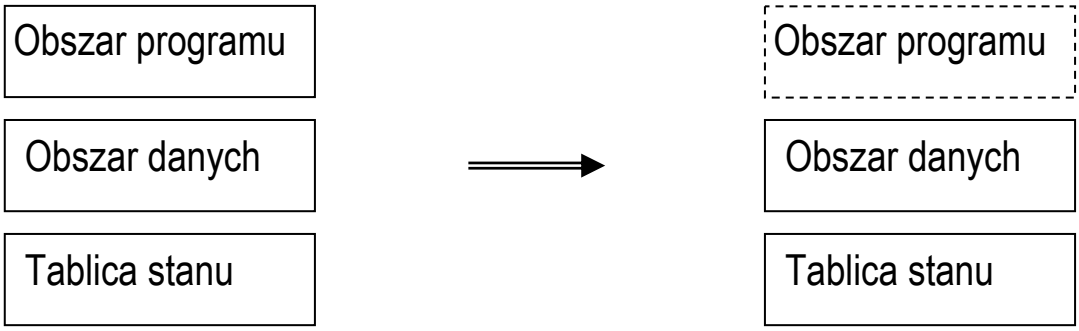
POSIX: IEEE Std 1003.1 – 2001
(Issue 6, 2004 edition)

- **Podstawowe rekomendacje przejęte z UNIXa**
  - wielodostęp
  - wielozadaniowość
  - system plików
  - terminal
  - gniazda
- **Rekomendacje dla obszaru czasu rzeczywistego**
  - strategie szeregowania
  - rozszerzony mechanizm sygnałów
  - synchronizacja i komunikacja zadań
  - kontrola zarządzania pamięcią
  - wspólne obszary pamięci
  - programowe liczniki czasu
  - asynchroniczne operacje wejścia-wyjścia

Tworzenie i usuwanie procesów

- **Utworzenie** kopii procesu

  pid_t fork( )



- **Wymiana** ciała procesu

  int exec-- (plik, argumenty, środowisko)

- **Przykład**

```
.....
pid=fork();
if ( p!=0 ) {
   // ... proces macierzysty
} else {
   exec(new,..,..);
   // ... nowy proces potomny
}
.....
```

- **Utworzenie** nowego procesu

→POSIX_SPAWN

*int posix_spawn (pid_t *pid, char *path,*
*posix_spawn_file_actions_t *actions,*
*posix_spawnattr_t *attr,*
*char *argv[ ], char *envp[ ]);*

- **Zakończenie** procesu

void exit (int status)

- **Oczekiwanie** na zakończenie procesu potomnego

*pid_t wait (int *status);*
*pid_t waitpid (pid_t pid, int *status, int options);*

Tworzenie i usuwanie wątków

→POSIX_THREADS

- **Utworzenie** wątku

  *int pthread_create (pthread_t *thread,*
  *const pthread_attr_t *attr,*
  *void *(*body)(void *),*
  *void *arg)*

