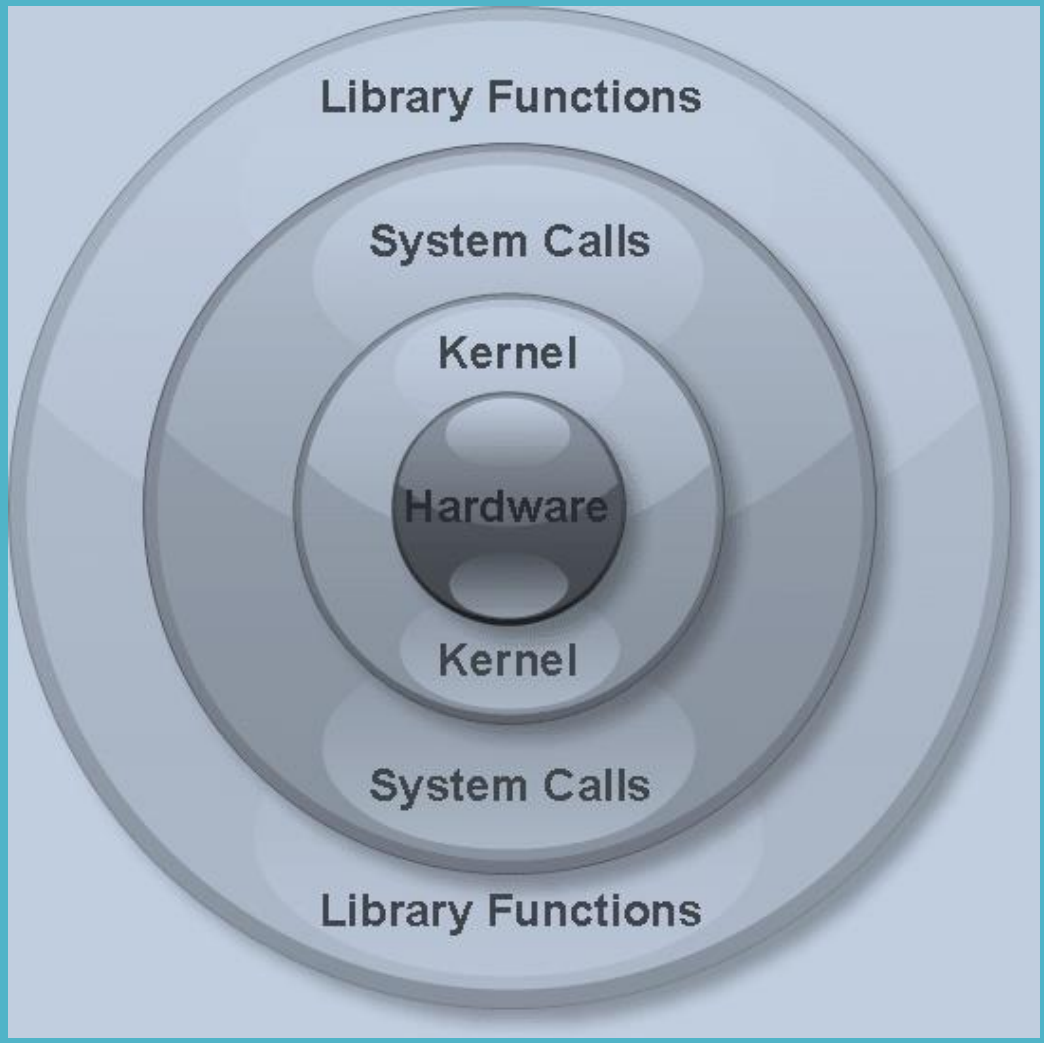
Κλήσεις συστήματος



Κλήσεις συστήματος

Στο λειτουργικό σύστημα Linux οι πάσης φύσεως εφαρμογές εκτελούνται σε δύο διαφορετικές καταστάσεις λειτουργίας:

- Κατάσταση λειτουργίας χρήστη (User Mode) → σε αυτή την κατάσταση λειτουργίας εκτελούνται οι εφαρμογές των χρηστών.

- Κατάσταση λειτουργίας πυρήνα (Kernel Mode) → σε αυτή την κατάσταση λειτουργίας εκτελούνται οι οδηγοί συσκευών και άλλα κρίσιμα προγράμματα υπό καθεστώς αυξημένης προστασίας έτσι ώστε να μην προκαλείται κατάρρευση του συστήματος. Ο κώδικας που εκτελείται σε kernel mode έχει πρόσβαση σε όλο το σύστημα.

Ο στόχος των εφαρμογών που εκτελούνται σε kernel mode είναι η εξυπηρέτηση των εφαρμογών που εκτελούνται σε user mode.

Μία κλήση συστήματος είναι ο τρόπος με τον οποίο ένα πρόγραμμα που εκτελείται σε user mode ζητά εξυπηρέτηση από ένα πρόγραμμα που εκτελείται σε kernel mode.

