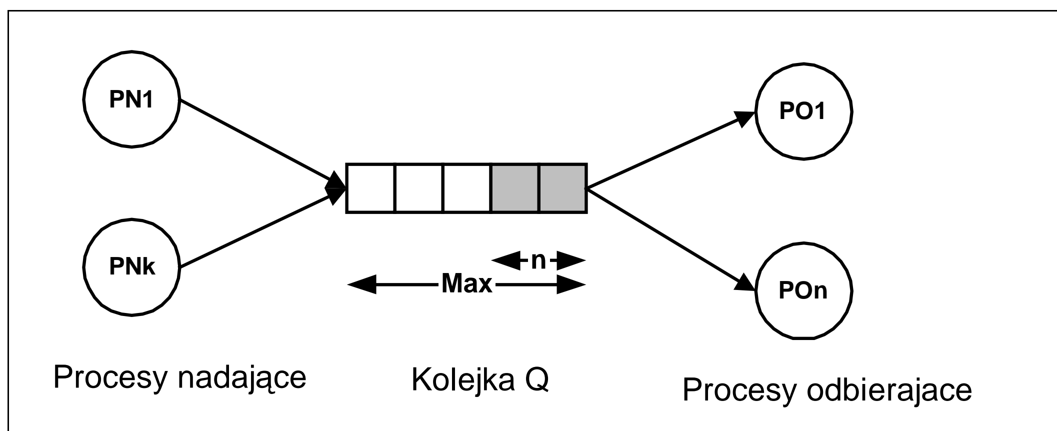
13. Kolejki komunikatów POSIX

13.1 Wstęp

Kolejki komunikatów (mailboxy, bufory) są bardzo popularnym mechanizmem komunikacji międzyprocesowej. Występują w prawie każdym systemie operacyjnym.

Kolejka komunikatów Q posiada następujące własności:

- Posiada określoną pojemność N komunikatów (długość bufora komunikatów).
- Posiada nazwę którą procesy mogą zidentyfikować.
- Więcej niż jeden proces może czytać lub pisać z/do kolejki.



Rys. 1 Procesy komunikują się za pomocą kolejki komunikatów

Blokowanie procesów podczas operacji zapisu i odczytu zależy od liczby n komunikatów w kolejce i od jej pojemności Max.

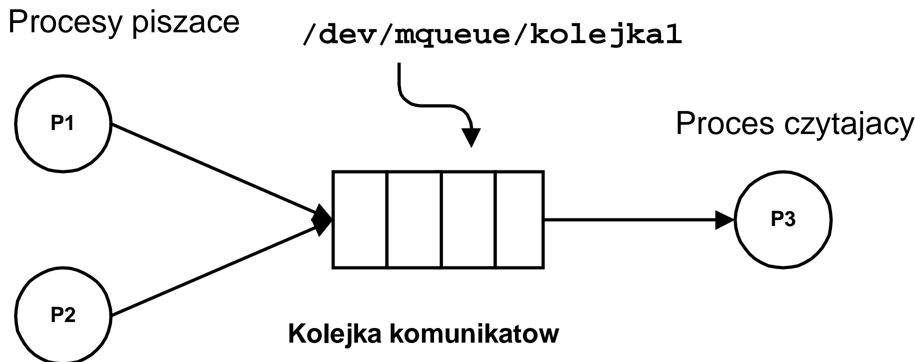
Liczba komunikatów n w kolejce Q	Wysłanie komunikatu	Odbiór komunikatu
n = Max	Blokada lub sygnalizacja błędu	Bez blokady
0< n < Max	Bez blokady	Bez blokady
n = 0	Bez blokady	Blokada lub sygnalizacja błędu

Tab. 13-1 Przebieg operacji na kolejce komunikatów w zależności od liczby komunikatów n w jej buforze

PDF created with pdfFactory trial version www.pdffactory.com

13.2 Kolejki komunikatów – POSIX

Istnieje wiele implementacji kolejek komunikatów. Tutaj omówione będą kolejki komunikatów POSIX.



