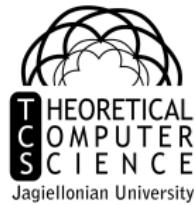


Systemy Operacyjne.

Jakub Kozik

Informatyka Analityczna
tcs@jagiellonian



Zasady

"This is not Nam. [...] There are rules."

Zaliczenie i egzamin

- Przynajmniej 3 z przynajmniej 4 dużych zadań.
- Submity poprzez Satori.
- Obrony projektów/zadań.

Program Wykładu

- ① POSIX - strona użytkownika.
- ② MINIX - strona systemu.

Zagadnienia:

- Procesy.
- Wejście/Wyjście.
- Pamięć.
- System plików.

THE MINIX BOOK



Andrew S Tanenbaum, Albert S Woodhull,
Operating Systems Design and Implementation,
3rd Edition, Pearson Prentice Hall 2009

- Andrew S. Tanenbaum, **Systemy operacyjne**
 - Abraham Silberschatz, James L. Peterson, Peter B. Galvin, **Podstawy systemów operacyjnych**
-
- ① <http://www.minix3.org/>
 - ② POSIX.1-2008 - IEEE Std 1003.1TM-2008 - The Open Group Technical Standard Base Specifications, Issue 7.

Outline

1 Zasady

2 POSIX

- Wstęp

- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

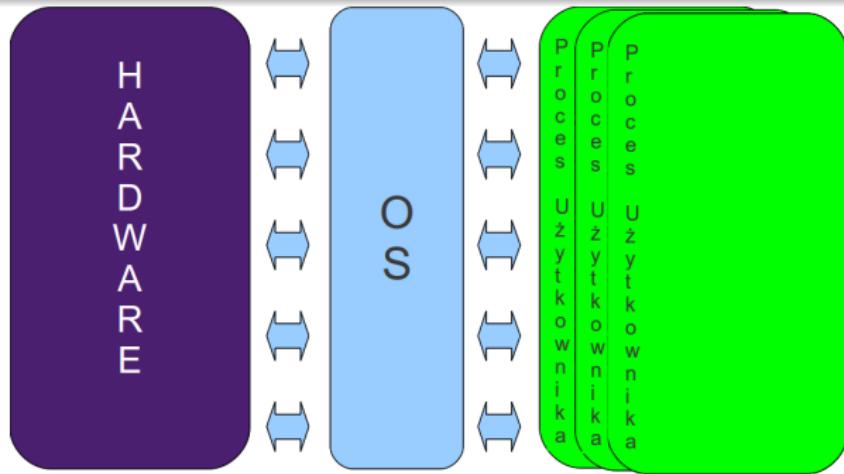
5 Scheduling

- Batch systems

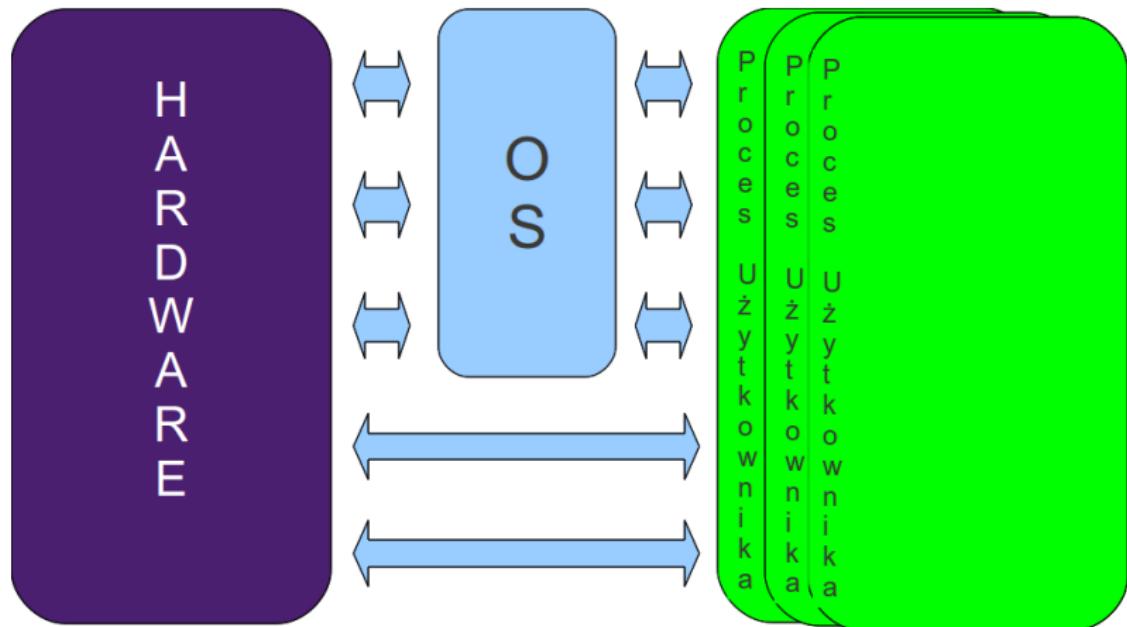
System Operacyjny

Główne funkcje systemu.

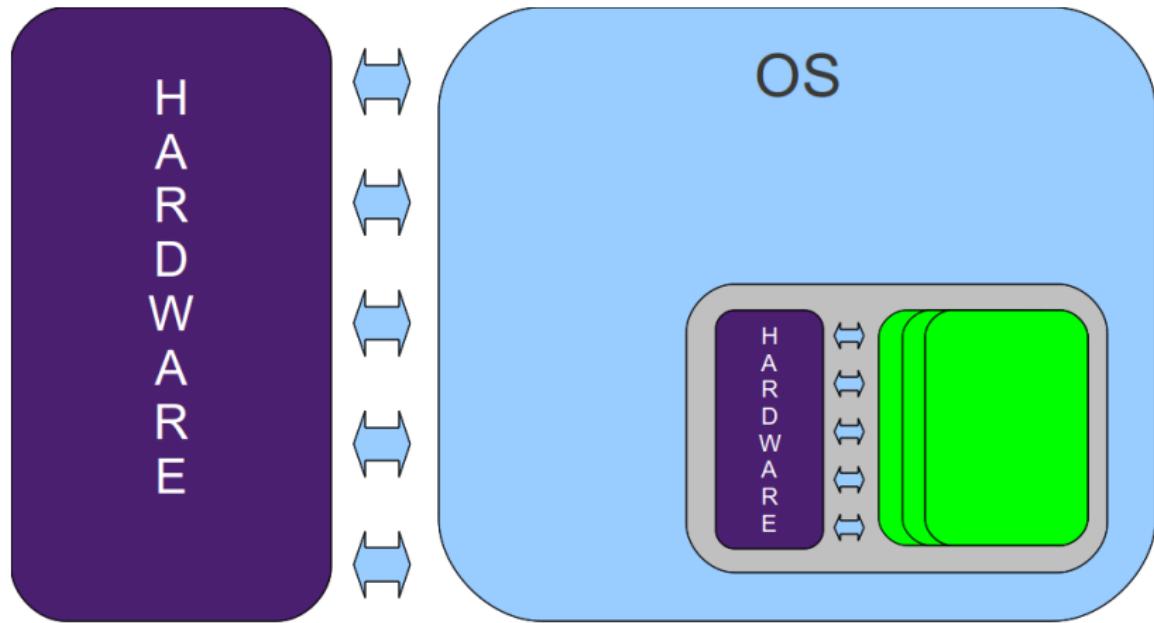
- Extended Machine
- Resource Management



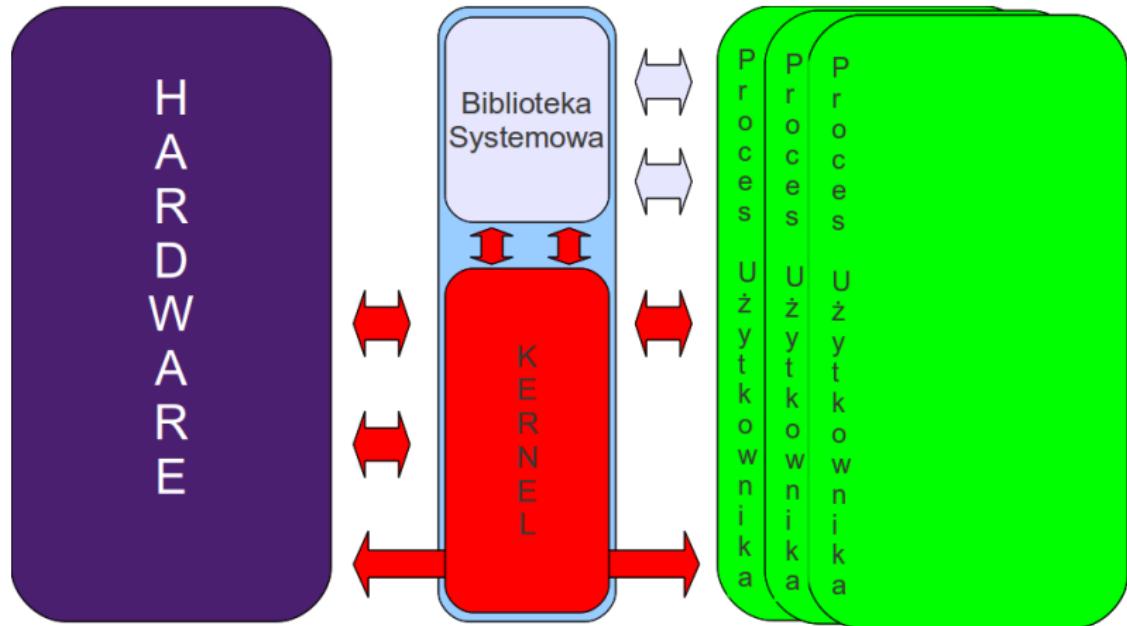
Bez zarządzania zasobami.



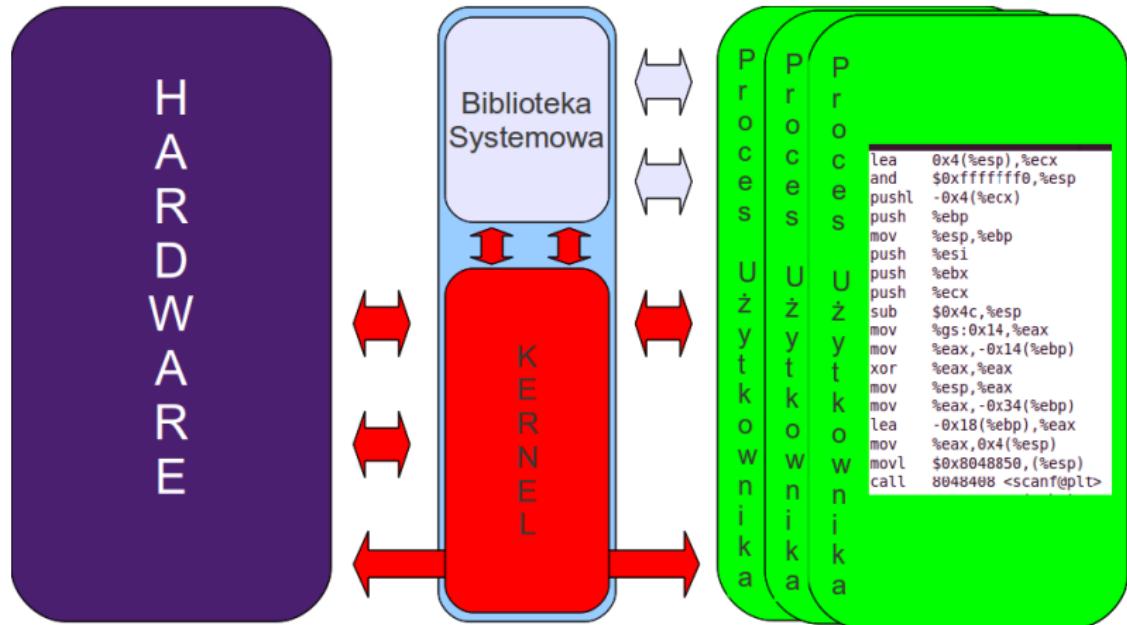
Wirtualna maszyna.



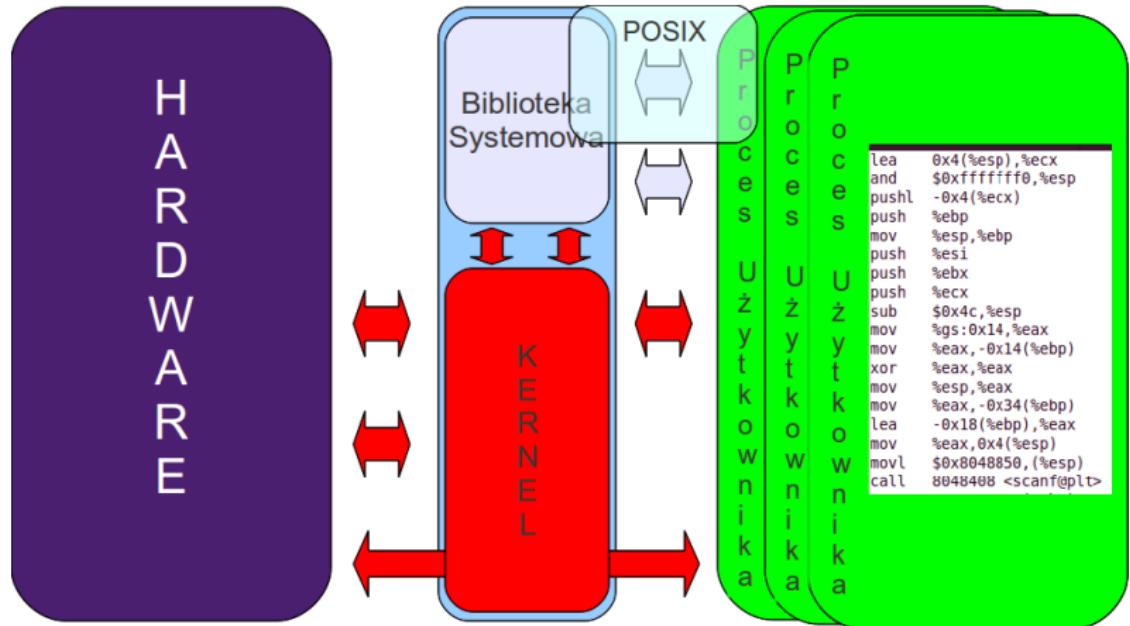
“Złoty środek.”



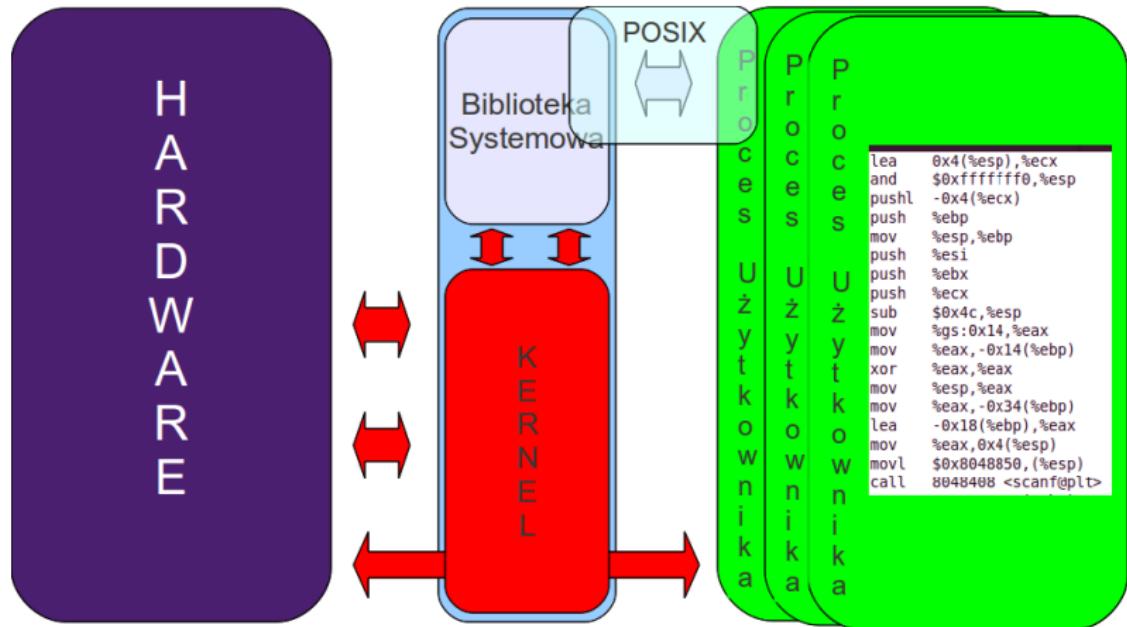
System calls - wywołania systemowe.



POSIX



POSIX programming.



Outline

1 Zasady

2 POSIX

- Wstęp
- **POSIX - standard**
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

5 Scheduling

- Batch systems

POSIX

Portable Operating System Interface

"**POSIX.1-2008** is simultaneously IEEE Std 1003.1TM-2008 and The Open Group Technical Standard Base Specifications, Issue 7."

POSIX principles:

- Application-Oriented
- Interface, Not Implementation
- Source, Not Object, Portability
- The C Language (ISO C)
- No Superuser, No System Administration
- Minimal Interface, Minimally Defined
- Broadly Implementable
- Minimal Changes to Historical Implementations
- Minimal Changes to Existing Application Code

POSIX.1-1990

IEEE Std. 1003.1-1990 Standard for Information Technology –
Portable Operating System Interface (POSIX) –
ART 1. System Application Programming Interface (API)
[C Language].

Donald Lewine, POSIX Programmers Guide, O'Reilly Media 1991

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- **POSIX - procesy**
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

5 Scheduling

- Batch systems

Proces

Program w trakcie wykonywania.

Procesy - system calls

```
1 #include <stdio.h>
2
3 int main(argc, argv)
4 int argc;
5 char** argv;
6 {
7     printf("Hey, you sass that hoopy Ford Prefect?\n");
8 }
```

Procesy - system calls

```
1 #include <stdio.h>
2
3 int main(argc, argv)
4 int argc;
5 char** argv;
6 {
7     printf("Hey, you sass that hoopy Ford Prefect?\n");
8 }
```

exit

```
#include <unistd.h>
void _exit(int status);

#include <stdlib.h>
void exit(int status);
```

Procesy - system calls

Linux - x86

```
08048080 <_start>:  
 8048080: b8 04 00 00 00          mov    $0x4,%eax  
 8048085: bb 01 00 00 00          mov    $0x1,%ebx  
 804808a: b9 a0 90 04 08          mov    $0x80490a0,%ecx  
 804808f: ba 06 00 00 00          mov    $0x6,%edx  
 8048094: cd 80                  int    $0x80  
 8048096: b8 01 00 00 00          mov    $0x1,%eax  
 804809b: cd 80                  int    $0x80
```

exit

```
#include <unistd.h>  
void _exit(int status);  
  
#include <stdlib.h>  
void exit(int status);
```

fork

```
#include <unistd.h>
pid_t fork(void);
```

```
int main(argc, argv)
int argc;
char* argv[];
{
    int k;

    printf("%d,%d\n",
           getpid(), getppid());
    k= fork();
    printf("%d,%d,%d\n",
           k, getpid(), getppid());
}
```

- unique process ID.
- different parent process ID
- own copy of the parent's descriptors.
- no pending signals,
inactive alarm timer

Fork bomb.

```
int main(){
    while (1) fork();
}
```

exec

execve

```
#include <unistd.h>

int execve(const char *path, char *const argv[], char *const envp[]);
```

```
extern char **environ;
```

```
int execl(const char *path, const char *arg0, ... /*, (char *)0 */);
int execle(const char *path, const char *arg0, ... /*,
            (char *)0, char *const envp[]*/);
int execlp(const char *file, const char *arg0, ... /*, (char *)0 */);
int execv(const char *path, char *const argv[]);
int execve(const char *path, char *const argv[], char *const envp[]);
int execvp(const char *file, char *const argv[]);
int fexecve(int fd, char *const argv[], char *const envp[]);
```

Deskryptory procesu wywołującego exec pozostają otwarte (domyślnie).

exec

```
#include <unistd.h>

int execve(const char *path, char *const argv[], char *const envp[]);
```

tic.c

```
1 #include <stdio.h>
2
3 int main(argc, argv)
4 int argc;
5 char** argv;
6 {
7     int i;
8     char* str= argv[1];
9
10    for (i=0; i<10; i++) {
11        printf("%s\n", str);
12        sleep(1);
13    }
14 }
```

tictac.c

```
1 #include <unistd.h>
2
3 int main(argc, argv)
4 int argc;
5 char** argv;
6 {
7     char* str;
8
9     if (fork()) str="tic";
10    else str="tac";
11
12    execl("tic", "tic", str, NULL)
13 }
```



waitpid

waitpid

```
#include <sys/wait.h>

pid_t wait(int *stat_loc);
pid_t waitpid(pid_t pid, int *stat_loc, int options);
```

`wait(stat_loc) ≡ waitpid(-1, stat_loc, 0)`

waitpid

```
1 #include <unistd.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <sys/types.h>
5 #define BSIZE 100
6
7 int main(){
8     char str[BSIZE];
9     pid_t chld_pid;
10
11    while (fgets(str, BSIZE, stdin)){
12        chld_pid= fork();
13        if (!chld_pid){
14            execl("echo", "echo", str, NULL);
15            exit(1);
16        } else
17            waitpid(chld_pid, NULL, 0);
18    }
19 }
```

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- **POSIX - pliki**
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

5 Scheduling

- Batch systems

Deskryptory plików

File Descriptor

“A **per-process unique, non-negative integer** used to identify an open file for the purpose of file access.

The value of a file descriptor is from zero to OPEN_MAX.”

limits.h

```
#define _POSIX_OPEN_MAX      16 /* a process may have 16 files open */  
...  
#define OPEN_MAX            20 /* # open files a process may have */
```

Open File Description

“A record of how a process or group of processes is accessing a file. Each file descriptor refers to exactly one open file description, but an open file description can be referred to by more than one file descriptor. The file offset, file status, and file access modes are attributes of an open file description.”

Deskryptory plików

Domyślnie otwarte deskryptory.

0 - stdin

1 - stdout

2 - stderr

open

```
#include <sys/types.h>
#include <fcntl.h>

int open(const char *path, int flags [, mode_t mode]);
```

(zwraca deskryptor dla otwartego pliku)

O_RDONLY	open for reading only
O_WRONLY	open for writing only
O_RDWR	open for reading and writing
O_NONBLOCK	do not block on open
O_APPEND	append on each write
O_CREAT	create file if it does not exist
O_TRUNC	truncate size to 0
O_EXCL	error if create and file exists

Semafor na plikach.

Atomic lock.

(O_CREAT | O_EXCL) - open() shall fail if the file exists.

```
#define LOCKFILE "/etc/ptmp"
...
int pfd; /* Integer for file descriptor returned by open() call. */
...
if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
    S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
{
    fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
    exit(1);
}
...
```

creat & close

close

```
#include <unistd.h>

int close(int d);
```

creat

```
#include <sys/types.h>
#include <fcntl.h>

int creat(const char *name, mode_t mode)
```

creat(path, mode) ≡ open(path, O_WRONLY|O_CREAT|O_TRUNC, mode)

Czytanie.

read

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

ssize_t read(int d, void *buf, size_t nbytes);
```

(zwraca liczbę przeczytanych byte'ów) (0 → EOF)

If a `read()` is interrupted by a signal before it reads any data, it shall return `-1` with `errno` set to `[EINTR]`.

If a `read()` is interrupted by a signal after it has successfully read some data, it shall return the number of bytes read.

Pisanie.

write

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

ssize_t write(int d, const void *buf, size_t nbytes);
```

(zwraca liczbę zapisanych byte'ów)

If `write()` is interrupted by a signal before it writes any data, it shall return `-1` with `errno` set to `[EINTR]`.

If `write()` is interrupted by a signal after it successfully writes some data, it shall return the number of bytes written.

lseek

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

#define SEEK_SET 0      /* offset is absolute */
#define SEEK_CUR 1      /* relative to current position */
#define SEEK_END 2      /* relative to end of file */

off_t lseek(int d, off_t offset, int whence)
```

```
1 #include <sys/stat.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4
5 int main(int argc, char* argv[]){
6     int fd=open("foo", O_RDWR|O_CREAT|O_TRUNC, S_IRUSR|S_IWUSR);
7
8     lseek(fd,1000000000L,SEEK_CUR); /*10GB*/
9     write(fd,"a",1);
10    close(fd);
11 }
```

pipe

```
#include <unistd.h>

int pipe(int fildes[2])
```

```
1 #include <stdio.h>
2
3 int main(int argc, char* argv[]){
4     int fd[2];
5
6     if (pipe(fd)<0) return 1;
7
8     if (fork()){
9         write(fd[1], "say_something", 13);
10    } else{
11        char buf[21];
12        int n= read(fd[0], buf, 20);
13        buf[n]=0;
14        printf("%s\n", buf+4);
15    }
16    return 0;
17 }
```



Named pipe - FIFO

mkfifo & mknod

```
#include <sys/types.h>
#include <unistd.h>
#include <sys/stat.h>

int mknod(const char *path, mode_t mode, dev_t dev)
int mkfifo(const char *path, mode_t mode)
```

pipe

Mknod may be invoked only by the super-user, unless it is being used to create a fifo.

The call `mkfifo(path, mode)` is equivalent to

```
mknod(path, (mode & 0777) | S_IFIFO, 0)
```

Pipe r/w rules.

Bad news

The behavior of multiple concurrent **reads** on the same pipe, FIFO, or terminal device is **unspecified**.

From read() - rationale

I/O is intended to be atomic to ordinary files and pipes and FIFOs. Atomic means that all the bytes from a single operation that started out together end up together, without interleaving from other I/O operations. It is a known attribute of terminals that this is not honored, and terminals are explicitly (and implicitly permanently) excepted, making the behavior unspecified. The behavior for other device types is also left unspecified, but the wording is intended to imply that future standards might choose to specify atomicity (**or not**).

Pipe r/w rules.

Good news

Write requests of PIPE_BUF bytes or less shall not be interleaved with data from other processes doing writes on the same pipe.

Writes of greater than PIPE_BUF bytes may have data interleaved, on arbitrary boundaries, with writes by other processes

```
1 #include <sys/stat.h>
2 #include <string.h>
3 #include <stdio.h>
4 int main(){
5     int fd[2], n;
6     char buf[4];
7     pipe(fd);
8     if (!fork()) {
9         while ((n = read(fd[0], buf, 3)) > 0){
10             buf[n] = 0;
11             printf("%s\n", buf);
12         }
13     } else {
14         sleep(1);
15         if (fork()) strcpy(buf, "tic");
16         else strcpy(buf, "tac");
17
18         for (n=1; n<10; n++){
19             write(fd[1], buf, 3);
20             sleep(1);
21         }
22     }
23 }
```



fcntl - file descriptor control functions

```
#include <fcntl.h>

int fcntl(int fd, int cmd, [data])
```

fcntl(fd, F_DUPFD, int fd2)

```
1 #include <fcntl.h>
2 #include <unistd.h>
3
4 int main(){
5     int fd[2];
6     pipe(fd);
7     if (!fork()) {
8         close(0);
9         close(fd[1]);
10        fcntl(fd[0], F_DUPFD, 0);
11        execlp("cat", "cat", NULL);
12    } else {
13        write(fd[1], "say-hello\n", 10);
14        close(fd[1]);
15        wait(NULL);
16    }
17 }
```

fcntl(fd, F_GETFD, int fd2) - fd flags

```
1 #include <fcntl.h>
2 #include <unistd.h>
3
4 int main(){
5     int fd[2];
6     pipe(fd);
7
8     int flags=fcntl(fd[1],F_GETFD);
9     flags|=FD_CLOEXEC;
10    fcntl(fd[1],F_SETFD, flags);
11
12    if (!fork()) {
13        close(0);
14    /*    close(fd[1]); */
15        fcntl(fd[0], F_DUPFD,0);
16        execlp("cat","cat",NULL);
17    } else {
18        write(fd[1],"say_hello\n",10);
19        close(fd[1]);
20    }
21 }
```

`fcntl(fd, F_GETFL, int fd2)` - file status flags

```
fcntl(fd, F_GETFL)
```

Return the file status flags and file access modes associated with the file associated with file descriptor fd.

```
fcntl(fd, F_SETFL, int flags)
```

Set the file status flags of the file referenced by fd to flags. Only O_NONBLOCK and O_APPEND may be changed. Access mode flags are ignored.

```
1 #include <fcntl.h>
2 #include <errno.h>
3 #include <unistd.h>
4 int main(){
5     int fd[2], n;
6     pipe(fd);
7     int flags=fcntl(fd[0], F_GETFL);
8     fcntl(fd[0], F_SETFL, flags | O_NONBLOCK);
9     if (!fork()) {
10         char buf[20];
11         close(fd[1]);
12         while ((n=read(fd[0], buf, 20))!=0){
13             if (n>0) write(0, buf, n);
14             else if (errno!=EAGAIN) return 1;
15             else write(0, "still nothing\n", 14);
16             sleep(1);
17         }
18     } else
19     for (n=0; n<5; n++){
20         sleep(3);
21         write(fd[1], "I am a walrus.\n", 16);
22     }
23 }
```



O_NONBLOCK for open

```
1 #include <fcntl.h>
2 #include <errno.h>
3 #include <unistd.h>
4 #include <sys/stat.h>
5 #include <stdio.h>
6
7 int main(){
8     int fd ,n; char buf[20];
9     mkfifo("fifo" , S_IWUSR | S_IRUSR );
10
11    if (!fork ()) {
12        fd=open("fifo" ,O_RDONLY|O_NONBLOCK);
13        write(1,"opened\n" ,7);
14        sleep(10);
15        while ((n=read(fd ,buf ,20))>0)
16            write(1,buf ,n);
17    } else{
18        sleep(10);
19        fd=open("fifo" ,O_WRONLY);
20        write(fd , "hello\n" ,6);
21        write(1,"done\n" ,5);
22    }
23 }
```

Advisory record locking.

```
fcntl(fd, F_GETLK, struct flock *lkp)
```

Find out if some other process has a lock on a segment of the file associated by file descriptor fd that overlaps with the segment described by the flock structure pointed to by lkp. [...]

```
fcntl(fd, F_SETLK, struct flock *lkp)
```

Register a lock on a segment of the file associated with file descriptor fd. [...] This call returns an error if any part of the segment is already locked.

```
fcntl(fd, F_SETLKW, struct flock *lkp)
```

Register a lock on a segment of the file associated with file descriptor fd. [...] This call blocks waiting for the lock to be released if any part of the segment is already locked.

```
struct flock {
    short    l_type;      /* F_RDLCK, F_WRLCK, or F_UNLCK */
    short    l_whence;    /* SEEK_SET, SEEK_CUR, or SEEK_END */
    off_t    l_start;     /* byte offset to start of segment */
    off_t    l_len;       /* length of segment */
    pid_t    l_pid;       /* process id of the locks' owner */
};
```

```
1 #include <fcntl.h>
2 #include <unistd.h>
3 #include <sys/stat.h>
4 int main(int argc, char * argv[]){
5     int fd;
6     struct flock fl;
7
8     fd= open("lock", O_CREAT | O_RDWR, S_IWUSR | S_IRUSR );
9     /*...*/
10    fl.l_type = F_WRLCK;
11    fl.l_whence = SEEK_SET;
12    fl.l_start = 0;
13    fl.l_len = 3;
14
15    fcntl(fd,F_SETLKW, &fl );
16    lseek(fd,0,SEEK_SET);
17    write(fd,argv[1],3);
18    sleep(30);
19    fl.l_type = F_UNLCK;
20    fcntl(fd,F_SETLK, &fl );
21    /*...*/
22    close(fd);
23 }
```



Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- **POSIX - sygnały**
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

5 Scheduling

- Batch systems

Sygnały

Sygnal

Informacja o **asynchronicznym** zdarzeniu/błędzie.

Ctrl-c

Ctrl-c powoduje wysłanie sygnału SIGINT do wszystkich procesów z *foreground process group*.

Dzielenie przez 0

Dzielenie liczby (int) przez (int) 0 powoduje wysłanie sygnału SIGFPE do procesu.

Źródła sygnałów.

Terminal Ctrl-C SIGINT,
Ctrl-\SIGQUIT

Hardware dzielenie przez 0 SIGFPE,
niewłaściwe odwołanie do pamięci SIGSEGV,...

Proces syscall kill, domyślny sygnał SIGTERM

System - Software conditions SIGALARM,
SIGPIPE (broken pipe)

Sygnały które nie docierają do adresata

SIGKILL

SIGSTOP

Wysyłanie sygnałów.