- **Zakończenie** wątku

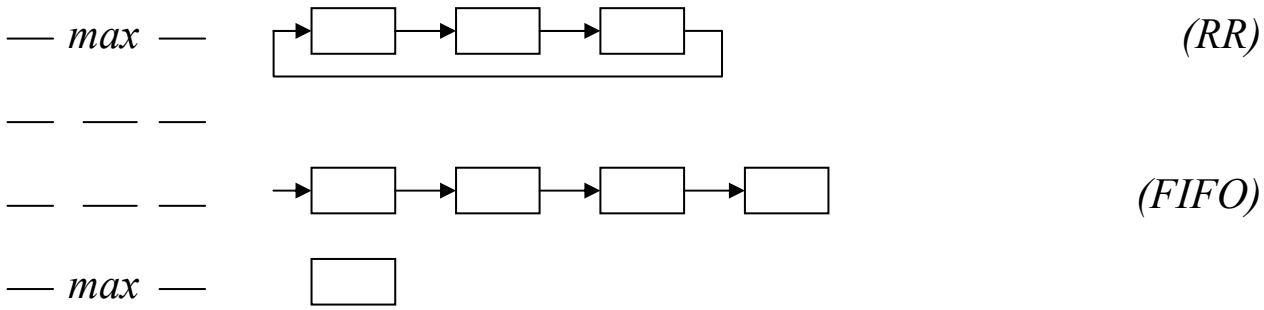
  *void pthread_exit (void *value)*

- **Oczekiwanie** na zakończenie wątku

  *int pthread_join (pthread_t thread, void **value)*

Algorytmy szeregowania

- **Poziomy priorytetu**
- **Wywłaszczanie zadań niższych poziomów**



- **Strategie szeregowania**
  - algorytm karuzelowy (*SCHED_RR*)
  - algorytm kolejkowy (*SCHED_FIFO*)
  - serwer sporadyczny (*SCHED_SPORADIC*)
  - algorytm dowolny (*SCHED_OTHER*)
- **Zwolnienie procesora**

  int sched_yield ( );

- **Ustalanie strategii szeregowania (*policy*) i priorytetu**

Procesy

*int sched_setparam (pid_t pid, struct sched_param *param)*

int sched_setscheduler (pid_t pid, int policy,
*struct sched_param *param)*

Wątki

int pthread_setschedprio (pthread_t thread, int prio)

int pthread_setschedparam (pthread_t thread, int policy,
*struct sched_param *param)*

contentionscope

PTHREAD_SCOPE_PROCESS

PTHREAD_SCOPE_SYSTEM

Sygnały

- **Generacja sygnału**

  system operacyjny

  int kill (pid_t pid, int sig)

  int raise (int sig)

  int pthread_kill (pthread_t thread, int sig)

- **Obsługa sygnałów**

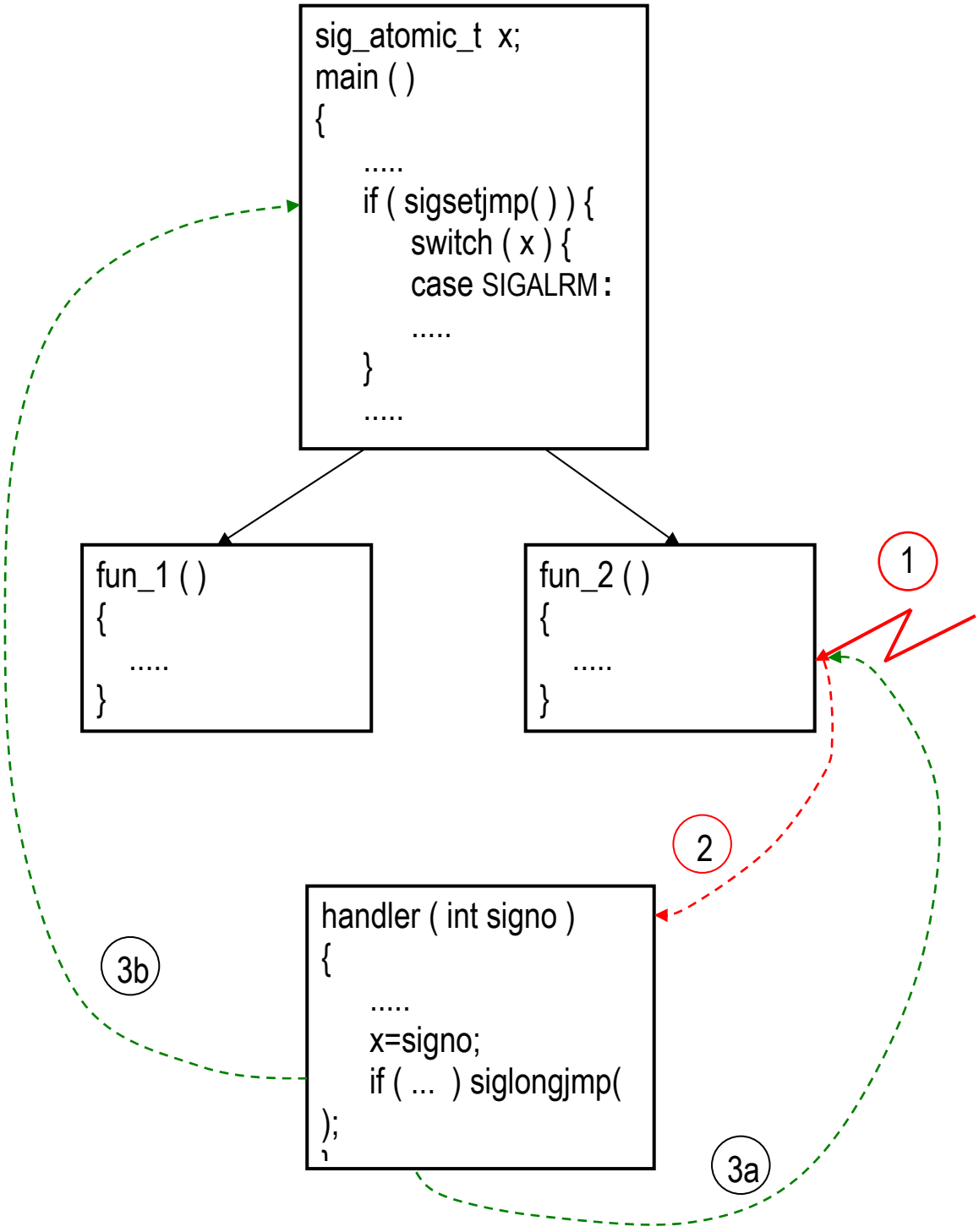
  *void (*signal (int sig, void (*func) (int))) (int)*