Κλήσεις συστήματος

Με τις κλήσεις συστήματος μπορούμε να πραγματοποιήσουμε πολλές και διαφορετικές λειτουργίες, όπως είναι για παράδειγμα, οι ακόλουθες:

- Διαχείριση αρχείων → open, create, delete, κ.τ.λ.
- Έλεγχος διεργασιών → kill, wait, κ.τ.λ.
- Διαχείριση συσκευών → request, release, κ.τ.λ.
- Ανάκτηση και ορισμός τιμών παραμέτρων → set time, get time, κ.τ.λ.
- Διαχείριση επικοινωνιών και δικτύων → send, receive, κ.τ.λ

Κλήσεις συστήματος

Λόγω του καθεστώτος αυξημένης προστασίας στο kernel mode ο τρόπος χρήσης των κλήσεων συστήματος από τις συναρτήσεις του χρήστη είναι πολύ περιοριστικός:

- Επιτρέπεται μόνο η μεταβίβαση παραμέτρων σταθερού μήκους που είναι ίσο με το μήκος που χρησιμοποιείται για τους δείκτες (pointers).

- Ως κωδικοί επιστροφής για τις κλήσεις συστήματος χρησιμοποιούνται μόνο προσημασμένοι ακέραιοι. Ένας κωδικός επιστροφής ίσος με το μηδέν, υποδηλώνει επιτυχή ολοκλήρωση, ενώ ένας αρνητικός κωδικός παραπέμπει στην εκδήλωση σφάλματος.

Ο κωδικός επιστροφής της τελευταίας κλήσης συστήματος αποθηκεύεται στη μεταβλητή errno.

Υπάρχουν τρεις συναρτήσεις ανάγνωσης του σφάλματος:

perror () → εκτυπώνει ένα μήνυμα σφάλματος.
strerror () → επιστρέφει φράση σταθερού μήκους που περιγράφει το σφάλμα.
sys_errlist () → χρησιμοποιεί δείκτες σε μηνύματα συστήματος αλλά δεν είναι standard συνάρτηση στο Linux.

Κλήσεις συστήματος

Οι κωδικοί επιστροφής (συνήθως) είναι αποθηκευμένοι στο αρχείο /usr/include/asm/errno.h

```
#define EPERM            1      /* Operation not permitted */
#define ENOENT           2      /* No such file or directory */
#define ESRCH            3      /* No such process */
#define EINTR            4      /* Interrupted system call */
#define EIO              5      /* I/O error */
#define ENXIO            6      /* No such device or address */
#define E2BIG            7      /* Arg list too long */
#define ENOEXEC          8      /* Exec format error */
#define EBADF            9      /* Bad file number */
#define ECHILD           10     /* No child processes */
#define EAGAIN           11     /* Try again */
#define ENOMEM           12     /* Out of memory */
#define EACCES           13     /* Permission denied */
#define EFAULT           14     /* Bad address */
#define ENOTBLK          15     /* Block device required */
#define EBUSY            16     /* Device or resource busy */
#define EEXIST           17     /* File exists */
#define EXDEV            18     /* Cross-device link */
#define ENODEV           19     /* No such device */
#define ENOTDIR          20     /* Not a directory */
#define EISDIR           21     /* Is a directory */
#define EINVAL           22     /* Invalid argument */
#define ENFILE           23     /* File table overflow */
#define EMFILE           24     /* Too many open files */
#define ENOTTY           25     /* Not a typewriter */
#define ETXTBSY          26     /* Text file busy */
#define EFBIG            27     /* File too large */
#define ENOSPC           28     /* No space left on device */
#define ESPIPE           29     /* Illegal seek */
#define EROFS            30     /* Read-only file system */
#define EMLINK           31     /* Too many links */
#define EPIPE            32     /* Broken pipe */
#define EDOM             33     /* Math argument out of domain of func */
#define ERANGE           34     /* Math result not representable */
#define EDEADLK          35     /* Resource deadlock would occur */
#define ENAMETOOLONG     36     /* File name too long */
#define ENOLCK           37     /* No record locks available */
#define ENOSYS           38     /* Function not implemented */
#define ENOTEMPTY        39     /* Directory not empty */
#define ELOOP            40     /* Too many symbolic links encountered */
#define EWOULDBLOCK      EAGAIN /* Operation would block */
#define ENOMSG           42     /* No message of desired type */
#define EIDRM            43     /* Identifier removed */
#define ECHRNG           44     /* Channel number out of range */
#define EL2NSYNC         45     /* Level 2 not synchronized */
#define EL3HLT           46     /* Level 3 halted */
#define EL3RST           47     /* Level 3 reset */
#define ELNRNG           48     /* Link number out of range */
#define EUNATCH          49     /* Protocol driver not attached */
#define ENOCSI           50     /* No CSI structure available */
#define EL2HLT           51     /* Level 2 halted */
#define EBADE            52     /* Invalid exchange */
#define EBADR            53     /* Invalid request descriptor */
#define EXFULL           54     /* Exchange full */
#define ENOANO           55     /* No anode */
#define EBADRQC          56     /* Invalid request code */
#define EBADSLT          57     /* Invalid slot */

#define EDEADLOCK        EDEADLK

#define EBFONT           59     /* Bad font file format */
#define ENOSTR           60     /* Device not a stream */
#define ENODATA          61     /* No data available */
#define ETIME            62     /* Timer expired */
#define ENOSR            63     /* Out of streams resources */
#define ENONET           64     /* Machine is not on the network */
#define ENOPKG           65     /* Package not installed */
#define EREMOTE          66     /* Object is remote */
#define ENOLINK          67     /* Link has been severed */
#define EADV             68     /* Advertise error */
#define ESRMNT           69     /* Srmount error */
#define ECOMM            70     /* Communication error on send */
#define EPROTO           71     /* Protocol error */
#define EMULTIHOP        72     /* Multihop attempted */
#define EDOTDOT          73     /* RFS specific error */
#define EBADMSG          74     /* Not a data message */
#define EOVERFLOW        75     /* Value too large for defined data type */
#define ENOTUNIQ         76     /* Name not unique on network */
#define EBADFD           77     /* File descriptor in bad state */
#define EREMCHG          78     /* Remote address changed */
#define ELIBACC          79     /* Can not access a needed shared library */
#define ELIBBAD          80     /* Accessing a corrupted shared library */
#define ELIBSCN          81     /* .lib section in a.out corrupted */
#define ELIBMAX          82     /* Attempting to link in too many shared libraries */
#define ELIBEXEC         83     /* Cannot exec a shared library directly */
#define EILSEQ           84     /* Illegal byte sequence */
#define ERESTART         85     /* Interrupted system call should be restarted */
#define ESTRPIPE         86     /* Streams pipe error */
#define EUSERS           87     /* Too many users */
#define ENOTSOCK         88     /* Socket operation on non-socket */
#define EDESTADDRREQ     89     /* Destination address required */
#define EMSGSIZE         90     /* Message too long */
#define EPROTOTYPE       91     /* Protocol wrong type for socket */
#define ENOPROTOOPT      92     /* Protocol not available */
#define EPROTONOSUPPORT  93     /* Protocol not supported */
#define ESOCKTNOSUPPORT  94     /* Socket type not supported */
#define EOPNOTSUPP       95     /* Operation not supported on transport endpoint */
#define EPFNOSUPPORT     96     /* Protocol family not supported */
#define EAFNOSUPPORT     97     /* Address family not supported by protocol */
#define EADDRINUSE       98     /* Address already in use */
#define EADDRNOTAVAIL    99     /* Cannot assign requested address */
#define ENETDOWN         100    /* Network is down */
#define ENETUNREACH      101    /* Network is unreachable */
#define ENETRESET        102    /* Network dropped connection because of reset */
#define ECONNABORTED     103    /* Software caused connection abort */
#define ECONNRESET       104    /* Connection reset by peer */
#define ENOBUFS          105    /* No buffer space available */
#define EISCONN          106    /* Transport endpoint is already connected */
#define ENOTCONN         107    /* Transport endpoint is not connected */
#define ESHUTDOWN        108    /* Cannot send after transport endpoint shutdown */
#define ETOOMANYREFS     109    /* Too many references: cannot splice */
#define ETIMEDOUT        110    /* Connection timed out */
#define ECONNREFUSED     111    /* Connection refused */
#define EHOSTDOWN        112    /* Host is down */
#define EHOSTUNREACH     113    /* No route to host */
#define EALREADY         114    /* Operation already in progress */
#define EINPROGRESS      115    /* Operation now in progress */
#define ESTALE           116    /* Stale NFS file handle */
#define EUCLEAN          117    /* Structure needs cleaning */
#define ENOTNAM          118    /* Not a XENIX named type file */
#define ENAVAIL          119    /* No XENIX semaphores available */
#define EISNAM           120    /* Is a named type file */
#define EREMOTEIO        121    /* Remote I/O error */
#define EDQUOT           122    /* Quota exceeded */

#define ENOMEDIUM        123    /* No medium found */
#define EMEDIUMTYPE      124    /* Wrong medium type */
```

