QNX Neutrino
(v 6.3)

- System operacyjny czasu rzeczywistego
- Wielozadaniowy, architektura z mikrojądrem
- API zgodne ze standardem POSIX
- Rozproszony, przezroczysta praca w sieci
- Mechanizmy wykrywania/tolerowania błędów
- Wsparcie SMP
- Wsparcie wielu platform docelowych
- Wiele systemów plików
- Narzędzia programistyczne natywne i skrośne

Architektura klient-serwer

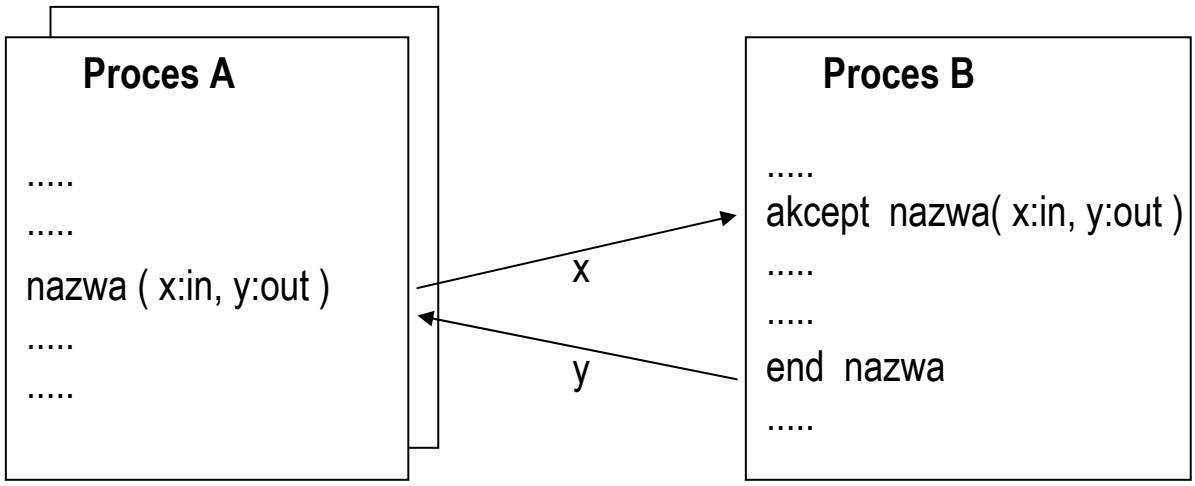
Funkcje mikrojądra

- podział czasu procesora
- sygnały
- komunikacja
- obsługa czasu

Procesy systemowe