  - domyślnie (*SIG_DFL*)
  - ignorowanie (*SIG_IGN*)
  - przechwycenie

    ```
    void handler ( int signo )
    {
      ......
    }
    ```

  sig_atomic_t

  sigsetjmp ( ), siglongjmp ( )


sig_atomic_t x;
main ( )
{
.....
if (sigsetjmp( )) {
switch (x) {
case SIGALRM:
.....
}
.....
fun_1 ( )
{
.....
}
fun_2 ( )
{
.....
}
1
2
handler (int signo)
{
.....
x=signo;
if (...) siglongjmp(
);
3a
3b

- **Maskowanie sygnałów**

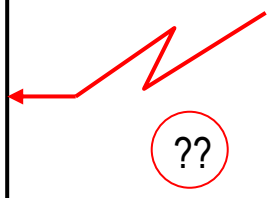
sigemptyset ( ), sigfillset ( ), sigaddset ( ), sigdelset ( )
sigprocmask ( )

pthread_sigmask ( )

sigismember ( ), sigpending ( )

sigsuspend ( )
sigwait ( ), sigwaitinfo ( ), sigtimedwait ( )

```
fun ( )
{
      .....
      signal ( SIGUSR1 , &handler );
      sigprocmask ( ... );
      pause ( )
      .....
}
```




```
handler ( )
{ }
```

- **Rodzaje sygnałów (28)**

si_signo	si_code	default	
SIGBUS	rodzaj błędu	Z	Błędny adres fizyczny na magistrali
SIGILL	rodzaj błędu	Z	Błędna instrukcja
SIGFPE	rodzaj błędu	Z	Błąd operacji arytmetycznej.
SIGSEGV	rodzaj błędu	Z	Naruszenie ochrony pamięci
SIGABRT	↑	Z	Nienormalne zakończenie procesu (*abort*)
SIGALRM		Z	Upływ czasu (*alarm clock*)
SIGKILL		Z	Usunięcie procesu – bez przechwycenia
SIGTERM		Z	Normalne zakończenie procesu (*exit*)
SIGHUP	SI_USER	Z	Zakończenie sesji terminala
SIGINT	SI_QUEUE	Z	Przerwanie z terminala
SIGQUIT	SI_TIMER	Z	Zakończenie pracy z terminala
SIGSTOP	SI_MESGQ	S	Zatrzymanie procesu – bez przechwycenia
SIGCONT	SI_ASYNCIO	C	Kontynuacja zatrzymanego procesu
SIGPIPE	↓	Z	Zapis do zamkniętego potoku
SIGSYS		Z	Błędne odwołanie do systemu
SIGCHLD		I	Zakończenie procesu potomnego
SIGUSR1		Z	User-defined signal 1.
SIGUSR2		Z	User-defined signal 2.