```
#include <signal.h>

int kill(pid_t pid, int sig);
```

Permission

...the real or effective user ID of the sending process shall match the real or saved set-user-ID of the receiving process.

Adresaci - pod warunkiem że można do nich wysyłać

`pid>0` proces, którego ID jest równe `pid`

`pid=0` procesy z tej samej grupy

`pid=-1` wszystkie procesy

`pid <-1` wszystkie procesy z grupy o ID równym $|pid|$

```
int raise(int sig);
```

Obsługa sygnałów - ISO C

```
#include <signal.h>

void (*signal(int signo, void (*func)(int)))(int);

typedef void Sigfunc(int);

Sigfunc *signal(int, Sigfunc *);
```

Obsługa sygnałów

SIG_DFL domyślna obsługa sygnału

SIG_IGN sygnał jest ignorowany

wskaźnik do funkcji która ma obsłużyć sygnał

Znikające i nieobsłużone sygnały.

```
1 #include <stdio.h>
2 #include <signal.h>
3
4 void handler(int sig_nb){
5     write(1,"If everything seems under control ,\"
6 you're just not going fast enough.\n",70);
7     sleep(1);
8     signal(SIGINT, handler);
9 }
10
11 int main(){
12     signal(SIGINT, handler);
13
14     while (1)
15         pause();
16 }
```

Obsługa sygnałów - POSIX

```
#include <signal.h>

int sigaction(int sig, const struct sigaction *restrict act,
              struct sigaction *restrict oact);
```

void (*sa_handler)(int)	Pointer to a signal-catching function or one of the SIG_IGN or SIG_DFL.
sigset_t sa_mask	Set of signals to be blocked during execution of the signal handling function.
int sa_flags	Special flags.
void (*sa_sigaction)(int, siginfo_t *, void *)	Pointer to a signal-catching function.

sigset_t

```
#include <signal.h>

int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signo);
int sigdelset(sigset_t *set, int signo);
int sigismember(const sigset_t *set, int signo);
```

Signal mask for the duration of the signal-catching function

This mask is formed by taking the union of the current signal mask and the value of the `sa_mask` for the signal being delivered, and unless `SA_NODEFER` or `SA_RESETHAND` is set, then including the signal being delivered.

```
1 #include <unistd.h>
2 #include <sys/types.h>
3 #include <signal.h>
4
5 int ready=0;
6 void handler(int sig_nb){
7     ready=1;
8 }
9
10 int main(){
11     pid_t other;
12     char* str="tic\n";
13     struct sigaction act;
14
15     act.sa_handler= handler;
16     act.sa_flags=0;
17     sigemptyset(&act.sa_mask);
18     sigaction(SIGUSR1,&act ,NULL);
19
20     if (!(other=fork())){
21         str="tac\n";
22         other= getppid();
23     } else ready=1;
24
25     while (1) {
26         if (ready){
27             ready = 0;
28             sleep(1);
29             write(1,str ,4);
30             kill(other ,SIGUSR1);
31         }
32         pause();
33     }
34 }
```

Flaga SA_SIGINFO

If SA_SIGINFO is set and the signal is caught,
the signal-catching function shall be entered as:

```
void func(int signo, siginfo_t *info, void *context);
```

info the reason why the signal was generated;

context the receiving thread's context that was interrupted when the signal was delivered.

Syscalla przerwane sygnałami.

read - przypomnienie

If a `read()` is interrupted by a signal before it reads any data, it shall return `-1` with `errno` set to `[EINTR]`.

If a `read()` is interrupted by a signal after it has successfully read some data, it shall return the number of bytes read.

write - przypomnienie

If `write()` is interrupted by a signal before it writes any data, it shall return `-1` with `errno` set to `[EINTR]`.

If `write()` is interrupted by a signal after it successfully writes some data, it shall return the number of bytes written.

```
1 #include <unistd.h>
2 #include <sys/types.h>
3 #include <signal.h>
4 #include <errno.h>
5
6 #define BSIZE 100
7 void handler(int sig_nb){}
8
9 int main(){
10    int n,k,w;
11    char buf[BSIZE];
12    pid_t parent, child;
13    struct sigaction act;
14
15    act.sa_handler= handler;
16    act.sa_flags=0;
17    sigemptyset(&act.sa_mask);
18    sigaction(SIGUSR1,&act ,NULL);
19
20    if (!(child=fork())){
21        parent= getppid();
22        while (1) kill(parent,SIGUSR1);
23        exit(1);
24    }
25
26    while (n = read(0,buf, BSIZE)){
27        if ((n<0) && (errno!=EINTR)) break;
28        k=0;
29        while(k<n){
30            w= write( 1, buf+k,n-k);
31            if ((w<0) && (errno!=EINTR)) goto end;
32            k+=w;
33        }
34    }
35 end:   kill(child, SIGTERM);
36 }
```



Flaga SA_RESTART

SA_RESTART

If set, and a function specified as interruptible is interrupted by this signal, the function shall restart and shall not fail with [EINTR] unless otherwise specified.

Przykłady

`read, write, open, waitpid, fcntl (F_SETLKW)`

```
1 #include <unistd.h>
2 #include <sys/types.h>
3 #include <signal.h>
4 #include <errno.h>
5
6 #define BSIZE 100
7 void handler(int sig_nb){}
8
9 int main(){
10    int n,k,w;
11    char buf[BSIZE];
12    pid_t parent, child;
13    struct sigaction act;
14
15    act.sa_handler= handler;
16    act.sa_flags=SA_RESTART;
17    sigemptyset(&act.sa_mask);
18    sigaction(SIGUSR1,&act,NULL);
19
20    if (!(child=fork())){
21        parent= getppid();
22        while (1) kill(parent,SIGUSR1);
23        exit(1);
24    }
25
26    while ((n = read(0,buf, BSIZE))>0){
27        k=0;
28        while (k<n){
29            w= write( 1, buf+k,n-k);
30            if (w<0) goto end;
31            k+=w;
32        }
33    }
34 end:   kill(child, SIGTERM);
35 }
```

UWAGA na errno!

```
1 #include <unistd.h>
2 #include <sys/types.h>
3 #include <signal.h>
4 #include <errno.h>
5
6 #define BSIZE 100
7 void handler(int sig_nb){ errno =0;}
8
9 int main(){
10    int n,k,w;
11    char buf[BSIZE];
12    pid_t parent, child;
13    struct sigaction act;
14
15    act.sa_handler= handler;
16    act.sa_flags=0;
17    sigemptyset(&act.sa_mask);
18    sigaction(SIGUSR1,&act ,NULL);
19
20    if (!(child=fork())){
21        parent= getppid();
22        while (1) kill(parent ,SIGUSR1);
23        exit(1);
24    }
25
26    while (n = read(0,buf, BSIZE)){
27        if ((n<0) && (errno!=EINTR)) break;
28        write( 1, buf,n); // nie dbamy o przerwane write'y
29    }
30 end:   kill(child , SIGTERM);
31 }
```

Flaga SA_NOCLDSTOP

SIGCHLD

Child process terminated, stopped, or continued.

SA_NOCLDSTOP

Do not generate SIGCHLD when children stop or stopped children continue.

Uwaga

If a process sets the action for the SIGCHLD signal to SIG_IGN, the behavior is unspecified.

Normalne sygnały NIE są kolejkowane!

```
1 #include <unistd.h>
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <sys/types.h>
5 #include <signal.h>
6
7 int s=0;
8 void handler(int sig_nb){ s++; }
9
10 int main(){
11     int n,k,w;
12     pid_t parent;
13     struct sigaction act;
14
15     act.sa_handler= handler;
16     act.sa_flags=0;
17     sigemptyset(&act.sa_mask);
18     sigaction(SIGUSR1,&act ,NULL);
19
20     if (!fork()){
21         parent= getppid();
22         for (n=0; n<10; n++) kill(parent ,SIGUSR1);
23         exit(1);
24     }
25
26     pause();
27     while (s-- >0) {
28         printf(" received\n");
29         sleep(3);
30     }
31 }
```

```
1 #include <unistd.h>
2 #include <stdio.h>
3 #include <sys/types.h>
4 #include <sys/wait.h>
5 #include <signal.h>
6
7 #define CHILDREN 10
8 volatile int z=CHILDREN;
9 void handler(int sig_nb){
10     pid_t child;
11     do{
12         child = waitpid(-1,NULL,WNOHANG);
13         if (child>0) z--;
14     } while (child>0);
15     sleep(1);
16 }
17
18 int main(){
19     int n;
20     struct sigaction act;
21     act.sa_handler= handler;
22     act.sa_flags=0;
23     sigemptyset(&act.sa_mask);
24     sigaction(SIGCHLD,&act ,NULL);
25
26     for (n=0; n<CHILDREN ; n++)
27         if (!fork()) {
28             sleep(n);
29             return 0;
30         }
31
32     while (z >0) {
33         printf("%d_children/zombies_left.\n",z );
34         sleep(1);
35     }
36     printf("No more zombies.\n");
37 }
```



SIG_IGN dla SIGCHLD

```
1 #include <unistd.h>
2 #include <signal.h>
3
4 #define CHILDREN 10
5
6 int main(){
7     int n;
8     struct sigaction act;
9     act.sa_handler= SIG_IGN;
10    act.sa_flags=0;
11    sigemptyset(&act.sa_mask);
12    sigaction(SIGCHLD,&act ,NULL);
13
14    for (n=0; n<CHILDREN ; n++)
15        if (!fork()) {
16            return 0;
17        }
18    sleep(10);
19 }
```

LINUX

Ignoring SIGCHLD can be used to prevent the creation of zombies.

async-signal-safe functions

safe

_exit, close, kill, read, write, ...

unsafe

malloc, exit, printf ...

alarm() function → SIGALRM signal

```
#include <unistd.h>

unsigned alarm(unsigned seconds);
```

Co może pójść źle w poniższym programie?

```
1 #include <unistd.h>
2 #include <stdio.h>
3 #include <signal.h>
4
5 #define TIME 5
6 void handler(int sig_nb){ }
7
8 int main(){
9     int n;
10    struct sigaction act;
11
12    act.sa_handler= handler;
13    act.sa_flags=0;
14    sigemptyset(&act.sa_mask);
15    sigaction(SIGALRM,&act ,NULL);
16
17    alarm(TIME);
18    pause();
19    printf("No_time... no_time_to_lose.\n");
20 }
```

sigprocmask

```
#include <signal.h>

int sigprocmask(int how, const sigset_t *restrict set,
                sigset_t *restrict oset);
```

how values

SIG_BLOCK The resulting set shall be the union of the current set and the signal set pointed to by set.

SIG_SETMASK The resulting set shall be the signal set pointed to by set.

SIG_UNBLOCK The resulting set shall be the intersection of the current set and the complement of the signal set pointed to by set.

alarm()

Trochę lepsze rozwiązanie.

```
1 #include <unistd.h>
2 #include <stdio.h>
3 #include <signal.h>
4
5 #define TIME 5
6 void handler(int sig_nb){ }
7
8 int main(){
9     int n;
10    struct sigaction act;
11    sigset(SIGALRM, handler);
12
13
14    act.sa_handler= handler;
15    act.sa_flags=0;
16    sigemptyset(&act.sa_mask );
17    sigaction(SIGALRM,&act ,NULL);
18    sigaction(SIGINT,&act ,NULL);
19
20    sigfillset(&mask );
21    sigdelset(&mask ,SIGALRM );
22
23    sigprocmask(SIG_SETMASK, &mask , NULL);
24    alarm(TIME);
25    pause();
26    printf("No_time... no_time to close.\n");
27 }
```

sigsuspend

```
#include <signal.h>

int sigsuspend(const sigset_t *sigmask);
```

```
1 #include <unistd.h>
2 #include <stdio.h>
3 #include <signal.h>
4 #define TIME 5
5 void handler(int sig_nb){ }
6
7 int main(){
8     struct sigaction act;
9     sigset_t mask;
10
11    act.sa_handler= handler;
12    act.sa_flags=0;
13    sigemptyset(&act.sa_mask );
14    sigaction(SIGALRM,&act ,NULL);
15    sigaction(SIGINT,&act ,NULL);
16
17    sigemptyset(&mask );
18    sigaddset(&mask ,SIGALRM );
19    sigprocmask(SIG_BLOCK, &mask , NULL);
20
21    sigfillset(&mask );
22    sigdelset(&mask , SIGALRM );
23    alarm(TIME);
24    if(sigsuspend(&mask ) != -1)
25        printf("No_time....no_time_to_lose.\n");
26 }
```



sigpending

```
#include <signal.h>

int sigpending(sigset_t *set);
```

The `sigpending()` function shall store, in the location referenced by the `set` argument, the set of signals that are blocked from delivery to the calling thread and that are pending on the process or the calling thread.

read with timeout - prawie poprawne

```
1 #include <unistd.h>
2 #include <stdio.h>
3 #include <signal.h>
4
5 volatile int time_is_up;
6 void handler(int sig_nb){ time_is_up=1;}
7
8 int tread(char * buf, int n, int timeout){
9     int r;
10    struct sigaction act, oact;
11    sigset(SIGALRM, handler);
12    act.sa_handler = handler;
13    act.sa_flags = 0;
14    sigemptyset(&act.sa_mask);
15    sigaction(SIGALRM, &act, &oact);
16
17    time_is_up = 0;
18    alarm(timeout);
19    do r = read(0, buf, n);
20    while ((!time_is_up) && (r < 0));
21    alarm(0);
22
23    if (r < 0) return 0;
24    sigaction(SIGALRM, &oact, NULL);
25    if (r < 0) return 0;
26    return r;
27 }
28 int main(){
29     char buf[100];
30     int n = tread(buf, 100, 5);
31     write(1, buf, n);
32 }
```



select

```
#include <sys/select.h>

int select(int nfds, fd_set *restrict readfds,
           fd_set *restrict writefds, fd_set *restrict errorfds,
           struct timeval *restrict timeout);
void FD_CLR(int fd, fd_set *fdset);
int FD_ISSET(int fd, fd_set *fdset);
void FD_SET(int fd, fd_set *fdset);
void FD_ZERO(fd_set *fdset);
```

Upon successful completion, the pselect() or select() function shall modify the objects pointed to by the readfds, writefds, and errorfds arguments to indicate which file descriptors are ready for reading, ready for writing, or have an error condition pending, respectively, and shall return the total number of ready descriptors in all the output sets.

```
struct timeval {
    long      tv_sec;          /* seconds */
    long      tv_usec;         /* microseconds */
};
```

read with timeout - poprawne

```
1 #include <unistd.h>
2 #include <stdio.h>
3 #include <errno.h>
4 #include <sys/select.h>
5 #include <sys/time.h>
6
7 int tread(char * buf, int n, int timeout){
8     int r;
9     fd_set rfds;
10    struct timeval timeout_s;
11
12    FD_ZERO(&rfds);
13    FD_SET(0,&rfds);
14
15    timeout_s.tv_sec=timeout;
16    timeout_s.tv_usec=0;
17
18    do r= select(1, &rfds, NULL, NULL, &timeout_s);
19    while ((r<0) && (errno== EINTR));
20
21    if (r<=0) return 0;
22
23    return (read(0, buf, n));
24 }
25 int main(){
26     char buf[100];
27     int n= tread(buf,100,5);
28     write(1,buf,n);
29 }
```

pselect

```
#include <sys/select.h>

int pselect(int nfds, fd_set *restrict readfds,
            fd_set *restrict writefds, fd_set *restrict errorfds,
            const struct timespec *restrict timeout,
            const sigset_t *restrict sigmask);
```

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- **POSIX - remanent**

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

5 Scheduling

- Batch systems

Procesy

getpriority, setpriority - get and set scheduling priority
setsid, getpgrp - create process group, get process group id
setuid, setgid - set user or group ID's
brk, sbrk - change data segment size

File System

access - determine accessibility of file
chmod - change mode of file
chown - change owner and group of a file
link - make a hard link to a file
mkdir - make a directory file
mount, umount - mount or umount a file system
rename - change the name of a file
rmdir - remove a directory file
stat, lstat, fstat - get file status
sync, fsync - update dirty buffers and super-block
unlink - remove directory entry
umask - set file creation mode mask
utime - set file times

Info

gettimeofday - get date and time
getuid, geteuid - get user identity
time, stime - get/set date and time
times - get process times
uname - get system info

Inne

chroot - change root directory

ptrace - process trace

reboot - close down the system or reboot

svrctl - special server control functions

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

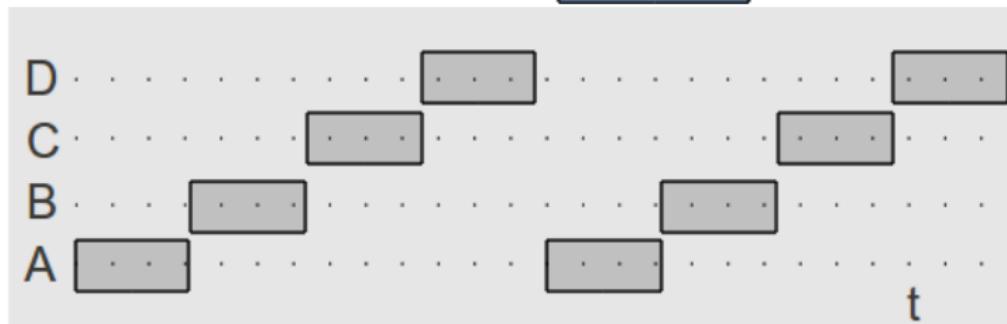
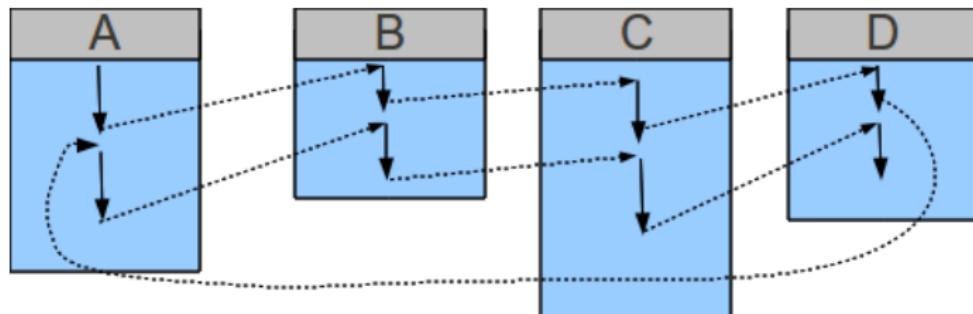
5 Scheduling

- Batch systems

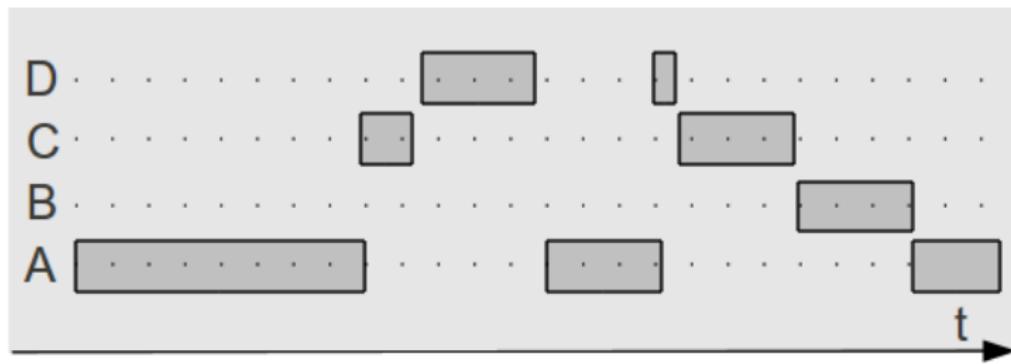
Multiprogramming (multitasking)

“Wirtualne procesory”

Każdy proces pracuje jak gdyby miał kopię procesora (z ograniczoną funkcjonalnością) dla siebie.

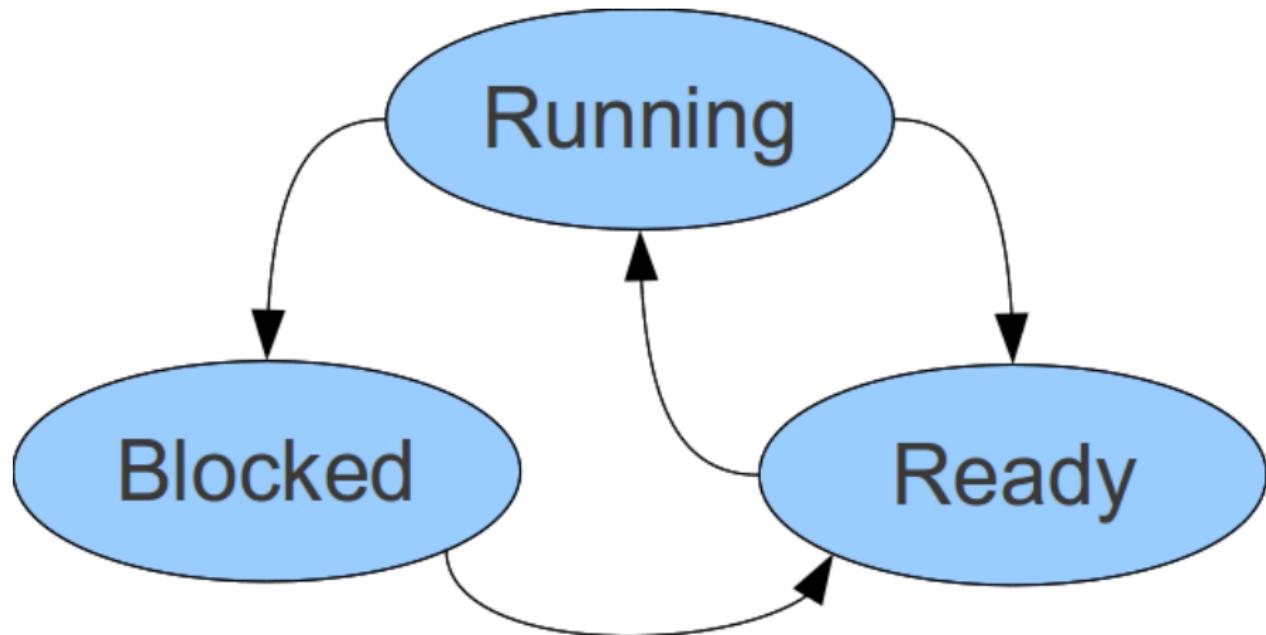


Multiprogramming (multitasking)



nierównomierny postęp w czasie → nie wolno robić założeń dotyczących czasu rzeczywistego.

Stany procesu



Opis procesu

Kernel

- Registers
- Program counter
- Program status word
- Stack Pointer
- Process state
- Current scheduling priority
- Maximum scheduling priority
- Scheduling ticks left
- Quantum size
- CPU time used
- Message queue pointers
- Pending signal bits
- Various flag bits
- Process name

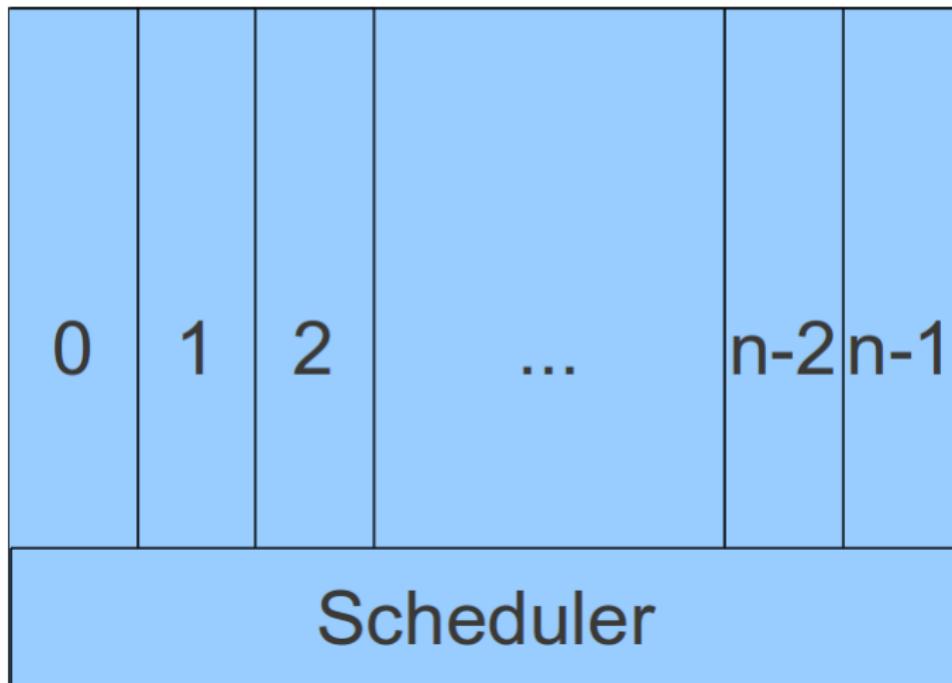
Process management

- Pointer to text segment
- Pointer to data segment
- Pointer to bss segment
- Exit status
- Signal status
- **Proces ID**
- Parent Process
- Process group
- Children's CPU time
- Real UID
- Effective UID
- Real GID
- Effective GID
- File info for sharing text
- Bitmaps for signals
- Various flag bits
- Process name

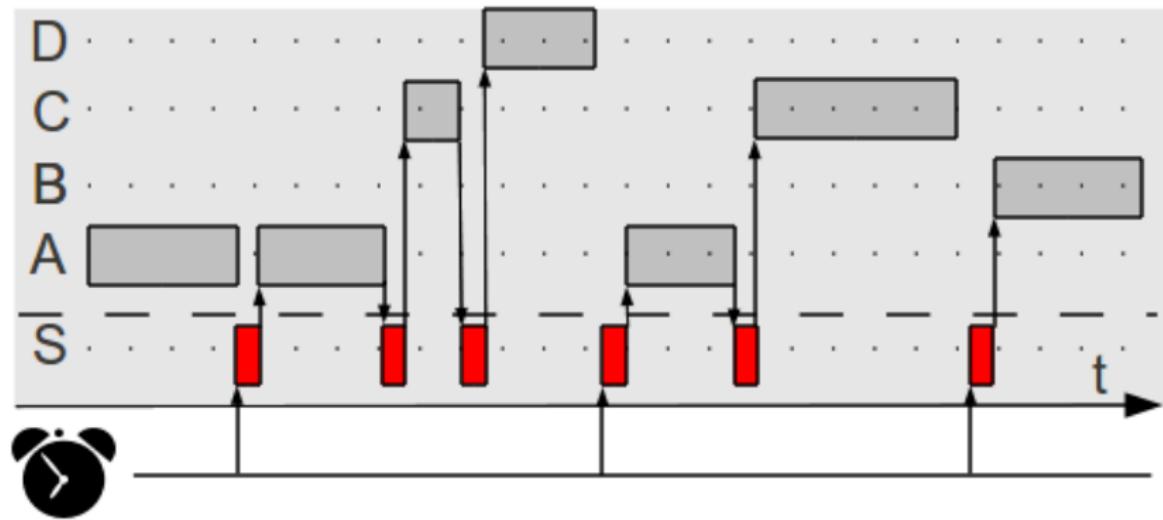
File management

- UMASK mask
- Root directory
- Working directory
- File descriptors
- Real UID
- Effective UID
- Real GID
- Effective GID
- Controlling tty
- Save area for read/write
- System call parameters
- Various flag bits

Procesy



Przerwania zegarowe.



MINIX - 60 przerwań na sekundę.

Pamięć - zmiany kontekstu

Kernel

- Registers
- Program counter
- Program status word
- Stack Pointer
- Process state
- Current scheduling priority
- Maximum scheduling priority
- Scheduling ticks left
- Quantum size
- CPU time used
- Message queue pointers
- Pending signal bits
- Various flag bits
- Process name

Process management

- Pointer to text segment
- Pointer to data segment
- Pointer to bss segment
- Exit status
- Signal status
- Process ID
- Parent Process
- Process group
- Children's CPU time
- Real UID
- Effective UID
- Real GID
- Effective GID
- File info for sharing text
- Bitmaps for signals
- Various flag bits
- Process name

File management

- UMASK mask
- Root directory
- Working directory
- File descriptors
- Real UID
- Effective UID
- Real GID
- Effective GID
- Controlling tty
- Save area for read/write
- System call parameters
- Various flag bits

Threads - wątki

Wiele “lekkich procesów” w tej samej przestrzeni adresowej.

Zmiana na inny wątek w tym samym procesie

- Registers
- Program counter
- Stack pointer
- State

wspólna pamięć → konieczna współpraca (synchronizacja).

systemowe / użytkownika

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

4 Architektura

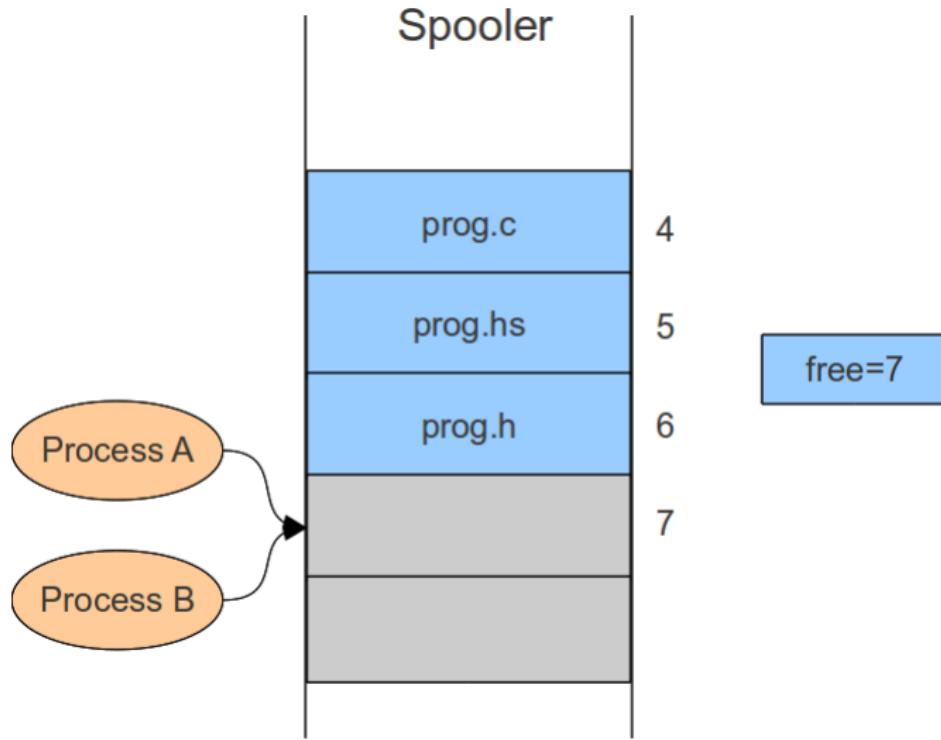
- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

5 Scheduling

- Batch systems

Współdzielona pamięć.

Race condition



Sekcje krytyczne

Sekcja krytyczna

Fragment programu z odwołaniami do współdzielonej pamięci.

Cel

Zawsze co najwyżej jeden proces/wątek w sekcji krytycznej.

Dodatkowo wymagamy:

- żaden proces działający poza sekcją krytyczną nie może blokować innego procesu,
- żaden proces nie powinien czekać w nieskończoność na wejście do sekcji krytycznej,
- mechanizm powinien działać niezależnie od szybkości i liczby procesorów.

Wyłączanie przerwań.

CLI - processor instruction

Clear Interrupt Flag – Clears the interrupt flag (IF) in the rFLAGS register to zero, thereby masking external interrupts received on the INTR input.

Wady

- konieczne uprawnienia do blokowania przerwań
- nieskuteczne w systemach wieloprocesorowych

Busy waiting - spin lock

```
while (TRUE) {  
    while (lock!=0);  
    lock = 1;  
    critical_region();  
    lock = 0;  
    noncritical_region();  
}
```

```
while (TRUE) {  
    while (lock!=0);  
    lock = 1;  
    critical_region();  
    lock = 0;  
    noncritical_region();  
}
```

Busy waiting - spin lock + turns

```
while (TRUE) {  
    while (turn!=0);  
    critical_region();  
    turn = 1;  
    noncritical_region();  
}
```

```
while (TRUE) {  
    while (turn!=1);  
    critical_region();  
    turn = 0;  
    noncritical_region();  
}
```

*Dude, [...] this determines who enters the next round robin. Am I wrong?
Am I wrong?*

Rozwiązańe Peterson'a

```
int turn;      //shared
int interested[2]; //shared

void enter_region(int process){
    int other;

    other = 1 - process;
    interested[process] = TRUE;
    turn = process;
    while ((turn==process) && (interested[other]==TRUE));
}

void leave_region(int process){
    interested[process] = FALSE;
}
```

Test and Set Lock instruction

TSL

TSL reg, lock - wczytuje zawartość pamięci lock do rejestru reg oraz zapisuje niezerową wartość pod adresem lock.

enter_region:

```
TSL eax, lock  
CMP eax, 0  
JNE enter_region  
RET
```

leave_region:

```
MOV lock, 0  
RET
```

x64

CMPXCHG reg/mem32, reg32

Compare EAX register with a 32-bit register or memory location. If equal, copy the second operand to the first operand. Otherwise, copy the first operand to EAX.

Kiedy używać spin-locków

Busy Waiting → Priority Inversion Problem

Sleep/Wakeup

```
#define N 100
int count=0;
void producer(void){
    int item;
    while (TRUE){
        item = produce_item();
        if (count==N) sleep();
        insert_item(item);
        count++;
        if (count==1) wakeup(consumer);
    }
}
void consumer(void){
    int item;
    while (TRUE){
        if (count==0) sleep();
        item = remove_item();
        count--;
        if (count==N-1) wakeup(producer);
        consume_item(item);
    }
}
```

Semaforы

```
#define N 100
semaphore mutex = 1;
semaphore empty = N;
semaphore full = 0;

void producer(void){
    int item;
    while (TRUE){
        item = produce_item();
        down(&empty);
        down(&mutex);
        insert_item(item);
        up(&mutex);
        up(&full);
    }
}

void consumer(void){
    int item;
    while (TRUE){
        down(&full);
        down(&mutex);
        item = remove_item();
        up(&mutex);
        up(&empty);
        consume_item(item);
    }
}
```

Semaforы - UWAGA!