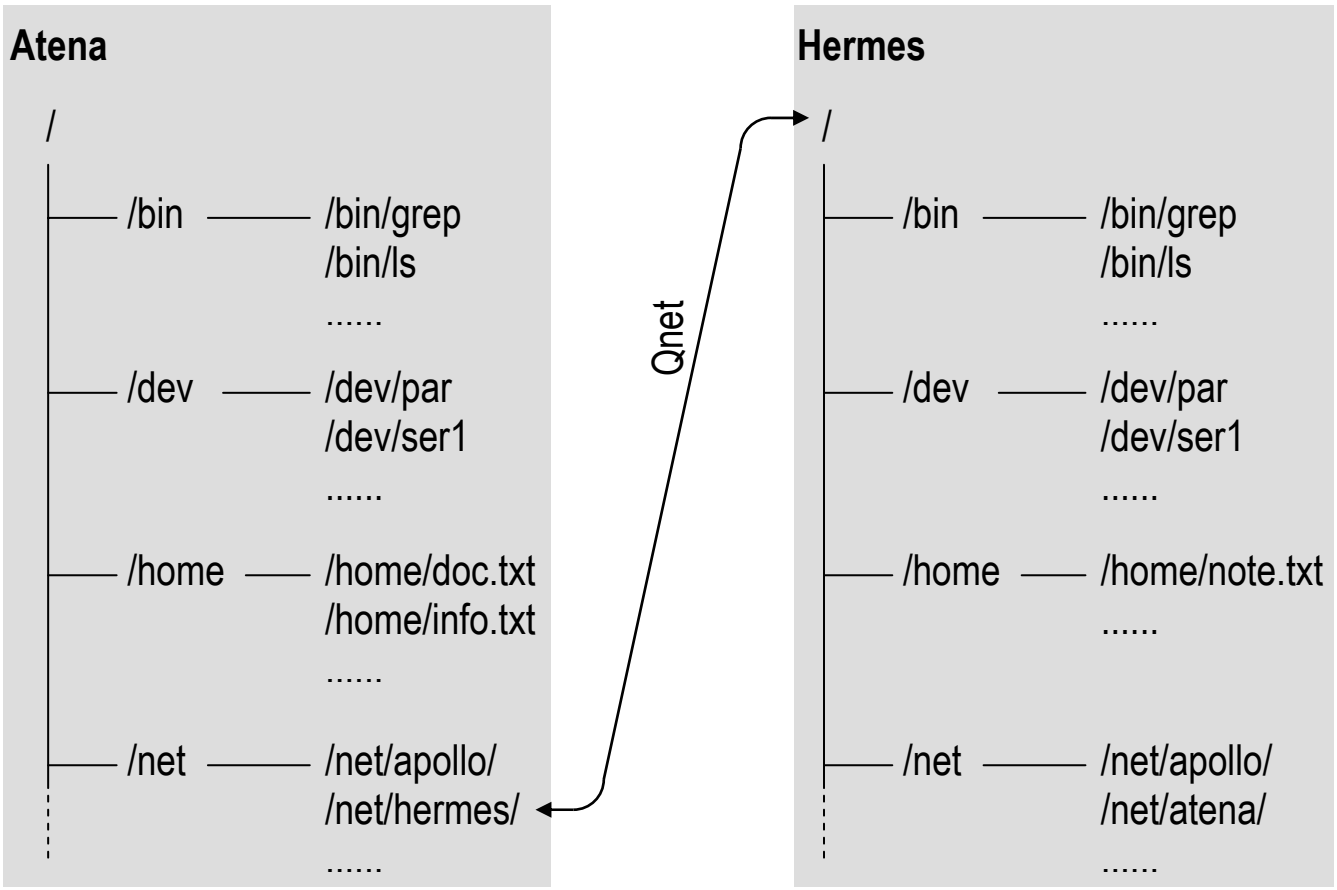
- Administrator procesów, wątków i pamięci (*procnto*)
- Administrator komunikacji znakowej (*devc-par, devc-ser8250, devc-con, ...*)
- Administrator dysku (*devb-eide, devb-fdc, devf-generic, ...*)
- Administrator systemu plików (*fs-dos.so, fs-qnx4.so, fs-ext2.so, fs-cd.so, ...*)
- Administrator sieci (*io-net, npm-qnet.so, devn-ne2000.so, ...*)
- Administrator karty graficznych (*io-graphics, devg-s3.so, ...*)
- .....

Spotkanie



- wymiana komunikatów ?
- zdalne wywołanie usługi ?