SIGRTMIN ... SIGRTMAX

- **Kolejkowanie sygnałów**

→POSIX_REALTIME_SIGNALS

int sigqueue (pid_t pid, int sig, union sigval value)

*int sigaction (int sig, struct sigaction *act,*
*struct sigaction *oact)*

struct sigaction

sa_flags
SA_SIGINFO
sa_handler
sa_sigaction

*void handler (int signo, siginfo_t *info, void *context)*

siginfo_t

int si_signo
int si_code
union sigval si_value

union sigval

int sival_int
*void *sival_ptr*

- **Przykład**

```
static sig_atomic_t  num, val, code;

void funh(int sig, siginfo_t *inf, void *con)
{
   num=sig;
   val=info->si_value.sival_int;
   code=info->si_code;
}

void main( )
{
   struct sigaction act;
   sigset_t         set;

   sigemptyset(&set);
   sigaddset(&set,SIGUSR1);
   sigaddset(&set,SIGUSR2);

   act.sa_flags=SA_SIGINFO;
   act.sa_mask=set
   act.sa_sigaction=&funh;
   sigaction(SIGUSR1,&act,NULL);

   pause();
   printf("signo=%d val=%d code=%d\n",
           num,val,code);
}
```

Zarządzanie pamięcią

- **Blokowanie stron w pamięci**

→ POSIX_MEMLOCK

int mlockall (int flags)

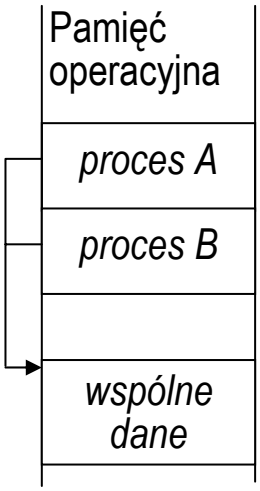
int munlockall ( )

→ POSIX_MEMLOCK_RANGE

*int mlock (void *addr, size_t len)*

*int munlock (void *addr, size_t len)*

- **Wspólne obszary danych**



- włączenie obszaru w przestrzeń adresową
- identyfikacja obszaru przez procesy
- ograniczenie rodzaju operacji
- trwałość obszaru

- **Odwzorowanie pliku w pamięci**

→ POSIX_MAPPED_FILES

*void *mmap (void *addr, size_t len, int prot,*
int flags, int file, off_t offset)

flags — MAP_SHARED, MAP_PRIVATE, MAP_FIXED

*int munmap (void *addr, size_t len)*

*int msync (void *addr, size_t len, int flags)*
flags — MS_ASYNC, MS_SYNC, MS_INVALIDATE

- **Dzielony obiekt pamięciowy**

→ POSIX_SHARED_MEMORY_OBJECTS

*int shm_open (char *name, int oflags, mode_t mode)*

*int shm_unlink (char *name)*

int close (int file)

int ftruncate (int file, off_t length)

Narzędzia synchronizacji i komunikacji zadań

- **Semafor**

→ POSIX_SEMAPHORES

*int sem_init (sem_t *sem, int shared, int value)*

*int sem_destroy (sem_t *sem)*

*int sem_wait (sem_t *sem)*

*int sem_post (sem_t *sem)*

*int sem_trywait (sem_t *sem)*

*int sem_timedwait (sem_t *sem,*
*struct timespec *abs_timeout)*

Semafor nazwany

*sem_t *sem_open (char *name, int oflag,*
mode_t mode, unsigned value)

oflags — O_CREAT, O_EXCL

*int sem_close (sem_t *sem)*

*int sem_unlink (char *name)*

Przykład

```
...
fd=shm_open(”semafor”,O_RDWR|O_CREAT,0777);
size=ltrunc(fd,sizeof(sem_t),SEEK_SET);
addr=mmap(0,sizeof(sem_t),PROT_READ|PROT_WRITE,
          MAP_SHARED,fd,0);
sem=(sem_t *)addr;
sem_init(sem,1,0);              // pshared
...
fork();
...
sem_wait(sem);
...
sem_destroy(sem);
munmap(addr,sizeof(sem_t);
close(fd);
shm_unlink(”semafor”);
...

/* drugi proces - bez wykorzystania fork() */
...
fd=shm_open(”semafor”,O_RDWR);
addr=mmap(0,sizeof(sem_t),PROT_READ|PROT_WRITE,
          MAP_SHARED,fd,0);
sem=(sem_t *)addr;
...
sem_post(sem)
...
munmap(addr,sizeof(sem_t);
close(fd);
...
```