Κλήσεις συστήματος

Οι κλήσεις συστήματος καλούνται από τις συναρτήσεις του χρήστη

Παράδειγμα → άνοιγμα αρχείου

Χρησιμοποιώντας τη C → FILE * fopen (const char * path, const char * mode)

FILE * fp = fopen (“myfile.txt”, “r+”)

Χρησιμοποιώντας την κλήση συστήματος open

int open (const char * path, int flags)

int fd = open (“myfile.txt”, O_RDWR)

Η fopen είναι portable, η open γενικά όχι

LINUX System Call Quick Reference

Jialong He
Jialong_he@bigfoot.com
http://www.bigfoot.com/~jialong_he

Introduction

System call is the services provided by Linux kernel. In C programming, it often uses functions defined in **libc** which provides a wrapper for many system calls. Manual page section 2 provides more information about system calls. To get an overview, use “man 2 intro” in a command shell.

It is also possible to invoke **syscall()** function directly. Each system call has a function number defined in **<syscall.h>** or **<unistd.h>**. Internally, system call is invokded by software interrupt 0x80 to transfer control to the kernel. System call table is defined in Linux kernel source file “**arch/i386/kernel/entry.S**”.

System Call Example

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

int main(void) {

        long ID1, ID2;
        /*----------------------------*/
        /* direct system call         */
        /* SYS_getpid (func no. is 20) */
        /*----------------------------*/
        ID1 = syscall(SYS_getpid);
        printf ("syscall(SYS_getpid)=%ld\n", ID1);