```
#define N 100
semaphore mutex = 1;
semaphore empty = N;
semaphore full = 0;

void producer(void){
    int item;
    while (TRUE){
        item = produce_item();
        down(&mutex); //zmiana
        down(&empty); // kolejności
        insert_item(item);
        up(&mutex);
        up(&full);
    }
}

void consumer(void){
    int item;
    while (TRUE){
        down(&full);
        down(&mutex);
        item = remove_item();
        up(&mutex);
        up(&empty);
        consume_item(item);
    }
}
```

Monitor

```
monitor ProducerConsumer
    condition full, empty;
    int count;

    void insert(int item)
        if count==N then wait(full);
        insert_item(item);
        count++;
        if count==1 then signal(empty);

    int remove()
        if count==0 then wait(empty);
        remove = remove_item();
        count--;
        if (count= N-1) then signal(full);
```

Bez współdzielonej pamięci.

Message Passing (buffered)

```
#define N 100

void producer(void){
    int item;
    message m;

    while (TRUE){
        item = produce_item();
        receive(consumer, &m);
        build_message(&m, item);
        send(consumer, &m);
    }
}

void consumer(void){
    int item,i;
    message m;

    for (i=0; i<N; i++)
        send(producer,&m);
    while (TRUE){
        receive(producer,&m);
        item= extract_item(&m);
        send(producer,&m);
        consume_item(item);
    }
}
```

Message Passing - rendezvous

```
#define N 100

void producer(void){
    int item;
    message m;

    while (TRUE){
        item = produce_item();
        receive(consumer, &m);
        build_message(&m, item);
        send(consumer, &m);
    }
}

void consumer(void){
    int item,i;
    message m;

    send(producer,&m);
    while (TRUE){
        receive(producer,&m);
        item= extract_item(&m);
        send(producer,&m);
        consume_item(item);
    }
}
```

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

5 Scheduling

- Batch systems

Monolithic kernel - Linux

Procesy użytkownika pracują w dwóch trybach

- tryb użytkownika - user mode
- tryb jądra - kernel mode

Przejście do trybu jądra (system calls) poprzez specjalne bramki - w x86 przerwania programowe (int 80h) i specjalne instrukcje (sysenter, syscall).

Skutek: wiele procesów (użytkownika) może pracować równocześnie w trybie jądra.

Obsługa przerwań

- Scheduling.
- Drivery urządzeń w trybie jądra!

Do tego wątki systemu odpowiedzialne za takie rzeczy jak

- zapisywanie buforów (pdflush),
- kolejkę (drobnych) zadań jądra (keventd),
- zwalnianie/swaponowanie stron pamięci (kswapd),

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

4 Architektura

- Monolithic kernel
- **Micro kernel**
- Tanenbaum - Torvalds debate

5 Scheduling

- Batch systems

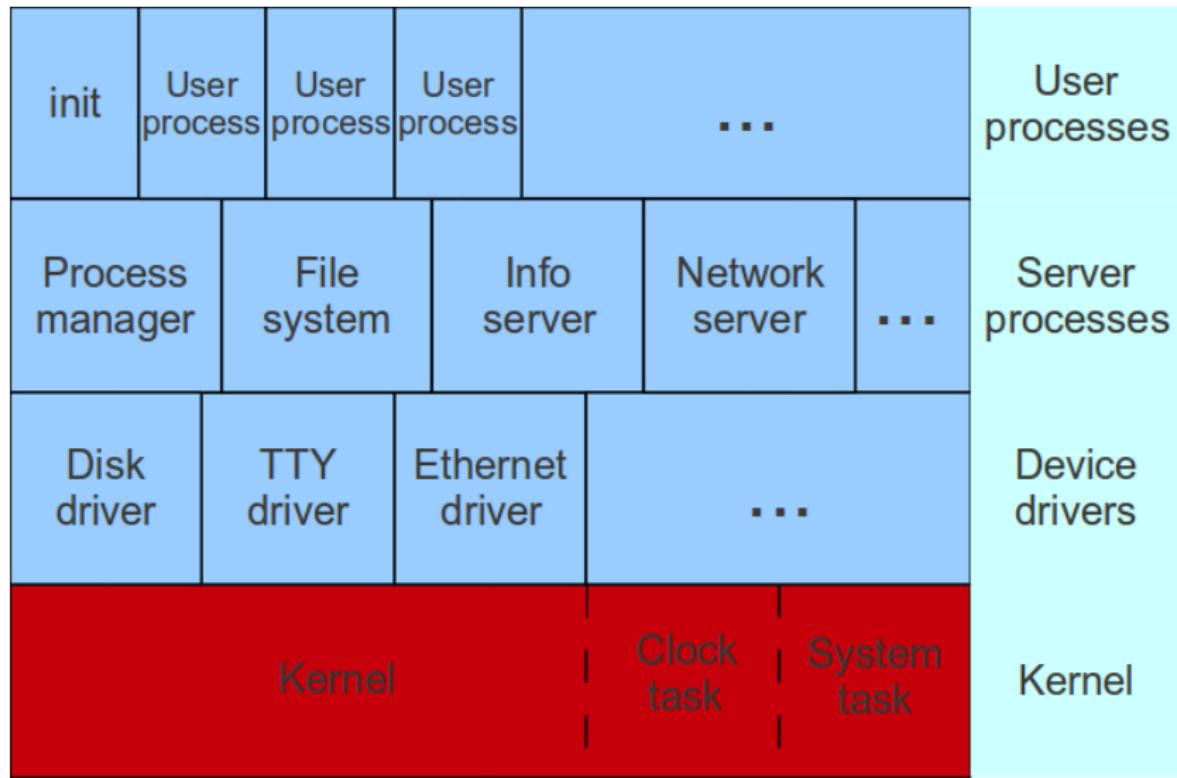
- Minimalizacja kodu pracującego w trybie uprzywilejowanym (kernel mode).
- Funkcjonalności systemu są zaimplementowane jako serwery (process manager, file system, network server).
- Komunikacja z serwerami - message passing.

Odpowiedzialność jądra

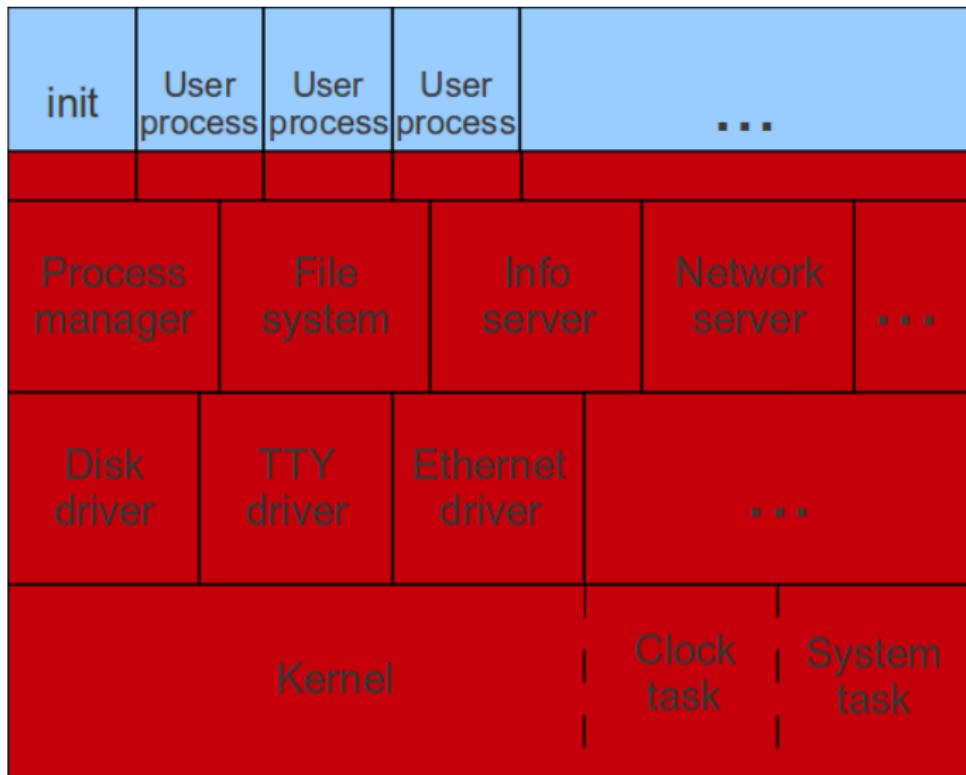
- multiprogramming,
- message - passing,
- mechanizmy dla obsługi urządzeń - drivery mogą pracować w trybie użytkownika.

Procesy użytkownika NIGDY nie pracują w trybie jądra.

Micro kernel - Minix



Monolithic kernel analogue



Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

4 Architektura

- Monolithic kernel
- Micro kernel
- **Tanenbaum - Torvalds debate**

5 Scheduling

- Batch systems

Tanenbaum - Torvalds debate

The Tanenbaum-Torvalds Debate, Open Sources: Voices from the Open Source Revolution, O'Reilly 1999

ast: ... LINUX is a monolithic style system. This is a giant step back into the 1970s. That is like taking an existing, working C program and rewriting it in BASIC. To me, writing a monolithic system in 1991 is a truly poor idea...

torvalds: ... True, linux is monolithic, and I agree that microkernels are nicer. With a less argumentative subject, I'd probably have agreed with most of what you said. From a theoretical (and aesthetical) standpoint linux looses. If the GNU kernel had been ready last spring, I'd not have bothered to even start my project: the fact is that it wasn't and still isn't. Linux wins heavily on points of being available now...

Tanenbaum - Torvalds debate

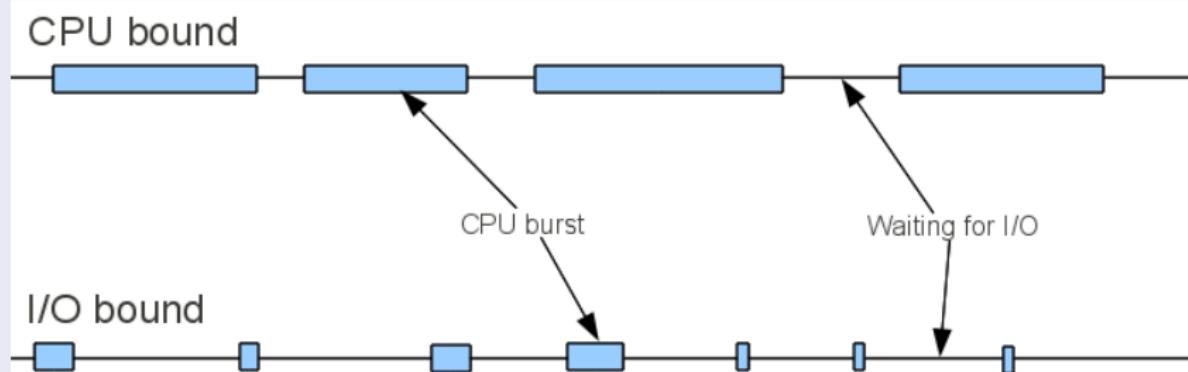
ast: ... Of course 5 years from now that will be different, but 5 years from now everyone will be running free GNU on their 200 MIPS, 64M SPARCstation-5...

PART II

Tanenbaum-Torvalds Debate: Part II, Tanenbaum's web page

Zachowanie procesów.

CPU-bound / I/O-bound



Kategorie procesów/schedulerów

- interaktywne
- wsadowe
- procesy czasu rzeczywistego

Kiedy scheduler wkracza do akcji?

Konieczna decyzja

- działający proces się blokuje
- działający proces się kończy

Możliwa zmiana procesu

- utworzenie nowego procesu
- przerwanie I/O
- przerwanie zegarowe (nonpreemptive / preemptive)

Wymagania

Wszystkie rodzaje

- Fairness - podobne procesy są podobnie traktowane
- Policy enforcement - uwzględnienie specyfiki procesów (priorytety itp.)
- Balance - wszystkie części systemu równo obciążone

Systemy wsadowe

- Throughput - maksymalizacja zadań na godzinę
- Turnaround time - minimalizacja czasu pomiędzy przyjściem a zakończeniem zadania (najlepiej znormalizowana wielkością zadania)
- CPU utilization - maksymalizacja obciążenia procesora

Wymagania - c.d.

Systemy interaktywne

- Response time - szybka reakcja na bodźce
- Proportionality - opóźnienie reakcji systemu proporcjonalne do czasu trwania zadania

Systemy czasu rzeczywistego

- Meeting deadlines
- Predictability

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

5 Scheduling

- Batch systems

Batch systems: First-Come First-Served

- Procesy obsługiwane w kolejności przyjścia.
- Zablokowane procesy trafiają na koniec kolejki kiedy przechodzą w stan gotowości.

Wady

- Słabe wskaźniki: Turnaround time, CPU utilization, Throughput.
- Wiele procesów może pracować współbieżnie - duże obciążenie pamięci.
- Jeśli wszystkie procesy są zablokowane na I/O to dodawane są nowe - czas oczekiwania na I/O się wydłuża.

Batch systems: Shortest Job First

- Zakładamy, że potrafimy oszacować czas wykonania zadania (totalny czas, nie tylko CPU).
- Wybieramy proces o najkrótszym czasie.

Zalety

- Off line: Optymalny z punktu widzenia Turnaround time

Wady

- On line: nie optymalny. Zadania A,B,C,D,E o wagach 2,4,1,1,1,1 przychodzą w chwilach 0,0,3,3,3. Średnia 4.6. Jeśli wykonać B,C,D,E,A to średnia 4.4.
- Nie ma mechanizmów równoważenia procesów (CPU-bound/I/O-bound)

Batch systems: Shortest Remaining Time Next

- Wybieramy proces o najkrótszym czasie pozostałym do zakończenia.
- Jeśli pojawia się nowy proces to dopuszczalne są wywłaszczenia.

Zalety

- Krótkie zadania mają krótki czas realizacji.
- Można uwzględnić blokowanie.
- Optymalny ze względu na Turnaround time w wersji on-line. Po zaniedbaniu kosztu zmiany kontekstu, schedulera, i ewentualnej korzyści z równoważenia.

Wady

- Nie ma mechanizmów równoważenia procesów (CPU-bound/I/O-bound).
- Pesymistyczny Turnaround-time może być bardzo duży.

Batch systems: Three-Level Scheduling

Admission scheduler

- Wybiera zadania które zaczną być wykonywane współbieżnie.
- Równoważenie zadań CPU-bound / I/O-bound.
- Minimalizacja normalizowanego Turnaround time.

Memory scheduler

- Działa jeśli procesy nie mieszczą się w pamięci.
- Wybiera procesy, które zostaną swapowane na dysk.
- Równoważenie zadań CPU-bound / I/O-bound.

CPU scheduler

- Wybiera następne zadanie do wykonania.
- Równoważenie zadań CPU-bound / I/O-bound.

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

5 Scheduling

- Batch systems

Interactive systems: Round robin

- Każdy proces dostaje kwant czasu (quantum).
- Procesy ustawione są w kolejce.
- Proces może obliczać tak długo jak trwa jego kwant.
- Proces który zużyje kwant, jest przerywany, dostaje nowe kwant i trafia na koniec kolejki.
- Proces, który się blokuje przed wykorzystaniem quantu, po przejściu w stan Ready, trafia na początek kolejki z pozostałym czasem.

Ważny parametr: długość kwantu

- kwant za duży - system może mieć długi czas reakcji.
- kwant za mały - duży odsetek pracy procesora tracony na zmiany kontekstu.
- w praktyce 20-50 msec.

Wady

- Wszystkie procesy są równo traktowane.

Interactive systems: Priority scheduling

Uogólnienie Round-robin. Wiele możliwości:

- Kwanty mogą zależeć od priorytetu.
- Procesy o wyższym priorytecie mają pierwszeństwo.
- Dla każdego priorytetu lista procesów - Round - robin.
- Dynamiczne/statyczne przydzielanie priorytetu (np. jeśli proces zużył $1/f$ swojego kwantu to dostaje priorytet f).
- Narastające kwanty w niższych priorytetach.

Ważny parametr: długość kwantu

- kwant za duży - system może mieć długi czas reakcji.
- kwant za mały - duży odsetek pracy procesora tracony na zmiany kontekstu.
- w praktyce 20-50 msec.

Guaranteed scheduling / Fair Share scheduling

- Każdy proces/użytkownik ma zagwarantowane $x\%$ procesora.
- Scheduler wybiera procesy tak aby wypełnić zobowiązanie.

Lottery scheduling

- Każdy proces dostaje kilka żetonów z puli.
- Scheduler wybiera losowo żeton i przydziela procesor właścielowi.
- Współpracujące procesy mogą sobie przekazywać żetony.

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

5 Scheduling

- Batch systems

- hard real time / soft real time.
- periodyczne, aperiodyczne zdarzenia.
- szeregowanie statyczne/dynamiczne.

Schedulable system

C_i - czas obsługi zdarzenia i

P_i - częstotliwość zdarzenia i

$$\sum_i \frac{C_i}{P_i} \leq 1$$

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

5 Scheduling

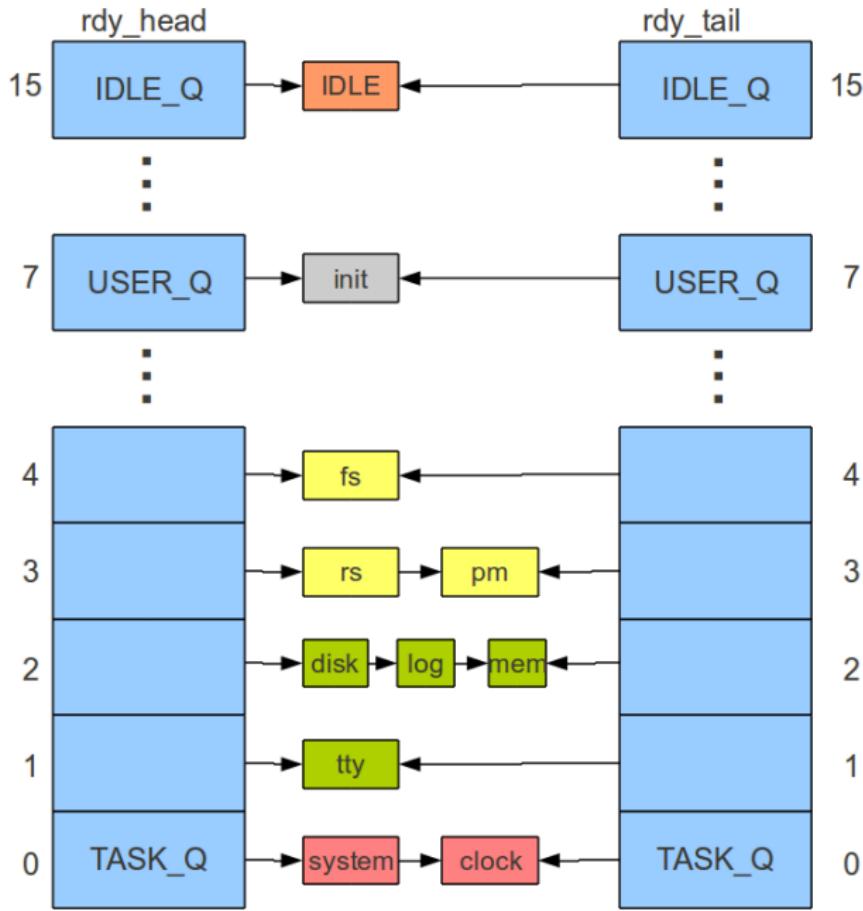
- Batch systems

Multilevel/Priority scheduling

- Dla każdego priorytetu kolejka.
- W obrębie kolejki round-robin.
- W kolejkach tylko procesy w stanie ready.
- Jeśli proces przechodzi w stan ready to:
 - Trafia na początek kolejki, jeśli nie wykorzystał całego czasu (kwantu) przed zablokowaniem,
 - wpp - trafia na koniec kolejki z nowym kwantem.

Wybór kolejnego procesu:

Weź pierwszy proces z niepustej kolejki o największym priorytecie.



Degradacja priorytetu

- +1 Proces zużył cały kwant i dwa razy z rzędu został mu przydzielony procesor.
- 1 wpp.

- Nie dotyczy zadań jądra (kernel tasks) oraz procesu IDLE.
- Priorytet nie może być większy niż IDLE_Q-1.
- Priorytet nie może być mniejszy niż maksymalny priorytet zależny od rodzaju procesu.

```
PUBLIC int _syscall(int who, int syscallnr, message * msgptr){  
    int status;           /*lib/other/syscall.c*/  
  
    msgptr->m_type = syscallnr;  
    status = _sendrec(who, msgptr);  
    if (status != 0) {  
        /* 'sendrec' itself failed. */  
        msgptr->m_type = status;  
    }  
    if (msgptr->m_type < 0) {  
        errno = -msgptr->m_type;  
        return(-1);  
    }  
    return(msgptr->m_type);  
}
```

```
--sendrec:          ! lib/i386/rts/_ipc.s  
    push ebp  
    mov ebp, esp  
    push ebx  
    mov eax, SRC_DST(ebp)  ! eax = dest-src  
    mov ebx, MESSAGE(ebp)  ! ebx = message pointer  
    mov ecx, SENDREC      ! _sendrec(srcdest, ptr)  
    int SYSVEC            ! trap to the kernel  
    pop ebx  
    pop ebp  
    ret
```

```
_s_call:           ! kernel/mpx386.s
_p_s_call:
    cld          ! set direction flag to a known value
    sub esp, 6*4 ! skip RETADR, eax, ecx, edx, ebx, est
    push ebp      ! stack already points into proc table
    push esi
    push edi
    o16 push ds
    o16 push es
    o16 push fs
    o16 push gs
    mov dx, ss
    mov ds, dx
    mov es, dx
    incb (_k_reenter)
    mov esi, esp  ! assumes P_STACKBASE == 0
    mov esp, k_stktop
    xor ebp, ebp  ! for stacktrace
                  ! end of inline save
                  ! now set up parameters for sys_call()
    push ebx      ! pointer to user message
    push eax      ! src/dest
    push ecx      ! SEND/RECEIVE/BOTH
    call _sys_call ! sys_call(function, src_dest, m_ptr)
                  ! caller is now explicitly in proc_ptr
    mov AXREG(esi), eax ! sys_call MUST PRESERVE si
```

```
! Fall into code to restart proc/task running.  
_restart:                      ! kernel/mpx386.s  
  
! Restart the current process or the next process if it is set.  
    cmp (_next_ptr), 0      ! see if another process is scheduled  
    jz 0f  
    mov  eax, (_next_ptr)  
    mov  (_proc_ptr), eax    ! schedule new process  
    mov  (_next_ptr), 0  
0:   mov esp, (_proc_ptr)    ! will assume P_STACKBASE == 0  
    lldt P_LDT_SEL(esp) ! enable process' segment descriptors  
    lea eax, P_STACKTOP(esp) ! arrange for next interrupt  
    mov (_tss+TSS3_S_SP0), eax ! to save state in process table  
restart1:  
    decb (_k_reenter)  
    o16 pop gs  
    o16 pop fs  
    o16 pop es  
    o16 pop ds  
    popad  
    add esp, 4      ! skip return adr  
    iretd          ! continue process
```

```
1 PUBLIC int sys_call(call_nr, src_dst, m_ptr)
2 int call_nr;          /* system call number and flags */
3 int src_dst;          /* src to receive from or dst to send to */
4 message *m_ptr;       /* pointer to message in the caller's space */
5 {
6     /.../
7     int function = call_nr & SYSCALL_FUNC; /* get system call function
8     /...// check several conditions
9     switch(function) {
10         case SENDREC:
11             /* A flag is set so that notifications cannot interrupt SENDREC */
12             priv(caller_ptr)->s_flags |= SENDREC_BUSY;
13             /* fall through */
14         case SEND:
15             result = mini_send(caller_ptr, src_dst, m_ptr, flags);
16             if (function == SEND || result != OK) {
17                 break;           /* done, or SEND failed */
18             }                  /* fall through for SENDREC */
19         case RECEIVE:
20             if (function == RECEIVE)
21                 priv(caller_ptr)->s_flags &= ~SENDREC_BUSY;
22             result = mini_receive(caller_ptr, src_dst, m_ptr, flags);
23             break;
24     /...// NOTIFY, ECHO, default
25 }
26 return(result);
27 }
```



```

1 PRIVATE int mini_send(caller_ptr, dst, m_ptr, flags)
2 register struct proc *caller_ptr; /* who is trying to send a message? */
3 int dst; /* to whom is message being sent? */
4 message *m_ptr; /* pointer to message buffer */
5 unsigned flags; /* system call flags */
6 {
7 /* Send a message from 'caller_ptr' to 'dst'. */
8 register struct proc *dst_ptr = proc_addr(dst);
9 register struct proc **xpp;
10 register struct proc *xp;
11 /*.../* Check for deadlock by 'caller_ptr' and 'dst' sending to each other. */
12
13 /* Check if 'dst' is blocked waiting for this message. The destination's
14 * SENDING flag may be set when its SENDREC call blocked while sending.
15 */
16 if ((dst_ptr->p_rts_flags & (RECEIVING | SENDING)) == RECEIVING &&
17     (dst_ptr->p_getfrom == ANY || dst_ptr->p_getfrom == caller_ptr->p_nr)) {
18     /* Destination is indeed waiting for this message. */
19     CopyMess(caller_ptr->p_nr, caller_ptr, m_ptr, dst_ptr, dst_ptr->p_messbuf);
20     if ((dst_ptr->p_rts_flags & ~RECEIVING) == 0) enqueue(dst_ptr);
21 } else if ( !(flags & NON_BLOCKING)) {
22     /* Destination is not waiting. Block and dequeue caller. */
23     caller_ptr->p_messbuf = m_ptr;
24     if (caller_ptr->p_rts_flags == 0) dequeue(caller_ptr);
25     caller_ptr->p_rts_flags |= SENDING;
26     caller_ptr->p_sendto = dst;
27     /* Process is now blocked. Put in on the destination's queue. */
28     xpp = &dst_ptr->p_caller_q; /* find end of list */
29     while (*xpp != NIL_PROC) xpp = &(*xpp)->p_q_link;
30     *xpp = caller_ptr; /* add caller to end */
31     caller_ptr->p_q_link = NIL_PROC; /* mark new end of list */
32 } else {
33     return(ENOTREADY);
34 }
35 return(OK);
36 }

```



```

1 PRIVATE void enqueue(rp)
2 register struct proc *rp; /* this process is now runnable */
3 {
4     int q;           /* scheduling queue to use */
5     int front;       /* add to front or back */
6
7     /* Determine where to insert to process. */
8     sched(rp, &q, &front);
9
10    /* Now add the process to the queue. */
11    if (rdy_head[q] == NIL_PROC) { /* add to empty queue */
12        rdy_head[q] = rdy_tail[q] = rp; /* create a new queue */
13        rp->p_nextready = NIL_PROC; /* mark new end */
14    }
15    else if (front) { /* add to head of queue */
16        rp->p_nextready = rdy_head[q]; /* chain head of queue */
17        rdy_head[q] = rp; /* set new queue head */
18    }
19    else { /* add to tail of queue */
20        rdy_tail[q]->p_nextready = rp; /* chain tail of queue */
21        rdy_tail[q] = rp; /* set new queue tail */
22        rp->p_nextready = NIL_PROC; /* mark new end */
23    }
24
25    /* Now select the next process to run. */
26    pick_proc();
27 }

```

```

1 /*
2 *      pick_proc          *
3 */
4 PRIVATE void pick_proc()
5 {
6 /* Decide who to run now. A new process is selected by setting 'next_ptr'.
7 * When a billable process is selected, record it in 'bill_ptr', so that the
8 * clock task can tell who to bill for system time.
9 */
10 register struct proc *rp;      /* process to run */
11 int q;                      /* iterate over queues */
12
13 /* Check each of the scheduling queues for ready processes. The number of
14 * queues is defined in proc.h, and priorities are set in the image table.
15 * The lowest queue contains IDLE, which is always ready.
16 */
17 for (q=0; q < NR_SCHED_QUEUES; q++) {
18     if ( (rp = rdy_head[q]) != NIL_PROC) {
19         next_ptr = rp;        /* run process 'rp' next */
20         if (priv(rp)->s_flags & BILLABLE)
21             bill_ptr = rp;    /* bill for system time */
22         return;
23     }
24 }
25 }

```