Rysunek 13-1 Dwa procesy piszą do jednej kolejki

Podstawowe własności kolejek komunikatów POSIX:

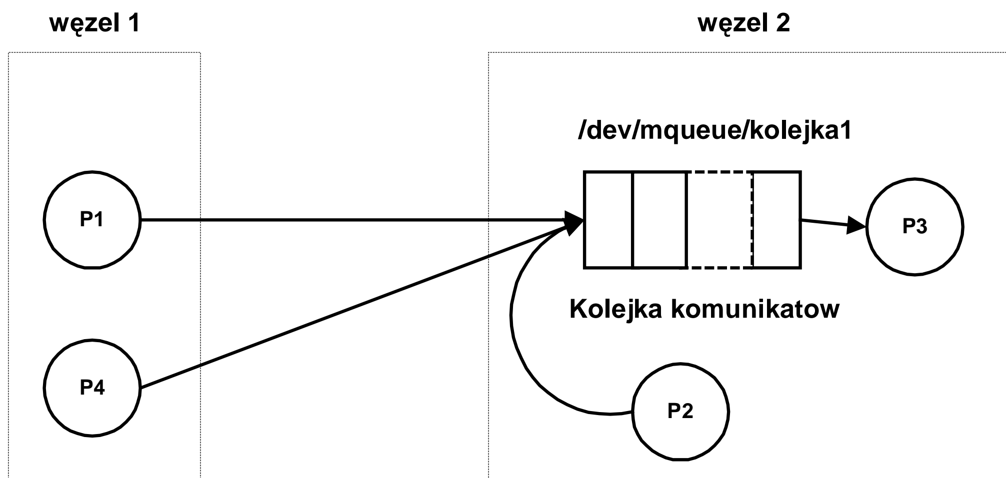
1. Kolejki komunikatów są pośrednim obiektem komunikacyjnym widzianym jako plik specjalny. Komunikujące się procesy nie muszą znać swoich identyfikatorów.
2. Komunikaty odczytywane z kolejki zachowują strukturę – są separowane. W kolejce mogą znajdować się komunikaty różnej długości. Własności tej nie mają kolejki FIFO.
3. Można zadać maksymalną długość kolejki komunikatów. Gdy zostanie ona przekroczona, proces piszący do kolejki komunikatów będzie zablokowany.
4. Kolejka widziana jest w systemie plików jako plik specjalny. Operacje zapisu / odczytu mogą być zabezpieczane prawami dostępu tak jak w przypadku plików regularnych.
5. Można testować status kolejki (np. liczbę komunikatów w kolejce). Nie jest to możliwe w przypadku kolejek FIFO.
6. Komunikatom można nadać priorytet. Komunikaty wyższym priorytecie będą umieszczane na początku kolejki.

PDF created with pdfFactory trial version www.pdffactory.com

Zastosowanie kolejki komunikatów jest wygodnym rozwiązaniem w następujących przypadkach:

1. Proces wysyłający komunikaty nie może być wstrzymany.
2. Proces wysyłający komunikaty nie potrzebuje szybkiej informacji zwrotnej o tym czy komunikat dotarł do adresata.
3. Zachodzi potrzeba przekazywania danych z procesu w którym one powstają (producent) do procesu w którym są one przetwarzane (konsument)

Uwaga!
W systemie QNX6 Neutrino kolejki komunikatów działają przez sieć.



Rysunek 13-2 Kolejka komunikatów dostępna w sieci Qnet

PDF created with pdfFactory trial version www.pdffactory.com

Podstawowe typy i plik nagłówkowy

Kolejka komunikatów jest typu mqd_t. Typ ten jest zdefiniowany w pliku nagłówkowym <mqueue.h>. Modyfikowalne atrybuty kolejki komunikatów zdefiniowane są w strukturze **mq_attr**.

```
struct mq_attr {
  long mq_maxmsg;     // Maks. liczba komunikatów w kolejce
  long mq_msgsize;    // Maks. wielkość poj. komunikatu
  long mq_curmsg;     // Aktualna liczba kom. w kolejce
  long mq_flags;  // Flagi
  long mq_sendwait; // Liczba proc. zablok. na op. zapisu
  long mq_recvwait; // Liczba proc. zablok. na op. odczytu
}
```