- **Muteks**

Utworzenie

*int pthread_mutex_init (pthread_mutex_t *mutex,*
*pthread_mutexattr_t * attr)*

pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;

Usunięcie

*int pthread_mutex_destroy (pthread_mutex_t *mutex)*

Sekcja krytyczna

*int pthread_mutex_lock (pthread_mutex_t *mutex)*

.....

*int pthread_mutex_unlock (pthread_mutex_t *mutex)*

Inwersja priorytetów

protocol

PTHREAD_PRIO_NONE

PTHREAD_PRIO_INHERIT

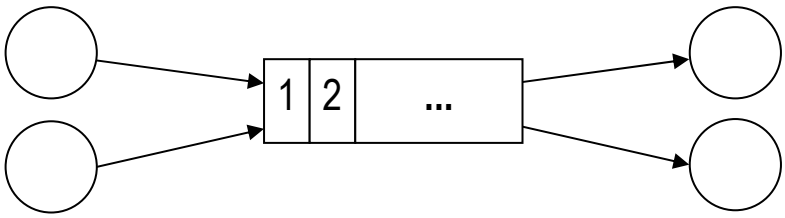
PTHREAD_PRIO_PROTECT

Ograniczenie oczekiwania

*int pthread_mutex_trylock (pthread_mutex_t *mutex)*

*int pthread_mutex_timedlock (pthread_mutex_t *mutex,*
*struct timespec *abs_timeout)*

- **Zmienna warunkowa**



Producent

```
.....
wpisz wiadomość ( w );
s=s+1;
.....
```

Konsument

```
.....
while ( s == 0 ) czekaj;
pobierz wiadomość ( p );
s=s−1;
.....
```

Producent

```
.....
mutex_lock ( m );
wpisz wiadomość ( w );
s=s+1;
mutex_unlock ( m );
.....
```

Konsument

```
.....
mutex_lock ( m );
while ( s == 0 ) czekaj;
s=s−1;
pobierz wiadomość ( p );
mutex_unlock ( m );
.....
```

Operacje

*int pthread_cond_init (pthread_cond_t *cond,*
*pthread_condattr_t * attr)*

*int pthread_cond_wait (pthread_cond_t *cond,*
*pthread_mutex_t *mutex)*

*int pthread_cond_signal (pthread_cond_t *cond)*

*int pthread_cond_timedwait (pthread_cond_t *cond,*
*pthread_mutex_t *mutex,*
*const struct timespec *abstime)*