```

1 PRIVATE void sched(rp, queue, front)
2 register struct proc *rp;           /* process to be scheduled */
3 int *queue;                      /* return: queue to use */
4 int *front;                      /* return: front or back */
5 {
6     static struct proc *prev_ptr = NIL_PROC; /* previous without time */
7     int time_left = (rp->p_ticks_left > 0); /* quantum fully consumed */
8     int penalty = 0;                  /* change in priority */
9
10    if ( ! time_left ) {            /* quantum consumed ? */
11        rp->p_ticks_left = rp->p_quantum_size; /* give new quantum */
12        if ( prev_ptr == rp ) penalty++; /* catch infinite loops */
13        else penalty--;           /* give slow way back */
14        prev_ptr = rp;             /* store ptr for next */
15    }
16
17    /* Determine the new priority of this process. The bounds are determined
18     * by IDLE's queue and the maximum priority of this process. Kernel task
19     * and the idle process are never changed in priority.
20    */
21    if (penalty != 0 && !iskernelp(rp)) {
22        rp->p_priority += penalty; /* update with penalty */
23        if (rp->p_priority < rp->p_max_priority) /* check upper bound */
24            rp->p_priority=rp->p_max_priority;
25        else if (rp->p_priority > IDLE_Q-1) /* check lower bound */
26            rp->p_priority = IDLE_Q-1;
27    }
28
29    /* If there is time left, the process is added to the front of its queue,
30     * so that it can immediately run. The queue to use simply is always the
31     * process' current priority.
32    */
33    *queue = rp->p_priority;
34    *front = time_left;
35 }

```

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

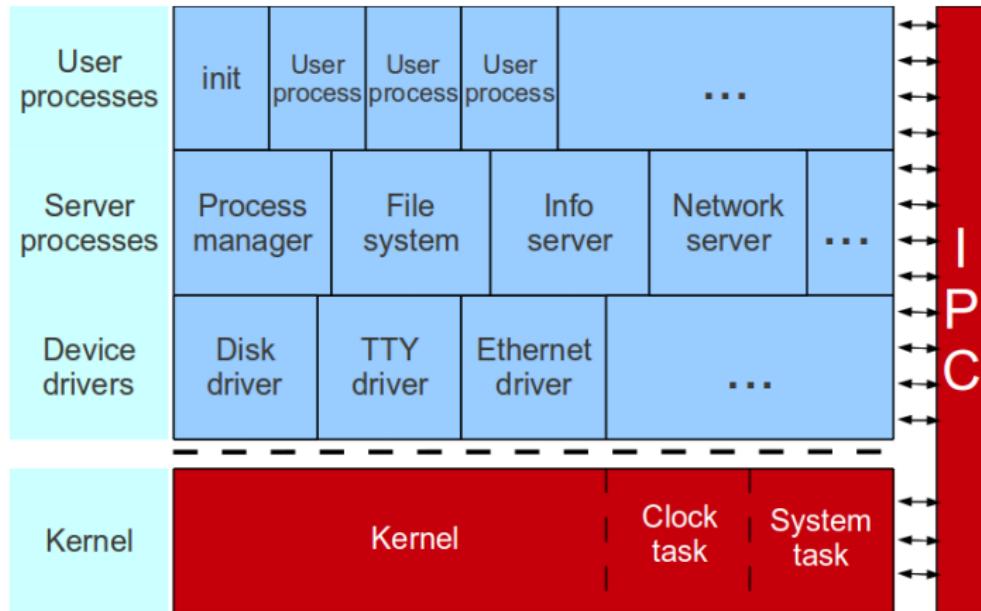
4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

5 Scheduling

- Batch systems

IPC - message passing



Message Passing Primitives

- SEND=1
- RECEIVE=2
- SENDREC=3
- NOTIFY=4

Kernel trap

```
push    ebp
mov     ebp,  esp
push    ebx
mov     eax, SRC_DST(ebp) ! eax = dest-src
mov     ebx, MESSAGE(ebp) ! ebx = message pointer
mov     ecx, {SEND|RECEIVE|SENDREC|NOTIFY}
int    SYSVEC           ! 33
pop    ebx
pop    ebp
ret
```

```
1 PUBLIC int sys_call(call_nr, src_dst, m_ptr)
2 int call_nr;          /* system call number and flags */
3 int src_dst;          /* src to receive from or dst to send to */
4 message *m_ptr;       /* pointer to message in the caller's space */
5 {
6     /* ... */
7     int function = call_nr & SYSCALL_FUNC; /* get system call function
8     /* ... // check several conditions
9     switch(function) {
10         case SENDREC:
11             /* A flag is set so that notifications cannot interrupt SENDREC */
12             priv(caller_ptr)->s_flags |= SENDREC_BUSY;
13             /* fall through */
14         case SEND:
15             result = mini_send(caller_ptr, src_dst, m_ptr, flags);
16             if (function == SEND || result != OK) {
17                 break;           /* done, or SEND failed */
18             }                  /* fall through for SENDREC */
19         case RECEIVE:
20             if (function == RECEIVE)
21                 priv(caller_ptr)->s_flags &= ~SENDREC_BUSY;
22             result = mini_receive(caller_ptr, src_dst, m_ptr, flags);
23             break;
24         case NOTIFY:
25             result = mini_notify(caller_ptr, src_dst);
26             break;
27     /* ... // ECHO, default */
```

Message Passing Primitives

- SEND ,RECEIVE,SENDREC - są blokujące (rendez-vous). Uwaga na deadlock!
- NOTIFY - nie blokujące, nie buforowane
- Wiadomości wychodzące z jądra (kernel tasks) nie przechodzą przez sys_call.

Testy w sys_call

- Wiadomości mogą być wysyłane do zadań jądra wyłącznie przez SENDREC.
- Każdy proces ma maskę operacji IPC które może wykonywać (w strukturze priv) - procesy użytkownika mogą używać jedynie SENDREC.
- Argument src_dst musi być prawidłowym identyfikatorem procesu/zadania lub ANY dla RECEIVE.
- Wskaźnik do wiadomości musi się znajdować w pamięci procesu wołającego.
- Każdy proces ma maskę procesów do których może wysyłać (w strukturze priv) - procesy użytkownika mogą wysyłać do PM, FS, RS.
- Proces wskazywany przez src_dst musi być żywy lub ANY.

```

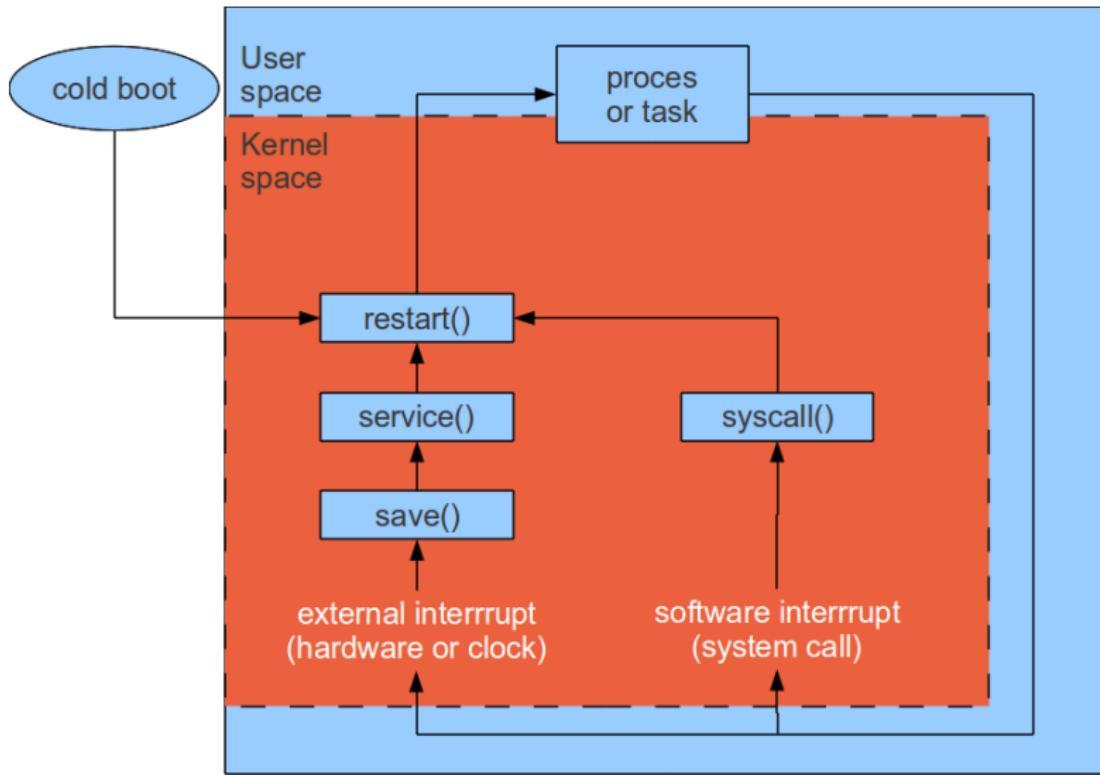
1 PRIVATE int mini_notify(caller_ptr, dst)
2 register struct proc *caller_ptr; /* sender of the notification */
3 int dst; /* which process to notify */
4 {
5     register struct proc *dst_ptr = proc_addr(dst);
6     int src_id; /* source id for late delivery */
7     message m; /* the notification message */
8
9     /* Check to see if target is blocked waiting for this message. A process
10    * can be both sending and receiving during a SENDREC system call.
11    */
12    if ((dst_ptr->p_rts_flags & (RECEIVING|SENDING)) == RECEIVING &&
13        ! (priv(dst_ptr)->s_flags & SENDREC_BUSY) &&
14        (dst_ptr->p_getfrom == ANY || dst_ptr->p_getfrom == caller_ptr->p_nr)) {
15
16        /* Destination is indeed waiting for a message. Assemble a notification
17        * message and deliver it. Copy from pseudo-source HARDWARE, since the
18        * message is in the kernel's address space.
19        */
20        BuildMess(&m, proc_nr(caller_ptr), dst_ptr);
21        CopyMess(proc_nr(caller_ptr), proc_addr(HARDWARE), &m,
22                  dst_ptr, dst_ptr->p_messbuf);
23        dst_ptr->p_rts_flags &= ~RECEIVING; /* deblock destination */
24        if (dst_ptr->p_rts_flags == 0) enqueue(dst_ptr);
25        return(OK);
26    }
27
28    /* Destination is not ready to receive the notification. Add it to the
29    * bit map with pending notifications. Note the indirectness: the system id
30    * instead of the process number is used in the pending bit map.
31    */
32    src_id = priv(caller_ptr)->s_id;
33    set_sys_bit(priv(dst_ptr)->s_notify_pending, src_id);
34    return(OK);
35 }

```

```
1 PRIVATE int mini_receive(caller_ptr, src, m_ptr, flags)
2 register struct proc *caller_ptr; /* process trying to get message */
3 int src;                      /* which message source is wanted */
4 message *m_ptr;                /* pointer to message buffer */
5 unsigned flags;                /* system call flags */
6 {
7     ...
8     if (!(caller_ptr->p_rts_flags & SENDING)) {
9
10    /* Check if there are pending notifications , except for SENDREC. */
11    if (! (priv(caller_ptr)->s_flags & SENDREC_BUSY)) {
12        map = &priv(caller_ptr)->s_notify_pending;
13        for (chunk=&map->chunk[0]; \
14             chunk<&map->chunk[NR_SYS_CHUNKS]; \
15             chunk++) {
16            /* Find a pending notification from the requested source. */
17            ...
18            /* Found a suitable source ,
19             * deliver the notification message. */
20            BuildMess(&m, src_proc_nr, caller_ptr); /* assemble message */
21            CopyMess(src_proc_nr, proc_addr(HARDWARE), \
22                      &m, caller_ptr, m_ptr);
23            return(OK);           /* report success */
24        }
25    }
26 ...
```

```
1 ...
2     /* Check caller queue. Use pointer pointers to keep code simple. */
3     xpp = &caller_ptr->p_caller_q;
4     while (*xpp != NIL_PROC) {
5         if (src == ANY || src == proc_nr(*xpp)) {
6             /* Found acceptable message. Copy it and update status. */
7             CopyMess((*xpp)->p_nr,*xpp,(*xpp)->p_messbuf,caller_ptr,m);
8             if (((*xpp)->p_rts_flags&= ~SENDING) == 0) enqueue(*xpp);
9             *xpp = (*xpp)->p_q_link;      /* remove from queue */
10            return(OK);                /* report success */
11        }
12        xpp = &(*xpp)->p_q_link;      /* proceed to next */
13    }
14 }
15 ...
```

```
1 ...
2 /* No suitable message is available or the caller couldn't
3 * send in SENDREC.
4 * Block the process trying to receive ,
5 * unless the flags tell otherwise .
6 */
7 if( !(flags & NON_BLOCKING)) {
8     caller_ptr->p_getfrom = src;
9     caller_ptr->p_messbuf = m_ptr;
10    if (caller_ptr->p_rts_flags == 0) dequeue(caller_ptr);
11    caller_ptr->p_rts_flags |= RECEIVING;
12    return(OK);
13 } else {
14     return(ENOTREADY);
15 }
16 }
```



```
1 PRIVATE int generic_handler(hook)
2 irq_hook_t *hook;
3 {
4 /* This function handles hardware interrupt in a simple and generic
5 * way. All interrupts are transformed into messages to a driver.
6 * The IRQ line will be reenabled if the policy says so.
7 */
8
9 /* As a side-effect, the interrupt handler gathers random
10 * information by timestamping the interrupt events.
11 * This is used for /dev/random.
12 */
13 get_randomness(hook->irq );
14
15 /* Add a bit for this interrupt to the process' pending interrupts.
16 * When sending the notification message, this bit map will be
17 * magically set as an argument.
18 */
19 priv(proc_addr(hook->proc_nr))->s_int_pending |=(1<<hook->notify_id );
20
21 /* Build notification message and return. */
22 lock_notify(HARDWARE, hook->proc_nr);
23 return(hook->policy & IRQ_REENABLE);
24 }
```

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

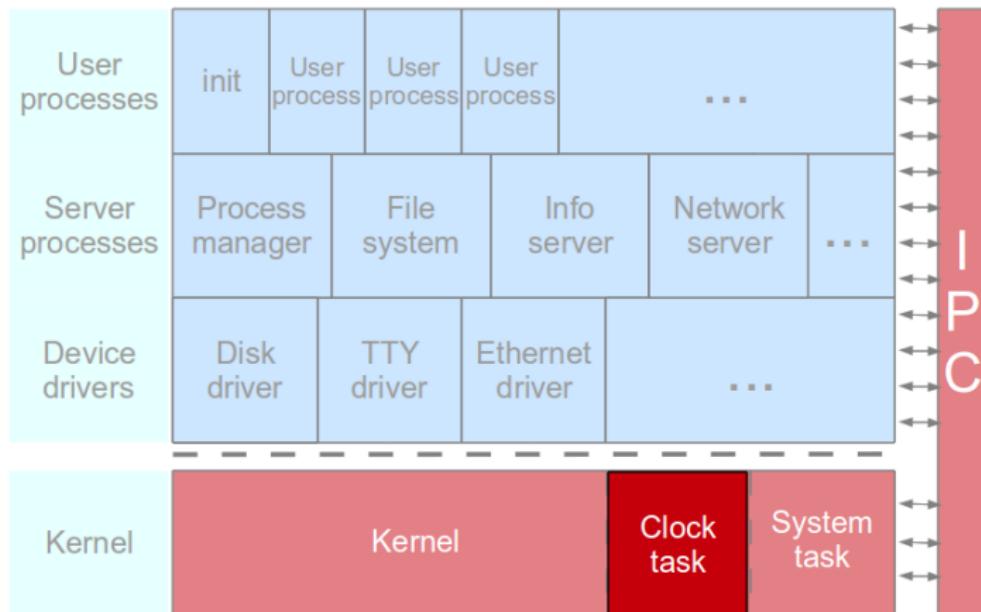
4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

5 Scheduling

- Batch systems

Clock task



Funkcje zegara

`clock_handler()` Obsługa przerwań zegara - aktualizacja czasu,
"księgowość", sprawdzanie alarmów i potrzeby wywłaszenia.

`clock_task()` Wywłaszczenie procesów których czas się wyczerpał,
uruchamianie alarmów.

`inne` Funkcje dotyczące zegara:

- `get_uptime()`
- `set_timer()`
- `reset_timer()`
- `read_clock()`
- `clock_stop()`

```
1 PRIVATE int clock_handler(hook)
2 irq_hook_t *hook;
3 {
4     register unsigned ticks;
5
6     /* Acknowledge the PS/2 clock interrupt. */
7     if (machine.ps_mca) outb(PORT_B, inb(PORT_B) | CLOCK_ACK_BIT);
8
9     /* Get number of ticks and update realtime. */
10    ticks = lost_ticks + 1;
11    lost_ticks = 0;
12    realtime += ticks;
13
14    proc_ptr->p_user_time += ticks;
15    if (priv(proc_ptr)->s_flags & PREEMPTIBLE) {
16        proc_ptr->p_ticks_left -= ticks;
17    }
18    if (! (priv(proc_ptr)->s_flags & BILLABLE)) {
19        bill_ptr->p_sys_time += ticks;
20        bill_ptr->p_ticks_left -= ticks;
21    }
22
23    /* Check if do_clocktick() must be called. Done for alarms and scheduling.
24     * Some processes , such as the kernel tasks , cannot be preempted.
25     */
26    if ((next_timeout <= realtime) || (proc_ptr->p_ticks_left <= 0)) {
27        prev_ptr = proc_ptr;      /* store running process */
28        lock_notify(HARDWARE, CLOCK); /* send notification */
29    }
30    return(1);           /* reenable interrupts */
31 }
```

```
1 PUBLIC void clock_task()
2 {
3 /* Main program of clock task. If the call is not HARD_INT it is an e
4 */
5 message m;          /* message buffer for both input and output */
6 int result;         /* result returned by the handler */
7
8 init_clock();        /* initialize clock task */
9
10 /* Main loop of the clock task. Get work, process it. Never reply . */
11 while (TRUE) {
12
13     /* Go get a message. */
14     receive(ANY, &m);
15
16     /* Handle the request. Only clock ticks are expected. */
17     switch (m.m_type) {
18         case HARD_INT:
19             result = do_clocktick(&m); /* handle clock tick */
20             break;
21         default:           /* illegal request type */
22             kprintf("CLOCK: illegal request %d from %d.\n", m.m_type, m.
23     }
24 }
25 }
```

```
1 PRIVATE int do_clocktick(m_ptr)
2 message *m_ptr;           /* pointer to request message */
3 {
4     if (prev_ptr->p_ticks_left <= 0 &&
5         priv(prev_ptr)->s_flags & PREEMPTIBLE) {
6         lock_dequeue(prev_ptr);    /* take it off the queues */
7         lock_enqueue(prev_ptr);   /* and reinsert it again */
8     }
9
10    /* Check if a clock timer expired and run its watchdog function. */
11    if (next_timeout <= realtime) {
12        tmrs_exptimers(&clock_timers, realtime, NULL);
13        next_timeout = clock_timers == NULL ?
14            TMR_NEVER : clock_timers->tmr_exp_time;
15    }
16
17    /* Inhibit sending a reply. */
18    return(EDONTREPLY);
19 }
```

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

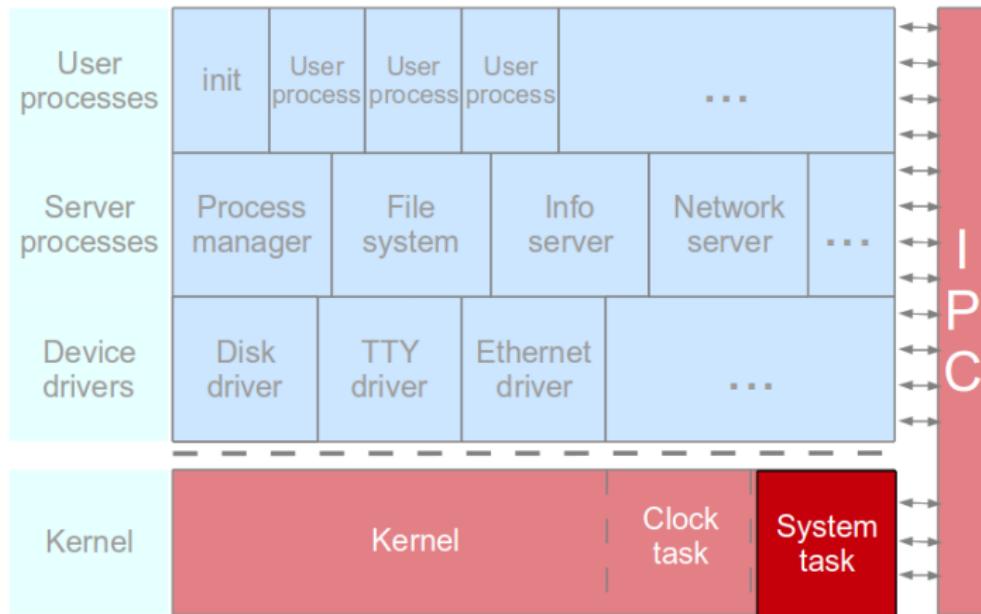
4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

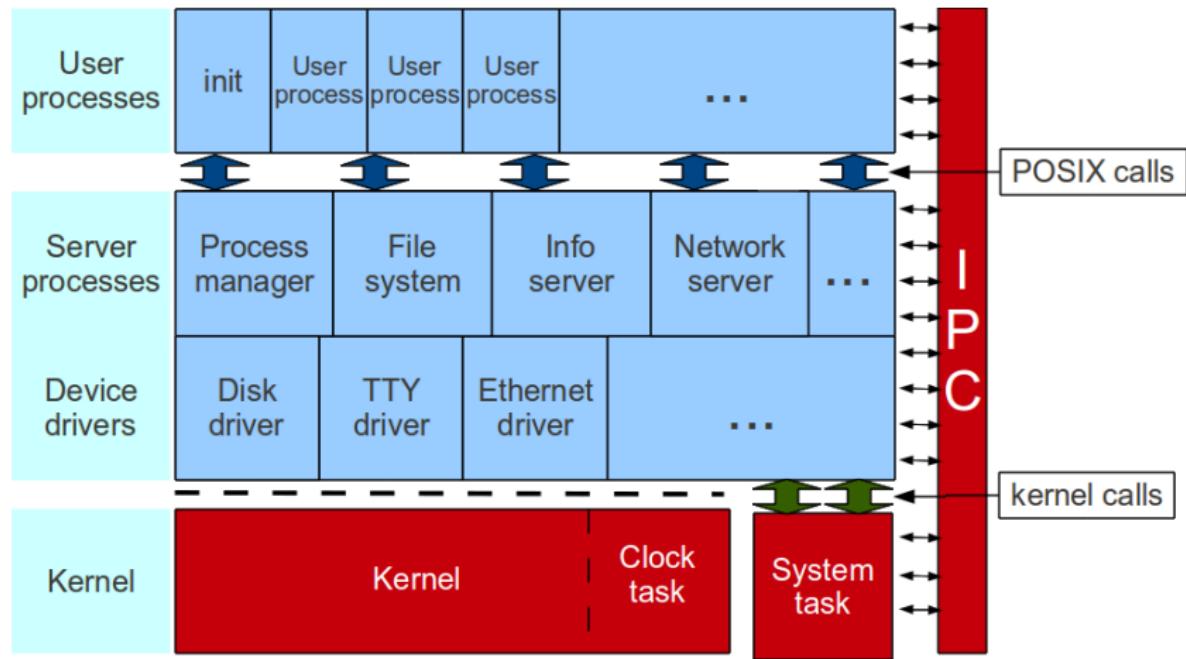
5 Scheduling

- Batch systems

System task



System task



Kernel calls

Message type	From	Meaning
sys_fork	PM	A process has forked.
sys_exec	PM	Set stack pointer after EXEC call
sys_exit	PM	A process has exited.
sys_nice	PM	Set scheduling priority.
sys_trace	PM	Carry out on operation of PTRACE call.
sys_kill	PM, FS, TTY	Send signal to a process after KILL call. (notify)
sys_getksig	PM	PM is checking for pending signals.
sys_endksig	PM	PM has finished processing signals.
sys_sigsend	PM	Send a signal to a process.
sys_sigreturn	PM	Cleanup after completion of a signal.
sys_newmap	PM	Set up a process memory map.
sys_memset	PM	Write char to memory area.
sys_times	PM	Get uptime and process times.
sys_setalarm	PM, FS, Drivers	Schedule synchronous alarm.
sys_abort	PM, TTY	Panic: MINIX is unable to continue

Kernel calls

Message type	From	Meaning
sys_privctl	RS	Set or change privileges.
sys_irqctl	Drivers	Enable, disable or configure interrupt.
sys_devio	Drivers	Read from or write to I/O port.
sys_sdevio	Drivers	Read or write string from/to I/O port.
sys_vdevio	Drivers	Carry out vector of I/O requests.
sys_int86	Drivers	Do a real-mode BIOS call
sys_segctl	Drivers	Add segment and get selector (far data access).
sys_umap	Drivers	Convert virtual address to physical addr.
sys_vircopy	FS, Drivers	Copy using pure virt. addressing.
sys_physcopy	Drivers	Copy using physical addressing.
sys_virvcopy	Any	Vector of VCOPY requests.
sys_physvcopy	Any	Vector of PHYSCOPY requests.
sys_getinfo	Any	Requests system information.

```

1 PUBLIC void sys_task()
2 {
3 /* Main entry point of sys_task. Get the message and dispatch on type. */
4 ...
5
6 initialize();
7
8 while (TRUE) {
9     /* Get work. Block and wait until a request message arrives. */
10    receive(ANY, &m);
11    call_nr = (unsigned) m.m_type - KERNEL_CALL;
12    caller_ptr = proc_addr(m.m_source);
13
14    /* See if the caller made a valid request and try to handle it. */
15    if (! (priv(caller_ptr)->s_call.mask & (1<<call_nr))) {
16        kprintf("SYSTEM: _request_%d_from_%d_denied.\n", call_nr, m.m_source);
17        result = ECALLDENIED;      /* illegal message type */
18    } else if (call_nr >= NR_SYS_CALLS) { /* check call number */
19        kprintf("SYSTEM: _illegal_request_%d_from_%d.\n", call_nr, m.m_source);
20        result = EBADREQUEST;    /* illegal message type */
21    }
22    else {
23        result = (*call_vec[call_nr])(&m); /* handle the kernel call */
24    }
25
26    /* Send a reply, unless inhibited by a handler function. Use the kernel
27     * function lock_send() to prevent a system call trap. The destination
28     * is known to be blocked waiting for a message.
29    */
30    if (result != EDONTREPLY) {
31        m.m_type = result; /* report status of call */
32        if (OK != (s=lock_send(m.m_source, &m))) {
33            kprintf("SYSTEM, _reply_to_%d_failed : %d\n", m.m_source, s);
34        }
35    }
36 }

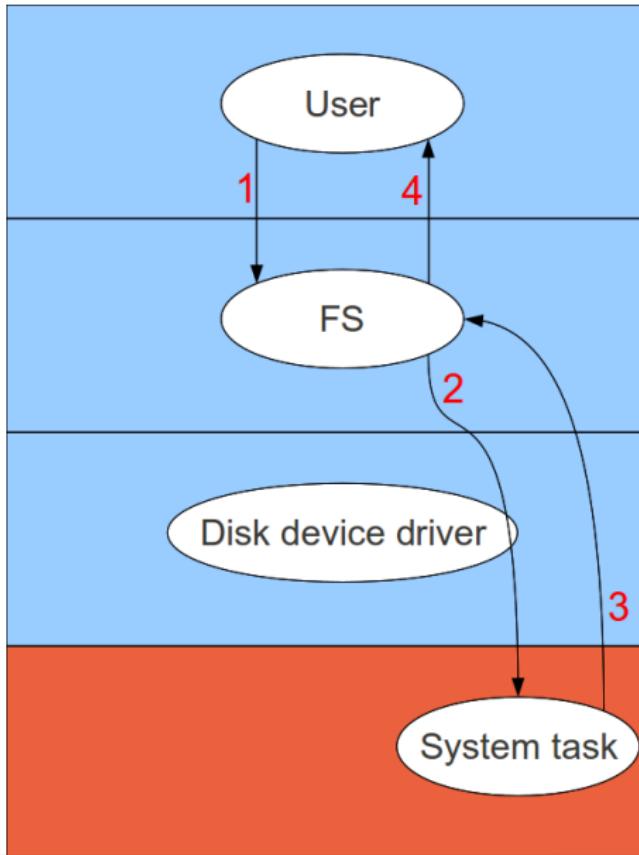
```

```
1 PRIVATE void initialize(void)
2 {
3     register struct priv *sp;
4     int i;
5
6     /* Initialize IRQ handler hooks. Mark all hooks available. */
7     for (i=0; i<NR_IRQ_HOOKS; i++) {
8         irq_hooks[i].proc_nr = NONE;
9     }
10
11    /* Initialize all alarm timers for all processes. */
12    for (sp=BEG_PRIV_ADDR; sp < END_PRIV_ADDR; sp++) {
13        tmr_inittimer(&(sp->s.alarm_timer));
14    }
15
16    /* Initialize the call vector to a safe default handler. */
17    for (i=0; i<NR_SYS_CALLS; i++) {
18        call_vec[i] = do_unused;
19    }
20
21    /* Process management. */
22    map(SYS_FORK, do_fork);      /* a process forked a new process */
23    map(SYS_EXEC, do_exec);     /* update process after execute */
24    /.../
25 }
```

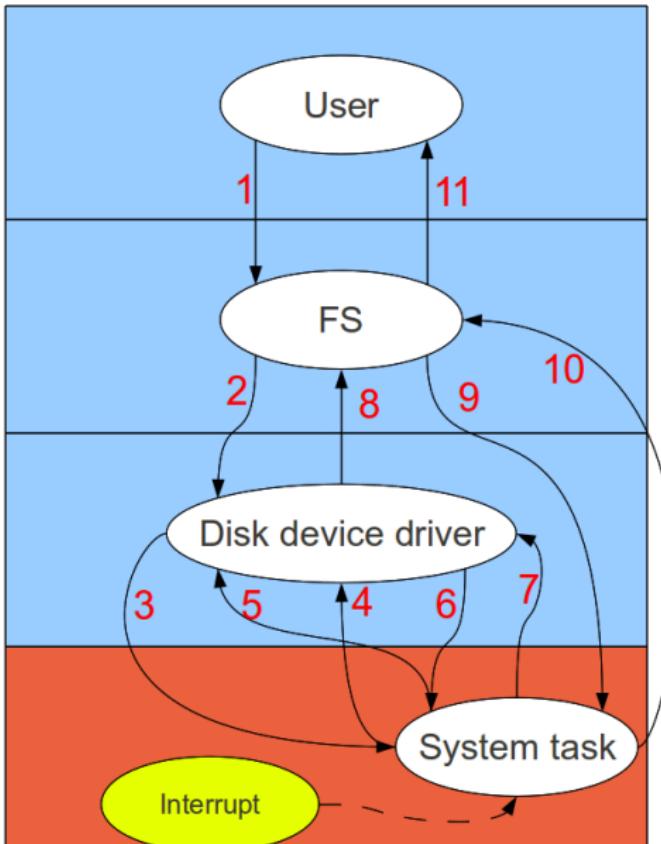
```
1 PUBLIC int do_setalarm(m_ptr)
2 message *m_ptr; /* pointer to request message */
3 {
4 /* A process requests a synchronous alarm, or wants to cancel its alarm. */
5 /* ... */
6 /* Extract shared parameters from the request message. */
7 exp_time = m_ptr->ALRM_EXP_TIME; /* alarm's expiration time */
8 use_abs_time = m_ptr->ALRM_ABS_TIME; /* flag for absolute time */
9 proc_nr = m_ptr->m_source; /* process to interrupt later */
10 rp = proc_addr(proc_nr);
11 if (! (priv(rp)->s_flags & SYS_PROC)) return (EPERM);
12
13 /* Get the timer structure and set the parameters for this alarm. */
14 tp = &(priv(rp)->s_alarm_timer);
15 tmr_arg(tp)->ta_int = proc_nr;
16 tp->tmr_func = cause_alarm;
17
18 /* Return the ticks left on the previous alarm. */
19 uptime = get_uptime();
20 if ((tp->tmr_exp_time != TMR_NEVER) && (uptime < tp->tmr_exp_time) ) {
21     m_ptr->ALRM_TIME_LEFT = (tp->tmr_exp_time - uptime);
22 } else {
23     m_ptr->ALRM_TIME_LEFT = 0;
24 }
25
26 /* Finally, (re)set the timer depending on the expiration time. */
27 if (exp_time == 0) {
28     reset_timer(tp);
29 } else {
30     tp->tmr_exp_time = (use_abs_time) ? exp_time : exp_time + get_uptime();
31     set_timer(tp, tp->tmr_exp_time, tp->tmr_func);
32 }
33 return (OK);
34 }
```