Utworzenie i otwarcie kolejki komunikatów

Kolejkę komunikatów tworzy się za pomocą funkcji:

```
mqd_t  mq_open(char *name,int oflag,int
mode,mq_attr *attr)
```

name	Łańcuch identyfikujący kolejkę komunikatów. Kolejki tworzone są w katalogu **/dev/mqueue**
oflag	Tryb tworzenia kolejki. Tryby te są analogiczne jak w zwykłej funkcji open.
mode	Prawa dostępu do kolejki (r - odczyt, w - zapis) dla właściciela pliku, grupy i innych, analogicznie jak w przypadku plików regularnych. Atrybut x - wykonanie jest ignorowany.
attr	Atrybuty kolejki

PDF created with pdfFactory trial version www.pdffactory.com

Ważniejsze tryby tworzenia kolejki komunikatów:

Tryb	Znaczenie
O_RDONLY	Tylko odczyt z kolejki
O_WRONLY	Tylko zapis do kolejki
O_RDWR	Odczyt i zapis
O_CREAT	Utwórz kolejkę o ile nie istnieje
O_NONBLOCK	• Domyślnie flaga jest wyzerowana co powoduje że operacje odczytu (**mq_receive**) i zapisu (**mq_send**) mogą być blokujące. • Gdy flaga jest ustawiona operacje te nie są blokujące i kończą się błędem.

Tab. 13-2 Podstawowe flagi używane przy tworzeniu kolejek komunikatów

Domyślne atrybuty:

Atrybut	Wartość domyślna
mq_maxmsg	**1024**
mq_msgsize	**4096**
mq_flags	**0**

Gdy kolejka już istnieje parametry 3 i 4 funkcji mq_open są ignorowane.

Funkcja **mq_open** zwraca:

1. W przypadku pomyślnego wykonania wynik jest nieujemny – jest to identyfikator kolejki komunikatów
2. W przypadku błędu funkcja zwraca –1.

Uwaga!

- Gdy nazwa kolejki zaczyna się od „/” to kolejka tworzona jest w katalogu **/dev/mqueue**
- Gdy nazwa kolejki zaczyna się od znaku innego niż „/” to kolejka tworzona jest w katalogu bieżącym.

PDF created with pdfFactory trial version www.pdffactory.com

Wysłanie komunikatu do kolejki

Wysłanie komunikatu do kolejki komunikatów odbywa się za pomocą funkcji:

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

Znaczenie parametrów:

mq identyfikator kolejki komunikatów,
***msg** adres bufora wysyłanego komunikatu,
len długość wysyłanego komunikatu,
mprio priorytet komunikatu (od 0 do MQ_PRIORITY_MAX).

Wywołanie funkcji powoduje przekazanie komunikatu z bufora msg do kolejki mq. Można wyróżnić dwa zasadnicze przypadki:

1) W kolejce jest miejsce na komunikaty. Wtedy wykonanie funkcji nie spowoduje zablokowania procesu bieżącego.
2) W kolejce brak miejsca na komunikaty. Wtedy wykonanie funkcji spowoduje zablokowania procesu bieżącego. Proces ulegnie odblokowaniu gdy zwolni się miejsce w kolejce.

Zachowanie się funkcji uzależnione jest od stanu flagi O_NONBLOCK. Flaga ta jest domyślnie wyzerowana. W ogólności funkcja zwraca:

0 Sukces
-1 Błąd

Wysłanie komunikatu do kolejki – czekanie z przeterminowaniem

W przypadku gdy na wysłanie komunikatu nie można czekać w nieskończoność istnieje wersja funkcji z przeterminowaniem.

```
int mq_timedsend(mqd_t mq, char *msg, size_t len,
unsigned int mprio, struct timespec *timeout)
```

Znaczenie parametrów:

mq Identyfikator kolejki komunikatów,
***msg** Adres bufora wysyłanego komunikatu,
len Długość wysyłanego komunikatu,
mprio Priorytet komunikatu (od 0 do MQ_PRIORITY_MAX).
timeout Czas absolutny po którym wystąpi timeout

PDF created with pdfFactory trial version www.pdffactory.com

Gdy upłynie chwila **timeout** a komunikat nie zostanie umieszczony w kolejce funkcja odblokuje się, zwróci –1 a zmienna errno zawierać będzie kod błędu **ETIMEDOUT** .

