Systemy Operacyjne.

Jakub Kozik

semestr zimowy 2016/2017

Informatyka Analityczna
tcs@jagiellonian

Zasady

“This is not Nam. [..] There are rules.”

Zaliczenie i egzamin

- 2 duże zadania + obrony projektów (2 x 30 punktów),
- 2 małe zadania (2 x 10 punktów),
- aktywność na ćwiczeniach (po 1 punkcie na każde ćwiczenia)
- egzamin pisemny (20 punktów).

Za każde z dużych zadań oraz za egzamin trzeba uzyskać przynajmniej 50% możliwych punktów.

Przeliczenie punktów na ocenę:

50-60 3.0; 60-70 3.5; 70-80 4.0; 80-90 4.5; 90-100 5.0

Zaliczenie w II-gim terminie

Po terminie oddania maksymalna liczba punktów za każde z zadań spada liniowo do 50% w ciągu dwóch tygodni.

Program

Program Wykładu

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

Zagadnienia:

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

Literatura

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 1003.1$^{\text{TM}}$-2008 - The Open Group Technical Standard Base Specifications, Issue 7.

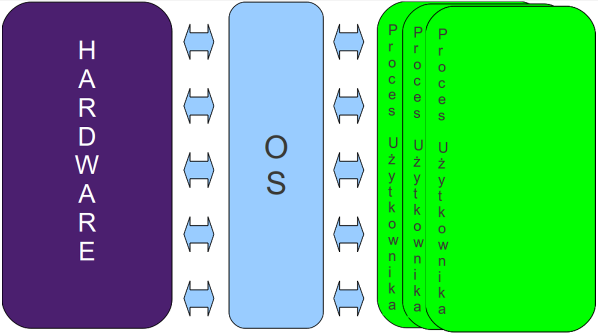
Outline

1. Zasady
2. POSIX
   - Wstęp
   - POSIX - standard
   - POSIX - procesy
   - POSIX - pliki
   - POSIX - sygnały
   - POSIX - remanent
3. Sequentiall Processes
   - Multiprogramming
   - IPC - InterProcess Communication

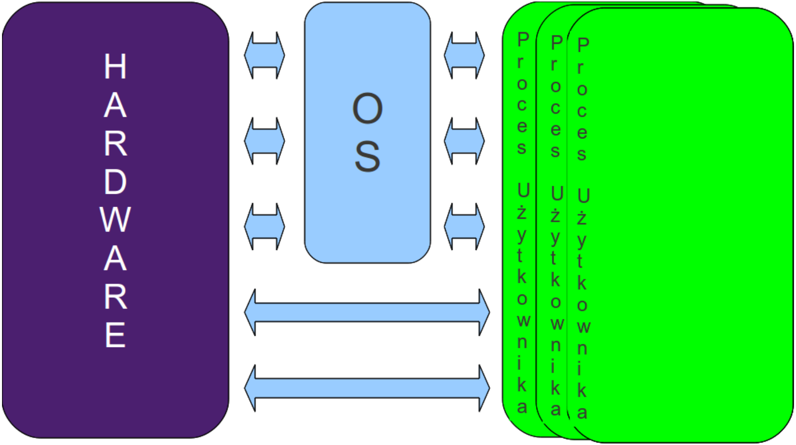
System Operacyjny

Główne funkcje systemu.

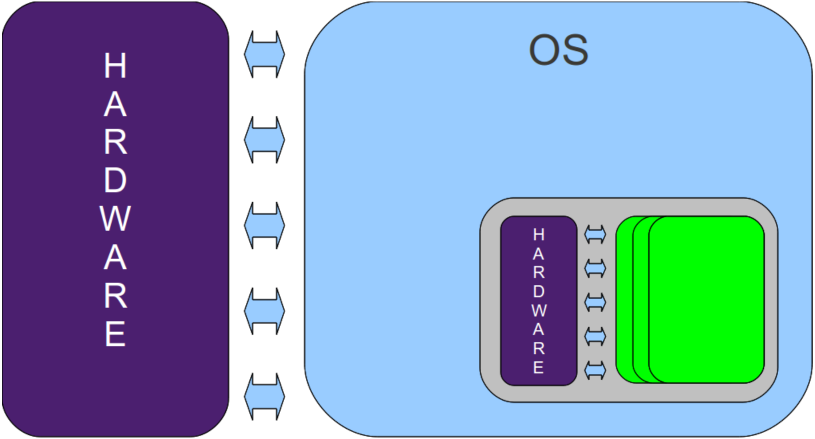
- Extended Machine
- Resource Management