```
1 PRIVATE void cause_alarm(tp)
2 timer_t *tp;
3 {
4 /* Routine called if a timer goes off and the process requested
5 * a synchronous alarm.
6 * The process number is stored in timer argument 'ta_int'.
7 * Notify that process with a notification message from CLOCK.
8 */
9 int proc_nr = tmr_arg(tp)->ta_int;      /* get process number */
10 lock_notify(CLOCK, proc_nr);           /* notify process */
11 }
```

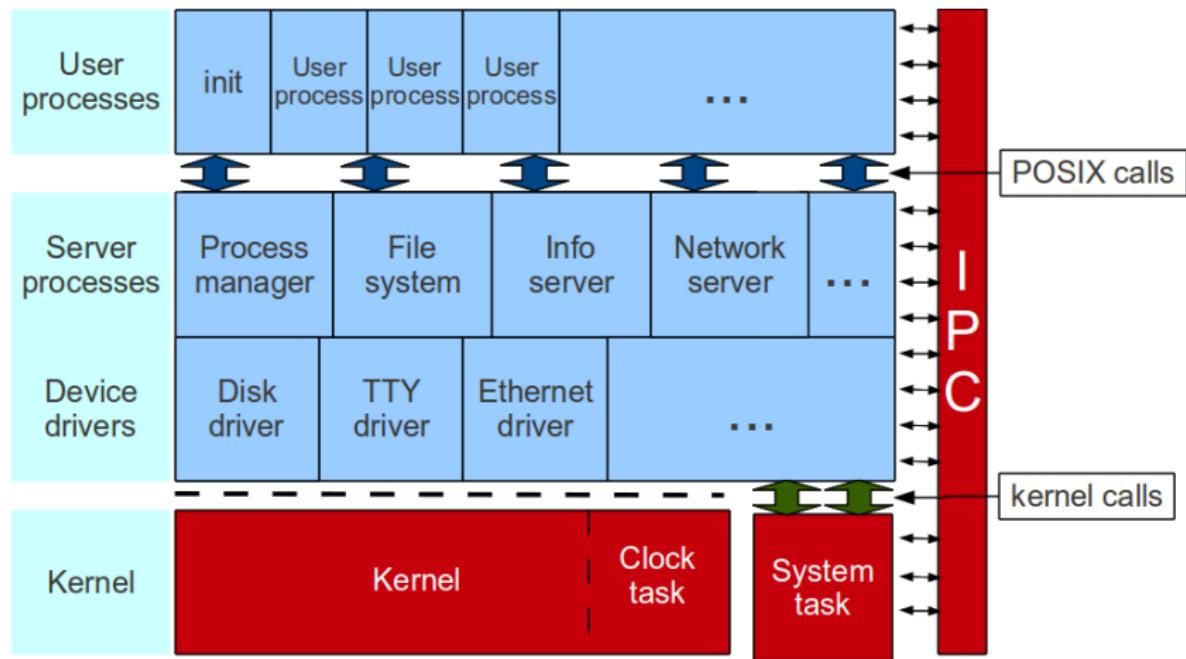
read()



read()



Minix kernel



Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

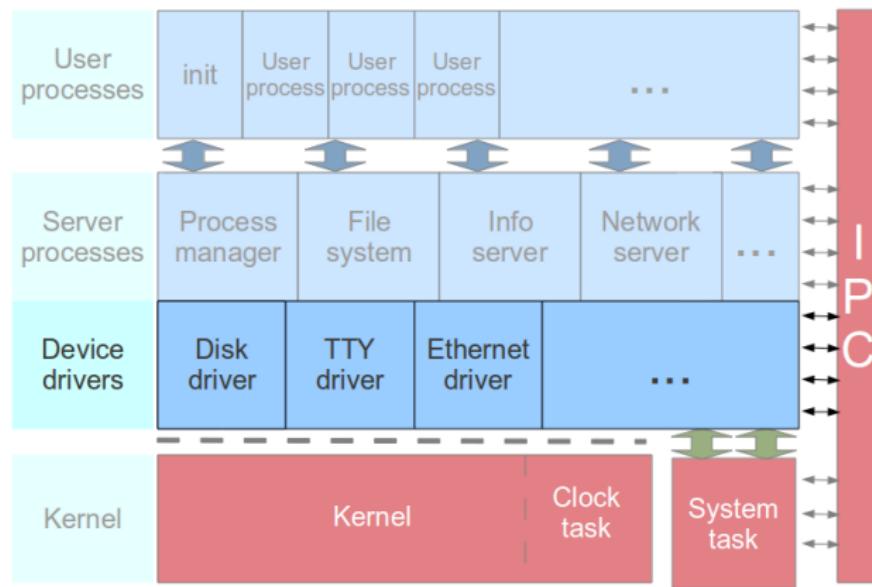
4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

5 Scheduling

- Batch systems

Device drivers



Urządzenia blokowe

Swobodny(random) dostęp do danych. Czas dostępu do każdego fragmentu tego samego rzędu. Dane zorganizowane w bloki (np. dysk, CD/DVD-ROM)

Urządzenia znakowe

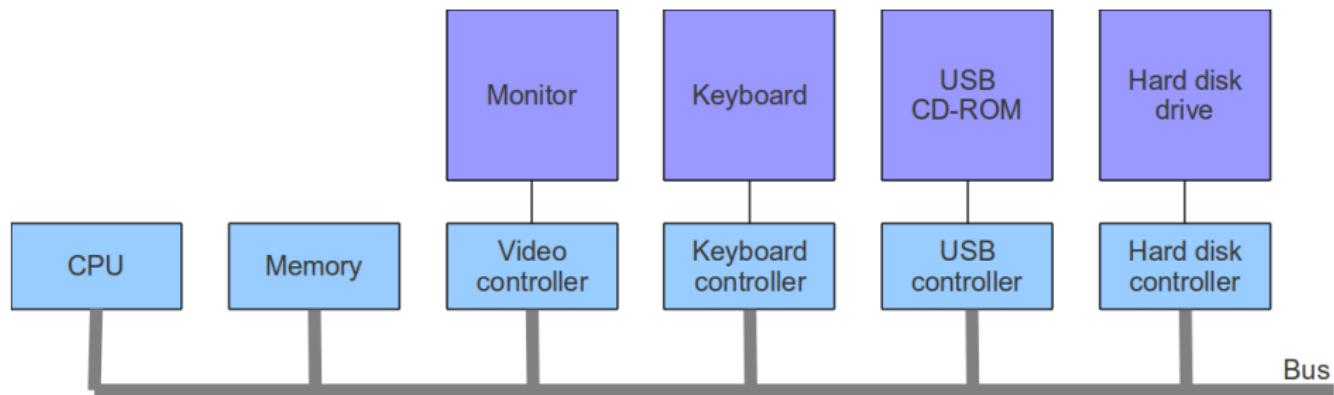
Operuje na ciągach znaków. Sekwencyjny dostęp (zapis/odczyt). (np. klawiatura, mysz, modem).

Prędkości urządzeń

Device	Data rate
Klawiatura	10bytes/s
Mysz	100bytes/s
56K modem	7KB/s
52xCD-ROM	8MB/s
USB 2.0	60MB/s
GigabitEthernet	125MB/s
Serial ATA disk	200MB/s
PCI bus	528 MB/s

Dużo więcej prędkości - tutaj ([wikipedia](#)).

Device controller



I/O port

W architekturze x86 procesor ma do dyspozycji 2^{16} 8-bitowych portów.
Porty mogą być grupowane dla zwiększenia transferu.

Instrukcje:

- IN REG, PORT
- OUT PORT, REG
- instrukcje dla ciągów: INS, OUTS

Oddzielna przestrzeń adresowa.

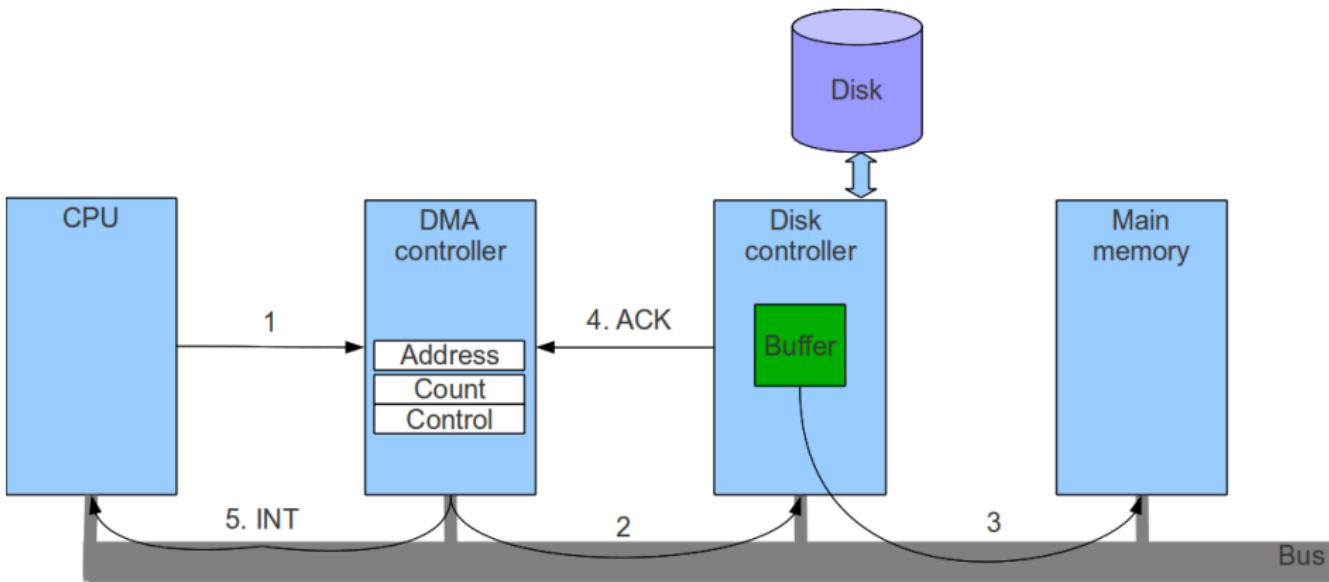
memory mapped I/O

Rejestry urządzeń mogą być zmapowane do przestrzeni adresowej pamięci procesora. Dostęp jak do pamięci (np. MOV).

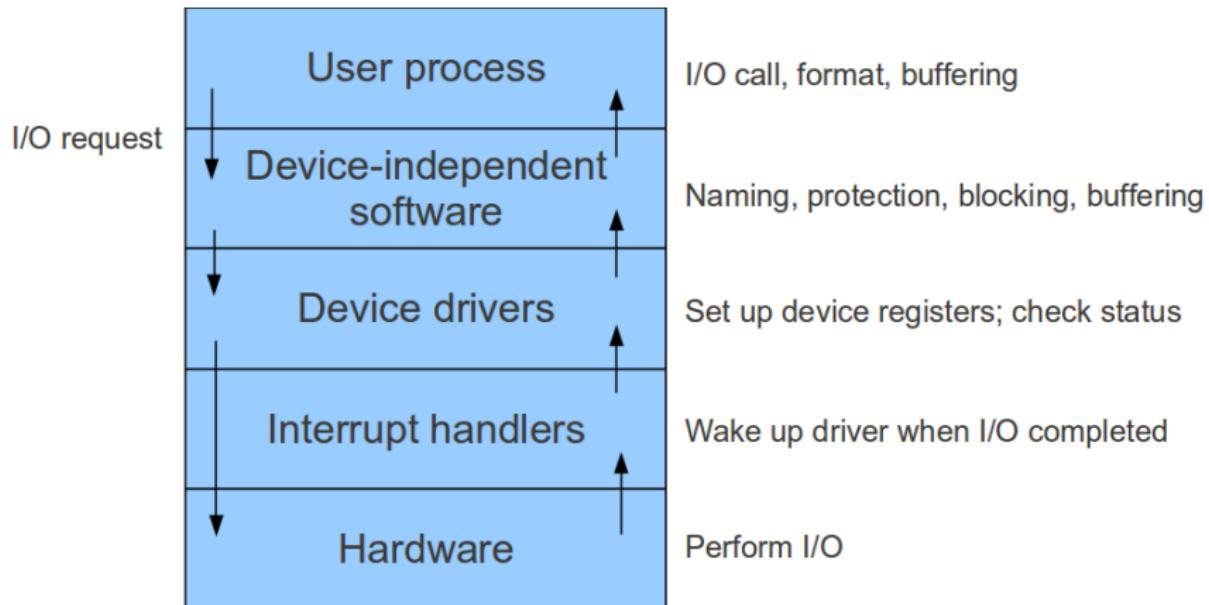
Asynchroniczna komunikacja

- status bits polling
- przerwania

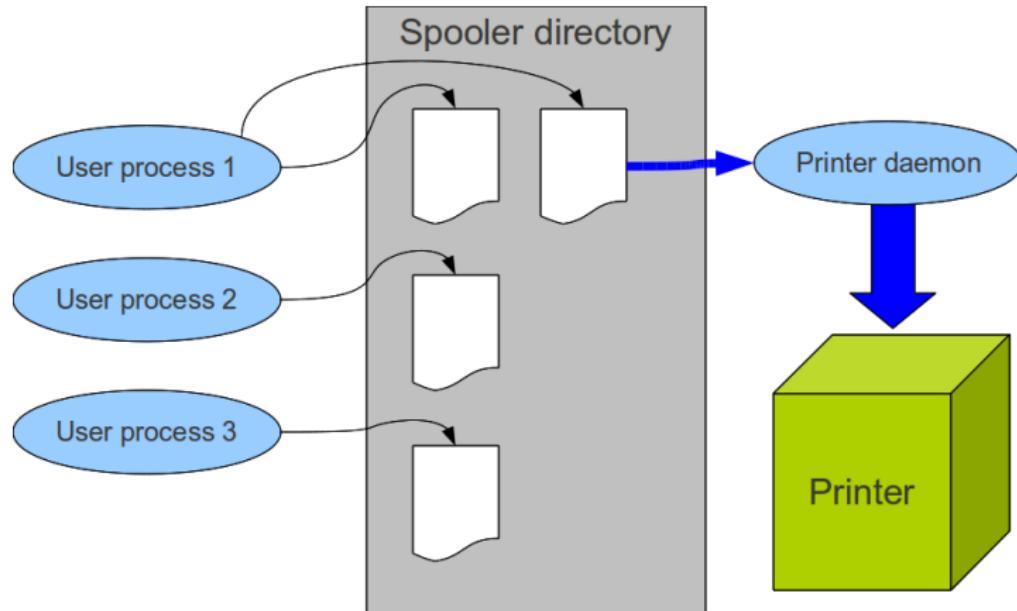
DMA



Device independent software



Spooling



Rodzaje urządzeń/zasobów

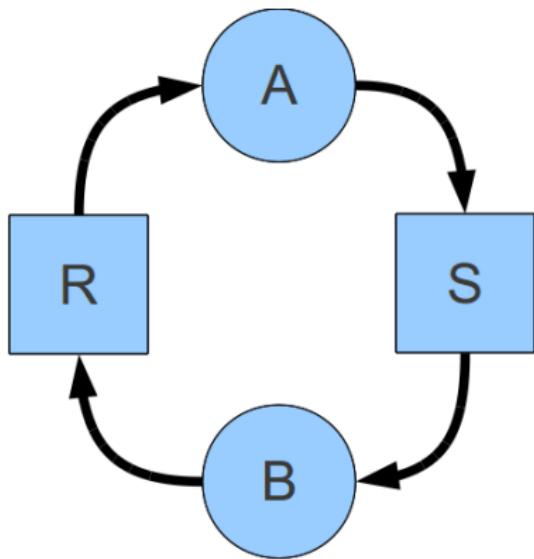
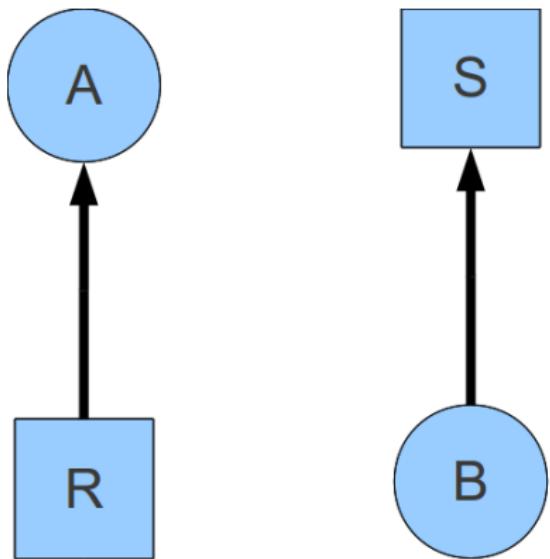
- unikalne (np. drukarka, wiersz w bazie danych)
 - fungible (zastępowalne?) (np. CPU)
-
- wywłaszcjalne (np. pamięć)
 - niewywłaszcjalne (np. nagrywarka CD)

A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.

Warunki istnienia deadlock'ów (przykładowe)

- ① Mutual exclusion condition - każdy zasób jest albo dostępny albo przypisany jakiemuś procesowi.
- ② Hold and wait condition - proces będący w posiadaniu jakiegoś zasobu może zarządzać następnych.
- ③ No preemption condition - zasoby nie mogą być odbierane.
- ④ Circular wait condition - może pojawić się cykl procesów z których każdy czeka na zasób będący w posiadaniu następnego procesu w cyklu.

Deadlock - modelowanie



Strategie radzenia sobie z deadlock'ami

- ① Ignoruj problem.
- ② Wykrywaj i usuwaj.
- ③ Ustal zasady korzystania z zasobów, które wykluczają możliwość zaistnienia deadlock'u (structural prevention).
- ④ Starannie przydzielaj zasoby tak aby nie dopuścić do deadlock'u.

Wykrywanie i usuwanie deadlock'ów

- Sprawdzaj co jakiś czas graf/sieć przydziału zasobów.
- Jeśli w sieci jest cykl zabij jeden z procesów w cyklu (potem ew. restart).

Zastosowanie: batch systems.

Uwaga na zmiany w systemie spowodowane przez zabity proces.

Mutual exclusion condition

Każdy zasób jest albo dostępny albo przypisany jakiemuś procesowi.

SPOOLING

Problemy:

- nie wszystkie zasoby mogą być w ten sposób obsłużone
- przerzucenie ciężaru na inne zasoby (dysk)

Structural prevention

Hold and wait condition

Proces będący w posiadaniu jakiegoś zasobu może zarządzać następujących.

- ① Proces może zarządzać zasobów tylko przed rozpoczęciem działania.
- ② Proces może zarządzać zbioru zasobów (wszystkie na raz) ale przedtem musi zwolnić wszystkie posiadane.

Problemy:

- trudno ocenić jakie zasoby będą potrzebne przed rozpoczęciem działania,
- niektórych zasobów nie wolno zwolnić przed zakończeniem pracy (niewywłaszcjalne),
- słabe wykorzystanie zasobów.

Z tego samego powodu:

No preemption condition

Zasoby nie mogą być odbierane.

Circular wait condition

Mожет zaistnieć cykl procesów z których każdy czeka na zasób będący w posiadaniu następnego procesu w cyklu.

- ① Każdy proces może używać co najwyżej jednego zasobu na raz.
- ② Zasoby są uporządkowane liniowo. Każdy proces może zarządzać tylko zasobów "większych" od największego posiadanejgo zasobu.

Problemy:

- jeden zasób na raz to za mało dla procesu (skanowanie + zapis na CD),
- trudno dobrać odpowiedni porządek,
- trudno uwzględnić zasoby wymienialne,
- niewydajne wykorzystanie zasobów,
- duży rozmiar porządku (wiersze w bazie danych).

Unikanie deadlock'ów

Przydzielaj zasoby tak, aby deadlock nie powstał.

Algorytm bankiera - Dijkstra 1965

Cena:

- musimy wiedzieć ile maksymalnie danego zasobu będzie potrzebował proces,
- zakładamy że każdy proces się kiedyś skończy (lub zwolni wszystkie swoje zasoby)

	Has	Max
A	0	6
B	0	5
C	0	4
D	0	7

Free: 10

	Has	Max
A	1	6
B	1	5
C	2	4
D	4	7

Free: 2

	Has	Max
A	1	6
B	2	5
C	2	4
D	4	7

Free: 1

Algorytm bankiera - wiele zasobów

przypisane (RA):

Pr	TD	Plo.	Pr.	CD
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1

potrzebne (RN):

Pr	TD	Plo.	Pr.	CD
A	1	1	0	0
B	0	1	1	2
C	0	0	1	0
D	2	1	1	0

Zasoby dostępne: $R = (1, 0, 2, 0)$

- ① znajdź proces x dla którego $R \leq RA(x)$ (jeśli nie ma to deadlock)
- ② $R+ = RA(x)$
- ③ usuń wiersz x z macierzy RN, RA
- ④ jeśli macierze niepuste goto 1
- ⑤ sukces - brak deadlocku

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

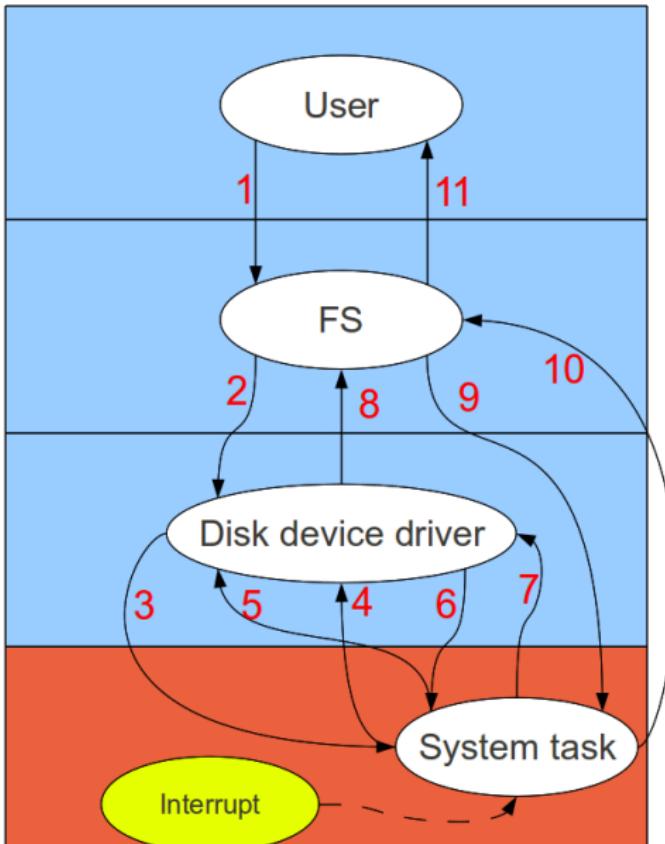
4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

5 Scheduling

- Batch systems

read()



Format wiadomości FS - DD

Request

Filed	Type	Meaning
m.m_type	int	Operation requested
m.DEVICE	int	Minor device to use
m.PROC_NT	int	Process requesting I/O
m.COUNT	int	Byte count or ioctl code
m.POSITION	long	Position on device
m.ADDRESS	char *	Address within requesting process

Answer

Filed	Type	Meaning
m.m_type	int	Always DRIVER_REPLY
m.REP_PROC_NR	int	Same as PROC_NR in request
m.REP_STATUS	int	Bytes transferred or error number

Operacje

- OPEN
- CLOSE
- READ
- WRITE
- IOCTL
- SCATTERED_IO
- ...

```

1 PUBLIC void driver_task(struct driver *dp){ /*dp - device dependent entry points.*/
2 init_buffer(); // DMA
3 while (TRUE) {
4 if(receive(ANY, &mess) != OK) continue;
5 device_caller = mess.m_source;
6 proc_nr = mess.PROC_NR;
7
8 switch(mess.m_type) {
9 case DEV_OPEN: r = (*dp->dr_open)(dp, &mess); break;
10 case DEV_CLOSE: r = (*dp->dr_close)(dp, &mess); break;
11 case DEV_IOCTL: r = (*dp->dr_ioctl)(dp, &mess); break;
12 case CANCEL: r = (*dp->dr_cancel)(dp, &mess); break;
13 case DEV_SELECT: r = (*dp->dr_select)(dp, &mess); break;
14 case DEV_READ:
15 case DEV_WRITE: r = do_rdwt(dp, &mess); break;
16 case DEV_GATHER:
17 case DEV_SCATTER: r = do_vrdwt(dp, &mess); break;
18 case HARD_INT: /* leftover interrupt or expired timer. */
19     if(dp->dr_hw_int) {
20         (*dp->dr_hw_int)(dp, &mess);
21     }
22     continue;
23 case SYS_SIG: (*dp->dr_signal)(dp, &mess);
24     continue; /* don't reply */
25 case SYN_ALARM: (*dp->dr_alarm)(dp, &mess);
26     continue; /* don't reply */
27 default: /.../
28 }
29 (*dp->dr_cleanup)();
30 if (r != EDONTREPLY) {
31     mess.m_type = TASK.REPLY;
32     mess.REP_PROC_NR = proc_nr;
33     /* Status is # of bytes transferred or error code. */
34     mess.REP_STATUS = r;
35     send(device_caller, &mess);
36 }
37

```



```
1 /*  
2 *      at_wincchester_task  
3 */  
4 PUBLIC int main()  
5 {  
6 /* Set special disk parameters  
7 * then call the generic main loop.  
8 */  
9     init_params();  
10    driver_task(&w_dtab);  
11    return(OK);  
12 }
```

DMA buffer alignment (64K)

```
1 PRIVATE void init_buffer()
2 {
3 /* Select a buffer that can safely be used for DMA transfers. It may
4 * be used to read partition tables and such. Its absolute address is
5 * 'tmp_phys', the normal address is 'tmp_buf'.
6 */
7
8 #if (CHIP == INTEL)
9     unsigned left;
10
11    tmp_buf = buffer;
12    sys_uimap(SELF, D, (vir_bytes)buffer, (phys_bytes)sizeof(buffer), &tr
13
14    if ((left = dma_bytes_left(tmp_phys)) < DMA_BUF_SIZE) {
15        /* First half of buffer crosses a 64K boundary, can't DMA into that
16        tmp_buf += left;
17        tmp_phys += left;
18    }
19#endif /* CHIP == INTEL */
20 }
```

```

1 PRIVATE int do_rdwrt(dp, mp)
2 struct driver *dp; /* device dependent entry points */
3 message *mp; /* pointer to read or write message */
4 {
5 /* Carry out a single read or write request. */
6 iovec_t iovec1;
7 int r, opcode;
8 phys_bytes phys_addr;
9
10 /* Disk address? Address and length of the user buffer? */
11 if (mp->COUNT < 0) return(EINVAL);
12
13 /* Check the user buffer. */
14 sys_lumap(mp->PROC_NR, D, (vir_bytes) mp->ADDRESS, mp->COUNT, &phys_addr);
15 if (phys_addr == 0) return(EFAULT);
16
17 /* Prepare for I/O. */
18 if ((*dp->dr_prepare)(mp->DEVICE) == NIL_DEV) return(ENXIO);
19
20 /* Create a one element scatter/gather vector for the buffer. */
21 opcode = mp->m_type == DEV_READ ? DEV_GATHER : DEV_SCATTER;
22 iovec1.iov_addr = (vir_bytes) mp->ADDRESS;
23 iovec1.iov_size = mp->COUNT;
24
25 /* Transfer bytes from/to the device. */
26 r = (*dp->dr_transfer)(mp->PROC_NR, opcode, mp->POSITION, &iovec1, 1);
27
28 /* Return the number of bytes transferred or an error code. */
29 return(r == OK ? (mp->COUNT - iovec1.iov_size) : r);
30 }

```

```

1 PRIVATE int do_vrdwt(dp, mp)
2 struct driver *dp; /* device dependent entry points */
3 message *mp; /* pointer to read or write message */
4 {
5     static iovec_t iovec[NR_IOREQS];
6     iovec_t *iov;
7     phys_bytes iovec_size;
8     unsigned nr_req;
9     int r;
10
11    nr_req = mp->COUNT; /* Length of I/O vector */
12
13    if (mp->m_source < 0) {
14        /* Called by a task, no need to copy vector. */
15        iov = (iovec_t *) mp->ADDRESS;
16    } else {
17        /* Copy the vector from the caller to kernel space. */
18        if (nr_req > NR_IOREQS) nr_req = NR_IOREQS;
19        iovec_size = (phys_bytes) (nr_req * sizeof(iovec[0]));
20
21        if (OK != sys_datacopy(mp->m_source, (vir_bytes) mp->ADDRESS,
22                               SELF, (vir_bytes) iovec, iovec_size))
23            panic((*dp->dr_name)(), "bad_I/O_vector_by", mp->m_source);
24        iov = iovec;
25    }
26    /* Prepare for I/O. */
27    if ((*dp->dr_prepare)(mp->DEVICE) == NIL_DEV) return(ENXIO);
28    /* Transfer bytes from/to the device. */
29    r = (*dp->dr_transfer)(mp->PROC_NR, mp->m_type, mp->POSITION, iov, nr_req);
30    /* Copy the I/O vector back to the caller. */
31    if (mp->m_source >= 0) {
32        sys_datacopy(SELF, (vir_bytes) iovec,
33                     mp->m_source, (vir_bytes) mp->ADDRESS, iovec_size);
34    }
35    return(r);
36 }

```

Urządzenia

- 0:/dev/ram
- 1:/dev/mem
- 2:/dev/kmem
- 3:/dev/null
- 4:/dev/boot
- 5:/dev/zero

```
PRIVATE struct driver m_dtab = {  
    m_name, /* current device's name */  
    m_do_open, /* open or mount */  
    do_nop, /* nothing on a close */  
    m_ioctl, /* specify ram disk geometry */  
    m_prepare, /* prepare for I/O on a given minor devi  
    m_transfer, /* do the I/O */  
    nop_cleanup, /* no need to clean up */  
    m_geometry, /* memory device "geometry" */  
    nop_signal, /* system signals */  
    nop_alarm,  
    nop_cancel,  
    nop_select,  
    NULL, /* other */  
    NULL /* hw_int */  
};
```

```

1 PRIVATE void m_init()
2 {
3     int i, s;
4
5     if (OK != (s=sys_getkinfo(&kinfo))) {
6         panic("MEM", "Couldn't_get_kernel_information.", s);
7     }
8
9     /* Install remote segment for /dev/kmem memory. */
10    m_geom[KMEM.DEV].dv_base = cvul64(kinfo.kmem_base);
11    m_geom[KMEM.DEV].dv_size = cvul64(kinfo.kmem_size);
12    if (OK != (s=sys_segctl(&m_seg[KMEM.DEV], (u16_t *) &s, (vir_bytes *) &s,
13        kinfo.kmem_base, kinfo.kmem_size))) {
14        panic("MEM", "Couldn't_install_remote_segment.", s);
15    }
16    ...
17    /* Initialize /dev/zero. Simply write zeros into the buffer. */
18    for (i=0; i<ZERO_BUF_SIZE; i++) dev_zero[i] = '\0';
19
20    /* Set up memory ranges for /dev/mem. */
21    if (OK != (s=sys_getmachine(&machine))) {
22        panic("MEM", "Couldn't_get_machine_information.", s);
23    }
24    if (! machine.protected) {
25        m_geom[MEM.DEV].dv_size = cvul64(0x100000); /* 1M for 8086 systems */
26    } else {
27 #if _WORD_SIZE == 2
28        m_geom[MEM.DEV].dv_size = cvul64(0x1000000); /* 16M for 286 systems */
29 #else
30        m_geom[MEM.DEV].dv_size = cvul64(0xFFFFFFFF); /* 4G-1 for 386 systems */
31 #endif
32    }
33 }

```