        /*----------------------------*/
        /* "libc" wrapped system call  */
        /* SYS_getpid (Func No. is 20) */
        /*----------------------------*/
        ID2 = getpid();
        printf ("getpid()=%ld\n", ID2);

        return(0);
}
```

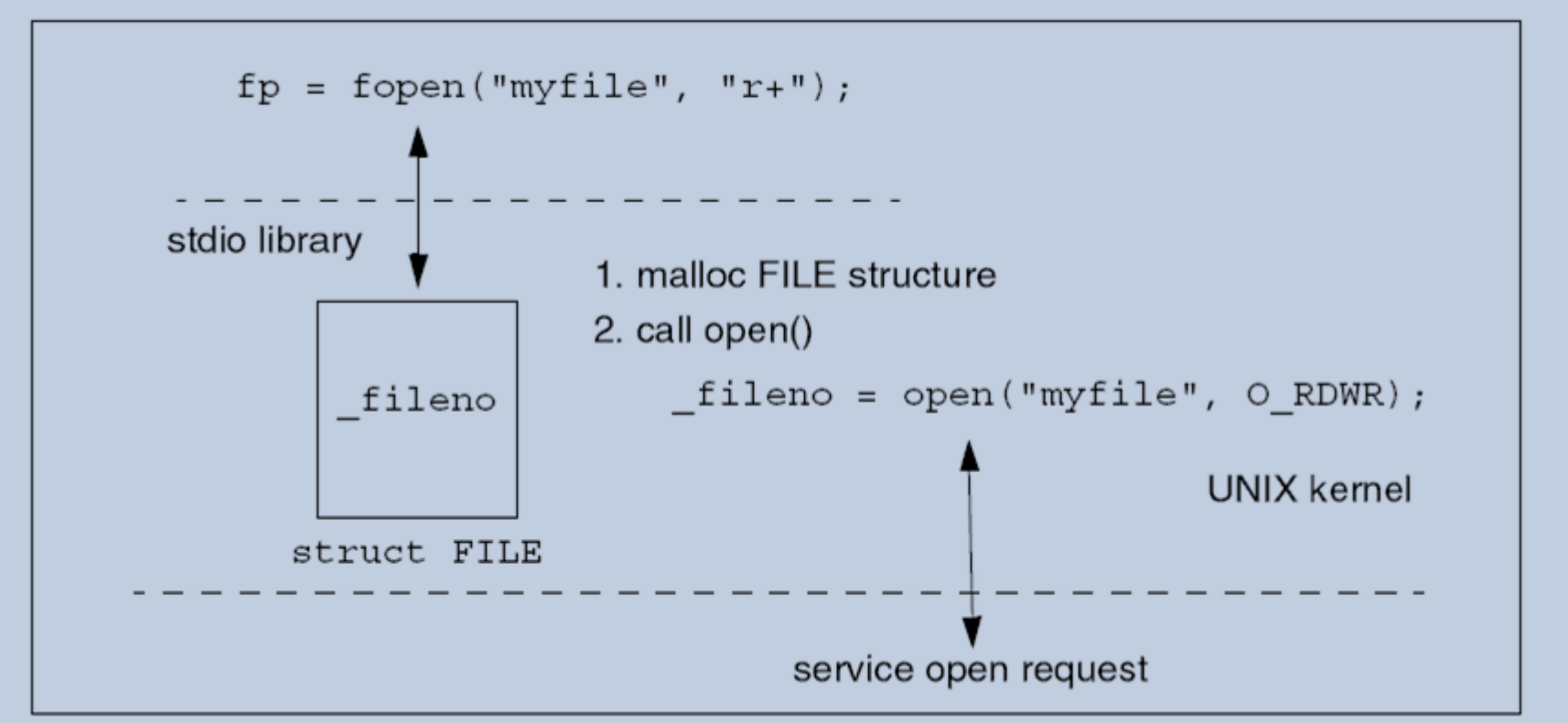
System Call Quick Reference

No	Func Name	Description	Source
1	exit	terminate the current process	*kernel/exit.c*
2	fork	create a child process	*arch/i386/kernel/process.c*
3	read	read from a file descriptor	*fs/read_write.c*
4	write	write to a file descriptor	*fs/read_write.c*
5	open	open a file or device	*fs/open.c*
6	close	close a file descriptor	*fs/open.c*
7	waitpid	wait for process termination	*kernel/exit.c*
8	creat	create a file or device ("man 2 open" for information)	*fs/open.c*
9	link	make a new name for a file	*fs/namei.c*
10	unlink	delete a name and possibly the file it refers to	*fs/namei.c*
11	execve	execute program	*arch/i386/kernel/process.c*
12	chdir	change working directory	*fs/open.c*
13	time	get time in seconds	*kernel/time.c*
14	mknod	create a special or ordinary file	*fs/namei.c*
15	chmod	change permissions of a file	*fs/open.c*
16	lchown	change ownership of a file	*fs/open.c*
18	stat	get file status	*fs/stat.c*
19	lseek	reposition read/write file offset	*fs/read_write.c*
20	getpid	get process identification	*kernel/sched.c*
21	mount	mount filesystems	*fs/super.c*
22	umount	unmount filesystems	*fs/super.c*
23	setuid	set real user ID	*kernel/sys.c*
24	getuid	get real user ID	*kernel/sched.c*
25	stime	set system time and date	*kernel/time.c*
26	ptrace	allows a parent process to control the execution of a child process	*arch/i386/kernel/ptrace.c*
27	alarm	set an alarm clock for delivery of a signal	*kernel/sched.c*
28	fstat	get file status	*fs/stat.c*
29	pause	suspend process until signal	*arch/i386/kernel/sys_i386.c*
30	utime	set file access and modification times	*fs/open.c*
33	access	check user's permissions for a file	*fs/open.c*
34	nice	change process priority	*kernel/sched.c*
36	sync	update the super block	*fs/buffer.c*
37	kill	send signal to a process	*kernel/signal.c*
38	rename	change the name or location of a file	*fs/namei.c*
39	mkdir	create a directory	*fs/namei.c*
40	rmdir	remove a directory	*fs/namei.c*
41	dup	duplicate an open file descriptor	*fs/fcntl.c*
42	pipe	create an interprocess channel	*arch/i386/kernel/sys_i386.c*
43	times	get process times	*kernel/sys.c*
45	brk	change the amount of space allocated for the calling process's data segment	*mm/mmap.c*
46	setgid	set real group ID	*kernel/sys.c*
47	getgid	get real group ID	*kernel/sched.c*
48	sys_signal	ANSI C signal handling	*kernel/signal.c*
49	geteuid	get effective user ID	*kernel/sched.c*
50	getegid	get effective group ID	*kernel/sched.c*

51	acct	enable or disable process accounting	*kernel/acct.c*
52	umount2	unmount a file system	*fs/super.c*
54	ioctl	control device	*fs/ioctl.c*
55	fcntl	file control	*fs/fcntl.c*
56	mpx	(unimplemented)	
57	setpgid	set process group ID	*kernel/sys.c*
58	ulimit	(unimplemented)	
59	olduname	obsolete uname system call	*arch/i386/kernel/sys_i386.c*
60	umask	set file creation mask	*kernel/sys.c*
61	chroot	change root directory	*fs/open.c*
62	ustat	get file system statistics	*fs/super.c*
63	dup2	duplicate a file descriptor	*fs/fcntl.c*
64	getppid	get parent process ID	*kernel/sched.c*
65	getpgrp	get the process group ID	*kernel/sys.c*
66	setsid	creates a session and sets the process group ID	*kernel/sys.c*
67	sigaction	POSIX signal handling functions	*arch/i386/kernel/signal.c*
68	sgetmask	ANSI C signal handling	*kernel/signal.c*
69	ssetmask	ANSI C signal handling	*kernel/signal.c*
70	setreuid	set real and effective user IDs	*kernel/sys.c*
71	setregid	set real and effective group IDs	*kernel/sys.c*
72	sigsuspend	install a signal mask and suspend caller until signal	*arch/i386/kernel/signal.c*
73	sigpending	examine signals that are blocked and pending	*kernel/signal.c*
74	sethostname	set hostname	*kernel/sys.c*
75	setrlimit	set maximum system resource con sumption	*kernel/sys.c*
76	getrlimit	get maximum system resource con sumption	*kernel/sys.c*
77	getrusage	get maximum system resource con sumption	*kernel/sys.c*
78	gettimeofday	get the date and time	*kernel/time.c*
79	settimeofday	set the date and time	*kernel/time.c*
80	getgroups	get list of supplementary group IDs	*kernel/sys.c*
81	setgroups	set list of supplementary group IDs	*kernel/sys.c*
82	old_select	sync. I/O multiplexing	*arch/i386/kernel/sys_i386.c*
83	symlink	make a symbolic link to a file	*fs/namei.c*
84	lstat	get file status	*fs/stat.c*
85	readlink	read the contents of a symbolic link	*fs/stat.c*
86	uselib	select shared library	*fs/exec.c*
87	swapon	start swapping to file/device	*mm/swapfile.c*
88	reboot	reboot or enable/disable Ctrl-Alt-Del	*kernel/sys.c*
89	old_readdir	read directory entry	*fs/readdir.c*
90	old_mmap	map pages of memory	*arch/i386/kernel/sys_i386.c*
91	munmap	unmap pages of memory	*mm/mmap.c*
92	truncate	set a file to a specified length	*fs/open.c*
93	ftruncate	set a file to a specified length	*fs/open.c*
94	fchmod	change access permission mode of file	*fs/open.c*
95	fchown	change owner and group of a file	*fs/open.c*
96	getpriority	get program scheduling priority	*kernel/sys.c*
97	setpriority	set program scheduling priority	*kernel/sys.c*
98	profil	execut ion time profile	
99	statfs	get file system statistics	*fs/open.c*
100	fstatfs	get file system statistics	*fs/open.c*
101	ioperm	set port input/output permissions	*arch/i386/kernel/ioport.c*
102	socketcall	socket system calls	*net/so cket.c*