Przestrzeń nazw systemu



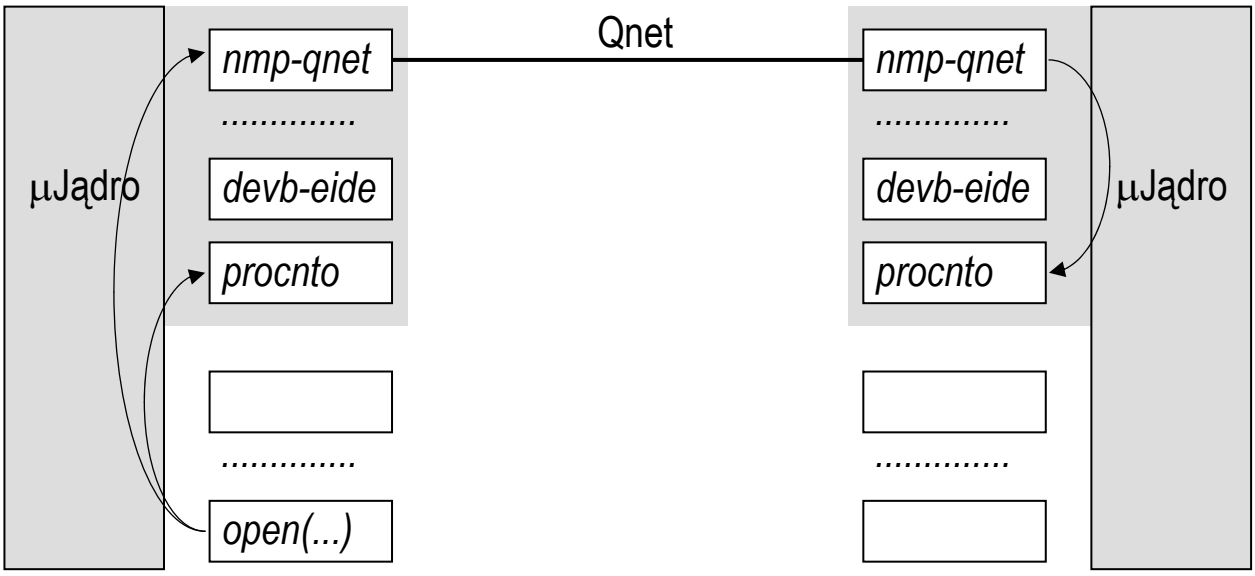
/home/doc.txt

/net/hermes/home/note.txt /home/note.txt

/dev/ser1

/net/hermes/dev/ser1 /dev/ser1

Wywołanie usług



open (/home/doc.txt, ...)

1. Spotkanie z *procnto*
2. Spotkanie z *devb-eide*

open (/net/hermes/home/note.txt, ...)

1. Spotkanie z *procnto*
2. Spotkanie z *nmp-qnet*
3. Spotkanie z *procnto* w odległym węźle
4. Spotkanie z *devb-eide* w odległym węźle

Zgodność ze standardem POSIX

POSIX

- procesy, wątki, algorytmy szeregowania
- sygnały
- synchronizacja i komunikacja zadań
- zarządzanie pamięcią
- uzależnienia czasowe

Odstępstwa

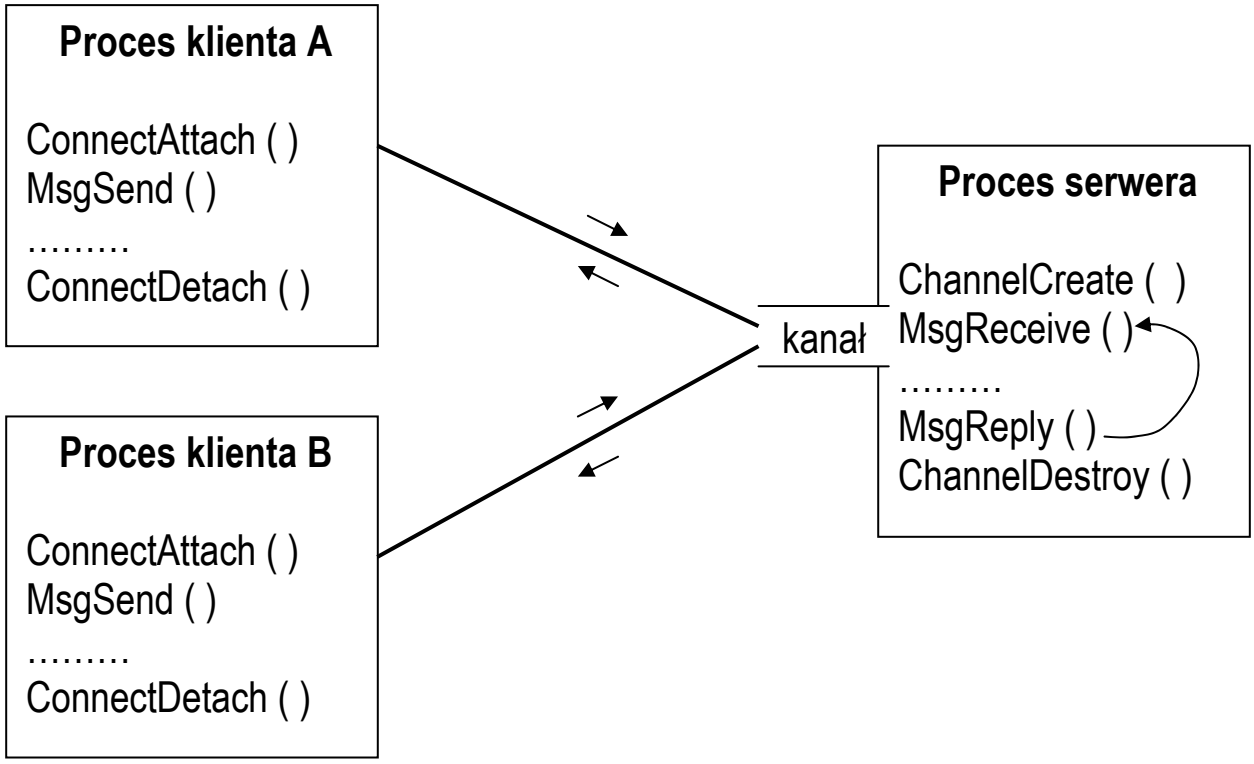
- kolejki wiadomości
- zdarzenia i liczniki czasu
- brak asynchronicznych operacji we/wy