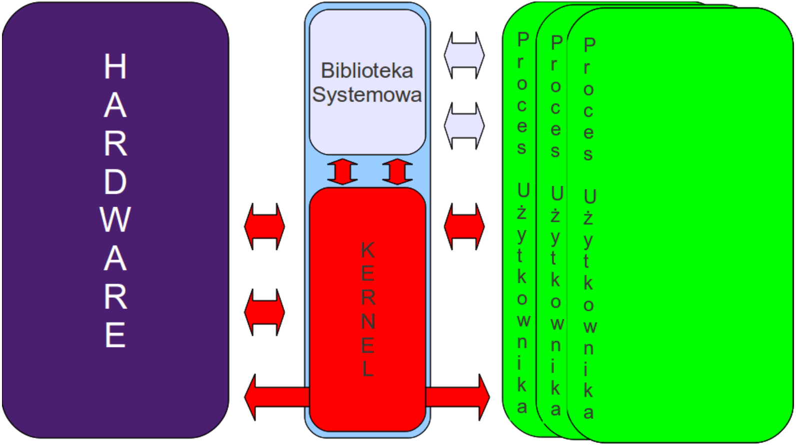
Bez zarządzania zasobami.



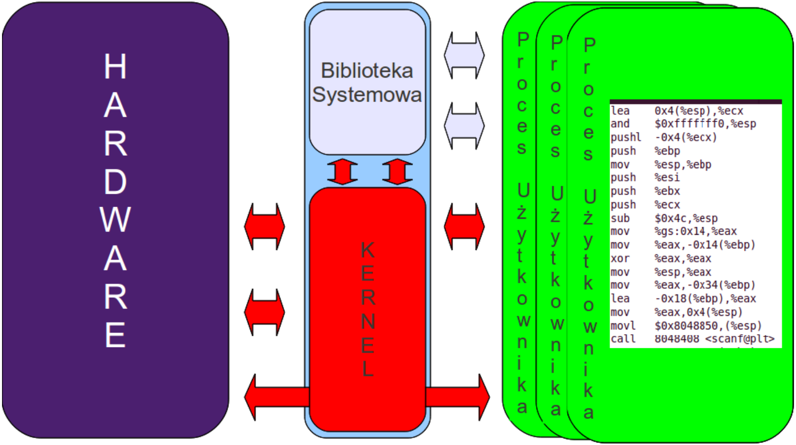
Wirtualna maszyna.



“Złoty środek.”

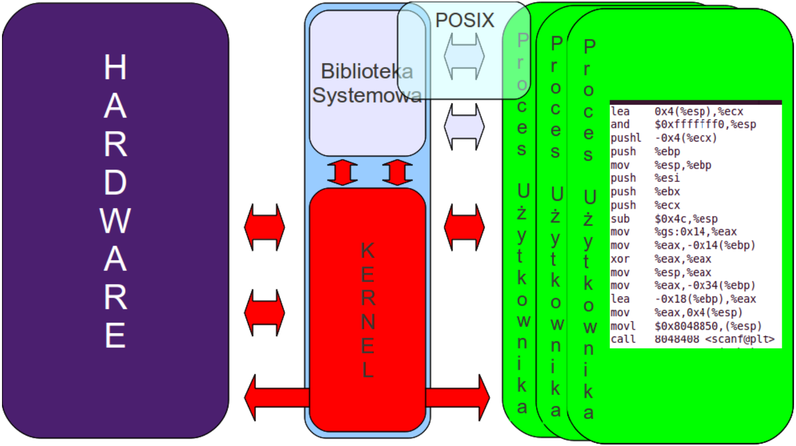


System calls - wywołania systemowe.



