SA_ONSTACK    0x0001
/* reset signal handler when signal caught */
#define SA_RESETHAND  0x0002
/* don't block signal while catching it */
#define SA_NODEFER    0x0004
/* automatic system call restart */
#define SA_RESTART    0x0008
/* extended signal handling */
#define SA_SIGINFO    0x0010
/* don't create zombies */
#define SA_NOCLDWAIT  0x0020
/* don't receive SIGCHLD when child stops */
#define SA_NOCLDSTOP  0x0040

```



```

// Signal handling information
sigset_t mp_ignore; // 1 ignore the signal
sigset_t mp_catch; // 1 catch the signal
sigset_t mp_sig2mess; // 1 transform into notify
sigset_t mp_sigmask; // signals to be blocked
sigset_t mp_sigmask2; // saved copy of mp_sigmask
sigset_t mp_sigpending; // pending signals
// as in sigaction(2)
struct sigaction mp_sigact[_NSIG + 1];
// address of C library __sigreturn function
vir_bytes mp_sigreturn;
// watchdog timer for alarm(2)
struct timer mp_timer;

```



Βοηθητική Συνάρτηση sigaction

- Ορίζεται στο αρχείο `/usr/src/lib/posix/_sigaction.c`

```

PUBLIC int sigaction(sig, act, oact)
int sig;
_CONST struct sigaction *act;
struct sigaction *oact;
{
    message m;
    m.ml_i2 = sig;
    m.ml_p1 = (char *) act;
    m.ml_p2 = (char *) oact;
    m.ml_p3 = (char *) __sigreturn;
    return(_syscall(MM, SIGACTION, &m));
}

```



Κλήση Συστήματος SIGACTION (1)

```

PUBLIC int do_sigaction() {
    int r;
    struct sigaction svec;
    struct sigaction *svp;
    if (m_in.sig_nr == SIGKILL) return(OK);
    if (m_in.sig_nr < 1 || m_in.sig_nr > _NSIG)
        return (EINVAL);
    svp = &mp->mp_sigact[m_in.sig_nr];
    if ((struct sigaction *) m_in.sig_osa !=
        (struct sigaction *) NULL) {
        r = sys_datacopy(PM_PROC_NR, (vir_bytes) svp,
            who_e, (vir_bytes) m_in.sig_osa,
            (phys_bytes) sizeof(svec));
        if (r != OK) return(r);
    }
}

```



Κλήση Συστήματος SIGACTION (2)

```
if ((struct sigaction *) m_in.sig_nsa ==
    (struct sigaction *) NULL)
    return(OK);

/* Read in the sigaction structure. */
r = sys_datacopy(who_e, (vir_bytes) m_in.sig_nsa,
                PM_PROC_NR, (vir_bytes) &svect,
                (phys_bytes) sizeof(svect));
if (r != OK) return(r);

if (svect.sa_handler == SIG_IGN) {
    sigaddset(&mp->mp_ignore, m_in.sig_nr);
    sigdelset(&mp->mp_sigpending, m_in.sig_nr);
    sigdelset(&mp->mp_catch, m_in.sig_nr);
    sigdelset(&mp->mp_sig2mess, m_in.sig_nr);
```



Κλήση Συστήματος SIGACTION (3)

```
} else if (svect.sa_handler == SIG_DFL) {
    sigdelset(&mp->mp_ignore, m_in.sig_nr);
    sigdelset(&mp->mp_catch, m_in.sig_nr);
    sigdelset(&mp->mp_sig2mess, m_in.sig_nr);
} else if (svect.sa_handler == SIG_MESS) {
    if (! (mp->mp_flags & PRIV_PROC)) return(EPERM);
    sigdelset(&mp->mp_ignore, m_in.sig_nr);
    sigaddset(&mp->mp_sig2mess, m_in.sig_nr);
    sigdelset(&mp->mp_catch, m_in.sig_nr);
} else {
    sigdelset(&mp->mp_ignore, m_in.sig_nr);
    sigaddset(&mp->mp_catch, m_in.sig_nr);
    sigdelset(&mp->mp_sig2mess, m_in.sig_nr);
}
```



Κλήση Συστήματος SIGACTION (4)