Usługi niestandardowe

- spotkanie
- nazwy zadań
- obsługa przerwań
- operacje we/wy

Spotkanie QNX

1. Utworzenie kanału
2. Nawiązanie połączenia
3. Wywołanie spotkania
4. Odpowiedź serwera



- **Operacje**

Tworzenie kanałów i połączeń

int ChannelCreate (unsigned flags)

flags — dziedziczenie priorytetu,
zdarzenia nadzwyczajne
struktura wiadomości

int ChannelDestroy (int chid)

int ConnectAttach (int node, pid_t pid, int chid,
unsigned index, int flags)

flags — _NTO_COF_CLOEXEC

int ConnectDetach (int coid)

Wysyłanie i odbiór wiadomości

*int MsgSend (int coid, void *smsg, int sbytes,*
*void *rmsg, int rbytes)*

*int MsgReceive (int chid, void *msg, int bytes,*
*struct _msg_info *info)*

*int MsgReply (int rcvid, int status, void *msg, int bytes)*

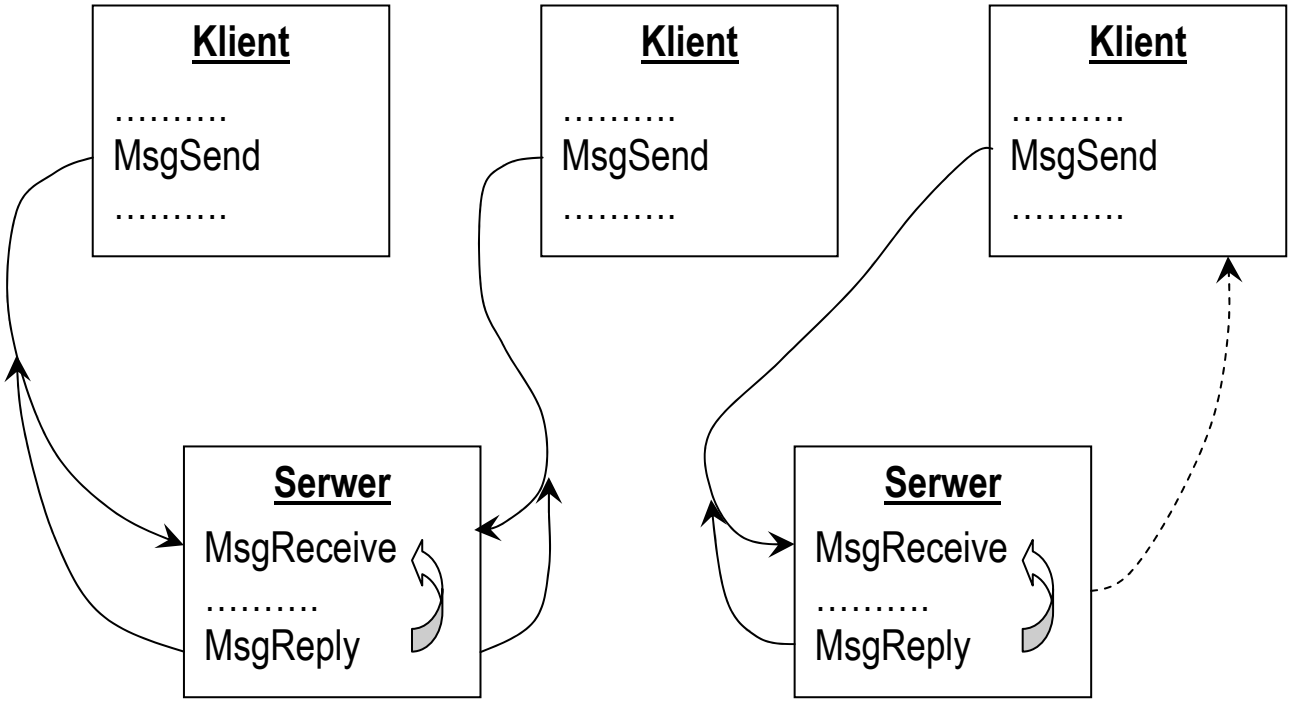
*int MsgRead (int rcvid, void *msg, int bytes, int offset)*