*int pthread_cond_broadcast (pthread_cond_t *cond)*

Przykład

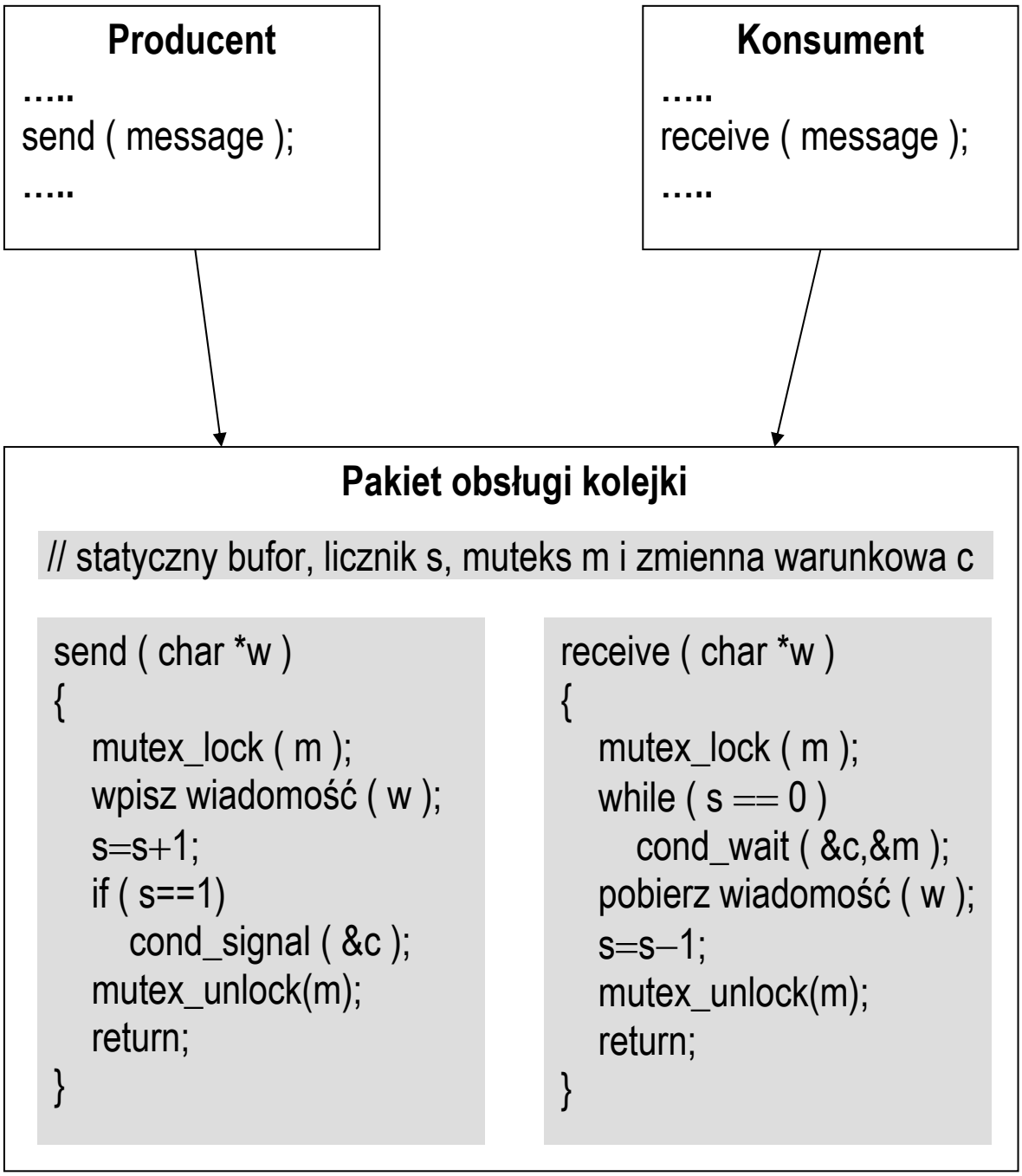
Producent

```
.....
mutex_lock ( m );
wpisz wiadomość ( w );
s=s+1;
if ( s==1)
   cond_signal ( &c );
mutex_unlock(m);
.....
```