103	syslog	read and/or clear kernel message ring buffer	*kernel/printk.c*
104	setitimer	set value of interval timer	*kernel/itimer.c*
105	getitimer	get value of interval timer	*kernel/itimer.c*
106	sys_newstat	get file status	*fs/stat.c*
107	sys_newlstat	get file status	*fs/stat.c*
108	sys_newfstat	get file status	*fs/stat.c*
109	olduname	get name and information about current kernel	*arch/i386/kernel/sys_i386.c*
110	iopl	change I/O privilege level	*arch/i386/kernel/ioport.c*
111	vhangup	virtually hangup the current tty	*fs/open.c*
112	idle	make process 0 idle	*arch/i386/kernel/process.c*
113	vm86old	enter virtual 8086 mode	*arch/i386/kernel/vm86.c*
114	wait4	wait for process termination, BSD style	*kernel/exit.c*
115	swapoff	stop swapping to file/device	*mm/swapfile.c*
116	sysinfo	returns information on overall system statistics	*kernel/info.c*
117	ipc	System V IPC system calls	*arch/i386/kernel/sys_i386.c*
118	fsync	synchronize a file's complete in-core state with that on disk	*fs/buffer.c*
119	sigreturn	return from signal handler and cleanup stack frame	*arch/i386/kernel/signal.c*
120	clone	create a child process	*arch/i386/kernel/process.c*
121	setdomainname	set domain name	*kernel/sys.c*
122	uname	get name and information about current kernel	*kernel/sys.c*
123	modify_ldt	get or set ldt	*arch/i386/kernel/ldt.c*
124	adjtimex	tune kernel clock	*kernel/time.c*
125	mprotect	set protection of memory mapping	*mm/mprotect.c*
126	sigprocmask	POSIX signal handling functions	*kernel/signal.c*
127	create_module	create a loadable module entry	*kernel/module.c*
128	init_module	initialize a loadable module entry	*kernel/module.c*
129	delete_module	delete a loadable module entry	*kernel/module.c*

130	get_kernel_syms	retrieve exported kernel and module symbols	*kernel/module.c*
131	quotactl	manipulate disk quotas	*fs/dquot.c*
132	getpgid	get process group ID	*kernel/sys.c*
133	fchdir	change working directory	*fs/open.c*
134	bdflush	start, flush, or tune buffer-dirty-flush daemon	*fs/buffer.c*
135	sysfs	get file system type information	*fs/super.c*
136	personality	set the process execution domain	*kernel/exec_domain.c*
137	afs_syscall	(unimplemented)	
138	setfsuid	set user identity used for file system checks	*kernel/sys.c*
139	setfsgid	set group identity used for file system checks	*kernel/sys.c*
140	sys_llseek	move extended read/write file pointer	*fs/read_write.c*
141	getdents	read directory entries	*fs/readdir.c*
142	select	sync. I/O multiplexing	*fs/select.c*
143	flock	apply or remove an advisory lock on an open file	*fs/locks.c*
144	msync	synchronize a file with a memory map	*mm/filemap.c*
145	readv	read data into multiple buffers	*fs/read_write.c*
146	writev	write data into multiple buffers	*fs/read_write.c*
147	sys_getsid	get process group ID of session leader	*kernel/sys.c*
148	fdatasync	synchronize a file's in-core data with that on disk	*fs/buffer.c*
149	sysctl	read/write system parameters	*kernel/sysctl.c*
150	mlock	lock pages in memory	*mm/mlock.c*
151	munlock	unlock pages in memory	*mm/mlock.c*
152	mlockall	disable paging for calling process	*mm/mlock.c*
153	munlockall	reenable paging for calling process	*mm/mlock.c*
154	sched_setparam	set scheduling parameters	*kernel/sched.c*
155	sched_getparam	get scheduling parameters	*kernel/sched.c*
156	sched_setscheduler	set scheduling algorithm parameters	*kernel/sched.c*
157	sched_getscheduler	get scheduling algorithm parameters	*kernel/sched.c*
158	sched_yield	yield the processor	*kernel/sched.c*
159	sched_get_priority_max	get max static priority range	*kernel/sched.c*
160	sched_get_priority_min	get min static priority range	*kernel/sched.c*
161	sched_rr_get_interval	get the SCHED_RR interval for the named process	*kernel/sched.c*
162	nanosleep	pause execution for a specified time (nano seconds)	*kernel/sched.c*
163	mremap	re-map a virtual memory address	*mm/mremap.c*
164	setresuid	set real, effective and saved user or group ID	*kernel/sys.c*
165	getresuid	get real, effective and saved user or group ID	*kernel/sys.c*
166	vm86	enter virtual 8086 mode	*arch/i386/kernel/vm86.c*
167	query_module	query the kernel for various bits pertain ing to modules	*kernel/module.c*
168	poll	wait for some event on a file descriptor	*fs/select.c*
169	nfsservctl	syscall interface to kernel nfs daemon	*fs/filesystems.c*
170	setresgid	set real, effective and saved user or group ID	*kernel/sys.c*
171	getresgid	get real, effective and saved user or group ID	*kernel/sys.c*
172	prctl	operations on a process	*kernel/sys.c*
173	rt_sigreturn		*arch/i386/kernel/signal.c*
174	rt_sigaction		*kernel/signal.c*
175	rt_sigprocmask		*kernel/signal.c*
176	rt_sigpending		*kernel/signal.c*
177	rt_sigtimedwait		*kernel/signal.c*
178	rt_sigqueueinfo		*kernel/signal.c*
179	rt_sigsuspend		*arch/i386/kernel/signal.c*
180	pread	read from a file descriptor at a given offset	*fs/read_write.c*
181	sys_pwrite	write to a file descriptor at a given offset	*fs/read_write.c*
182	chown	change ownership of a file	*fs/open.c*
183	getcwd	Get current working directory	*fs/dcache.c*
184	capget	get process capabilities	*kernel/capability.c*
185	capset	set process capabilities	*kernel/capability.c*
186	sigaltstack	set/get signal stack context	*arch/i386/kernel/signal.c*
187	sendfile	transfer data between file descriptors	*mm/filemap.c*
188	getpmsg	(unimplemented)	
189	putpmsg	(unimplemented)	
190	vfork	create a child process and block parent	*arch/i386/kernel/process.c*

Κλήσεις συστήματος



Η fopen (user mode) καλεί την open (kernel mode)

Κλήσεις συστήματος

FILE * fp = fopen (“myfile.txt”, “r+”)