*int MsgWrite (int rcvid, void *msg, int size, int offset)*

- **Przykład**

```
main()
{
   int pid,status;
   int chid,coid,rcvid;
   char bufc[SIZE],bufs[SIZE];
   .....
   chid=ChannelCreate(0);
   .....
   fork();
   if ( p!=0 ) {
      // proces macierzysty – serwer
      do {
         rcvid=MsgReceive(chid,bufs,SIZE,NULL);
         .....
         res=MsgReply(rcvid,0,bufs,SIZE);
      } while ( bufs[0]==0 );
      wait(&status);
   } else {
      // proces potomny - klient
      pid=getppid();
      coid=ConnectAttach (0,pid,chid,_NTO_SIDE_CHANNEL,0);
      do {
         .....
         MsgSend(coid,bufc,SIZE,bufc,SIZE);
      } while ( bufc[0]==0 );
   }
   exit(0);
}
```

Architektura klient-serwer



- wywołanie usług
- powiadomienie

Zdarzenia QNX

- przekazanie sygnału
- uruchomienie nowego wątku
- przekazanie impulsu

struct sigevent

```
int             sigev_notify;
union {
    int         sigev_signo;
    int         sigev_coid;
    int         sigev_id;
    void        (*sigev_notify_function)(union sigval);
}
union sigval    sigev_value;
union {
    struct {
        short   sigev_code;
        short   sigev_priority;
    } st;
    pthread_attr_t *__sigev_notify_attributes;
}
```

Impulsy

struct pulse

uint16_t	*type*
uint16_t	*subtype*
int8_t	*code*
uint8_t	*zero[3]*
union sigval	*value*
int32_t	*scoid*

- **Operacje**

int MsgSendPulse (int coid, int priority, int code, int value)

*int MsgDeliverEvent (int rcvid, struct sigevent *event)*

*int MsgReceivePulse (int chid, void *pulse, int bytes, NULL)*

*int MsgReceive (int chid, void *msg, int bytes,*
*struct _msg_info *info)*

Usługa nazewnicza (GNS)

/dev/name/global
/dev/name/local

- **Operacje**

*name_attach_t *name_attach (NULL, char *name,*
unsigned flags)

flags — NAME_FLAG_ATTACH_GLOBAL

name_attach_t

```
dispatch_t *dpp
int        chid
int        mntid
int        zero[2]
```

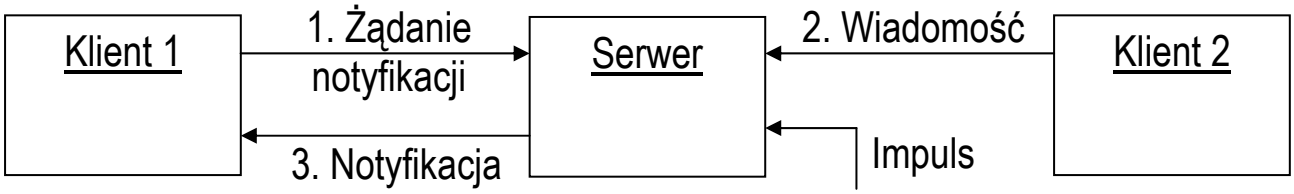
*int name_open (char *name, int flags)*

flags — NAME_FLAG_ATTACH_GLOBAL

int name_close (int coid)

*int name_detach (name_attach_t *attach, unsigned flags)*

Przykład (klient-serwer)



- Struktura wiadomości

```
union {
   struct _pulse       pls;            // impuls
   struct {                            // żądanie notyfikacji
      int              type;
      struct sigevent  event;
   } not;
   struct {                            // wiadomość
      int              type;
      char             text [ MAX ];
   } msg;
} message;
```

- Serwer

```
/* server.c */
int main()
{
   int                rcvid;
   union message      buf;
   name_attach_t      *attach;
   struct not         notify;
   attach=name_attach(NULL,”nazwa”,0);
   if ( attach==NULL ) ... exit(1);
   while (1) {
      rcvid=MsgReceive(attach->chid,&buf,
                                          sizeof(buf),NULL);
      if ( rcvid==–1 ) break;
      if ( rcvid==0 ) ... continue;
      if ( buf.not.type==0) {
         notify.event=buf.not.event;
         notify.type=rcvid;
      }
      ...
      MsgReply(rcvid,0,&buf,sizeof(buf));
      if ( buf.msg.type==9 )
         MsgDeliverEvent(notify.type,&notify.event);
   }
   name_detach(attach,0);
   return 1;
}
```