```
mp->mp_sigact[m_in.sig_nr].sa_handler =
    svect.sa_handler;
sigdelset(&svect.sa_mask, SIGKILL);
mp->mp_sigact[m_in.sig_nr].sa_mask =
    svect.sa_mask;
mp->mp_sigact[m_in.sig_nr].sa_flags =
    svect.sa_flags;
mp->mp_sigreturn = (vir_bytes) m_in.sig_ret;
return(OK);
}
```



Κλήσεις Συστήματος SIGPENDING και SIGPROCMASK

- ▶ Επιτρέπεται η επεξεργασία των ρυθμίσεων των σημάτων ακόμα και 'μέσα' από τον signal handler
- ▶ Η κλήση SIGPENDING επιστρέφει τα σήματα που είναι σε αναμονή
 - ▶ Δηλαδή: αυτά που πρόκειται να παραδοθούν
- ▶ Η κλήση SIGPROCMASK επιστρέφει τα σήματα που έχουν μπλοκαριστεί
 - ▶ Δηλαδή: αυτά που δεν πρόκειται να παραδοθούν

```
PUBLIC int do_sigpending() {
    mp->mp_reply.reply_mask = (long)
        mp->mp_sigpending;
    return OK;
}
```



Κλήση Συστήματος *KILL*

- ▶ Βασικός τρόπος για την αποστολή σημάτων

```
PUBLIC int do_kill() {
    /* Perform the kill(pid, signo) system call*/
    return check_sig(m_in.pid, m_in.sig_nr);
}
```

- ▶ Η συνάρτηση *check_sig* ανατρέχει τον πίνακα των διεργασιών για να εντοπίσει όλες τις διεργασίες που μπορούν/πρέπει να παραλάβουν το σήμα
 - ▶ Ελέγχει αν έχει δικαίωμα να λάβει το σήμα
 - ▶ Ελέγχει αν είναι διαχειριστής και δεν πρέπει να λάβει το σήμα
 - ▶ Ελέγχει αν ανήκει στην ίδια ομάδα/ιεραρχία διεργασιών
- ▶ Για κάθε διεργασία καλεί την συνάρτηση *sig_proc* για την παράδοση



Συνάρτηση *sig_proc* (1)

```
PUBLIC void sig_proc(rmp, signo)
/* pointer to the process to be signaled */
register struct mproc *rmp;
int signo; /* signal to send (1 to _NSIG) */
{
    vir_bytes new_sp;
    int s, slot, sigflags;
    struct sigmsg sm;
    slot = (int) (rmp - mproc);
    if ((rmp->mp_flags & (IN_USE | ZOMBIE)) != IN_USE)
        printf("PM: signal %d sent to %s process %d\n",
            signo,
            (rmp->mp_flags & ZOMBIE) ? "zombie" : "dead");
    panic(__FILE__, "", NO_NUM);
}
```



Συνάρτηση *sig_proc* (2)

```
if ((rmp->mp_flags & TRACED) && signo!=SIGKILL) {
    /* A traced process has special handling. */
    unpause(slot);
    stop_proc(rmp, signo); /* stop the proc */
    return;
}
/* Some signals are ignored by default. */
if (sigismember(&rmp->mp_ignore, signo)) {
    return;
}
if (sigismember(&rmp->mp_sigmask, signo)) {
    /* Signal should be blocked. */
    sigaddset(&rmp->mp_sigpending, signo);
    return;
}
```



Συνάρτηση *sig_proc* (3)

```
sigflags = rmp->mp_sigact[signo].sa_flags;
if (sigismember(&rmp->mp_catch, signo)) {
    if (rmp->mp_flags & SIGSUSPENDED)
        sm.sm_mask = rmp->mp_sigmask2;
    else
        sm.sm_mask = rmp->mp_sigmask;
    sm.sm_signo = signo;
    sm.sm_sighandler = (vir_bytes) rmp->mp_sigact[s
    sm.sm_sigreturn = rmp->mp_sigreturn;
    if ((s=get_stack_ptr(rmp->mp_endpoint, &new_sp))
        panic(__FILE__, "couldn't get new stack pointe
    sm.sm_stkptr = new_sp;
```



Συνάρτηση *sig_proc* (4)

```
/* Make room for the sigcontext and sigframe st
new_sp -= sizeof(struct sigcontext)
        + 3 * sizeof(char *) + 2 * sizeof(int);

if (adjust(rmp, rmp->mp_seg[D].mem_len, new_sp) !=
goto doterminate;