```
typedef struct
{
    int level;
    unsigned flags;
    char fd;
    unsigned char hold;
    int bsize;
    unsigned char_FAR* buffer;
    unsigned char_FAR* curp;
    unsigned istemp;
    short token;
}FILE;
```

int fd = open (“myfile.txt”, O_RDWR)

Το πεδίο fd (file descriptor) που επιστρέφει η open δεν είναι παρά ένα από τα πεδία της δομής FILE που επιστρέφει η fopen η οποία επομένως προσφέρει περισσότερες δυνατότητες διαχείρισης του αρχείου.

Κλήσεις συστήματος

- Όταν λοιπόν ένα πρόγραμμα πραγματοποιεί μία κλήση συστήματος, η λειτουργία του διακόπτεται και το σύστημα μεταφέρεται σε kernel mode.
- Η τρέχουσα κατάσταση της διεργασίας αποθηκεύεται έτσι ώστε αργότερα να συνεχίσει από εκεί που σταμάτησε.
- Ο πυρήνας προσδιορίζει τι ακριβώς είναι αυτό που έχει ζητηθεί και εάν το αίτημα της διεργασίας είναι έγκυρο και εάν η διεργασία που ζήτησε την εξυπηρέτηση δικαιούται να το κάνει.
- Εάν πληρούνται οι παραπάνω προϋποθέσεις, ο πυρήνας προσπελαύνει το κατάλληλο σε κάθε περίπτωση hardware μέσω των οδηγών συσκευών για την εξυπηρέτηση του αιτήματος.
- Όταν ολοκληρωθεί η εξυπηρέτηση του αιτήματος, ο πυρήνας αποκαθιστά την τρέχουσα κατάσταση της διεργασίας έτσι ώστε αυτή να συνεχίσει από εκεί που σταμάτησε και επιστρέφει τον έλεγχο στη διεργασία – το σύστημα επιστρέφει ξανά σε user mode.

Δομές και δείκτες στη C

Στη γλώσσα προγραμματισμού C μια δομή επιτρέπει τον ορισμό ενός σύνθετου τύπου δεδομένων.

ΠΑΡΑΔΕΙΓΜΑ → Ένα σημείο στις τρεις διαστάσεις περιγράφεται από συντεταγμένες x, y, z.

Για την περιγραφή ενός σημείου έχουμε δύο επιλογές: (α) να ορίσουμε τρεις ανεξάρτητες μεταξύ τους μεταβλητές του κατάλληλου τύπου π.χ.