Pobieranie komunikatu z kolejki

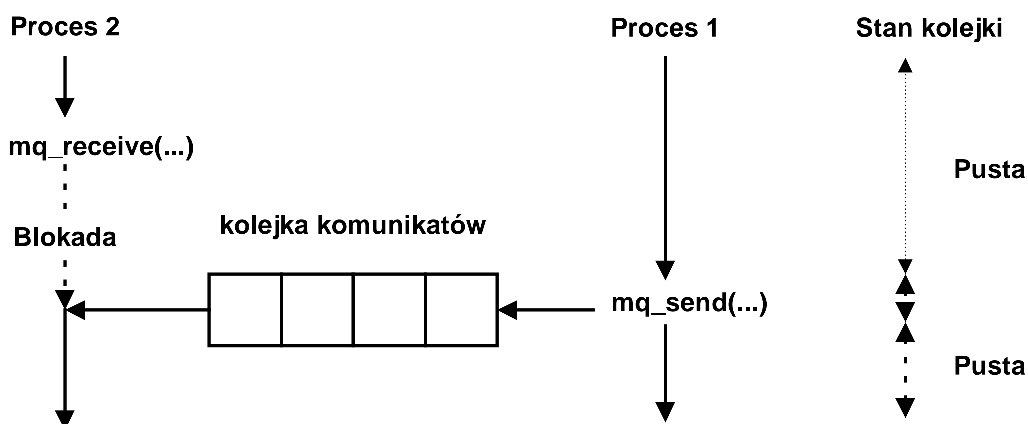
Pobieranie komunikatu z kolejki komunikatów odbywa się za pomocą funkcji **mq_receive.**

```
int mq_receive(mqd_t mq, char *msg, size_t len,
unsigned int *mprio)
```

Znaczenie parametrów:

mq identyfikator kolejki komunikatów,
***msg** adres bufora na odbierany komunikat,
len maksymalna długość odbieranego komunikatu,
mprio priorytet odebranego komunikatu.

1. Gdy w kolejce znajduje się przynajmniej jeden komunikat wywołanie funkcji **mq_receive** nie spowoduje zablokowania procesu bieżącego.
2. Gdy w kolejce brak komunikatów wywołanie funkcji **mq_receive** spowoduje zablokowania procesu bieżącego. Proces ulegnie odblokowaniu gdy w kolejce pojawi się jakiś komunikat.



Proces 2 blokuje się przy próbie odbioru komunikatu z kolejki.

W przypadku gdy więcej niż jeden proces czeka na komunikat – odblokowany będzie proces który najdłużej czekał. Zachowanie się

PDF created with pdfFactory trial version www.pdffactory.com

funkcji uzależnione jest także od stanu flagi O_NONBLOCK. Funkcja **mq_receive** zwraca:

>0 Rozmiar odebranego komunikatu gdy wynik jest większy od 0.

–1 Gdy wystąpił błąd.

Pobieranie komunikatu z kolejki – czekanie z przeterminowaniem

W przypadku gdy nie można czekać w nieskończoność istnieje wersja funkcji odbioru komunikatów z przeterminowaniem.

```
int mq_timedreceive(mqd_t mq, char *msg, size_t len, unsigned int *mprio, struct timespec *timeout)
```

Znaczenie parametrów:

mq Identyfikator kolejki komunikatów,
***msg** Adres bufora na odbierany komunikat,
len Maksymalna długość odbieranego komunikatu,
mprio Priorytet odebranego komunikatu.
timeout Czas absolutny po którym wystapi timeout

Gdy po upływie czasu **timeout** nie zostanie odebrany żaden komunikat to funkcja zwróci –1 a zmienna errno będzie zawierała kod błędu ETIMEDOUT

Przykład:

```
struct timespec tm;
clock_gettime(CLOCK_REALTIME,&tm);
// Dodana sekunda
tm.tv_sec += 1;
res = mq_timedreceive(mq,&msg,sizeof(msg),NULL,&tm)
if((res == -1) && (errno = = ETIMEDOUT)) {
  // Wystapil timeout
}
```

Przykład:

Procesy P1 i P2 komunikują się przy pomocy kolejki komunikatów – problem producenta konsumenta.