```
1 PRIVATE int m_transfer(int proc_nr, int opcode, off_t position, iovec_t *iov, unsigned nr_req){  
2     /* ... */  
3     /* Get minor device number and check for /dev/null. */  
4     dv = &m_geom[m_device];  
5     dv_size = cv64ul(dv->dv_size);  
6  
7     while (nr_req > 0) {  
8  
9         /* How much to transfer and where to / from. */  
10        count = iov->iov_size;  
11        user_vir = iov->iov_addr;  
12  
13        switch (m_device) {  
14  
15            /* No copying; ignore request. */  
16            case NULL_DEV:  
17                if (opcode == DEV_GATHER) return(OK); /* always at EOF */  
18                break;  
19  
20            /* Virtual copying. For RAM disk, kernel memory and boot device. */  
21            case RAM_DEV:  
22            case KMEM_DEV:  
23            case BOOT_DEV:  
24                if (position >= dv_size) return(OK); /* check for EOF */  
25                if (position + count > dv_size) count = dv_size - position;  
26                seg = m_seg[m_device];  
27  
28                if (opcode == DEV_GATHER) { /* copy actual data */  
29                    sys_vircopy(SELF, seg, position, proc_nr, D, user_vir, count);  
30                } else {  
31                    sys_vircopy(proc_nr, D, user_vir, SELF, seg, position, count);  
32                }  
33                break;  
34    ...
```

```
1  /* Physical copying. Only used to access entire memory. */
2  case MEM.DEV:
3      if (position >= dv_size) return(OK); /* check for EOF */
4      if (position + count > dv_size) count = dv_size - position;
5      mem_phys = cv64ul(dv->dv_base) + position;
6
7      if (opcode == DEV.GATHER) { /* copy data */
8          sys_physcopy(NONE, PHYS SEG, mem_phys,
9                      proc_nr, D, user_vir, count);
10     } else {
11         sys_physcopy(proc_nr, D, user_vir,
12                     NONE, PHYS SEG, mem_phys, count);
13     }
14     break;
15 ...
```

```
1  /* Null byte stream generator. */
2  case ZERO_DEV:
3      if (opcode == DEV_GATHER) {
4          left = count;
5          while (left > 0) {
6              chunk = (left > ZERO_BUF_SIZE) ? ZERO_BUF_SIZE : left;
7              if (OK != (s=sys_vircopy(SELF, D, (vir_bytes) dev_zero,
8                           proc_nr, D, user_vir, chunk)))
9                  report("MEM", "sys_vircopy failed", s);
10             left -= chunk;
11             user_vir += chunk;
12         }
13     }
14     break;
15
16 /* Unknown (illegal) minor device. */
17 default:
18     return(EINVAL);
19 }
20
21 /* Book the number of bytes transferred. */
22 position += count;
23 iov->iov_addr += count;
24 if ((iov->iov_size -= count) == 0) { iov++; nr_req--; }
25
26 }
27 return(OK);
28 }
```

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

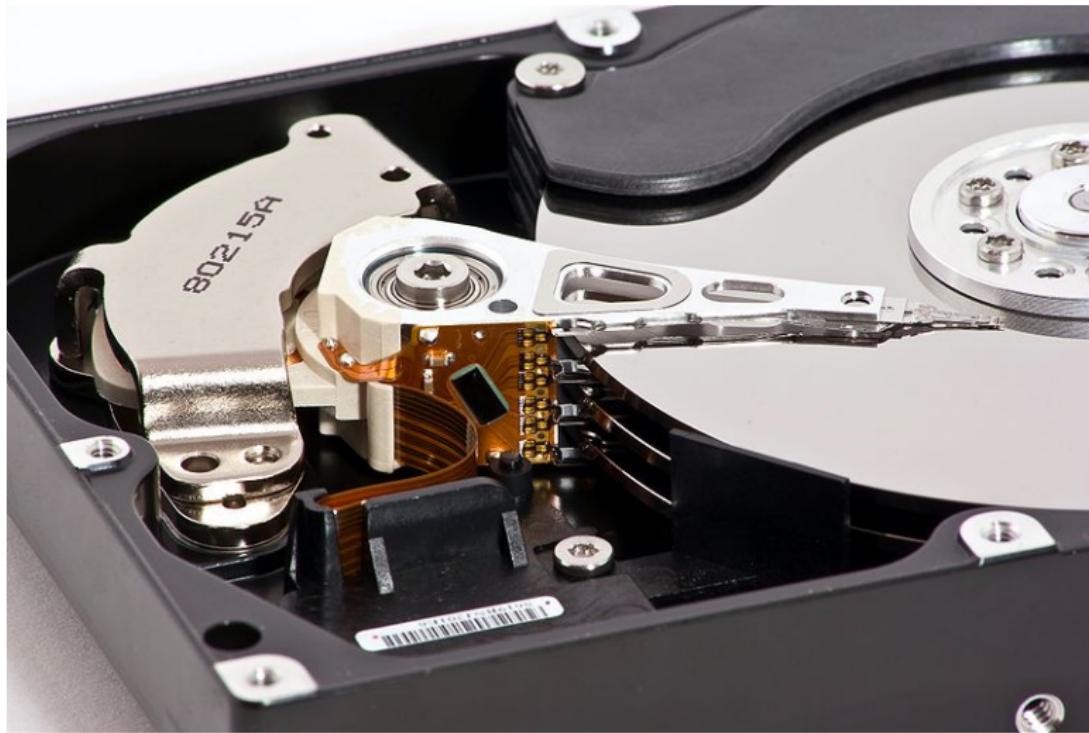
5 Scheduling

- Batch systems

Hard disk - budowa



Hard disk - budowa



Geometria dysku (typowe wartości)

- cylinder (16383)
- head (16)
- track
- sector (63 rozmiar 512)

W sumie ~8GB.

Logical block addressing

- LBA28 - limit 128 GB (Minix)
- LBA48 - limit 128 PB

Etapy dostępu

- *seek time*
- *rotational delay*
- *data transfer*

Disk arm scheduling

- First-Come First-Served
- Shortest Seek First
- Elevator alg.

Obsługa błędów.

IBM-AT winchester controller.

```
1 /* Entry points to this driver. */
2 PRIVATE struct driver w_dtab = {
3     w_name,      /* current device's name */
4     w_do_open,    /* open or mount request, initialize device */
5     w_do_close,   /* release device */
6     do_ioctl,     /* get or set a partition's geometry */
7     w_prepare,    /* prepare for I/O on a given minor device */
8     w_transfer,   /* do the I/O */
9     nop_cleanup,  /* nothing to clean up */
10    w_geometry,   /* tell the geometry of the disk */
11    nop_signal,   /* no cleanup needed on shutdown */
12    nop_alarm,    /* ignore leftover alarms */
13    nop_cancel,   /* ignore CANCELs */
14    nop_select,   /* ignore selects */
15    w_other,      /* catch-all for unrecognized commands and ioctls */
16    w_hw_int      /* leftover hardware interrupts */
17 };
```

```
1 PRIVATE int w_transfer(int proc_nr, int opcode, off_t position, iovec_t *iov, unsigned nr_req)
2     struct wini *wn = w.wn;
3     iovec_t *iop, *iov_end = iov + nr_req;
4     int r, s, errors;
5     unsigned long dv_size = cv64ul(w_dv->dv_size);
6     unsigned cylinder, head, sector, nbytes;
7     ...
8     while (nr_req > 0) {
9         /* How many bytes to transfer? */
10        block = div64u(add64ul(w_dv->dv_base, position), SECTOR_SIZE);
11        ...
12        if (!(wn->state & INITIALIZED) && w_specify() != OK) return(EIO);
13        /* Tell the controller to transfer nbytes bytes. */
14        r = do_transfer(wn, wn->precomp, ((nbytes >> SECTOR_SHIFT) & BYTE),
15                      block, opcode);
16
17        while (r == OK && nbytes > 0) {
18            /* For each sector, wait for an interrupt and fetch the data
19             * (read), or supply data to the controller and wait for an
20             * interrupt (write).
21            */
22
23            if (opcode == DEV_GATHER) {
24                /* First an interrupt, then data. */
25                if ((r = at_intr_wait()) != OK) {
26                    /* An error, send data to the bit bucket. */
27                    if (w.wn->w_status & STATUS_DRQ) {
28                        if ((s=sys_insw(wn->base_cmd + REG_DATA, SELF, tmp_buf, SECTOR_SIZE)) != OK)
29                            panic(w.name(), "Call_to_sys_insw() failed", s);
30                    }
31                    break;
32                }
33            }
```

```

1  /* Wait for data transfer requested. */
2  if (!w->waitfor(STATUS_DRQ, STATUS_DRQ)) { r = ERR; break; }
3
4  /* Copy bytes to or from the device's buffer. */
5  if (opcode == DEV_GATHER) {
6    if ((s=sys_insw(wn->base_cmd + REG_DATA, proc_nr, (void *) iov->iov_addr, SECTOR_SIZE)) != -1)
7      panic(w->name(), "Call_to_sys_insw()_failed", s);
8    } else {
9      if ((s=sys_outsw(wn->base_cmd + REG_DATA, proc_nr, (void *) iov->iov_addr, SECTOR_SIZE)) != -1)
10        panic(w->name(), "Call_to_sys_insw()_failed", s);
11
12      /* Data sent, wait for an interrupt. */
13      if ((r = at_intr_wait()) != OK) break;
14    }
15
16  /* Book the bytes successfully transferred. */
17  nbytes -= SECTOR_SIZE;
18  position += SECTOR_SIZE;
19  iov->iov_addr += SECTOR_SIZE;
20  if ((iov->iov_size -= SECTOR_SIZE) == 0) { iov++; nr_req--; }
21
22}
23
24/* Any errors? */
25if (r != OK) {
26  /* Don't retry if sector marked bad or too many errors. */
27  if (r == ERR_BAD_SECTOR || ++errors == max_errors) {
28    w_command = CMD_IDLE;
29    return(EIO);
30  }
31}
32
33w_command = CMD_IDLE;
34return(OK);
35
36}

```



Reg.	Read	Write
0	Data	Data
1	Error	Write Precompensation
2	Sector Count	Sector Count
3	Sector Number (0-7)	Sector Number (0-7)
4	Cylinder Low (8-15)	Cylinder Low (8-15)
5	Cylinder High (16-23)	Cylinder High (16-23)
6	Select Drive/Head (24-27)	Select Drive/Head (24-27)
7	Status	Command

7	6	5	4	3	2	1	0
1	LBA	1	D	HS3	HS2	HS1	HS0

D - master/slave

```
1 PRIVATE int do_transfer( struct wini *wn, unsigned int precomp, unsigned int count,
2   unsigned int sector, unsigned int opcode)
3 {
4     struct command cmd;
5     unsigned secspcyl = wn->pheads * wn->psectors;
6
7     cmd.precomp = precomp;
8     cmd.count = count;
9     cmd.command = opcode == DEV_SCATTER ? CMD_WRITE : CMD_READ;
10    /*
11    if (w_lba48 && wn->lba48) {
12    } else */
13    if (wn->ldhpref & LDH_LBA) {
14        cmd.sector = (sector >> 0) & 0xFF;
15        cmd.cyl_lo = (sector >> 8) & 0xFF;
16        cmd.cyl_hi = (sector >> 16) & 0xFF;
17        cmd.ldh = wn->ldhpref | ((sector >> 24) & 0xF);
18    } else {
19        int cylinder, head, sec;
20        cylinder = sector / secspcyl;
21        head = (sector % secspcyl) / wn->psectors;
22        sec = sector % wn->psectors;
23        cmd.sector = sec + 1;
24        cmd.cyl_lo = cylinder & BYTE;
25        cmd.cyl_hi = (cylinder >> 8) & BYTE;
26        cmd.ldh = wn->ldhpref | head;
27    }
28
29    return com_out(&cmd);
30 }
```

```

1 PRIVATE int com_out(cmd)
2 struct command *cmd; /* Command block */
3 {
4     ...
5
6     if (w_wn->state & IGNORING) return ERR;
7     if (!w_waitfor(STATUS_BSY, 0)) {
8         printf("%s:_controller_not_ready\n", w_name());
9         return (ERR);
10    }
11   /* Select drive. */
12   if ((s=sys_outb(base_cmd + REG_LDH, cmd->ldh)) != OK)
13     panic(w_name(), "Couldn't write register_to_select_drive", s);
14
15   if (!w_waitfor(STATUS_BSY, 0)) {
16       printf("%s:_com_out:_drive_not_ready\n", w_name());
17       return (ERR);
18   }
19   sys_setalarm(wakeup_ticks, 0);
20
21   wn->w_status = STATUS_ADMBSY;
22   w_command = cmd->command;
23   pv_set(outbyte[0], base_ctl + REG_CTL, wn->pheads >= 8 ? CTL_EIGHTHEADS : 0);
24   pv_set(outbyte[1], base_cmd + REG_PRECOMP, cmd->precomp);
25   pv_set(outbyte[2], base_cmd + REG_COUNT, cmd->count);
26   pv_set(outbyte[3], base_cmd + REG_SECTOR, cmd->sector);
27   pv_set(outbyte[4], base_cmd + REG_CYL_LO, cmd->cyl_lo);
28   pv_set(outbyte[5], base_cmd + REG_CYL_HI, cmd->cyl_hi);
29   pv_set(outbyte[6], base_cmd + REG_COMMAND, cmd->command);
30   if ((s=sys_voutb(outbyte, 7)) != OK)
31     panic(w_name(), "Couldn't write registers with sys_voutb()", s);
32   return (OK);
33 }

```

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

5 Scheduling

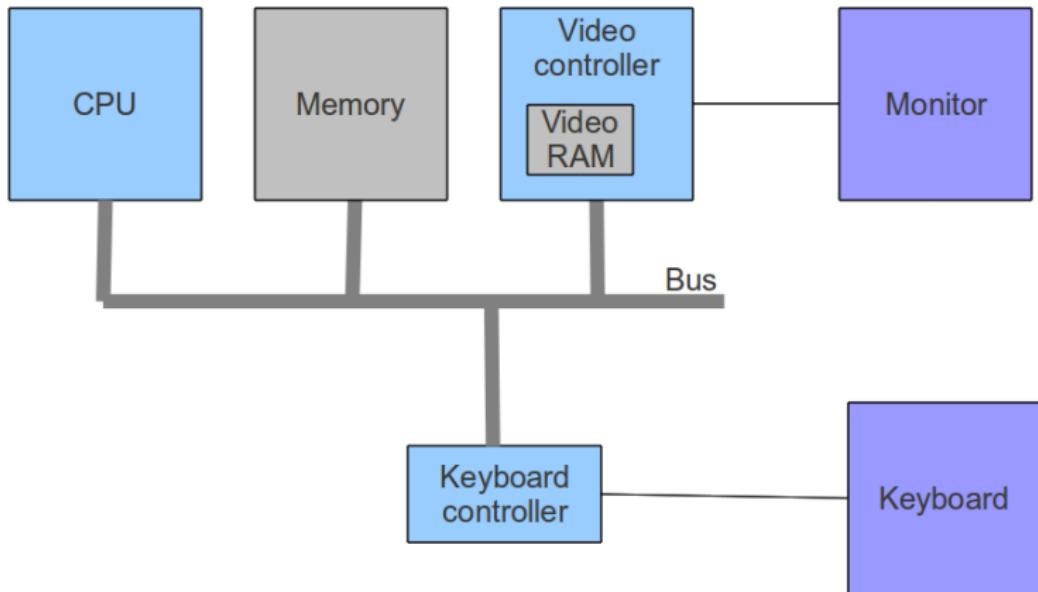
- Batch systems

Terminals

- Memory-mapped interface
- RS-232 interface
- Network interface



Memory-mapped terminal



Input processing

- character-oriented (raw mode / ~noncanonical (POSIX))
- line-orienter (cooked mode / canonical (POSIX))

Klawiatura - urządzenie

- Każde naciśnięcie/zwolnienie klawisza powoduje przerwanie sprzętowe (IRQ 1).
- Kod klawisza (scan code) można odczytać z portu 60h.
- Komunikacja z klawiaturą poprzez port 64h.

Code pages

$\text{scan code} \neq \text{ASCII code}$

Konfiguracja terminala - POSIX

tcgetattr, tcsetattr

```
#include <termios.h>

int tcgetattr(int fildes, struct termios *termios_p);

int tcsetattr(int fildes, int optional_actions,
              const struct termios *termios_p);
```

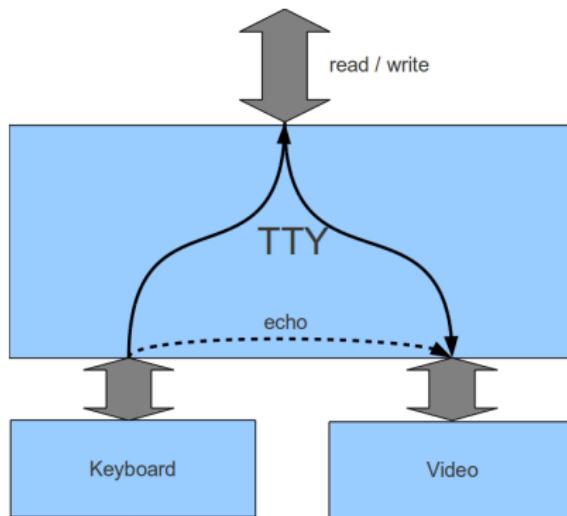
termios

The <termios.h> header shall define the termios structure, which shall include:

tcflag_t	c_iflag	Input modes.
tcflag_t	c_oflag	Output modes.
tcflag_t	c_cflag	Control modes.
tcflag_t	c_lflag	Local modes.
cc_t	c_cc[NCCS]	Control characters.

W MINIX realizowane przez ioctl().

Echo - canonical mode



- linefeed/carriage return
- TAB
- długie linie
- znaki sterujące

Znaki sterujące

Character	POSIX name	Comment
CTRL-D	EOF	End of file
CTRL-H	ERASE	Backspace one char.
CTRL-C	INTR	Interrupt process
CTRL-U	KILL	Erase entire line
CTRL-	QUIT	Force core dump
CTRL-Z	SUSP	Suspend
CTRL-Q	START	Start output
CTRL-S	STOP	Stop output
CTRL-R	REPRINT	Redisplay input (MINIX)
CTRL-V	LNEXT	Literal next (MINIX)
CTRL-O	DISCARD	Discard output (MINIX)
CTRL-M	CR	Carriage return
CTRL-J	NL	Linefeed

Noncanonical mode

MIN/TIME

	TIME = 0
MIN = 0	Zwróć natychmiast dostępne znaki.
MIN > 0	Poczekaj na MIN znaków i zwróć.
	TIME > 0
MIN = 0	Zwróć 1 odczytany znak lub 0 znaków jeśli nie udało się odczytać przed timeout'em.
MIN > 0	TIME oznacza maksymalne przerwy pomiędzy kolejnymi znakami. Zagar stratuje po wczytaniu pierwszego znaku. Zwraca przynajmniej 1 znak (MIN znaków jeśli żaden nie przekroczył czasu).

```
TIME = c_cc[VTIME] ;  
MIN   = c_cc[VMIN] ;
```

Output (memory-mapped)

video RAM

- w przestrzeni adresowej procesora (0xB0000)
- ekran - (24 linie) x (80 znaków) x (2 byte)
- scrolling hardware/software
- pozycja kurSORA
- znaki kontrolne

Output (memory-mapped)

Znaki kontrolne (ANSI escape sequences)

Escape sequence	Meaning
ESC[nA	Move up n lines
ESC[nB	Move down n lines
ESC[nC	Move right n spaces
ESC[nD	Move left n spaces
ESC[m;nH	Move cursor to ($y = m$, $x = n$)
...	...
ESC M	Scroll the screen backwards if the cursor is on the top line

Najważniejsze zdarzenia/wiadomości

- read (user pr. via FS)
- write (user pr. via FS)
- ioctl (user pr. via FS)
- keyboard interrupt (hardware via kernel)
- cancel previous request (FS)
- open
- close

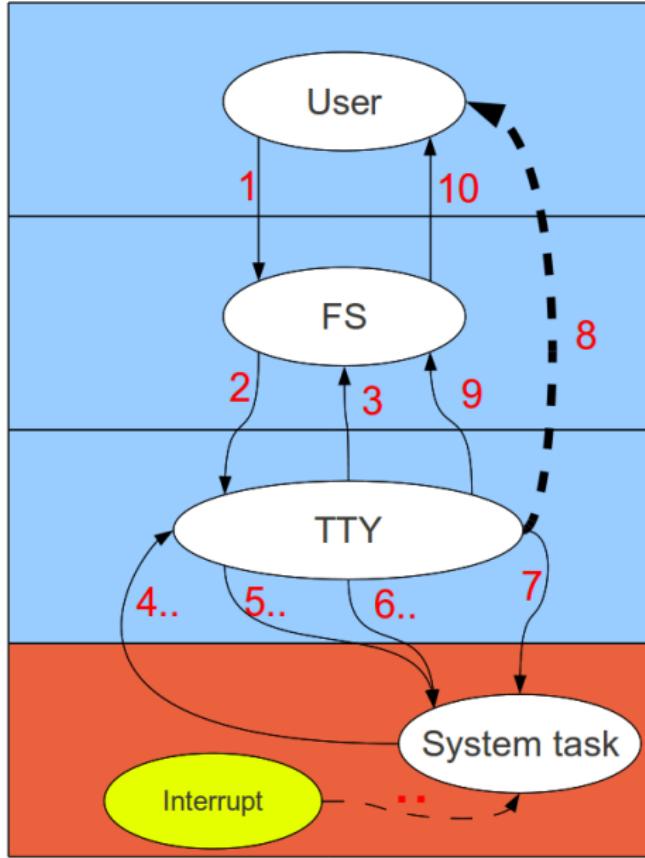
Terminal Driver - MINIX

Request

Filed	Type	Meaning
m.m_type	int	Operation requested
m.DEVICE	int	Minor device to use
m.PROC_NT	int	Process requesting I/O
m.COUNT	int	Byte count or ioctl code
m.POSITION	long	Position on device
m.ADDRESS	char *	Address within requesting process

tty_table - tablica struktur terminali

- kolejka znaków wprowadzonych ale nie odczytanych
- niespełnione żądania odczytu
- informacje o timeout'ach
- niespełnione żądania zapisu/wyświetlenia
- termios
- dane specyficzne dla urządzenia



```

1 PUBLIC void main(void){
2     ...
3     kb_init_once();
4     while (TRUE) {
5         /* Check for and handle any events on any of the ttys. */
6         for (tp = FIRST_TTY; tp < END_TTY; tp++) {
7             if (tp->tty_events) handle_events(tp);
8         }
9         receive(ANY, &tty_mess); /* Get a request message. */
10
11        switch (tty_mess.m_type) {
12            case SYN_ALARM: /* fall through */
13                expire_timers(); /* run watchdogs of expired timers */
14                continue; /* continue to check for events */
15            case HARD_INT: { /* hardware interrupt notification */
16                if (tty_mess.NOTIFY_ARG & kbd_irq_set)
17                    kbd_interrupt(&tty_mess); /* fetch chars from keyboard */
18                expire_timers(); /* run watchdogs of expired timers */
19                continue; /* continue to check for events */
20            }
21            case SYS_SIG: { /* system signal */
22                ...
23                if (sigismember(&sigset, SIGTERM)) cons_stop(); ...
24                continue; }
25            case PANIC_DUMPS: /* allow panic dumps */
26                cons_stop(); /* switch to primary console */
27                do_panic_dumps(&tty_mess);
28                continue;
29            case DIAGNOSTICS: /* a server wants to print some */
30                do_diagnostics(&tty_mess);
31                continue;
32            case FKEY_CONTROL: /* (un)register a fkey observer */
33                do_fkey_ctl(&tty_mess);
34                continue;
35            default: /* should be a driver request */
36                /* do nothing; end switch */
37        }

```



```

1 if (tty_mess.m_type == DEV_STATUS) {
2     do_status(&tty_mess);
3     continue;
4 }
5 line = tty_mess.TTY_LINE;
6 if ((line - CONS_MINOR) < NR_CONS) {
7     tp = tty_addr(line - CONS_MINOR);
8 } else /...
9 else {
10     tp = NULL;
11 }
12 /* If the device doesn't exist or is not configured return ENXIO. */
13 if (tp == NULL || ! tty_active(tp)) {
14     printf("Warning ,_TTY_got_illegal_request_%d_from_%d\n",
15            tty_mess.m_type, tty_mess.m_source);
16     tty_reply(TASK_REPLY, tty_mess.m_source,
17               tty_mess.PROC_NR, ENXIO);
18     continue;
19 }
20 /* Execute the requested device driver function. */
21 switch (tty_mess.m_type) {
22     case DEV_READ:    do_read(tp, &tty_mess);      break;
23     case DEV_WRITE:   do_write(tp, &tty_mess);     break;
24     case DEV_IOCTL:   do_ioctl(tp, &tty_mess);    break;
25     case DEV_OPEN:    do_open(tp, &tty_mess);     break;
26     case DEV_CLOSE:   do_close(tp, &tty_mess);    break;
27     case DEV_SELECT:  do_select(tp, &tty_mess);   break;
28     case CANCEL:     do_cancel(tp, &tty_mess);   break;
29     default:
30         printf("Warning ,_TTY_got_unexpected_request_%d_from_%d\n",
31                tty_mess.m_type, tty_mess.m_source);
32         tty_reply(TASK_REPLY, tty_mess.m_source,
33                   tty_mess.PROC_NR, EINVAL);
34 }
35 }
36 }

```

```
1 PUBLIC void handle_events(tty_y *tp){
2     ...
3     do {
4         tp->tty_events = 0;
5
6         /* Read input and perform input processing. */
7         (*tp->tty_devread)(tp, 0);
8
9         /* Perform output processing and write output. */
10        (*tp->tty_devwrite)(tp, 0);
11
12        /* ioctl waiting for some event? */
13        if (tp->tty_ioreq != 0) dev_ioctl(tp);
14    } while (tp->tty_events);
15
16    /* Transfer characters from the input queue to a waiting process. */
17    in_transfer(tp);
18
19    /* Reply if enough bytes are available. */
20    if (tp->tty_incum >= tp->tty_min && tp->tty_inleft > 0) {
21        if (tp->tty_inrepcode == REVIVE) {
22            notify(tp->tty_incaller);
23            tp->tty_inrevived = 1;
24        } else {
25            tty_reply(tp->tty_inrepcode, tp->tty_incaller,
26                      tp->tty_inproc, tp->tty_incum);
27            tp->tty_inleft = tp->tty_incum = 0;
28        }
29    }
30    if (tp->tty_select_ops)
31        select_retry(tp);
32 #if NR_PTYs > 0
33    if (ispty(tp))
34        select_retry_pty(tp);
35 #endif
36 }
```



```
1 PUBLIC void kbd_interrupt(m_ptr)
2 message *m_ptr;
3 {
4 /* A keyboard interrupt has occurred. Process it. */
5 int scode;
6 static timer_t timer; /* timer must be static! */
7
8 /* Fetch the character from the keyboard hardware and acknowledge it. */
9 scode = scan_keyboard();
10
11 /* Store the scancode in memory so the task can get at it later. */
12 if (icount < KB_IN_BYTES) {
13 *ihead++ = scode;
14 if (ihead == ibuf + KB_IN_BYTES) ihead = ibuf;
15 icount++;
16 tty_table[ccurrent].tty_events = 1;
17 if (tty_table[ccurrent].tty_select_ops & SEL_RD) {
18 select_retry(&tty_table[ccurrent]);
19 }
20 }
21 }
```

```
1 PRIVATE int scan_keyboard()
2 {
3 /* Fetch the character from the keyboard hardware and acknowledge it. */
4 pvb_pair_t byte_in[2], byte_out[2];
5
6 byte_in[0].port = KEYBD; /* get the scan code for the key struck */
7 byte_in[1].port = PORT_B; /* strobe the keyboard to ack the char */
8 sys_vinb(byte_in, 2); /* request actual input */
9
10 pv_set(byte_out[0], PORT_B, byte_in[1].value | KBIT); /* strobe bit high */
11 pv_set(byte_out[1], PORT_B, byte_in[1].value); /* then strobe low */
12 sys_voutb(byte_out, 2); /* request actual output */
13
14 return(byte_in[0].value); /* return scan code */
15 }
```

```
1 PRIVATE void set_leds()
2 {
3 /* Set the LEDs on the caps, num, and scroll lock keys */
4 int s;
5 if (! machine.pc_at) return; /* PC/XT doesn't have LEDs */
6
7 kb_wait(); /* wait for buffer empty */
8 if ((s=sys_outb(KEYBD, LED.CODE)) != OK)
9     printf("Warning, sys_outb couldn't prepare for LED values: %d\n", s);
10    /* prepare keyboard to accept LED values */
11 kb_ack(); /* wait for ack response */
12
13 kb_wait(); /* wait for buffer empty */
14 if ((s=sys_outb(KEYBD, locks[ccurrent])) != OK)
15     printf("Warning, sys_outb couldn't give LED values: %d\n", s);
16    /* give keyboard LED values */
17 kb_ack(); /* wait for ack response */
18 }
```

```
1 PRIVATE int kb_ack()
2 {
3 /* Wait until kbd acknowledges last command; return zero if this times out. */
4
5     int retries, s;
6     u8_t u8val;
7
8     retries = MAX_KB_ACK_RETRIES + 1;
9     do {
10         s = sys_inb(KEYBD, &u8val);
11         if (u8val == KB_ACK)
12             break; /* wait for ack */
13     } while(--retries != 0); /* continue unless timeout */
14
15     return(retries); /* nonzero if ack received */
16 }
```

```
1 PRIVATE void out_char(console_t * cons, int c){
2     if (cons->c_esc_state > 0) {
3         parse_escape(cons, c);
4         return;
5     }
6     switch(c) {
7         case 000: /* null is typically used for padding */
8             return; /* better not do anything */
9         case 007: /* ring the bell */
10            flush(cons); /* print any chars queued for output */
11            beep();
12            return;
13     /* ... */
14     default: /* printable chars are stored in ramqueue */
15         if (cons->c_column >= scr_width) {
16             if (!LINEWRAP) return;
17             if (cons->c_row == scr_lines - 1) {
18                 scroll_screen(cons, SCROLL_UP);
19             } else {
20                 cons->c_row++;
21             }
22             cons->c_column = 0;
23             flush(cons);
24         }
25         if (cons->c_rwords == buflen(cons->c_ramqueue)) flush(cons);
26         cons->c_ramqueue[cons->c_rwords++] = cons->c_attr | (c & BYTE);
27         cons->c_column++; /* next column */
28     }
29 }
30 }
```

```
1 PRIVATE void flush(cons)
2 register console_t *cons; /* pointer to console struct */
3 {
4 /* Send characters buffered in 'ramqueue' to screen memory, check the new
5 * cursor position, compute the new hardware cursor position and set it.
6 */
7 unsigned cur;
8 tty_t *tp = cons->c_tty;
9
10 /* Have the characters in 'ramqueue' transferred to the screen. */
11 if (cons->c_rwords > 0) {
12 mem_vid_copy(cons->c_ramqueue, cons->c_cur, cons->c_rwords);
13 cons->c_rwords = 0;
14
15 /* TTY likes to know the current column and if echoing messed up. */
16 tp->tty_position = cons->c_column;
17 tp->tty_reprint = TRUE;
18 }
19
20 /* Check and update the cursor position. */
21 if (cons->c_column < 0) cons->c_column = 0;
22 if (cons->c_column > scr_width) cons->c_column = scr_width;
23 if (cons->c_row < 0) cons->c_row = 0;
24 if (cons->c_row >= scr_lines) cons->c_row = scr_lines - 1;
25 cur = cons->c_org + cons->c_row * scr_width + cons->c_column;
26 if (cur != cons->c_cur) {
27 if (cons == curcons) set_6845(CURSOR, cur);
28 cons->c_cur = cur;
29 }
30 }
```

```
1 PRIVATE void scroll_screen(cons, dir)
2 register console_t *cons; /* pointer to console struct */
3 int dir; /* SCROLL_UP or SCROLL_DOWN */
4 {
5     unsigned new_line, new_org, chars;
6
7     flush(cons);
8     chars = scr_size - scr_width; /* one screen minus one line */
9
10    if (dir == SCROLL_UP) {
11        /* Scroll one line up in 3 ways: soft, avoid wrap, use origin. */
12        if (softscroll) {
13            vid_vid_copy(cons->c_start + scr_width, cons->c_start, chars);
14        } else {
15            if (!wrap && cons->c_org + scr_size + scr_width >= cons->c_limit) {
16                vid_vid_copy(cons->c_org + scr_width, cons->c_start, chars);
17                cons->c_org = cons->c_start;
18            } else {
19                cons->c_org = (cons->c_org + scr_width) & vid_mask;
20            }
21            new_line = (cons->c_org + chars) & vid_mask;
22        } else {
23            ...
24        }
25        /* Blank the new line at top or bottom. */
26        blank_color = cons->c_blank;
27        mem_vid_copy(BLANK_MEM, new_line, scr_width);
28
29        /* Set the new video origin. */
30        if (cons == curcons) set_6845(VID_ORG, cons->c_org);
31        flush(cons);
32 }
```