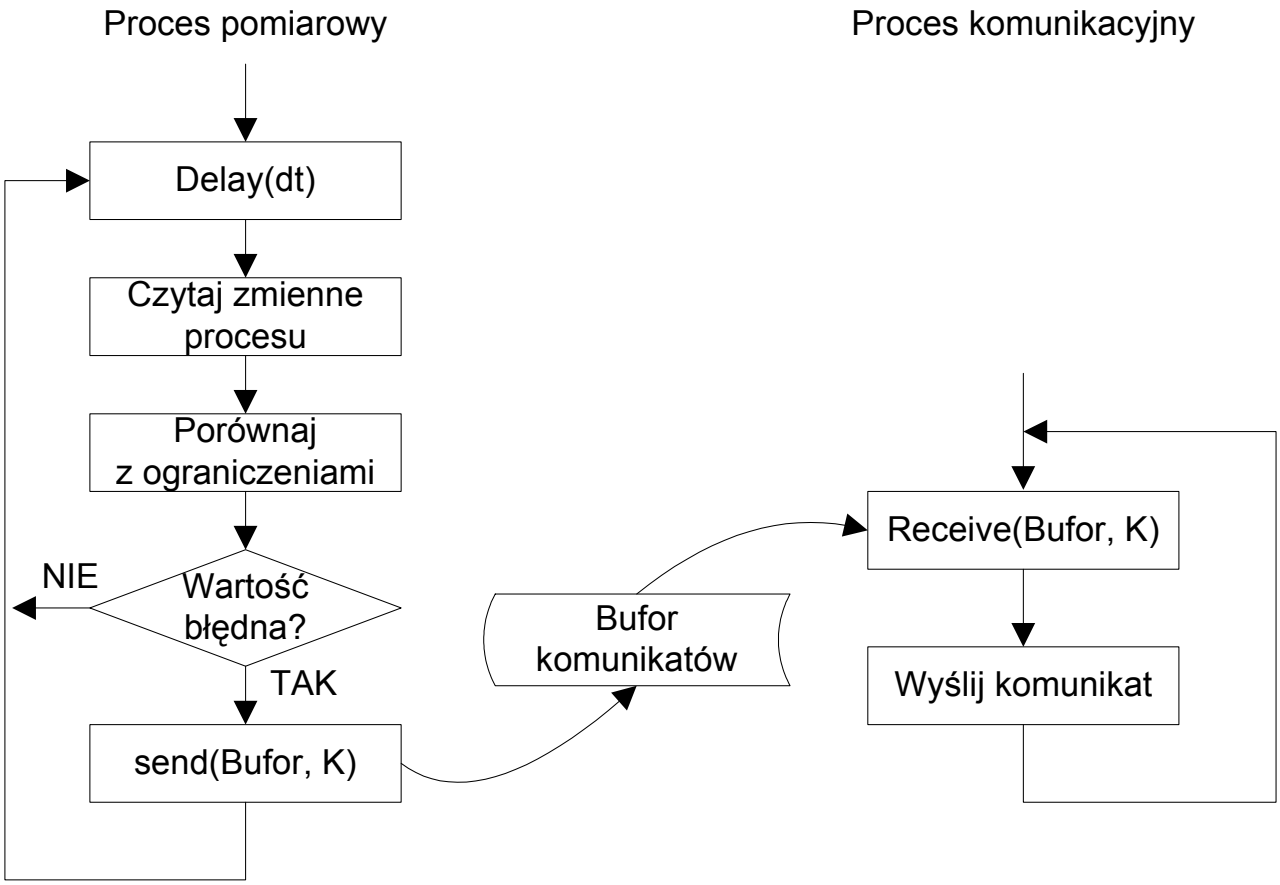
Konsument

```
.....
mutex_lock ( m );
while ( s == 0 )
    cond_wait ( &c,&m );
pobierz wiadomość ( w );
s=s−1;
mutex_unlock(m);
.....
```

- **Koncepcja monitora**



Kolejka wiadomości



Receive(Bufor, K):

if (jest wiadomość) **then** weź jedną **else** *zawieś zadanie*

Send(Bufor, K):

if (czeka zadanie) **then** wznów jedno **else** *wpisz wiadomość*

Operacje

→ POSIX_MESSAGE_PASSING

*mqd_t mq_open (char *name, int oflags,*
*mode_t mode, mq_attr *attr)*

attr — mq_maxmsg, mq_msgsize

oflags — O_CREAT, O_EXCL,
O_RDONLY, O_WRONLY, O_RDWR,
O_NONBLOCK

int mq_close (mqd_t mq)

*int mq_unlink (char *name)*

*int mq_send (mqd_t mq, char *msg, size_t len,*
unsigned prio)

*int mq_timedsend (mqd_t mq, char *msg, size_t len,*
*unsigned prio, struct timespec *abs_timeout)*

*ssize_t mq_receive (mqd_t mq, char *msg, size_t len,*
*unsigned *prio)*

*ssize_t mq_timedreceive (mqd_t mq, char *msg,*
*size_t len, unsigned * prio,*
*struct timespec *abs_timeout)*