PDF created with pdfFactory trial version www.pdffactory.com

```
// Proces P1 wysylajacy komunikaty do kolejki MQ1
#include <stdio.h>
#include <mqueue.h>
#define  SIZE    80
typedef struct {
      int  type;        // Typ komunikatu
      char text[SIZE];  // Tekst komunikatu
   } msg_tp;

main(int argc, char *argv[]) {
 int i;
 int res;
 mqd_t  mq;
 msg_tp msg;
 struct mq_attr attr;
 // Ustalenie atrybutów kolejki ----------------
 attr.mq_msgsize = sizeof(msg);
 attr.mq_maxmsg  = 8;
 // Utworzenie kolejki komunikatow ------------
 mq=mq_open("MQ1",O_RDWR | O_CREAT, 0666,&attr);
 if(mq < 0) { // Błąd
    perror("Kolejka MQ1");
    exit(-1);
 }
 for(i=0; i < 10 ;i++) {
    sprintf(msg.text,"Proces 1 komunikat %d",i);
    // Wysłanie komunikatu ----------------
    res = mq_send(mq,&msg,sizeof(msg),10);
    sleep(1);
  }
  mq_close(mq);
}
```

Przykład 13-1 Kod procesu wysyłającego komunikaty do kolejki MQ1 - producent

PDF created with pdfFactory trial version www.pdffactory.com

```
// Proces P2 odbierający komunikaty z kolejki MQ1
#include <stdio.h>
#include <mqueue.h>
#define  SIZE    80
typedef struct {
     int  type;        // Typ komunikatu
     char text[SIZE];  // Tekst komunikatu
  } msg_tp;

main(int argc, char *argv[]) {
 int i;
 int res;
 mqd_t  mq;
 msg_tp msg;
 struct mq_attr attr;
 // Ustalenie atrybutów kolejki ----------------
 attr.mq_msgsize = sizeof(msg);
 attr.mq_maxmsg  = 8;

 // Utworzenie kolejki komunikatow ------------
 mq=mq_open("MQ1",O_RDWR | O_CREAT, 0666,&attr);
 if(mq < 0) { // Błąd
    perror("Kolejka MQ1");
     exit(-1);
 }
 for(i=0; i < 10 ;i++) {
    sprintf(msg.text,"Proces 1 komunikat %d",i);
    // Odbiór komunikatu ----------------
    res = mq_receive(mq,&msg,sizeof(msg),NULL);
    printf("%s\n",msg.text);
 }

  mq_close(mq);
}
```

Przykład 13-2 Kod procesu odbierającego komunikaty z kolejki MQ1 - konsument

PDF created with pdfFactory trial version www.pdffactory.com

Testowanie statusu kolejki komunikatów

Testowanie statusu kolejki komunikatów odbywa się poprzez wykonanie funkcji:

```
int  mq_getattr(mqd_t mq, struct mq_attr *attr)
```

Znaczenie parametrów:

mq identyfikator kolejki komunikatów,
***attr** Adres bufora ze strukturą zawierającą atrybuty kolejki komunikatów

Użyteczne elementy struktury atrybutów:

mq_curmsg Aktualna liczba komunikatów w kolejce

mq_sendwait Liczba procesów zablokowanych na operacji zapisu
mq_recvwait Liczba procesów zablokowanych na operacji odczytu

PDF created with pdfFactory trial version www.pdffactory.com

Zawiadamianie procesu o pojawieniu się komunikatu w kolejce

1. Można spowodować aby pojawienie się komunikatu w pustej kolejce (a wiec zmiana stanu kolejki z „pusta” na „niepusta”) powodowało zawiadomienie procesu bieżącego.
2. Zawiadomienie może mieć postać sygnału lub impulsu (*ang.Pulse*).