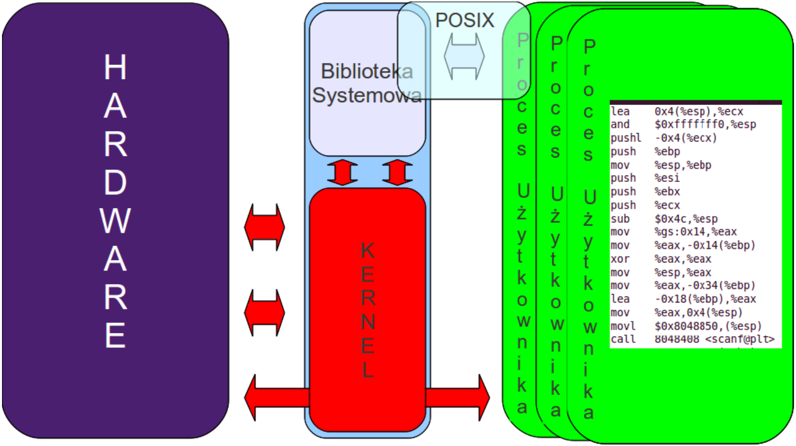
```
lea    0x4(%esp),%ecx
and    $0xfffffff0,%esp
pushl  -0x4(%ecx)
push   %ebp
mov    %esp,%ebp
push   %esi
push   %ebx
push   %ecx
sub    $0x4c,%esp
mov    %gs:0x14,%eax
mov    %eax,-0x14(%ebp)
xor    %eax,%eax
mov    %esp,%eax
mov    %eax,-0x34(%ebp)
lea    -0x18(%ebp),%eax
mov    %eax,0x4(%esp)
movl   $0x8048850,(%esp)
call   8048408 <scanf@plt>
```

POSIX



```
lea    0x4(%esp),%ecx
and    $0xfffffff0,%esp
pushl  -0x4(%ecx)
push   %ebp
mov    %esp,%ebp
push   %esi
push   %ebx
push   %ecx
sub    $0x4c,%esp
mov    %gs:0x14,%eax
mov    %eax,-0x14(%ebp)
xor    %eax,%eax
mov    %esp,%eax
mov    %eax,-0x34(%ebp)
lea    -0x18(%ebp),%eax
mov    %eax,0x4(%esp)
movl   $0x8048850,(%esp)
call   8048408 <scanf@plt>
```

POSIX programming.



```
lea     0x4(%esp),%ecx
and     $0xfffffff0,%esp
pushl   -0x4(%ecx)
push    %ebp
mov     %esp,%ebp
push    %esi
push    %ebx
push    %ecx
sub     $0x4c,%esp
mov     %gs:0x14,%eax
mov     %eax,-0x14(%ebp)
xor     %eax,%eax
mov     %esp,%eax
mov     %eax,-0x34(%ebp)
lea     -0x18(%ebp),%eax
mov     %eax,0x4(%esp)
movl    $0x8048850,(%esp)
call    8048408 <scanf@plt>
```

Outline

1. Zasady

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

3. Sequentiall Processes
   - Multiprogramming
   - IPC - InterProcess Communication

POSIX

Portable Operating System Interface

"**POSIX.1-2008** is simultaneously IEEE Std 1003.1^{TM}-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. Zasada

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

3. Sequentiall Processes
   - Multiprogramming
   - IPC - InterProcess Communication

POSIX - Procesy

Proces

Program w trakcie wykonywania.

Procesy - system calls

```c
#include <stdio.h>

int
main(int argc, char *argv[])
{
  printf("Hey, you sass that hoopy Ford Prefect?\n");
}
```

Procesy - system calls

```c
#include <stdio.h>

int
main(int argc, char *argv[])
{
  printf("Hey,␣you␣sass␣that␣hoopy␣Ford␣Prefect?\n");
}
```

exit

```c
#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(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

```
#include <stdio.h>

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

  for (i=0;i<10;i++) {
    printf("%s\n",argv[1]);
    sleep(1);
  }
}
```

tictac.c

```
#include <unistd.h>

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

  if (fork()) str = "tic";
  else str = "tac";

  execl("tic","tic",str,NULL);
}
```

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

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#define BSIZE 100

int main(){
  char str[BSIZE];
  pid_t chld_pid;

  while (fgets(str,BSIZE,stdin)){
    chld_pid = fork();
    if (!chld_pid){
      execlp("echo","echo",str,NULL);
      exit(1);
    } else
      waitpid(chld_pid,NULL,0);
  }
}
```

waitpid – exit(1) ???