if (OK == (s=sys_sigsend(rmp->mp_endpoint, &sm)) )
sigdelset(&rmp->mp_sigpending, signo);
/* If process is hanging on PAUSE, WAIT,
 * SIGSUSPEND, tty, pipe, etc., release it.
 */
unpause(slot);
return;
}
```



Κλήση Πυρήνα *SIGSEND*

- ▶ Τελικό στάδιο για την αποστολή ενός σήματος
- ▶ Εξυπηρετούνται μόνο οι κλήσεις που προέρχονται από τον διαχειριστή διεργασιών
- ▶ Οι πληροφορίες που αφορούν ένα μήνυμα μεταφέρονται μέσω μιας ειδικής δομής *sigmsg*
 - ▶ Αντιγράφεται η δομή στο kernel space
- ▶ Αποθηκεύει τα περιεχόμενα του *sigmsg* σε βοηθητικές δομές
 - ▶ Είναι απαραίτητα όταν θα γίνει η κλήση *SIGRETURN*
- ▶ Διακόπτει την ροή εκτέλεσης και σώζει τον program counter στο stack
- ▶ Ενημερώνει τους registers και τον program counter σύμφωνα με τον signal handler



Κλήση Συστήματος *SIGRETURN*

- ▶ Όταν ολοκληρωθεί η εκτέλεση του signal handler η διεργασία στέλνει ένα 'κρυφό' σήμα *SIGRETURN*
- ▶ Χρησιμοποιούμε αυτό το 'ενδιάμεσο' σήμα για την επαναφορά της ροής εκτέλεσης της διεργασίας
- ▶ Ο χειρισμός του *SIGRETURN* γίνεται από τον διαχειριστή διεργασιών
- ▶ Βασίζεται στην κλήση του πυρήνα *SYS_SIGRETURN*
- ▶ Επαναφέρει την ροή εκτέλεσης της διεργασίας
- ▶ Όταν ολοκληρωθεί η κλήση του πυρήνα *SYS_SIGRETURN* χρησιμοποιεί την συνάρτηση *check_pending*
 - ▶ Ελέγχει αν υπάρχουν σήματα σε εκκρεμότητα και τα προωθεί στις διεργασίες
 - ▶ Για κάθε συγκεκριμένο σήμα υπάρχει απλά 1 bit -- είτε έρθει 1 σήμα είτε 10 σήματα, μόνο 1 θα προωθηθεί



Σύνοψη 11^{ης} Διάλεξης

Προηγούμενο Μάθημα

Διαχείριση Μνήμης
Υλοποίηση Διαχείρισης Μνήμης
Δομές Διαχειριστή Διεργασιών

Λειτουργικό Σύστημα Minix

Σήματα
Λειτουργίες Σημάτων
Υλοποίηση Διαχείρισης Σημάτων

Σύνοψη Μαθήματος

Σύνοψη Μαθήματος
Βιβλιογραφία
Επόμενη Διάλεξη



- ▶ Σήματα – Σηματοδότηση
- ▶ Μηχανισμοί Διαχείρισης Σημάτων
- ▶ Εσωτερικές Δομές Διαχειριστή Διεργασιών
- ▶ Κλήσεις συστήματος για την σηματοδότηση



- ▶ Βιβλίο "Operating Systems Design and Implementation, Third Edition" (Andrew S. Tanenbaum)
 1. Κεφάλαιο 1: Introduction
 - ▶ Παράγραφος 1.4.2 System Calls for Signaling
 2. Κεφάλαιο 4: Memory Management
 - ▶ Παράγραφος 4.7.7 Signal Handling
 - ▶ Παράγραφος 4.8.6 Implementation of Signal Handling
- ▶ Βιβλίο "Σύγχρονα Λειτουργικά Συστήματα" (A.Tanenbaum)
 1. Κεφάλαιο 5: Συσκευές Εισόδου / Εξόδου
 2. Κεφάλαιο 10: Μελέτη Περίπτωσης 1: Unix και Linux
 - ▶ Παράγραφος 10.3 Οι διεργασίες στο Unix
 - ▶ Παράγραφος 10.3.1 Θεμελιώδεις έννοιες



- ▶ Σύστημα Αρχείων στο Λ.Σ. MINIX 3
- ▶ Επανάληψη από μάθημα "Λειτουργικά Συστήματα I"
 1. Κεφάλαιο 5: Συστήματα Αρχείων
- ▶ Βιβλίο "Σύγχρονα Λειτουργικά Συστήματα" (A.Tanenbaum)
 1. Κεφάλαιο 6: Συστήματα Αρχείων
 2. Κεφάλαιο 10: Μελέτη Περίπτωσης 1: Unix και Linux
 - ▶ Παράγραφος 10.6 Το Σύστημα Αρχείων στο UNIX
- ▶ Βιβλίο "Operating Systems Design and Implementation, Third Edition" (Andrew S. Tanenbaum)
 1. Κεφάλαιο 5: File System