Działanie	Symbol
Wysłanie impulsu	**SIGEV_PULSE**
Wysłanie do procesu sygnału zwykłego	**SIGEV_SIGNAL**
Wysłanie do procesu sygnału z 8 bitowym kodem	**SIGEV_SIGNAL_CODE**
Wysłanie do wątku sygnału z 8 bitowym kodem do wyspecyfikowanego wątku.	**SIGEV_SIGNAL_THREAD**
Zdarzenie odpowiadające przerwaniu	**SIGEV_INTR**

Tabela 13-1 Powiadomienia od kolejki komunikatów

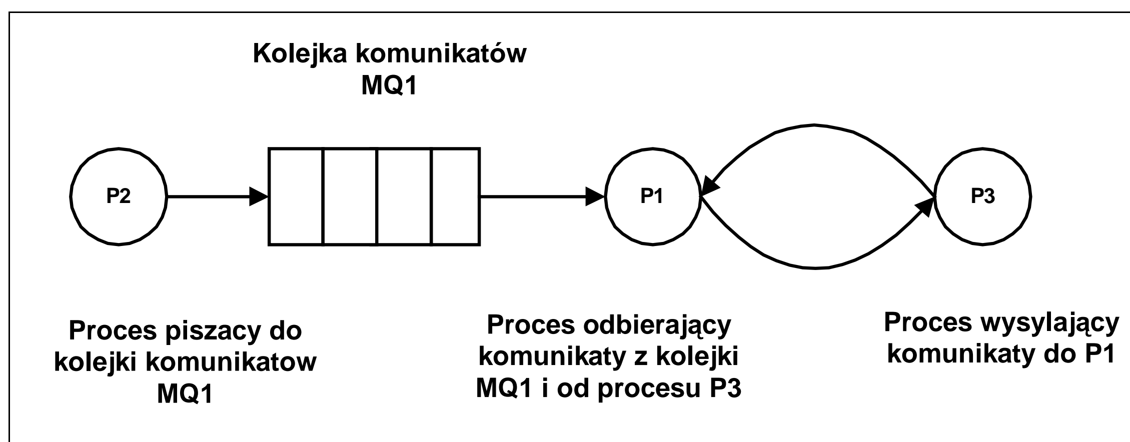
Ustawienie zawiadamiania odbywa się poprzez wykonanie funkcji **mq_notify():**

mq_notify() – ustawianie zawiadomienia kolejki	
int mq_notify(mqd_t mq, struct sigevent *notif)	
mq	Identyfikator kolejki komunikatów.
***notif**	Adres struktury typu **sigevent** specyfikującego sposób zawiadomienia.

Użyta w funkcji struktura typu **sigevent** powinna być wcześniej zainicjowana przy pomocy odpowiedniego makra. Np makra **SIGEV_PULSE_INIT(&event,coid,priority,code,value)** które inicjuje zdarzenie **event**.

PDF created with pdfFactory trial version www.pdffactory.com

Parametr	Opis
coid	Identyfikator kanału w którym impuls ma się pojawić
priority	Priorytet impulsu
code	Kod impulsu
value	Wartość impulsu



Proces P1 odbiera komunikaty z dwóch źródeł – kolejki MQ1 i procesu P3

Testując kod powrotu **`pid`** można rozróżnić przypadki pojawienia się impulsu i zwykłego komunikatu.

PDF created with pdfFactory trial version www.pdffactory.com

```
// Proces P1 - odbiór komunikatów z dwóch źródeł
// 1. Kolejki komunikatów MQ1
// 2. Innych procesów
// Wykorzystano zawiadomienie o zmianie stanu
// kolejki za pomocą impulsu

main(void) {
...
SIGEV_PULSE_INIT(&event,coid,priority,1,0);
mq = mq_open(MQ_NAME , O_RDWR | O_CREAT , 0660, &attr );
mq_notify(mq,&event);

for(i=0; i<11; i++) {
    pid = MsgReceive(chid,&msg,sizeof(msg),NULL);
    if(pid == 0) {
         printf("Komunikat w kolejce \n");
         mq_receive(mq,&buf,sizeof(buf),&prior);
    } else {
        printf("Komunikat z kanalu\n");
        res = MsgReply(pid,0,&msg,sizeof(msg));
    }
}
```

Przykład 13-3 Proces odbiera komunikaty z kanału i kolejki komunikatów

PDF created with pdfFactory trial version www.pdffactory.com

Zamknięcie i skasowanie kolejki komunikatów

Gdy proces przestanie korzystać z kolejki komunikatów powinien ją zamknąć. Do tego celu służy funkcja:

```
int mq_close(mqd_t mq)
```