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#define BSIZE 100

int main(){
  char str[BSIZE];
  pid_t chld_pid;

  while (fgets(str,BSIZE,stdin)){
    chld_pid = fork();
    if (!chld_pid){
      execlp("echo","echo",str,NULL);
      exit(1);
    } else
      waitpid(chld_pid,NULL,0);
  }
}
```

Outline

1. Zasady
2. POSIX
   - Wstęp
   - POSIX - standard
   - POSIX - procesy
   - POSIX - pliki
   - POSIX - sygnały
   - POSIX - remanent
3. Sequentiall Processes
   - Multiprogramming
   - IPC - InterProcess Communication

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 (zwraca deskryptor dla otwartego pliku)

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

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

```
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)
```

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

int main(int argc, char* argv[]){
  int fd=open("foo", O_RDWR|O_CREAT|O_TRUNC, S_IRUSR|S_IWUSR);

  lseek(fd,10000000000L, SEEK_CUR); /*~10GB*/
  write(fd, "a", 1);
  close(fd);
}
```

pipe

```
#include <unistd.h>

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

```
#include <stdio.h>

int main(int argc, char* argv[]){
  int fd[2];

  if (pipe(fd) != 0) return 1;
  if (fork()){
    write(fd[1],"say something",13);
  }else{
    char buf[21];
    int n;
    if (n = read(fd[0], buf,20) >= 4){
      buf[n] = 0;
      printf("%s\n",buf+4);
    }
  }
  return 0;
}
```

Named pipe - FIFO

mkfifo & mknode

```
#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

```c
#include <sys/stat.h>
#include <string.h>
#include <stdio.h>
int main(){
  int fd[2],n;
  char buf[4];
  pipe(fd);
  if (!fork()) {
    while ((n = read(fd[0], buf, 3))>0){
      buf[n] = 0;
      printf("%s\n", buf);
    }
  } else {
    sleep(1);
    if (fork()) strcpy(buf, "tic");
    else strcpy(buf, "tac");

    for (n=0; n<10; n++){
      write(fd[1], buf, 3);
      sleep(1);
    }
  }
}
```

fcntl - file descriptor control functions

```
#include <fcntl.h>

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

fcntl(fd, F_DUPFD, int fd2)

```c
#include <fcntl.h>
#include <unistd.h>

int main(){
  int fd[2];
  pipe(fd);
  if (!fork()) {
    close(0);
    close(fd[1]);
    fcntl(fd[0], F_DUPFD,0);
    execlp("cat", "cat", NULL);
  } else {
    write(fd[1], "say hello\n", 10);
    close(fd[1]);
    wait(NULL);
  }
}
```

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

```c
#include <fcntl.h>
#include <unistd.h>

int main(){
  int fd[2];
  pipe(fd);

  int flags = fcntl(fd[1],F_GETFD);
  flags |= FD_CLOEXEC;
  fcntl(fd[1],F_SETFD,flags);

  if (!fork()) {
    close(0);
/*    close(fd[1]); */
    fcntl(fd[0],F_DUPFD,0);
    execlp("cat","cat",NULL);
  } else {
    write(fd[1],"say hello\n",10);
    close(fd[1]);
  }
}
```

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 descrip-
          tor 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.
```

```c
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
int main(){
  int fd[2],n;
  pipe(fd);
  int flags=fcntl(fd[0], F_GETFL);
  fcntl(fd[0], F_SETFL, flags | O_NONBLOCK);
  if (!fork()) {
    char buf[20];
    close(fd[1]);
    while ((n=read(fd[0],buf,20))!=0){
      if (n>0) write(0,buf,n);
      else if (errno!=EAGAIN) return 1;
      else write(0,"still␣nothing\n",14);
      sleep(1);
    }
  } else
  for (n=0; n<5; n++){
    sleep(3);
    write(fd[1],"I␣am␣a␣walrus.\n",16);
  }
}
```

O_NONBLOCK for open

```c
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <sys/stat.h>
#include <stdio.h>

int main(){
  int fd,n; char buf[20];
  mkfifo("mfifo", S_IWUSR | S_IRUSR);

  if (!fork()) {
    fd=open("mfifo", O_RDONLY|O_NONBLOCK);
    write(1, "opened\n", 7);
    sleep(10);
    while ((n=read(fd, buf, 20))>0)
      write(1, buf, n);
  } else{
    sleep(5);
    fd=open("mfifo", O_WRONLY);
    write(fd, "hello\n", 6);
    write(1, "done\n", 5);
  }
}
```

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 over-
     laps 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 */
};
```

```c
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>
int main(int argc, char * argv[]){
  int fd;
  struct flock fl;

  fd= open("lock", O_CREAT | O_RDWR, S_IWUSR | S_IRUSR );
  /*...*/
  fl.l_type = F_WRLCK;
  fl.l_whence = SEEK_SET;
  fl.l_start = 0;
  fl.l_len = 3;

  fcntl(fd,F_SETLKW, &fl);
  lseek(fd,0,SEEK_SET);
  write(fd,argv[1],3);
  sleep(30);
  fl.l_type = F_UNLCK;
  fcntl(fd,F_SETLK, &fl);
  /*...*/
  close(fd);
}
```

Outline

1. Zasady
2. POSIX
   - Wstęp
   - POSIX - standard
   - POSIX - procesy
   - POSIX - pliki
   - POSIX - sygnały
   - POSIX - remanent
3. Sequentiall Processes
   - Multiprogramming
   - IPC - InterProcess Communication

Sygnały

Sygnał

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.