- **Klient**

```
/* client.c */
int main()
{
   int  chid,coid,srv_coid,rcvid;
   union message  buf;
   struct _pulse     pul;
   chid=ChannelCreate(0);
   coid=ConnectAttach(0,0,chid,_NTO_SIDE_CHANNEL,0);
   SIGEV_PULSE_INIT(&buf.not.event,coid,0,9,0);
   buf.not.type=0;
   srv_coid=name_open(”nazwa”,0);
   if ( srv_coid==-1 ) ... exit(1);
   while (1) {
      ...
      MsgSend(srv_coid,&buf,sizeof(buf),&buf, sizeof(buf));
      if ( ..... )
         res= MsgReceivePulse(chid,&pul,sizeof(pul),NULL);
      ...
   }
   name_close(srv_coid);
   ConnectDetach(coid);
   ChannelDestroy(chid);
   return 0;
}
```

Obsługa przerwań

- Asynchroniczna funkcja obsługi

```
int  InterruptAttach ( int  intr,
                       struct  sigevent *( *handler)(void  *, int ),
                       void  *area,  int  size,
                       unsigned  flags )
```

flags — kolejność obsługi przerwań dzielonych
przypisanie do procesu/wątku
śledzenie maski przerwań dzielonych

```
struct  sigevent  *handler ( void  *area,  int id )
{
  ......
}

int  InterruptWait ( 0,  NULL )
```

```
volatile
ThreadCtl ( _NTO_TCTL_IO,  0 )
```

- Notyfikacja procesu/wątku

  *int InterruptAttachEvent (int intr, struct sigevent *event, unsigned flags)*

  int InterruptDetach (int id)

- Blokowanie przerwań

  void InterruptDisable (void)

  void InterruptEnable (void)

  *void InterruptLock (intrspin_t *spinlock)*

  *void InterruptUnlock (intrspin_t *spinlock)*

  int InterruptMask (int intr, int id)

  int InterruptUnmask (int intr, int id)

Dostęp do rejestrów we/wy

- Dostępu do portów

  uint8_t in8 (uintptr_t port)

  void out8 (uintptr_t port, uint8_t val)

  *void *in8s (void* buff, unsigned len, uintptr_t port)*

  *void *out8s (void *buff, unsigned len, uintptr_t port)*

- Otwarcie przestrzeni we/wy

  uintptr_t mmap_device_io (size_t len, uint64_t io)

  *void *mmap_device_memory (void *addr, size_t len,*
  int prot, int flags,
  uint64_t physical)

- Krótkie oczekiwanie

int nanospin_ns (unsigned long nsec)

• Obsługa karty PCL 718

	7	6	5	4	3	2	1	0
BASE+2	*last channel*				*first channel*			
BASE+9	INT	nr przerwania			–	DMA	trigger	

```
#define BASE 0x300
#define INTR    3
#define SIZE  1000

static unsigned short buf[SIZE],head=0,tail=0;
int id;
struct sigevent event;
uintptr_t port;

void odczytAc(int rate,int last)
{
   ThreadCtl(_NTO_TCTL_IO,0);          //dostęp do io
   port=mmap_device_io(16,BASE);       //przestrzeń io
   counter_init(rate);                 //timery karty
   out8(port+2,last<<4);               //multiplekser
   SIGEV_INTR_INIT(&event);            //zdarzenie
   mlockall(MCL_CURRENT);              //wyłącz swap
   id=InterruptAttach(INTR,&handler,buf,2*SIZE+4,
                      _NTO_INTR_FLAGS_PROCESS);
   while(1) {
      out8(port+9,0x80|(INTR<<4)|3); //adc timer
      InterruptWait(0,NULL);
      ......................
   }
}

struct sigevent *handler(void *buf,int id)
{
   char al,ah;
   out8(port+8,0);                     //kasuj INTR
   al=in8(port+0);
   ah=in8(port+1);
   buf[tail]=(ah<<8)+al;
   tail=(tail+1)%SIZE;
   if ( (tail%(last+1))>0 ) {
      out8(port+9,0x80|(INTR<<4));    //adc program
      out8(port+0,0);                 //pomiar
      return NULL;
   } return &event;
}
```