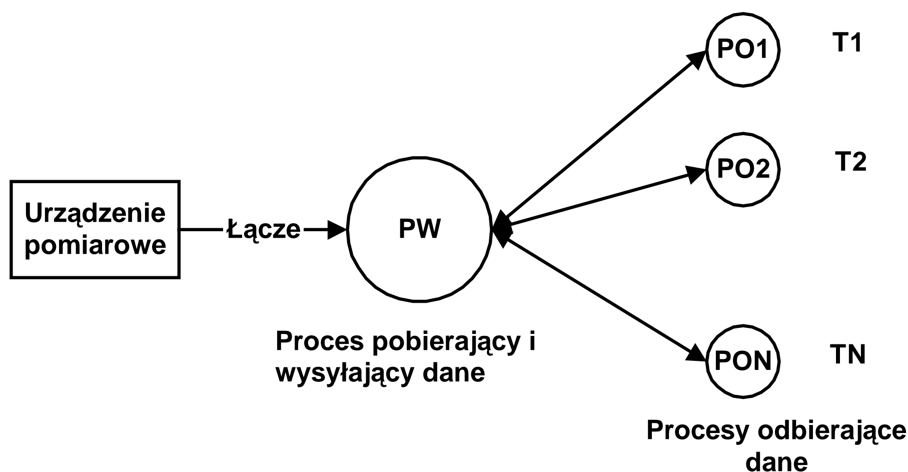
Kolejkę kasuje się za pomocą polecenia:

```
int mq_unlink(char *name)
```


PDF created with pdfFactory trial version www.pdffactory.com

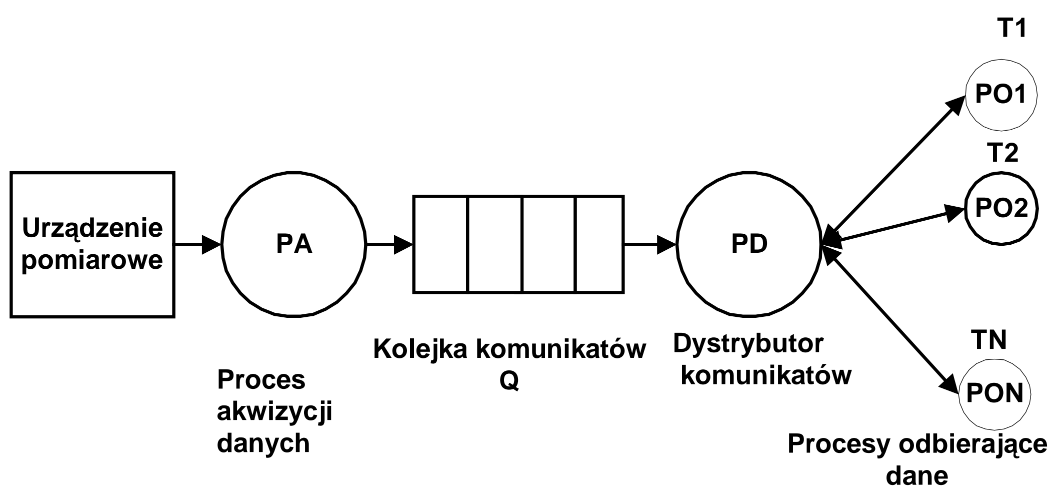
13.3 Przykład zastosowanie kolejki komunikatów w systemie akwizycji danych.

Urządzenie pomiarowe dostarcza komunikatów w nieregularnych odstępach czasu. Przesłanie komunikatów do procesów odbierających trwa pewien czas. Suma czasów T1+T2+...+TN może być większa niż okres pojawiania się pomiarów co może prowadzić do ich zagubienia.



Rys. 2 Proces akwizycji danych PW przesyła wyniki do N procesów PO odbierających dane za pomocą komunikatów

Zastosowanie kolejki komunikatów pozwala na buforowanie pomiarów co zmniejsza możliwość ich utraty.



Rys. 3 Akwizycja i dystrybucja danych odbywa się poprzez dwa procesy PA i PD połączone kolejką komunikatów Q.

PDF created with pdfFactory trial version www.pdffactory.com