```
#include <stdio.h>
#include <signal.h>

void handler(int sig_nb){
  write(1,"If␣everything␣seems␣under␣control,␣\
you're␣just␣not␣going␣fast␣enough.\n",70);
  sleep(1);
  signal(SIGINT,handler);
}

int main(){
  signal(SIGINT,handler);

  while (1)
    pause();
}
```

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.

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

volatile int ready = 0;
void handler(int sig_nb){
  ready = 1;
}

int main(){
  pid_t other;
  char* str="tic\n";
  struct sigaction act;

  act.sa_handler= handler;
  act.sa_flags = 0;
  sigemptyset(&act.sa_mask);
  sigaction(SIGUSR1,&act,NULL);

  if (!(other=fork())){
    str = "tac\n";
    other= getppid();
  } else ready = 1;

  while (1) {
    if (ready){
      ready = 0;
      sleep(1);
      write(1,str,4);
      kill(other,SIGUSR1);
    }
    pause();
  }
}
```

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.

Syscalle 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.

```c
#include <unistd.h>
#include <sys/types.h>
#include <signal.h>
#include <errno.h>

#define BSIZE 100
void handler(int sig_nb){}

int main(){
  int n,k,w;
  char buf[BSIZE];
  pid_t parent, child;
  struct sigaction act;

  act.sa_handler = handler;
  act.sa_flags = 0;
  sigemptyset(&act.sa_mask);
  sigaction(SIGUSR1, &act, NULL);

  if (!(child = fork())){
    parent = getppid();
    while (1) kill(parent, SIGUSR1);
    exit(1);
  }

  while (n = read(0, buf, BSIZE)){
    if ((n<0) && (errno!=EINTR)) break;
    k = 0;
    while(k<n){
      w = write(1, buf+k, n-k);
      if ((w<0) && (errno!=EINTR)) goto end;
      if (w>0) k+=w;
    }
  }
end:  kill(child, SIGTERM);
}
```

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)`

```c
#include <unistd.h>
#include <sys/types.h>
#include <signal.h>
#include <errno.h>

#define BSIZE 100
void handler(int sig_nb){}

int main(){
  int n,k,w;
  char buf[BSIZE];
  pid_t parent, child;
  struct sigaction act;

  act.sa_handler = handler;
  act.sa_flags = SA_RESTART;
  sigemptyset(&act.sa_mask);
  sigaction(SIGUSR1, &act, NULL);

  if (!(child = fork())){
    parent = getppid();
    while (1) kill(parent, SIGUSR1);
    exit(1);
  }

  while ((n = read(0, buf, BSIZE))>0){
    k = 0;
    while(k<n){
      w = write(1, buf+k, n-k);
      if (w<0) goto end;
      k += w;
    }
  }
end:  kill(child, SIGTERM);
}
```

UWAGA na errno!

```c
#include <unistd.h>
#include <sys/types.h>
#include <signal.h>
#include <errno.h>

#define BSIZE 100
void handler(int sig_nb){ errno = 0;}

int main(){
  int n,k,w;
  char buf[BSIZE];
  pid_t parent, child;
  struct sigaction act;

  act.sa_handler= handler;
  act.sa_flags=0;
  sigemptyset(&act.sa_mask);
  sigaction(SIGUSR1, &act, NULL);

  if (!(child = fork())){
    parent = getppid();
    while (1) kill(parent,SIGUSR1);
    exit(1);
  }

  while (n = read(0,buf, BSIZE)){
    if ((n<0) && (errno != EINTR)) break; else n = 0;
    write(1, buf, n);   // nie dbamy o przerwane write'y
  }
end:  kill(child, SIGTERM);
}
```