```
double x, y, z;
```

και να προχωρήσουμε στην αρχικοποίησή τους π.χ. γράφοντας

```
x=10;   y=20;   z=5;
```

ή να ορίσουμε μία δομή με όνομα point ως

```
struct point { double x; double y; double z;  };
```

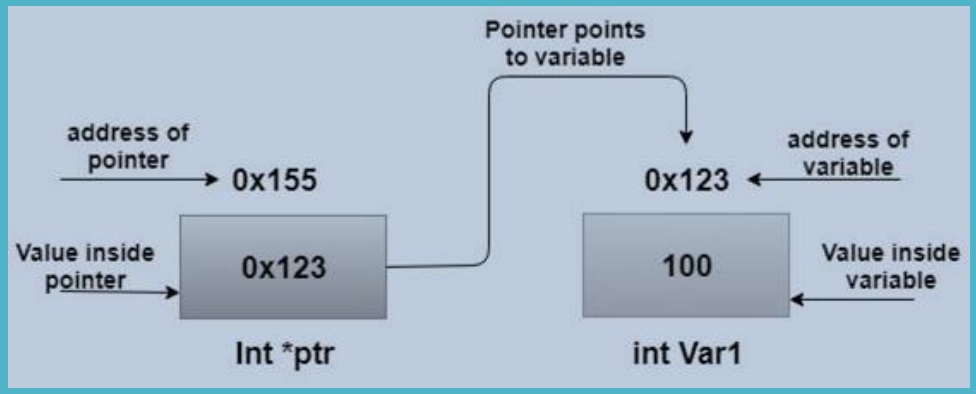
και αφού δηλώσουμε ένα point ως struct point p; να γράψουμε

```
p.x=10;      p.y=20;      p.z=5
```

όπου τώρα είναι εμφανές πως οι συντεταγμένες x, y και z αφορούν το ίδιο σημείο p.

Δομές και δείκτες στη C

Στη γλώσσα προγραμματισμού C ένας δείκτης (pointer) είναι μία μεταβλητή το περιεχόμενο της οποίας είναι η διεύθυνση μιας άλλης μεταβλητής.



Στη γλώσσα C ένας δείκτης προς μία μεταβλητή δηλώνεται χρησιμοποιώντας το σύμβολο * ενώ η διεύθυνση της μεταβλητής δηλώνεται χρησιμοποιώντας το σύμβολο & - θα είναι για παράδειγμα,

```
int Var1 = 100;
int *ptr = &Var1;
```

Θεωρώντας έναν πίνακα μεταβλητών κάποιου τύπου, π.χ έναν πίνακα ακεραίων, αυτός είναι ισοδύναμος με έναν δείκτη που δείχνει στο πρώτο κελί του πίνακα. Για παράδειγμα, εάν δηλώσουμε έναν πίνακα δέκα ακεραίων της μορφής int A [10], τα σύμβολα A και &A[0] εκφράζουν το ίδιο πράγμα.

Δομές και δείκτες στη C

Οι δείκτες αποτελούν σημαντικότατα στοιχεία της γλώσσας C αφού επιτρέπουν την πραγματοποίηση εξαιρετικά σημαντικών διαδικασιών που μεταξύ άλλων περιλαμβάνουν:

- Άμεση πρόσβαση στη μνήμη.
- Τη δυνατότητα επιστροφής περισσότερων από μία τιμές
- Τη μεταβίβαση πληροφοριών από την καλούσα συνάρτηση προς τη συνάρτηση που καλείται και αντίστροφα.
- Τη δυναμική δέσμευση και αποδέσμευση μνήμης.

Pass by value & Pass by reference

Θεωρώντας μία μεταβλητή π.χ. int n=5, μία μεταβλητή αναφοράς (reference variable) δεν είναι παρά ένα ψευδώνυμο αυτής της μεταβλητής που δηλώνεται ως int & r = n.

Αυτές οι μεταβλητές επιτρέπουν τη μεταβίβαση πληροφοριών ανάμεσα σε δύο συναρτήσεις func1 και func2 οι οποίες αλληλεπιδρούν καλώντας η μία την άλλη.

Οι παραπάνω αρχές ισχύουν στη γλώσσα C++ ενώ στην απλή C η αναφορά στις μεταβλητές γίνεται χρησιμοποιώντας δείκτες προς αυτές.

Δομές και δείκτες στη C

Παράδειγμα → η main καλεί τη συνάρτηση Twice που διπλασιάζει τις τιμές των ορισμάτων της