Zdarzenie notyfikacji

*int mq_notify (mqd_t mq, struct sigevent *notification)*

struct sigevent

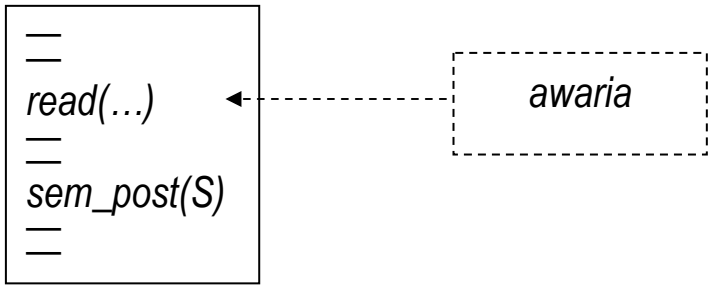
int	*sigev_notify*
int	*sigev_signo*
union sigval	*sigev_value*
void(*)(union sigval)	*sigev_notify_function*
*(pthread_attr_t *)*	*sigev_notify_attributes*

sigev_notify

SIGEV_NONE
SIGEV_SIGNAL
SIGEV_THREAD

Uzależnienia czasowe

- Opóźnienie
- Przeterminowanie



?

- Praca cykliczna

```
—
while (1) {
   sleep ( Δt );
   —
   —
}
—
```

?

Opóźnienie

unsigned sleep (unsigned sec)

unsigned alarm (unsigned sec)

→ POSIX_TIMERS

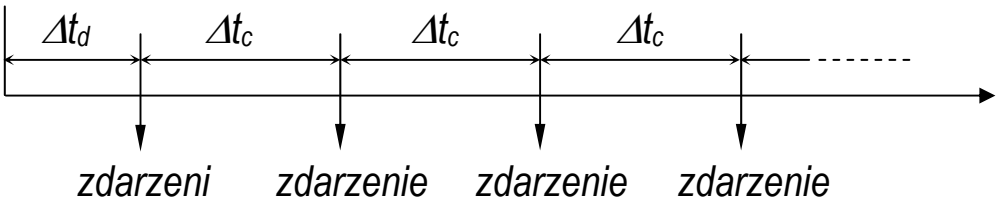
*int nanosleep (struct timespec *rq, struct timespec *rm)*

struct timespec

time_t tv_sec
long tv_nsec

Liczniki czasu (*timer*)

→POSIX_TIMERS



*int timer_create (clockid_t clock, struct sigevent *evp,*
*timer_t *timer)*

int timer_delete (timer_t timer)

int timer_settime (timer_t timer, int flags,
*struct itimerspec *value,*
*struct itimerspec *ovalue)*

flags — TIMER_ABSTIME

int timer_getoverrun (timer_t timer)

*int timer_gettime (timer_t timer, struct itimerspec *value)*

struct itimerspec

struct timespec it_interval
struct timespec it_value

Przykłady

- Przeterminowanie

```
struct itimerspec time;
timer_t           timer;
timer_create(CLOCK_REALTIME,NULL,&timer);
.....
time.it_value.tv_sec=0;
time.it_value.tv_nsec=50 000 000;
time.it_interval.tv_sec=0;
time.it_interval.tv_nsec=0;
timer_settime(timer,0,time,NULL);
result=read(fd,buf,n);
.....
```

- Praca cykliczna

```
.....
time.it_value.tv_sec=0;
time.it_value.tv_nsec=1 000 000;
time.it_interval.tv_sec=0;
time.it_interval.tv_nsec=50 000 000;
timer_settime(timer,0,time,NULL);
do {
   pause();
   .....
} while(run)
```

Asynchroniczne operacje we/wy

→ POSIX_POSIX_ASYNCHRONOUS_IO

struct aiocb

int aio_fildes
off_t aio_offset
*void *aio_buf*
size_t aio_nbytes
int aio_reqprio
struct sigevent aio_sigevent
int aio_lio_opcode Operation to be performed.

*int aio_read (struct aiocb *cb)*

*int aio_write (struct aiocb *cb)*

*int aio_error (struct aiocb *)*

*ssize_t aio_return (struct aiocb *)*

*int aio_suspend (struct aiocb *list[], ll, struct timespec *tt)*