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!

```c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <signal.h>

volatile int s=0;
void handler(int sig_nb){ s++; }

int main(){
  int n,k,w;
  pid_t parent;
  struct sigaction act;

  act.sa_handler = handler;
  act.sa_flags = 0;
  sigemptyset(&act.sa_mask);
  sigaction(SIGUSR1, &act, NULL);

  if (!fork()){
    parent= getppid();
    for (n=0; n<10; n++) kill(parent,SIGUSR1);
    exit(0);
  }

  pause();
  while (s-- >0) {
    printf("received\n");
    sleep(3);
  }
}
```

```c
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>

#define CHILDREN 10
volatile int z=CHILDREN;
void handler(int sig_nb){
  pid_t child;
  do{
    child = waitpid(-1,NULL,WNOHANG);
    if (child>0) z--;
  } while (child>0);
  sleep(1);
}

int main(){
  int n;
  struct sigaction act;
  act.sa_handler = handler;
  act.sa_flags = 0;
  sigemptyset(&act.sa_mask);
  sigaction(SIGCHLD,&act,NULL);

  for (n=0;n<CHILDREN;n++)
    if (!fork()) {
      sleep(n);
      return 0;
    }

  while (z>0) {
    printf("%d␣children/zombies␣left.\n",z);
    sleep(1);
  }
  printf("No␣more␣zombies.\n");
}
```

SIG_IGN dla SIGCHLD

```c
#include <unistd.h>
#include <signal.h>

#define CHILDREN 10

int main(){
  int n;
  struct sigaction act;
  act.sa_handler = SIG_IGN;
  act.sa_flags = 0;
  sigemptyset(&act.sa_mask);
  sigaction(SIGCHLD,&act,NULL);

  for (n=0;n<CHILDREN;n++)
    if (!fork()) {
      return 0;
    }
  sleep(10);
}
```

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?

```
#include <unistd.h>
#include <stdio.h>
#include <signal.h>

#define TIME 5
void handler(int sig_nb){ }

int main(){
  int n;
  struct sigaction act;

  act.sa_handler = handler;
  act.sa_flags = 0;
  sigemptyset(&act.sa_mask);
  sigaction(SIGALRM, &act,NULL);

  alarm(TIME);
  pause();
  printf("No time ... no time to lose.\n");
}
```

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.

```c
#include <unistd.h>
#include <stdio.h>
#include <signal.h>

#define TIME 5
void handler(int sig_nb){ }

int main(){
  int n;
  struct sigaction act;
  sigset_t mask;


  act.sa_handler = handler;
  act.sa_flags = 0;
  sigemptyset(&act.sa_mask);
  sigaction(SIGALRM, &act, NULL);
  sigaction(SIGINT, &act, NULL);

  sigfillset(&mask);
  sigdelset(&mask, SIGALRM);

  sigprocmask(SIG_SETMASK, &mask, NULL);
  alarm(TIME);
  pause();
  printf("No time ... no time to lose.\n");
}
```

sigsuspend

```
#include <signal.h>

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

```
#include <unistd.h>
#include <stdio.h>
#include <signal.h>
#define TIME 5
void handler(int sig_nb){ }

int main(){
  struct sigaction act;
  sigset_t mask;

  act.sa_handler = handler;
  act.sa_flags = 0;
  sigemptyset(&act.sa_mask);
  sigaction(SIGALRM, &act, NULL);
  sigaction(SIGINT, &act, NULL);

  sigemptyset(&mask);
  sigaddset(&mask, SIGALRM);
  sigprocmask(SIG_BLOCK, &mask, NULL);

  sigfillset(&mask);
  sigdelset(&mask, SIGALRM);
  alarm(TIME);
  sigsuspend(&mask);
  printf("No time ... no time to lose.\n");
}
```

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

```c
#include <unistd.h>
#include <stdio.h>
#include <signal.h>

volatile int time_is_up;
void handler(int sig_nb){ time_is_up = 1;}

int tread(char * buf, int n, int timeout){
  int r;
  struct sigaction act, oact;
  sigset_t mask, omask;

  act.sa_handler = handler;
  act.sa_flags = 0;
  sigemptyset(&act.sa_mask);
  sigaction(SIGALRM,&act,&oact);

  time_is_up = 0;
  alarm(timeout);
  do r= read(0,buf,n);
  while ((!time_is_up) && (r<0));
  alarm(0);

  sigaction(SIGALRM,&oact,NULL);
  if (r<0) return 0;
  return r;
}
int main(){
  char buf[100];
  int n = tread(buf,100,5);
  write(1,buf,n);
}
```

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

```c
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <sys/select.h>
#include <sys/time.h>

int tread(char * buf, int n, int timeout){
  int r;
  fd_set rfds;
  struct timeval timeout_s;