```
1 PRIVATE void set_6845(reg, val)
2 int reg;          /* which register pair to set */
3 unsigned val;      /* 16-bit value to set it to */
4 {
5 /* Set a register pair inside the 6845.
6 * Registers 12-13 tell the 6845 where in video ram to start
7 * Registers 14-15 tell the 6845 where to put the cursor
8 */
9 pvb_pair_t char_out[4];
10 pv_set(char_out[0], vid_port + INDEX, reg); /* set index register */
11 pv_set(char_out[1], vid_port + DATA, (val>>8) & BYTE); /* high byte */
12 pv_set(char_out[2], vid_port + INDEX, reg + 1); /* again */
13 pv_set(char_out[3], vid_port + DATA, val&BYTE); /* low byte */
14 sys_voutb(char_out, 4); /* do actual output */
15 }
16
17
18 PRIVATE void get_6845(reg, val)
19 int reg;          /* which register pair to set */
20 unsigned *val;      /* 16-bit value to set it to */
21 {
22     char v1, v2;
23 /* Get a register pair inside the 6845. */
24     sys_outb(vid_port + INDEX, reg);
25     sys_inb(vid_port + DATA, &v1);
26     sys_outb(vid_port + INDEX, reg+1);
27     sys_inb(vid_port + DATA, &v2);
28     *val = (v1 << 8) | v2;
29 }
```

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

5 Scheduling

- Batch systems

Hierarchia pamięci

Rejestry	1 - 3 ns
Level 1 Cache	2 - 8 ns
Level 2 Cache	5 - 12 ns
Memory	10-60 ns
Hard Disk	3 000 000 - 10 000 000 ns

na podstawie:

Understanding CPU caching and performance By Jon Stokes, **2002**

Główne zagadnienia

- Relokacja
- Ochrona/Bezpieczeństwo
- Przydzielanie pamięci / Swapping
- Współdzielenie pamięci (shared code / libraries)

Przydzielanie pamięci

Fragmentacja - Strategie alokacji

- first fit
- next fit
- best fit
- worst fit

Żądanie zwiększenia pamięci

- relokacja
- swap

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

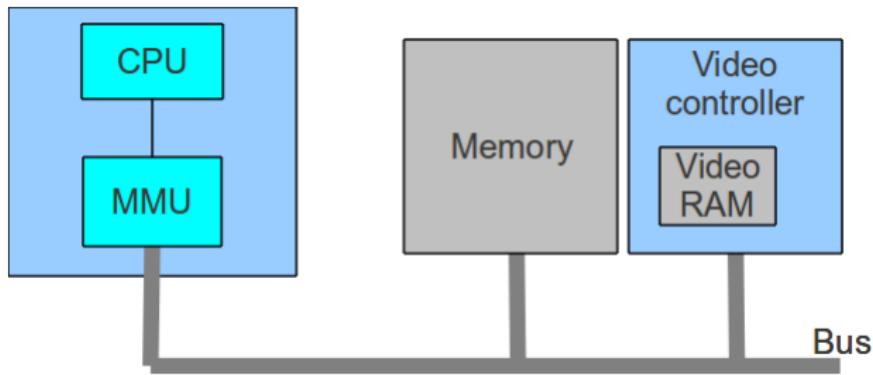
4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

5 Scheduling

- Batch systems

Pamięć wirtualna



Stronicowanie

- virtual address space \leftrightarrow pages
- physical memory \leftrightarrow page frames

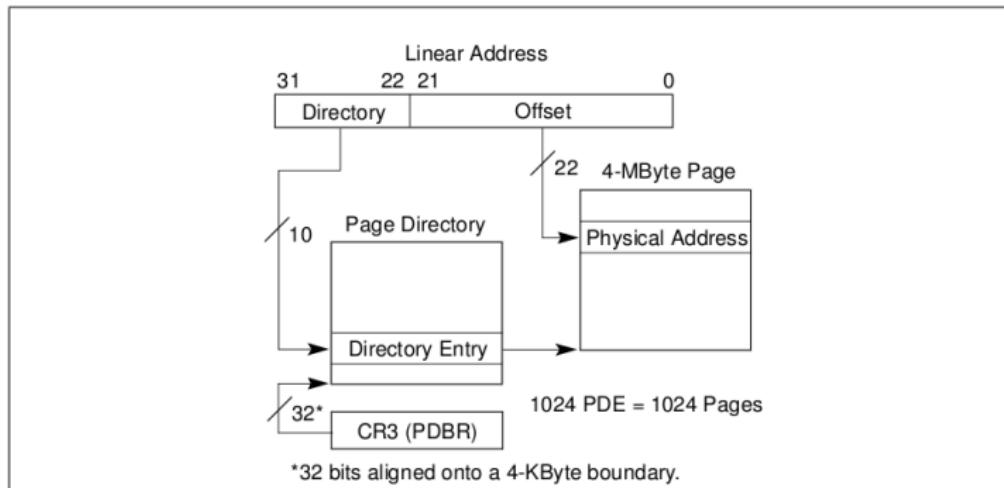
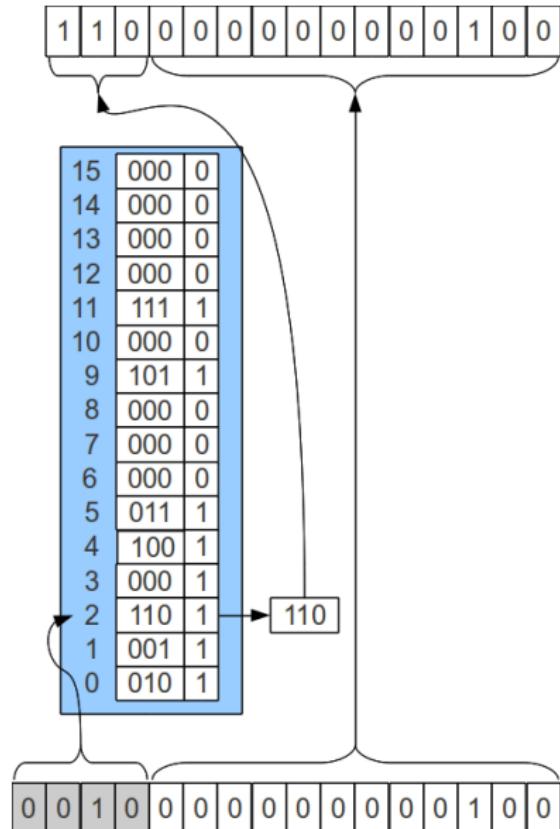


Figure 3-13. Linear Address Translation (4-MByte Pages)

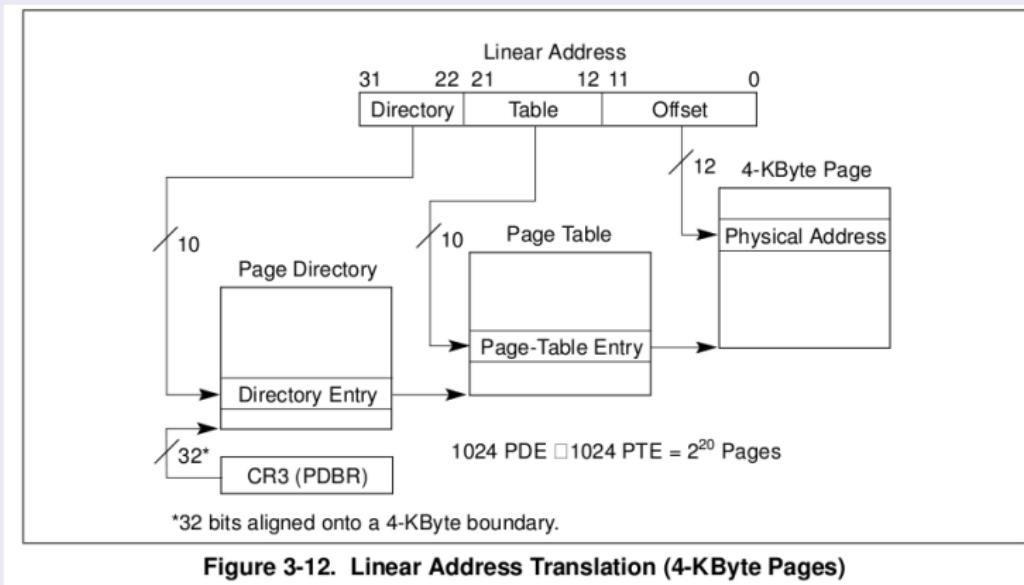
MMU & page fault



Tablica stron.

rozmiar vs szybkość

Wielopoziomowe tablice stron



x86 Page Table Entry

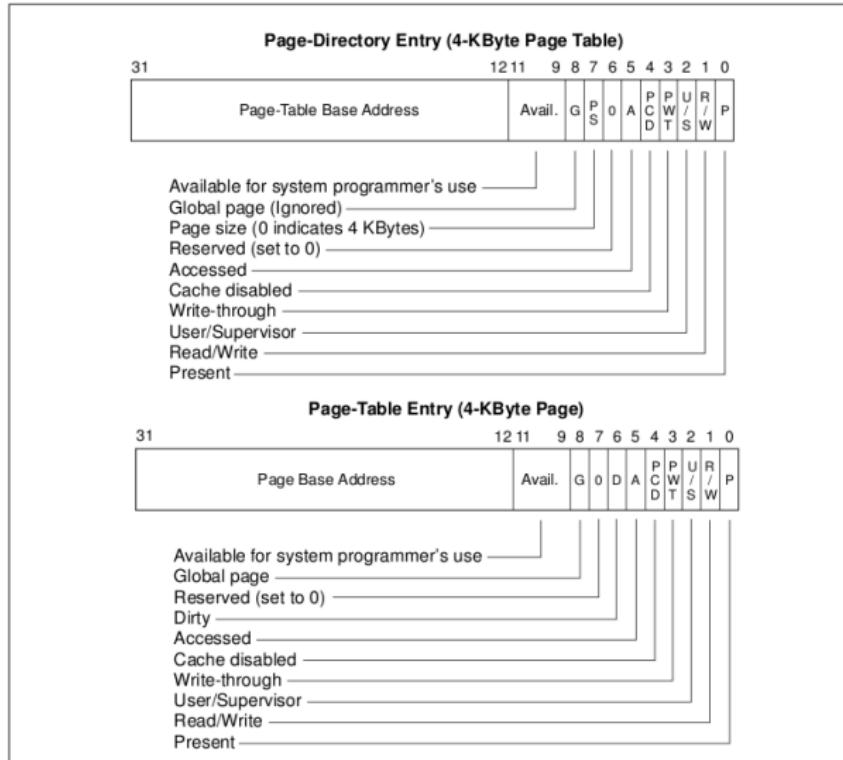
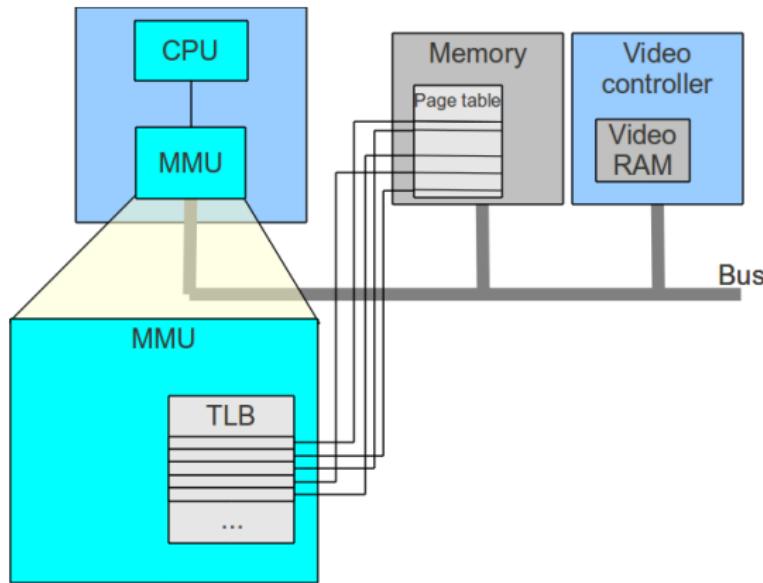


Figure 3-14. Format of Page-Directory and Page-Table Entries for 4-KByte Pages
and 32-Bit Physical Addresses

Intel Architecture Software Developer's Manual, Volume 3: System Programming

TLBs - Translation Lookaside Buffers

Locality of reference



RISC → software TLB management (TLB fault)

Strategie ładowania/zastępowania stron

Strategia optymalna

Zastąp stronę która będzie najpóźniej używana.
(nie do zrealizowania przy pierwszym wykonaniu)

FIFO

Ostatnio nieużywana

Bit w tabeli stron: **referenced** (Accessed), **modified** (Dirty)

Kolejność wybierania:

- not referenced , not modified
- not referenced, modified
- referenced, not modified
- referenced, modified

W obrębie danej klasy losowo.

Second Chance / Clock PRA

Modyfikacja FIFO:

- Usuwaj tylko jeśli bit `referenced == 0`,
- wpp. wyczyść bit `referenced` i przesuń stronę na koniec kolejki.

Zamiast kolejki lepiej używać *circular buffer* (clock).

Strategie ładowania/zastępowania stron

Least Recently Used PRA

Zastąp stronę która była używana najdawniej.

Wymaga wsparcia sprzętowego (update przy każdej referencji do pamięci).

Not Frequently Used PRA

Po każdym przerwaniu zegarowym, kasuj bit **referenced** i zwiększaj licznik stronom, które mają ten bit ustawiony.

Usuwaj strony z najmniejszą wartością licznika.

Aging PRA

Po każdym przerwaniu zegarowym, dla każdej strony przesuń bity licznika o 1 w prawo uzupełniając bitem **referenced** po czym skasuj bit **referenced**.

Usuwaj strony z najmniejszą wartością licznika.

- Local / Global allocation policy
- Page Fault Frequency
- On demand paging.

Rozmiar stron.

- s - średni rozmiar procesu
- e - rozmiar wiersza w tabeli stron
- p - rozmiar strony

$$\text{Overhead pre process} = \frac{s \cdot e}{p} + \frac{p}{2}.$$

$$p = \sqrt{2se}$$

$$s = 1MB, e = 8 \Rightarrow p = 4KB$$

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

5 Scheduling

- Batch systems

Segmentacja

Segmenty - wirtualne przestrzenie adresowe.

Adres = identyfikator segmentu + offset w obrębie segmentu

Podstawowe segmenty procesu:

- segment kodu (cs)
- segment danych (ds)
- segment stosu (ss)

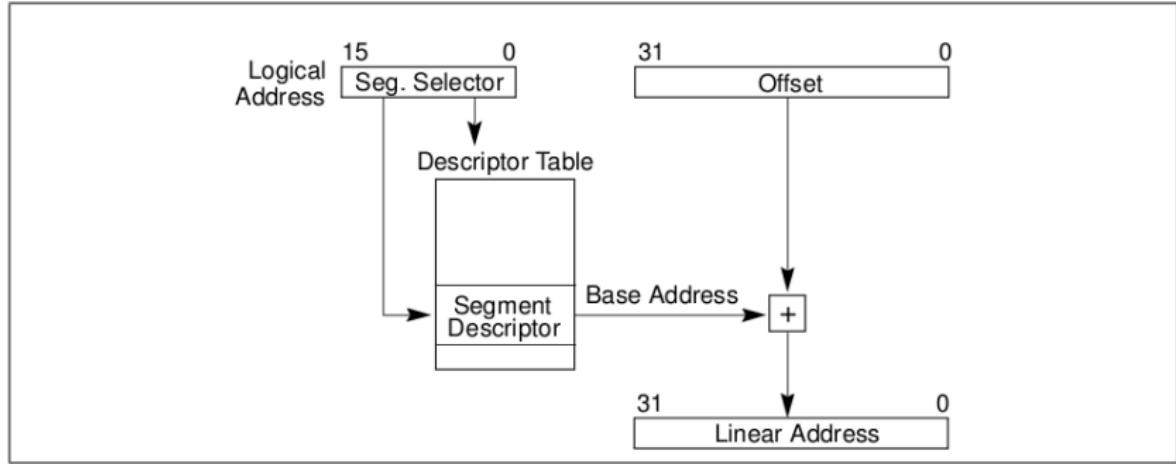


Figure 3-5. Logical Address to Linear Address Translation

Intel Architecture Software Developer's Manual, Volume 3: System Programming

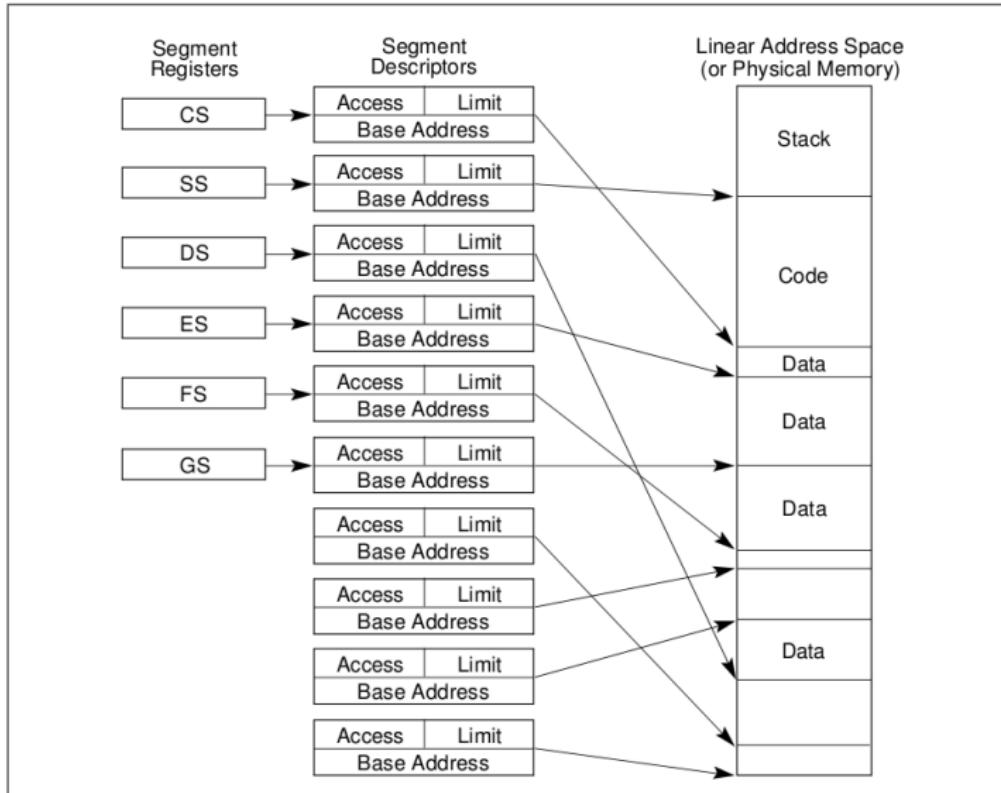
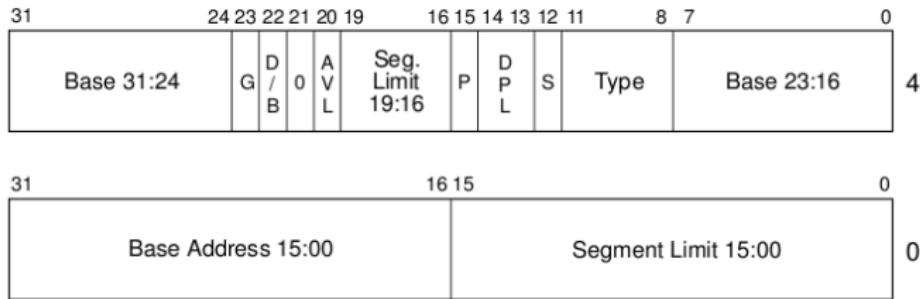


Figure 3-4. Multisegment Model



AVL — Available for use by system software

BASE — Segment base address

D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)

DPL — Descriptor privilege level

G — Granularity

LIMIT — Segment Limit

P — Segment present

S — Descriptor type (0 = system; 1 = code or data)

TYPE — Segment type

Figure 3-8. Segment Descriptor

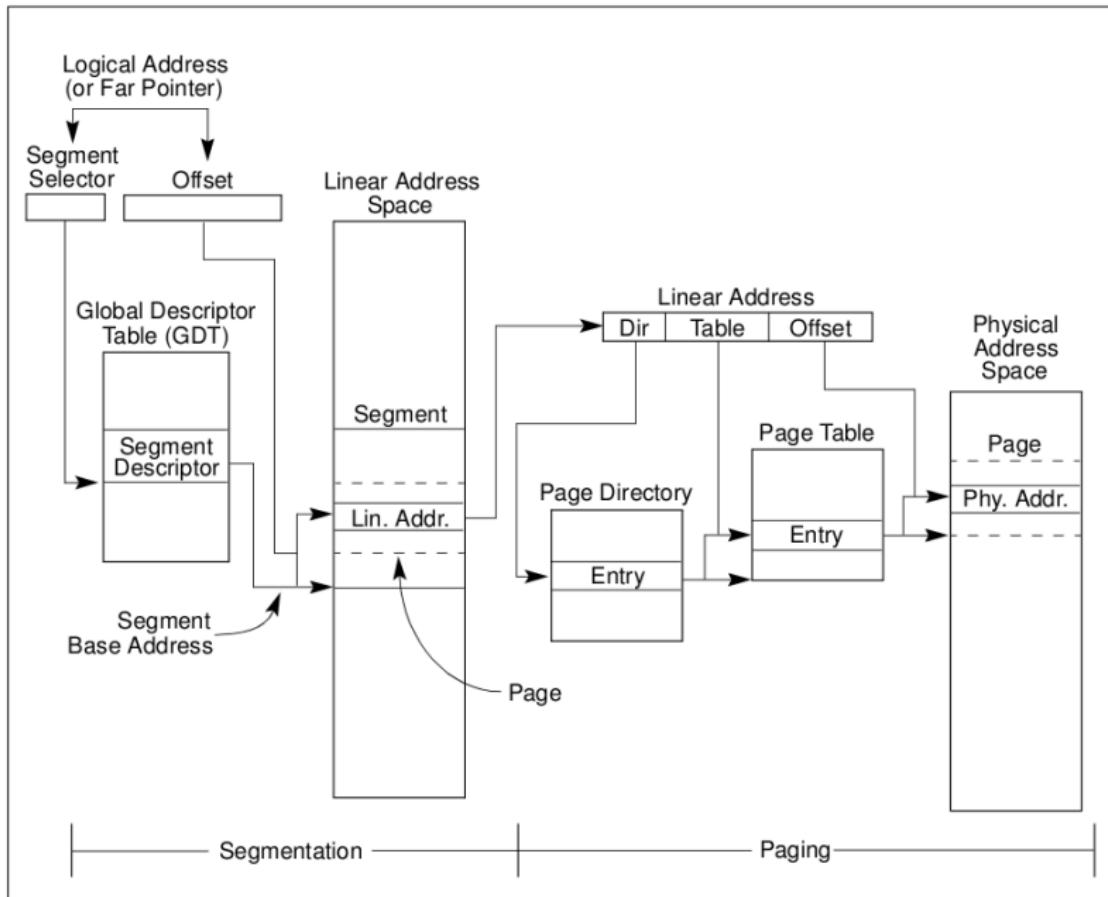


Figure 3-1. Segmentation and Paging

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

5 Scheduling

- Batch systems

Procesy i pamięć

- fork
- exit
- wait
- waitpid
- brk
- exec

Sygnały

- kill
- alarm
- pause
- sigaction
- sigsuspend
- sigpending
- sigprocmask
- sigreturn

Inne

- getuid, getgid, getpid, setuid, setgid, setsid, getpgrp
- time, stime, times, ptrace, reboot, svrctl, getsysinfo, getprocnr
- memalloc, memfree
- getpriority, setpriority, gettimeofday

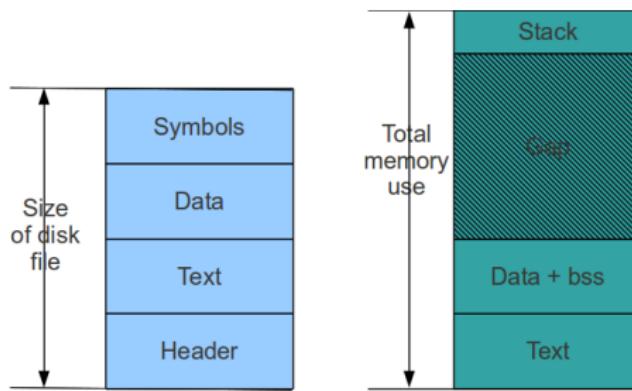
```
push    ebp  
mov     ebp, esp  
push    ebx  
mov     eax, SRC_DST(ebp)  
mov     ebx, MESSAGE(ebp)  
mov     ecx, SEND  
int    SYSVEC  
pop    ebx  
pop    ebp  
ret
```

Podstawowe segmenty

- segment kodu
- segment danych
- segment stosu

Organizacja pamięci - MINIX

- czysta segmentacja
- segment danych i stosu w jednym segmencie hardware'owym
- pamięć dla procesu jest przydzielana w momencie wykonywania exec/fork, przydział **nie może** być zwiększyony w trakcie działania
- segment kodu może być dzielony pomiędzy procesami, w przeciwnym przypadku jest łączony z danymi i stosem



chmem

PM - struktury danych

- `mproc` - tabela procesów
- `hole` - tabela wolnych obszarów pamięci

Kernel

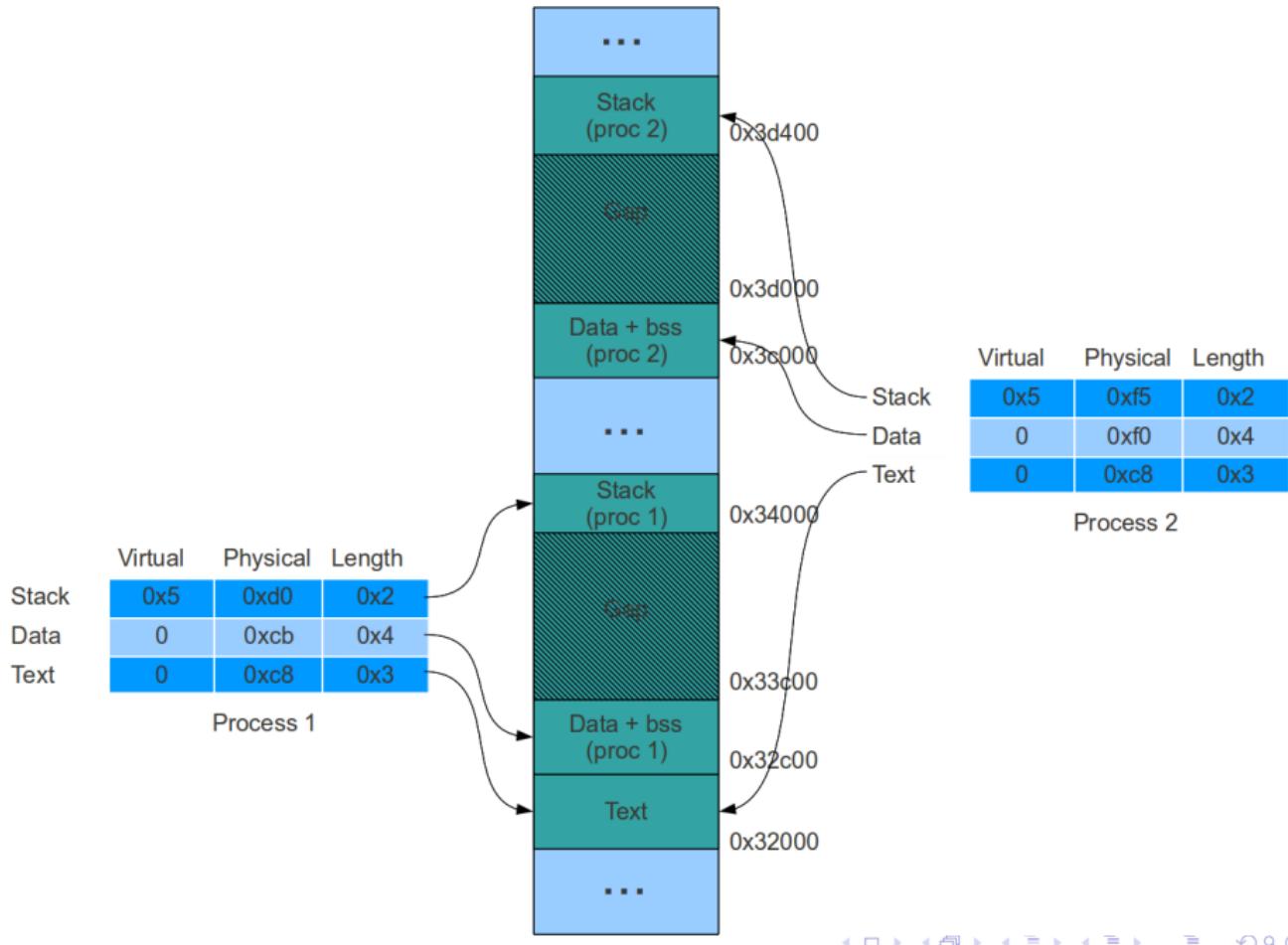
- Registers
- Program counter
- Program status word
- Stack Pointer
- Process state
- Current scheduling priority
- Maximum scheduling priority
- Scheduling ticks left
- Quantum size
- CPU time used
- Message queue pointers
- Pending signal bits
- Various flag bits
- Process name

PM - mproc

- Pointer to text segment
- Pointer to data segment
- Pointer to `bss` stack segment
- Exit status
- Signal status
- Process ID
- Parent Process
- Process group
- Children's CPU time
- Real UID
- Effective UID
- Real GID
- Effective GID
- File info for sharing text
- Bitmaps for signals
- Various flag bits

File management

- UMASK mask
- Root directory
- Working directory
- File descriptors
- Real UID
- Effective UID
- Real GID
- Effective GID
- Controlling tty
- Save area for read/write
- System call parameters
- Various flag bits



The Hole List

```
struct hole
PRIVATE struct hole {
    struct hole *h_next; /* pointer to next entry on the list */
    phys_clicks h_base; /* where does the hole begin? */
    phys_clicks h_len; /* how big is the hole? */
} hole[NR_HOLES];
```

click = 1024B

```

1 PUBLIC int do_brk(){
2 // The parameter, 'addr' is the new virtual address in D space.
3 register struct mproc *rmp;
4 int r;
5 vir_bytes v, new_sp;
6 vir_ticks new_ticks;
7
8 rmp = mp;
9 v = (vir_bytes) m_in.addr;
10 new_ticks = (vir_ticks) ( ((long) v + CLICK_SIZE - 1) >> CLICK_SHIFT );
11 if (new_ticks < rmp->mp_seg[D].mem_vir) {
12 rmp->mp_reply.reply_ptr = (char *) -1;
13 return(ENOMEM);
14 }
15 new_ticks -= rmp->mp_seg[D].mem_vir;
16 if ((r=get_stack_ptr(who, &new_sp)) != OK) /* ask kernel for sp value */
17 panic(FILE, "couldn't get stack pointer", r);
18 r = adjust(rmp, new_ticks, new_sp);
19 rmp->mp_reply.reply_ptr = (r == OK ? m_in.addr : (char *) -1);
20 return(r); /* return new address or -1 */
21 }
22
23 PUBLIC int adjust(rmp, data_ticks, sp)
24 /**
25 if (changed) sys_newmap((int)(rmp - mproc), rmp->mp_seg);
26 /**
27 }

```

fork

- ① Sprawdź, czy jest miejsce w tabeli procesów.
- ② Zaalokuj miejsce na stos i dane dziecka.
- ③ Skopiuj zawartość danych i stosu rodzica do pamięci dziecka.
- ④ Skopiuj wiersz rodzica w tabeli procesów do wiersza dziecka.
- ⑤ Uaktualnij `memory_map` w wierszu dziecka.
- ⑥ Wybierz PID dla dziecka.
- ⑦ Poinformuj jądro i FS o dziecku.
- ⑧ Przekaż `memory_map` jądru.
- ⑨ Wyślij odpowiedź do rodzica i do dziecka.

exit

- ① Usuń alarmy zgłoszone przez proces (PM).
- ② Powiadom jądro i FS że proces się zakończył.
- ③ Skasuj flagę REPLY - PM nie wyśle żadnych wiadomości.
- ④ Zwolnij pamięć procesu (data + ew. text)
- ⑤ Ustaw exitstatus w tabeli procesów.
- ⑥ Jeśli rodzic czeka na zakończenie dziecka (wait) wyślij exitstatus w wiadomości i zwolnij wiersz w tabeli procesów,
- ⑦ Jeśli nie, wyślij do rodzica SIGCHLD i ustaw flagę ZOMBIE.
- ⑧ Zmień rodziców wszystkich dzieci procesu na INIT.
- ⑨ Jeśli proces był liderem grupy procesów wyślij SIGHUP do wszystkich procesów z grupy.

- ① Sprawdź prawa dla pliku (rx).
- ② Wczytaj nagłówek i określ rozmiary segmentów.
- ③ Załaduj argumenty i zmienne środowiskowe z pr. wołającego.
- ④ Zaalokuj pamięć dla nowego programu i zwolnij starą (!).
- ⑤ Przygotuj stos dla nowego programu.
- ⑥ Skopiuj dane (+ew. text) nowego programu do pamięci.
- ⑦ Sprawdź setuid, setgid i uaktualnij.
- ⑧ Uaktualnij tabelę procesów.
- ⑨ Poinformuj jądro że proces jest runnable (oraz FS - FD_CLOEXEC).
- ⑩ no reply

exec - przygotowanie stosu

main

```
int main(int argc, char** argv, char** envp);
```

Polecenie:

```
ls -l f.c g.c
```

dla zmiennych środowiskowych HOME=/usr/ast .