```c
#include <stdio.h>

void Twice (int a, int b)  {
      a *= 2;
      b *= 2; }

int main() {
     int x = 5, y = 8;
     printf ("Old values ==> x=%d, y=%d\n", x,y);
     Twice(x,y);
     printf ("New values ==> x=%d, y=%d\n", x,y);
     return (0); }
```

```
amarg@amarg-vbox:~$ ./twice
Old values ==> x=5, y=8
New values ==> x=5, y=8
amarg@amarg-vbox:~$
```

Παρατηρήστε πως η τιμή των μεταβλητών x και y δεν άλλαξαν τιμή! Σε αυτόν τον τύπο κλήσης (pass by value), οι παράμετροι a και b της συνάρτησης λαμβάνουν αντίγραφα των μεταβλητών x και y. Ωστόσο, επειδή είναι τοπικές μεταβλητές που παύουν να υφίστανται όταν τερματιστεί η συνάρτηση, δεν έχουν καμία επίδραση στη συνάρτηση main που κάλεσε την Twice.

```
amarg@amarg-vbox:~$ ./twice
Old values ==> x=5, y=8
Old values ==> a=5, b=8
Old values ==> a=10, b=16
New values ==> x=5, y=8
amarg@amarg-vbox:~$
```

Δομές και δείκτες στη C

Παράδειγμα → η main καλεί τη συνάρτηση Twice που διπλασιάζει τις τιμές των ορισμάτων της

```c
#include <stdio.h>

void Twice (int * a, int * b)  {
      printf ("Old values ==> a=%d, b=%d\n", *a,*b);
      (*a) *= 2;
      (*b) *= 2;
      printf ("Old values ==> a=%d, b=%d\n", *a,*b); }

int main() {
     int x = 5, y = 8;
     printf ("Old values ==> x=%d, y=%d\n", x,y);
     Twice(&x,&y);
     printf ("New values ==> x=%d, y=%d\n", x,y);
     return (0); }
```

```
amarg@amarg-vbox:~$ ./twice
Old values ==> x=5, y=8
Old values ==> a=5, b=8
Old values ==> a=10, b=16
New values ==> x=10, y=16
amarg@amarg-vbox:~$
```

Στην προκειμένη περίπτωση (pass by reference) εκείνο που αντιγράφεται στις μεταβλητές a και b δεν είναι οι τιμές των x και y αλλά οι διευθύνσεις αυτών των τιμών, οι οποίες αποτελούν αναφορές προς τις θέσεις αποθήκευσης των πραγματικών μεταβλητών. Με άλλα λόγια, δεν χρησιμοποιείται αντίγραφο των x και y αλλά τα πραγματικά x και y των οποίων επομένως οι τιμές μεταβάλλονται.

Δομές και δείκτες στη C

Εναλλακτικά μπορούμε να επιστρέψουμε το αποτέλεσμα στην καλούσα συνάρτηση χρησιμοποιώντας τη δεσμευμένη λέξη return όπως φαίνεται στη συνέχεια.

```c
#include <stdio.h>

int sum (int a, int b)  {
     int c;
     c = a + b;
     return c; }

int main() {
     int x = 5, y = 8, z = 0;
     z = sum (x, y);
     printf ("z=%d\n", z);
     return (0); }
```

```
amarg@amarg-vbox:~$ ./pbyval
z=13
amarg@amarg-vbox:~$ _
```

Γιατί τώρα το αποτέλεσμα είναι σωστό παρά το γεγονός πως χρησιμοποιούμε τοπικές μεταβλητές? Διότι, πολύ απλά, αυτό υπολογίστηκε πριν την επιστροφή της συνάρτησης sum και εκείνο που επιστράφηκε στην καλούσα συνάρτηση είναι ένα αντίγραφο της υπολογιζόμενης τιμής.

Αυτός ο μηχανισμός επιστροφής αποτελέσματος (return by value) είναι ο πλέον κατάλληλος για την επιστροφή των τιμών μεταβλητών που έχουν δηλωθεί εντός της συνάρτησης ή για την επιστροφή των τιμών ορισμάτων που έχουν διαβιβαστεί με τη μέθοδο pass by value. Ωστόσο δεν χρησιμοποιείται για μεγάλες δομές επειδή είναι σχετικά αργός (εκεί προτιμάται η επιστροφή της διεύθυνσης της τιμής [return by address] όπου δεν επιστρέφεται η μεταβλητή αλλά η διεύθυνσή της στη μνήμη).