  FD_ZERO(&rfds);
  FD_SET(0,&rfds);

  timeout_s.tv_sec = timeout;
  timeout_s.tv_usec = 0;

  do r= select(1,&rfds,NULL,NULL,&timeout_s);
  while ((r<0) && (errno==EINTR));

  if (r<=0) return 0;

  return (read(0,buf,n));
}
int main(){
  char buf[100];
  int n = tread(buf,100,5);
  write(1,buf,n);
}
```

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. Sequentiall Processes
   - Multiprogramming
   - IPC - InterProcess Communication

Remanent

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
```

Remanent

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
```

Remanent

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
```

Remanent

Inne

```
chroot  - change root directory
ptrace  - process trace
reboot  - close down the system or reboot
mmap    - request memory mapping
```

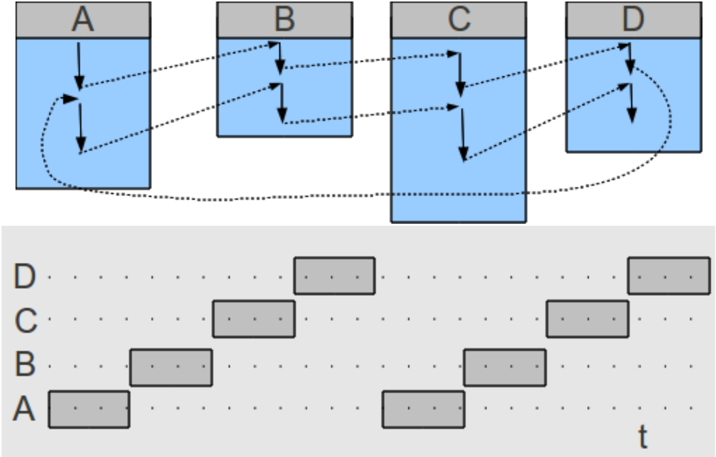
Outline

1. Zasada
2. POSIX
   - Wstęp
   - POSIX - standard
   - POSIX - procesy
   - POSIX - pliki
   - POSIX - sygnały
   - POSIX - remanent
3. Sequentiall Processes
   - Multiprogramming
   - IPC - InterProcess Communication

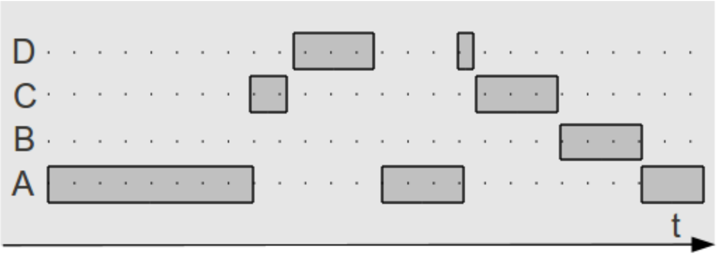
Multiprogramming (multitasking)

“Wirtualne procesory”

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

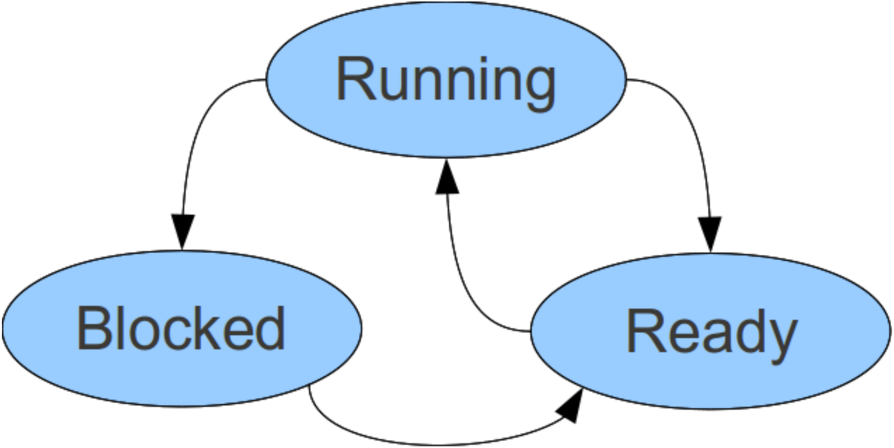


Multiprogramming (multitasking)



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
- Curent scheduling priority
- Maximum scheduling priority
- Scheduling ticks left
- Quantum size
- CPU time used
- Message queue pointers
- Pending signal bits
- Various flag bits
- Process name

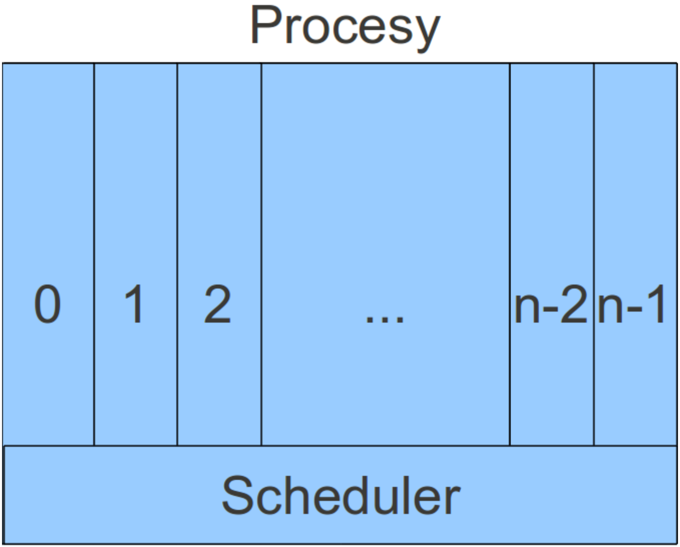
Process management

- 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
- Rreal GID
- Effective GID
- Controlling tty
- Save area for read/write
- System call parameters
- Various flag bits

Scheduler



Procesy

0

1

2

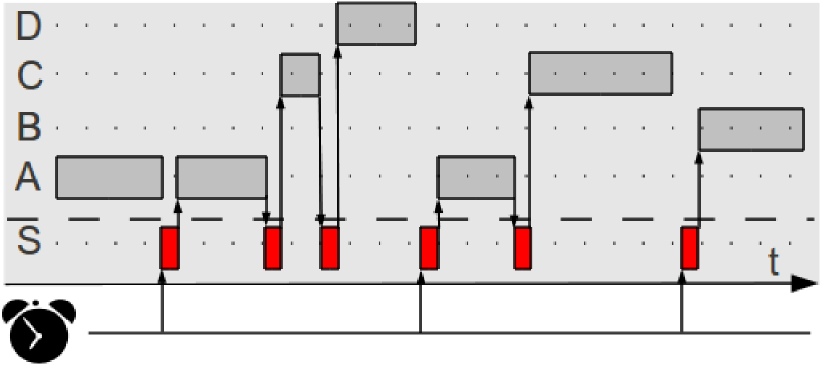
...

n-2

n-1

Scheduler

Przerwania zegarowe.



MINIX - 60 przerwań na sekundę.

Pamięć - zmiany kontekstu

Kernel

- **Registers**
- **Program counter**
- **Program status word**
- **Stack Pointer**
- **Process state**
- Curent 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
- Rreal 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

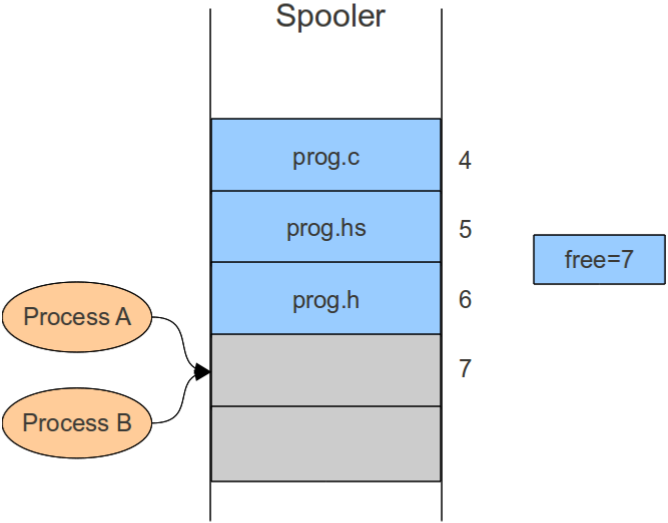
1. Zasady

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

3. Sequentiall Processes
   - Multiprogramming
   - IPC - InterProcess Communication

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,
- żeden proces nie powinien czekać w nieskończoność na wejście do sekcji krytycznej,
- mechanim 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?

Rozwiązanie 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);
    }
```

Semafory

```
#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);
    }
}
```

Semafory - 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);
    }
}
```

Monitory

```
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 - randezvous

```
#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);
    }
}
```