```
ls -l f.c g.c
```

\ 0	t	s	a	8188
/	r	s	u	8184
/	=	E	M	8180
O	H	\ 0	c	8176
.	g	\ 0	c	8172
.	f	\ 0	l	8168
-	\ 0	s	l	8164
		0		8160
		8178		8156
		0		8152
		8174		8148
		8170		8144
		8167		8140
		8164		8136
		8156		8132
		8136		8128
		4		8124
		return		8120

return ?

```
crtso:  
xor    ebp, ebp      ! clear for backtrace of core files  
mov    eax, (esp)    ! argc  
lea    edx, 4(esp)    ! argv  
lea    ecx, 8(esp)(eax*4)  ! envp  
/.../  
push   ecx          ! push envp  
push   edx          ! push argv  
push   eax          ! push argc  
/.../  
call   _main         ! main(argc, argv, envp)  
  
push   eax          ! push exit status  
call   _exit  
  
hlt    ! force a trap if exit fails
```

Obsługa sygnałów - POSIX

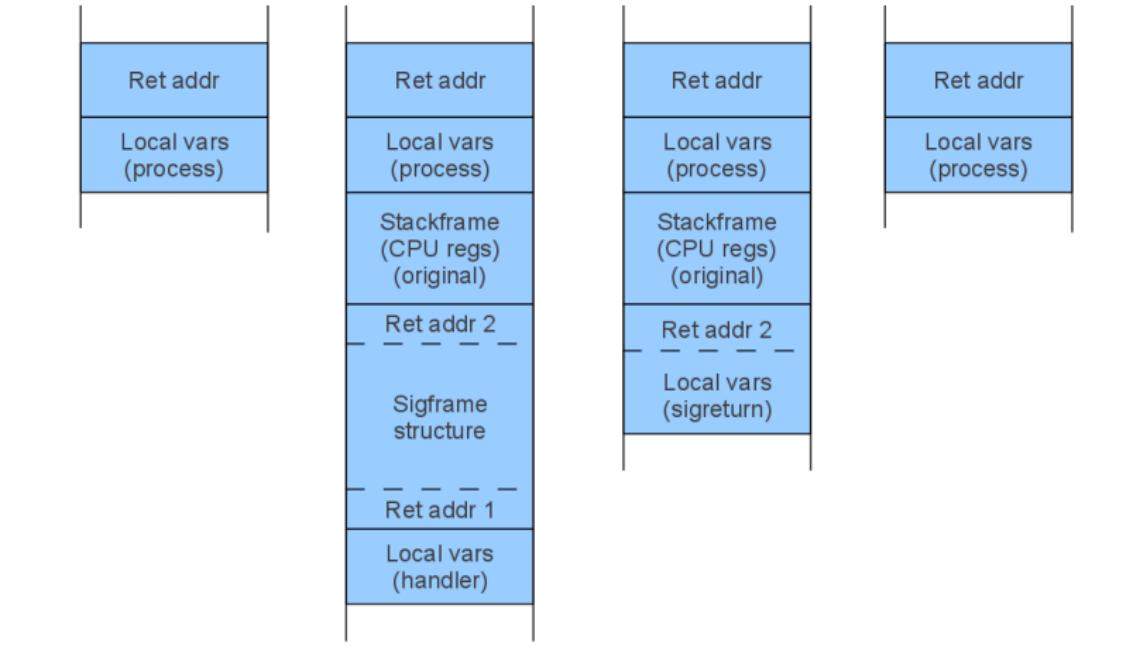
```
#include <signal.h>

int sigaction(int sig, const struct sigaction *restrict act,
              struct sigaction *restrict oact);

struct sigaction {
    __sighandler_t sa_handler; /* SIG_DFL, SIG_IGN, SIG_MESS ... */
    sigset_t sa_mask;
    int sa_flags;
}
```

Przykłady sygnałów + źródła

SIGINT	TTY
SIGILL	kernel (HARDWARE)
SIGUSR1	kill sysall
SIGPIPE	FS
SIGALARM	PM
SICHLD	PM
SIGKMESS	kernel



Obsługa sygnałów -c.d.

- sigreturn - system call
- Ret addr 2 - original ip (debug)
- EINTR - errno
- procesy które nie wznowią swojego programu po sygnale

User-Space Timers

PM przechowuje kolejkę alarmów procesów użytkowników.

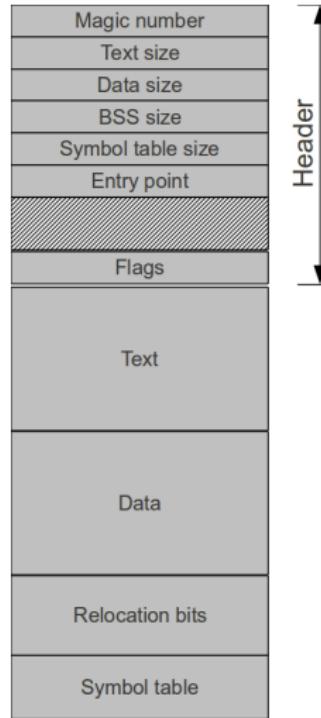
Rejestruje własny alarm w jądrze z czasem najbliższego alarmu z kolejki.

Struktura plików

- Ciąg byte'ów.
- Ciąg rekordów.
- Drzewo/tablica asocjacyjna (key-value).
- Ciąg strumieni byte'ów.

Typy plików

- pliki wykonywalne (binarne/skrypty)
- katalogi
- pliki specjalne znakowe/blokowe
- pozostałe - dane



Rodzaje dostępu

- Dostęp sekwencyjny.
- Dostęp swobodny (random).

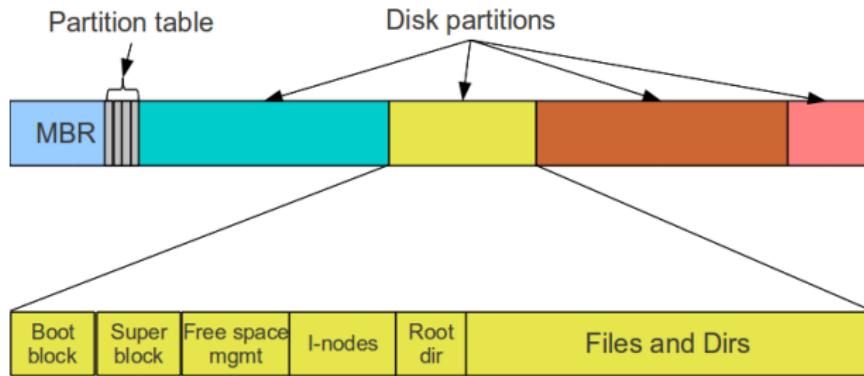
Operacje na plikach

- ① create
- ② delete
- ③ open
- ④ close
- ⑤ read
- ⑥ write
- ⑦ append
- ⑧ seek
- ⑨ get attributes
- ⑩ set attributes
- ⑪ rename
- ⑫ lock

Operacje na katalogach

- ① create
- ② delete
- ③ opendir
- ④ closedir
- ⑤ readdir
- ⑥ rename
- ⑦ link
- ⑧ unlink

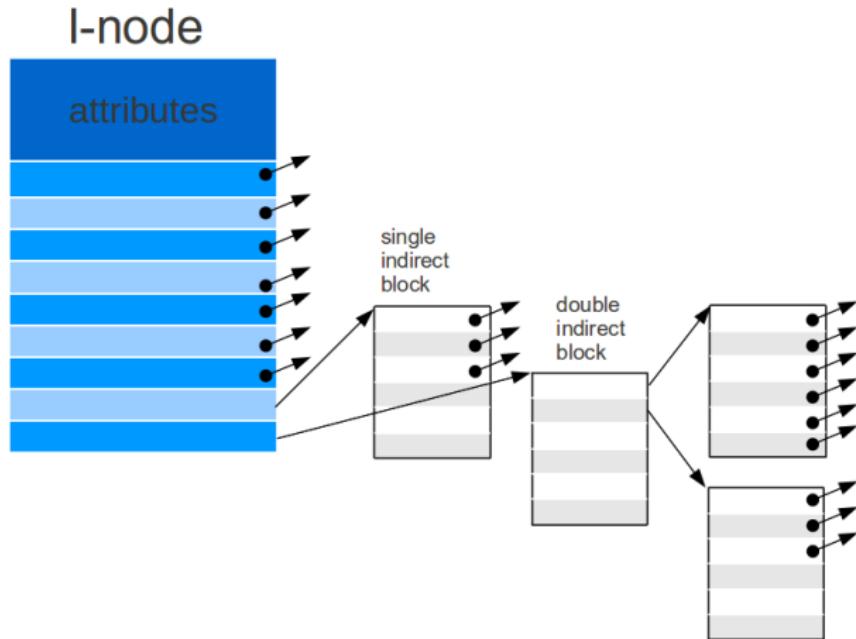
Organizacja dysku



Organizacja dysku - pliki

- ciągła alokacja
- lista bloków, dowiązania w blokach (rozmiar bloków nie jest potągą 2)
- lista bloków, dowiązania w tablicy alokacji (FAT)
- I-nodes

I-nodes



Katalogi

Katalog - lista referencji do plików

- ① Atrybuty pliku w strukturach katalogu (prawa dostępu, czasy ostatniego dostępu itp.)
- ② Atrybuty w I-node pliku. W strukturach katalogu nazwa i numer I-node pliku.

Rozmiary bloków

Performance vs Space utilization

Typowe rozmiary bloków: 1KB, 2KB, 4KB.

Zarządzanie wolnymi blokami

- Wolne bloki powiązane w liste.
- Lista tablic wolnych bloków (!)
- Bitmapa.

Kopie zapasowe

- physical dump
- logical dump
 - całościowe
 - przyrostowe

Dodatkowe problemy:

- wszelkie pliki przeczytane w jednej chwili
- struktura katalogów w przyrostowych kopach
- linki
- pliki z dziurami
- pliki specjalne (pipes)
- bad blocks (jeśli obsługiwane programowo)

Naprawianie systemu plików

Sprawdzanie bloków

Przeglądamy listę/mapę wolnych bloków oraz i-node'y.

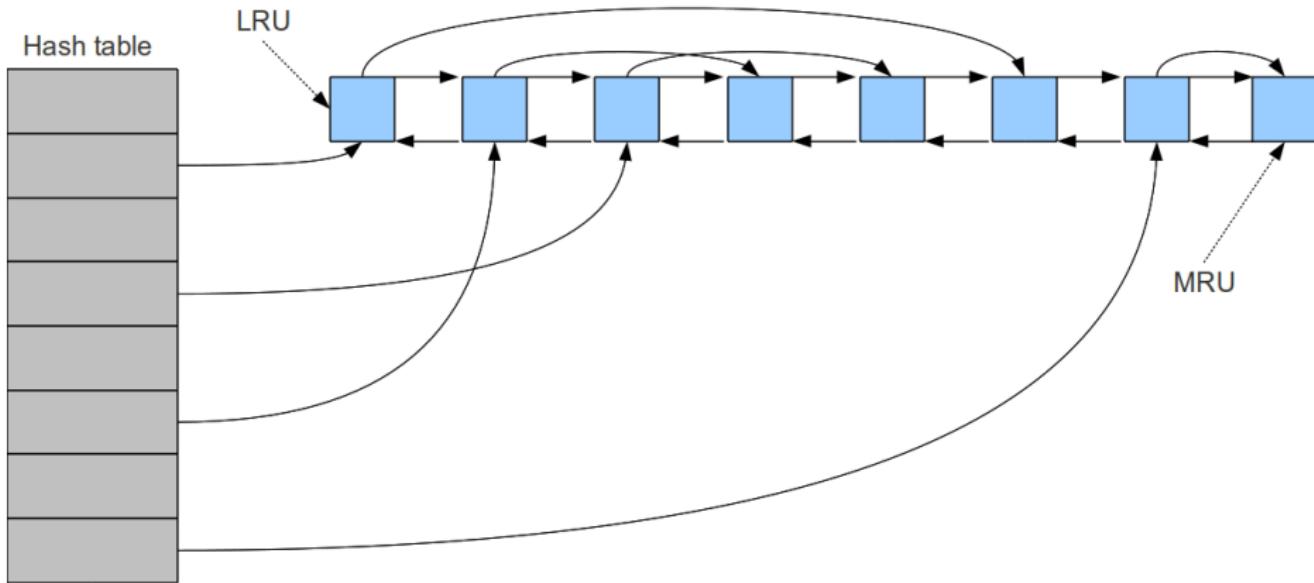
Problemy, które mogą się pojawić:

- Blok nie pojawia się.
- Blok pojawia się wielokrotnie na liście wolnych.
- Blok pojawia się wielokrotnie wśród I-node'ów.
- Zajęty blok pojawia się na liście wolnych.

Sprawdzanie struktury katalogów

Sprawdzamy, czy dla każdego pliku link count zgadza się z liczbą wystąpień w katalogach.

Wydajność - block cache

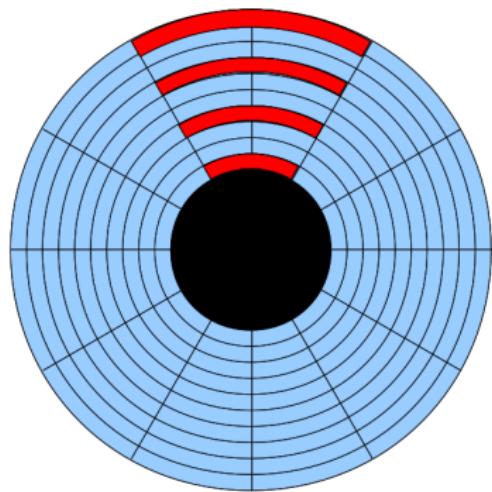
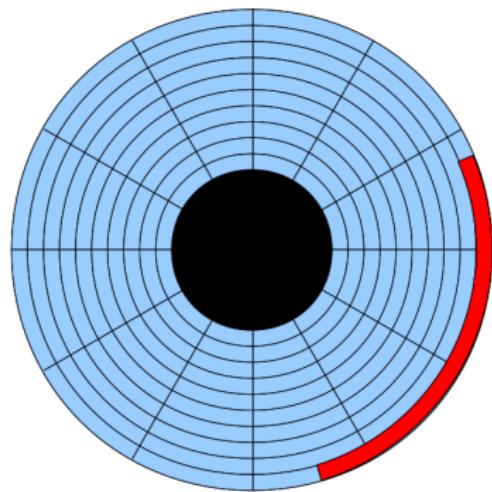


- zmiana kryterów odzyskiwania
- *write-through* cache

Wydajność - czytanie bloków z wyprzedzeniem

- sequential access mode
- random access mode

Wydajność - redukcja ruchów głowicy dysku



Ochrona zasobów

Dla każdego I-node'u:

- rwx - użytkownika
- rwx - grupy
- rwx - pozostałych

Możliwości zmiany praw dostępu dla procesu

- dodatkowy bit SETUID dla programów uruchamialnych,
- dodatkowy bit SETGID dla programów uruchamialnych,
- zmiana aktywnej grupy (w zależności od implementacji).

Outline

1 Zasady

2 POSIX

- Wstęp
- POSIX - standard
- POSIX - procesy
- POSIX - pliki
- POSIX - sygnały
- POSIX - remanent

3 Sequential Processes

- Multiprogramming
- IPC - InterProcess Communication

4 Architektura

- Monolithic kernel
- Micro kernel
- Tanenbaum - Torvalds debate

5 Scheduling

- Batch systems

MINIX - FILE SYSTEM - Interface

Wiadomości od procesów użytkownika

access, chdir, chmod, chown, chroot, close, creat, dup, fcntl, fstat, ioctl, link, lseek, mkdir, mknod, mount, open, pipe, read, rename, rmdir, stat, stime, **sync**, time, times, umask, umount, **unlink**, utime, write

Wiadomości od PM

exec, exit, fork, setgid, setsid, setuid

Pozostałe

revive, unpause

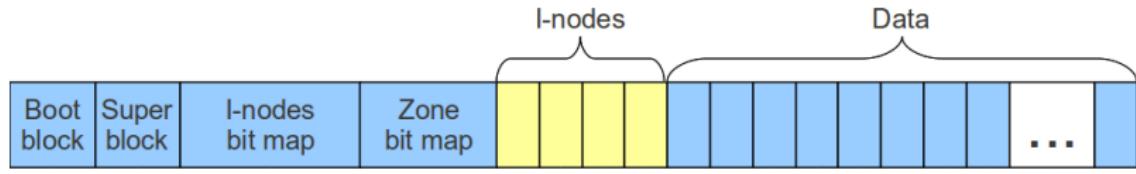
Organizacja dysku

Boot block

- ① zawsze 1024 byte'ów,
- ② zawiera kod programu ładującego system,
- ③ w ustalonym miejscu *magic number*

Super block

- ① zawsze 1024 byte'ów,
- ② zawiera rozmiar bloków
- ③ zawiera rozmiar stref (zones)
- ④ zone to block ratio



MINIX 3 superblock

Superblock na dysku:

Liczba i-node'ów (nieużywane)
Rozmiar bitmapy i-node'ów (w blokach)
Rozmiar bitmapy stref (w blokach)
Adres pierwszej strefy danych
$\log_2(\text{block}/\text{zone})$ (wypełnienie)
Maksymalny rozmiar pliku
Liczba stref
magic number (wypełnienie)
Rozmiar bloków
FS sub-version

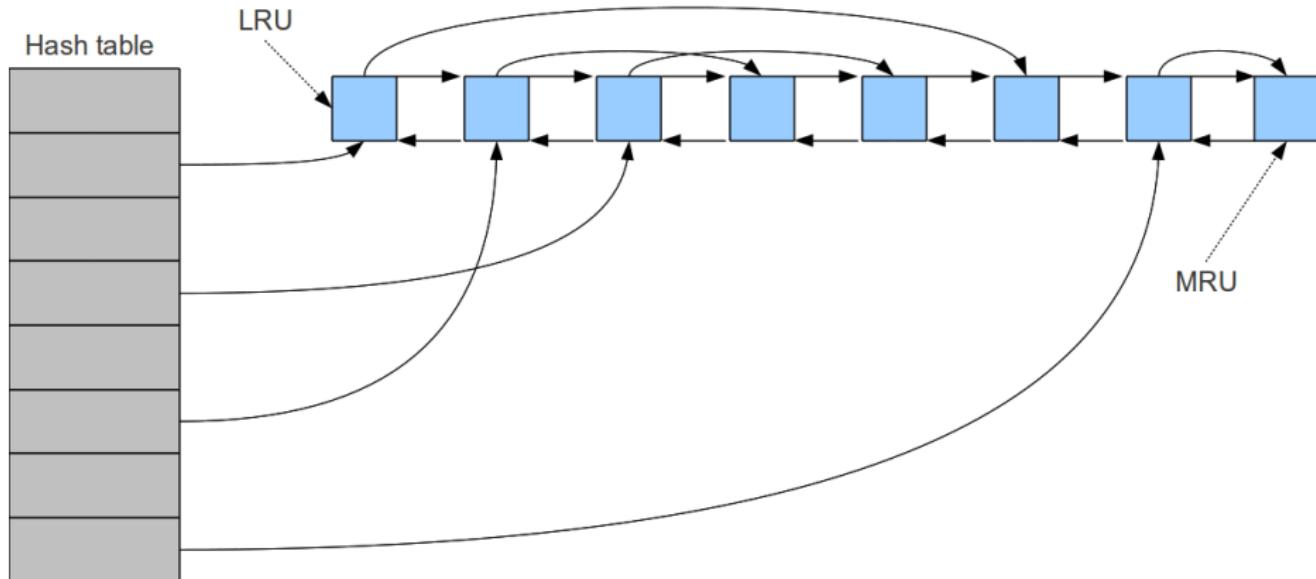
Superblock w pamięci:

Adres i-node dla katalogu root
Adres i-node dla ..
<i>inodes/block</i>
Device number
Read-only flag
FS-version
(wypełnienie)
<i>zones/i – node (liczba ref)</i>
<i>zones/(indir.block) (liczba ref)</i>
Pierwszy wolny bit w bitmapie i-node'ów
Pierwszy wolny bit w bitmapie stref

MINIX i-node (64B)

Mode
Liczba linków
UID
GID
Rozmiar pliku
Access time
Modification time
Status change time (i-node)
Zone 0
Zone 1
Zone 2
Zone 3
Zone 4
Zone 5
Zone 6
Indirect zone
Double indirect zone
(nieużywany)

MINIX - block cache

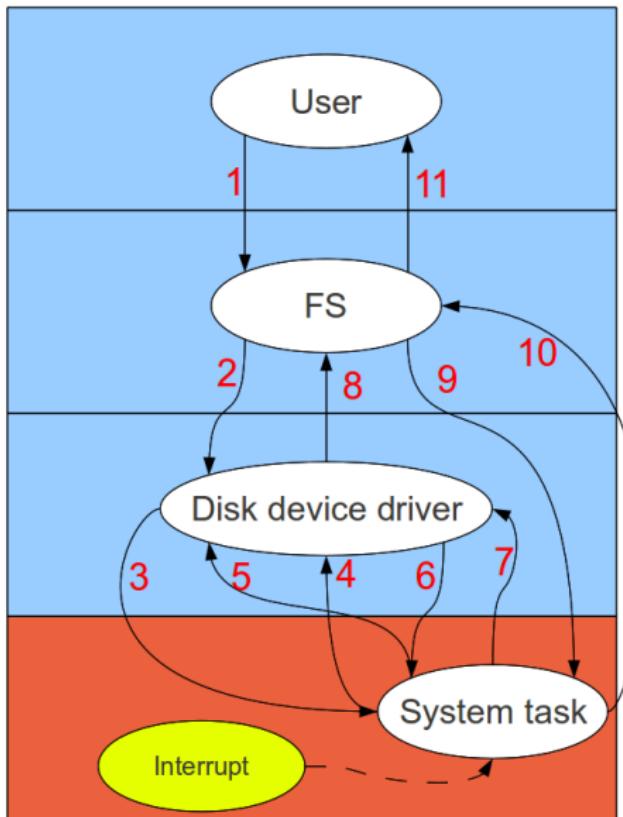


- bloki i-node'ów są dopisywane na początku
- zapełnione bloki danych są dopisywane na początku
- pozostałe na końcu

MINIX - filp table

```
1 EXTERN struct filp {  
2     mode_t filp_mode;      /* RW bits , telling how file is  
3     int filp_flags;        /* flags from open and fcntl */  
4     int filp_count;        /* how many file descriptors share  
5     struct inode *filp_ino; /* pointer to the inode */  
6     off_t filp_pos;        /* file position */  
7  
8     /.../  
9 } filp[NR_FILPS];
```

write() - ostatni raz



Bootstraping

Hardware

- ① czyta pierwszy sektor dysku (Master Boot Record)
- ② kopiuje w ustalone miejsce w pamięci
- ③ wykonuje

Dysk podzielony na partycje.

- ① przenosi się w inne miejsce w pamięci,
- ② odczytuje tablicę partycji,
- ③ czyta pierwszy sektor aktywnej partycji,
- ④ wykonuje.

Boot monitor

W końcu kod załadowany ładuje do pamięci boot monitor.

Boot monitor

- lokalizuje na dysku obraz systemu (używa systemu plików)
- ładuje do pamięci programy obrazu w predefiniowane miejsca
- odczytuje z obrazu informacje o systemie (np. 16/32 bts) oraz tablicę nagłówków a.out programów obrazu
- jeśli trzeba przechodzi w tryb 32-bit
- wrzuca na stos:
 - adres tablicy nagłówków a.out (32)
 - rozmiar i adres boot parameters (32)
 - adres segmentu kodu monitora (16)
 - adres powrotu do monitora (16)
- skacze do kodu obrazu (etykieta MINIX)

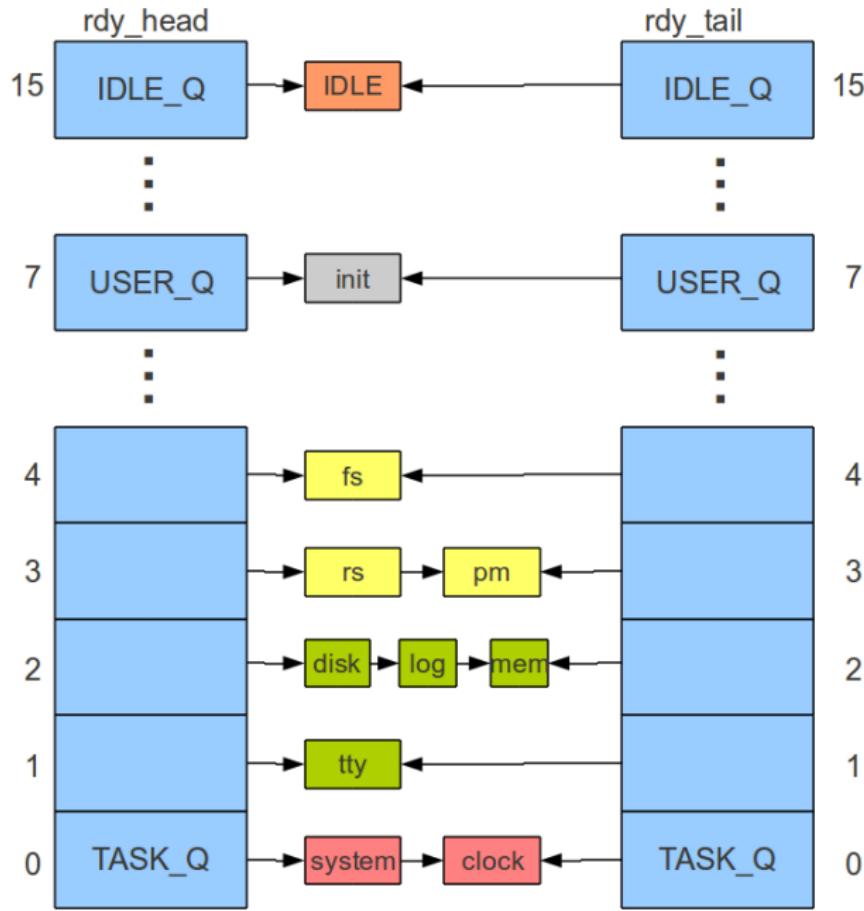
- przenosi GDT (Global Descriptor Table) monitora do przestrzeni adresowej jądra
- przełącza stos do przestrzeni jądra
- woła `cstart`
 - inicjalizuje GDT (segmenty dla jądra, LDT's dla procesów, TSS)
 - inicjalizuje IDT (Interrupt Desctiptor Table)
 - ustawia zmienne w jądrze zgonie z boot parameters (np. bus, video)
- ładuje GDT, IDT
- uaktywnia GDT
- skacze do `main`

```
        jmpf CS_SELECTOR:csinit
csinit:
    o16 mov ax, DS_SELECTOR
    ...

```

```
PUBLIC void main()
{
/* Start the ball rolling. */
```

- Inicjalizacja wektora przerwań.
- Inicjalizacja tablicy procesów i ppriv (na podstawie tablicy image) w tym
 - inicjalizacja LDT dla procesów
 - priorytety i kwanty dla procesów
 - początkowe wartości rejestrów
 - kolejkuje procesy w strukturach schedulera
- restart()



Pierwsze uruchomienie procesów

Kernel

- System Task
- Clock
 - zaprogramowanie zegara (60 przerwań na sek)
 - rejestracja obsługi przerwania zegarowego (`clock_handler`)
 - włączenie przerwania zegarowego

Drivery

- Inicjalizacja urządzeń (kontrolerów).
- Rejestracja obsługi i włączenie przerwań.

Reincarnation Server

Rejestracja obsługi sygnałów SIG_MESS dla SIGCHLD, SIGTERM, SIGABRT, SIGHUP.

Process Manager

- Wczytuje z prz. adr. jądra:
 - boot parameters
 - mapowanie pamięci zadań jądra
 - tablicę image
- Inicjalizuje tablicę procesów
 - Inicjalizuje wiersz na podstawie image
 - Przypisuje procesowi PID.
 - Wczytuje z jądra mapę pamięci dla procesu.
 - Wysyła do FS wiadomość (PROC_NR, PID).
- Wysyła (sendrec) do FS wiadomość (NONE, _).
- Inicjalizuje listę wolnych obszarów pamięci.

File System

- Inicjalizacja tabeli procesów
 - uzupełnianie wierszy dla każdej wiadomości od PM aż do NONE
 - potwierdzenie dla PM odczytania tabeli procesów
- Inicjalizacja struktur cache'u.
- Inicjalizacja mapowania urządzenie-driver.
- Montowanie katalogu root.
- Inicjalizacja katalogów roboczych procesów.

Init

- Rejestracja obsługi sygnałów (SIGHUP, SIGTERM, SIGABRT).
- Uruchomienie skryptu /etc/rc. W nim m.in.
 - Montowanie usr i home.
 - Startowanie dodatkowych driverów i serverów (np. lance i inet).
- Dla każdego terminala uruchomienie odpowiedniego procesu (np. gettys dla konsoli).

0652c00	0659400	26352	4996	44192	1024	tty
0665c00	0667000	4784	764	3012	4096	memory
0669000	066a800	5904	504	63276	4096	log
067b400	0681400	23680	10776	10932	8192	AT:at_wini
0688c00	068a800	7088	2284	1356	768	init

MINIX 3.1.0. Copyright 2006, Vrije Universiteit, Amsterdam, The Netherlands
 Executing in 32-bit protected mode.

Building process table: pm fs rs tty memory log driver init.

Physical memory: total 261706 KB, system 5732 KB, free 255974 KB.

Boot medium driver: AT driver mapped onto controller c0.

Multiuser startup in progress ...: floppy is cmos.

/dev/c0d0p0s2 is read-write mounted on /usr

/dev/c0d0p0s1 is read-write mounted on /home

Starting services: lance random.

Starting daemons: update usyslogd cron.

Minix Release 3 Version 1.0 (console)

192.168.56.3 login: