

# 1. Kolejki komunikatów POSIX

## 1.1 Podstawowe własności

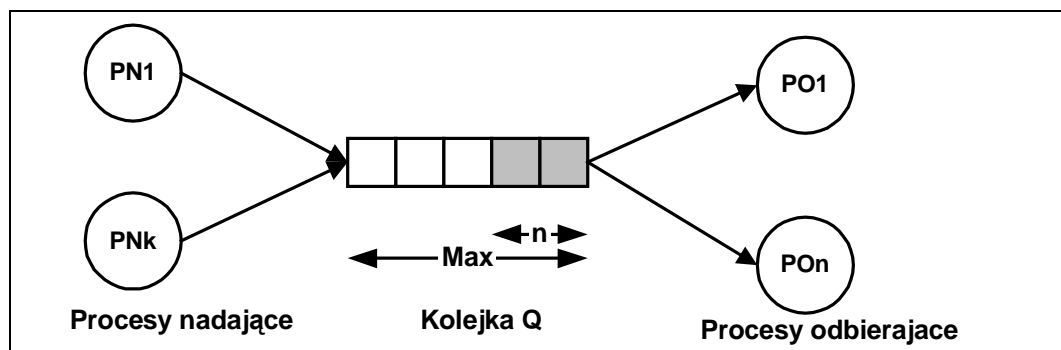
Kolejki FIFO mają następujące wady:

- Komunikaty pozbawione struktury
- Nie można testować stanu kolejki – liczba komunikatów, procesy zablokowane

Wad tych pozbawione są kolejki komunikatów POSIX

Podstawowe cechy kolejek komunikatów:

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 zabezpieczone 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.



Rys. 1-1 Procesy komunikują się za pomocą kolejki komunikatów

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)

## 1.2 Podstawowe typy i plik nagłówkowy

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

```
struct mq_attr {
    long mq_maxmsg;    // Maksymalna liczba komunikatów w kolejce
    long mq_msgsize;  // Maksymalna wielkość pojedynczego komunikatu
    long mq_curmsg;   // Aktualna liczba komunikatów w kolejce
    long mq_flags;    // Flagi
    long mq_sendwait; // Liczba procesów zablok. na operacji zapisu
    long mq_recvwait; // Liczba procesów zablok. na operacji odczytu
}
```

<code>mq_open</code>	Otwarcie kolejki komunikatów
<code>mq_getattr</code>	Pobranie atrybutów kolejki
<code>mq_setattr</code>	Ustawienie atrybutów kolejki
<code>mq_receive</code>	Odbiór komunikatu z kolejki
<code>mq_send</code>	Zapis komunikatu do kolejki
<code>mq_timedreceive</code>	Odbiór z kolejki komunikatów (z ogr.czasowym)
<code>mq_timedsend</code>	Zapis komunikatu do kolejki (z ogr.czasowym)
<code>mq_notify</code>	Żądanie zawiadomnienia gdy kolejka niepusta
<code>mq_unlink</code>	Skasowanie kolejki
<code>mq_close</code>	Zamknięcie kolejki

### 1.3 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 bieżącym

**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

Ważniejsze tryby tworzenia kolejki komunikatów:

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

Tab. 1-1 Podstawowe flagi używane przy tworzeniu kolejek komunikatów

Atrybut	Wartość domyślna
<code>mq_maxmsg</code>	1024
<code>mq_msgsize</code>	4096
<code>mq_flags</code>	0

Tab. 1-2 Domyślne atrybuty kolejki komunikatów:

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`.

## 1.4 Synchronizacja

Przebieg operacji zapisu i odczytu zależy od liczby `n` komunikatów w kolejce i od jej pojemności `Max`.

- Wysyłanie komunikatu: `mq_send(...)`
- Odbiór komunikatu: `mq_receive(...)`

Liczba komunikatów <code>n</code> w kolejce <code>Q</code>	Wysłanie komunikatu	Odbiór komunikatu
<code>n = Max</code>	Blokada lub sygnalizacja błędu	Bez blokady
<code>0 &lt; n &lt; Max</code>	Bez blokady	Bez blokady
<code>n = 0</code>	Bez blokady	Blokada lub sygnalizacja błędu

Rys. 1-2 Przebieg operacji na kolejce w zależności od liczby komunikatów `n` w jej buforze

## 1.5 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.

Funkcja zwraca:

- 0 Sukces
- 1 Błąd

## 1.6 Pobieranie komunikatu z kolejki

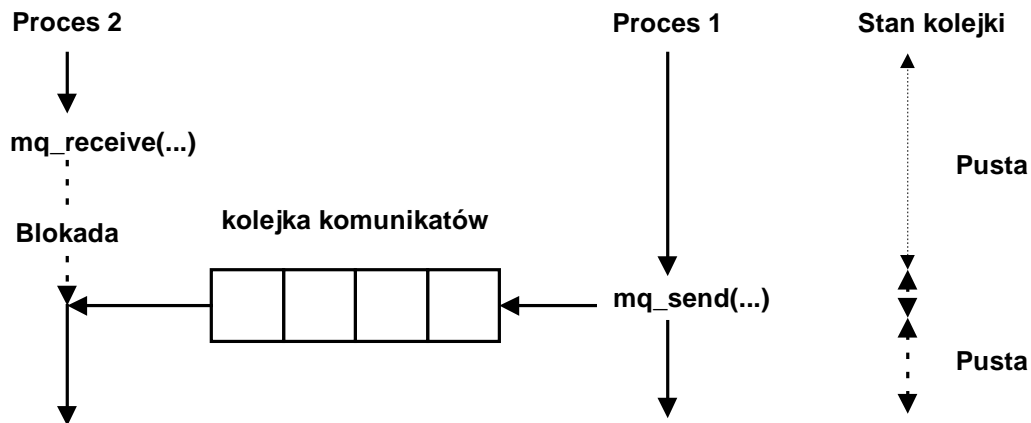
Pobieranie komunikatu z kolejki komunikatów odbywa się za pomocą funkcji `mq_receive`.

```
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.



Rys. 1-3 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 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.

Przykład:

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

```
// 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;
    attr.mq_flags = 0;

    // Utworzenie kolejki komunikatow -----
    mq=mq_open("MQ1",O_RDWR | O_CREAT, 0666,&attr);
    if(mq < 0) { // Bład
        perror("Kolejka MQ1");
        exit(-1);
    }
    for(i=0; i < 10 ;i++) {
        msg.type = 1;
        sprintf(msg.text,"Proces 1 komunikat %d",i);
        // Wysłanie komunikatu -----
        res = mq_send(mq,&msg,sizeof(msg),10);
        sleep(1);
    }
    mq_close(mq);
}
```

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

```
// 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, res, prio;
    mqd_t mq;
    msg_tp msg;
    struct mq_attr attr;
    // Ustalenie atrybutów kolejki -----
    attr.mq_msgsize = sizeof(msg);
    attr.mq_maxmsg = 8;
    attr.mq_flags = 0;
    // 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++) {
        // Odbiór komunikatu -----
        res = mq_receive(mq,&msg,sizeof(msg),&prio);
        printf("Typ: %d text: %s\n",msg.typ,msg.text);
    }
    mq_close(mq);
}
```

Przykład 1-2 Kod procesu odbierającego komunikaty z kolejki MQ1  
- konsument



## 1.7 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**            lidentyfikator 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

## 1.8 Odbieranie komunikatu z kolejki – wersja z ograniczeniem czasowym

Pobieranie komunikatu z kolejki komunikatów, ale z ograniczeniem czasowym, odbywa się za pomocą funkcji `mq_timedreceive`.

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

Znaczenie parametrów:

`mq` Identyfikator kolejki komunikatów,  
`*msg` Adres bufora na odbierany komunikat,  
`len` Maksymalna długość odbieranego komunikatu,  
`mprio` Priorytet odebranego komunikatu.  
`abs_timeout` Czas absolutny do którego komunikat musi zostać odebrany

```
struct timespec {
    time_t tv_sec;          /* seconds */
    long tv_nsec;          /* nanoseconds */
};
```

Gdy w kolejce nie pojawi się komunikat a upłynie czas `abs_timeout` to funkcja zakończy się z błędem  
Kod błędu `ETIMEDOUT`

## 1.9 Wysyłanie komunikatu z kolejki – wersja z ograniczeniem czasowym

Gdy kolejka komunikatów jest pełna to wysyłanie komunikatów funkcją `mq_send` spowoduje zablokowanie procesu bieżącego do czasu gdy nie zwolni się miejsce w kolejce.

Wysyłanie komunikatu do kolejki komunikatów, ale z ograniczeniem czasowym, odbywa się za pomocą funkcji `mq_timedsend`.

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

Znaczenie parametrów:

<code>mq</code>	Identyfikator kolejki komunikatów,
<code>*msg</code>	Adres bufora na wysyłany komunikat,
<code>len</code>	Maksymalna długość wysyłany komunikatu,
<code>mprio</code>	Priorytet wysyłanego komunikatu.
<code>abs_timeout</code>	Czas absolutny do którego komunikat musi zostać wysłany

```
struct timespec {
    time_t tv_sec;           /* seconds */
    long tv_nsec;          /* nanoseconds */
};
```

Gdy w kolejce nie pojawi się dość miejsca na pomieszczenie komunikatu a upłynie czas `abs_timeout` to funkcja zakończy się z błędem. Kod błędu `ETIMEDOUT`

### 1.10 Zawiadamianie procesu o pojawieniu się komunikatu w kolejce

1. Można spowodować aby pojawienie się komunikatu w pustej kolejce (a więc zmiana stanu kolejki z „pusta” na „niepusta”) powodowało zawiadomienie procesu bieżącego.
2. Zawiadomienie może mieć postać sygnału lub powodować uruchomienie wątku

```
int mq_notify(mqd_t mq, struct sigevent *notif)
```

Znaczenie parametrów:

**mq** Identyfikator kolejki komunikatów,

**\*notif** Adres struktury typu **sigevent** specyfikującego sposób zawiadomienia.

```
union sigval { /* Dane przekazywane z zawiadomieniem */
  int sival_int; /* Wartość int */
  void *sival_ptr; /* Wskaźnik */
};

struct sigevent {
  int sigev_notify; /* Metoda zawiadomienia */
  int sigev_signo; /* Sygnał zawiadomienia */
  union sigval sigev_value; /* Dane przekazywane z
                             zawiadomieniem */
  void (*sigev_notify_function)(union sigval);
  /* Funkcja do tworzenia wątku (SIGEV_THREAD) */
  void *sigev_notify_attributes;
  /* Atrybuty tworzonego wątku (SIGEV_THREAD) */
  pid_t sigev_notify_thread_id;
  /* Identyfikator wątku (SIGEV_THREAD_ID) */
};
```

Tab. 1-3 Deklaracja typu sigevent

Pole **sigev\_signo** interpretowane jest jako numer sygnału który będzie wysłany gdy w kolejce pojawi się komunikat. W procesie należy zdefiniować sposób obsługi tego sygnału.

SIGEV_NONE	brak akcji
SIGEV_SIGNAL	wysłanie sygnału
SIGEV_THREAD	uruchomienie wątku

Tab. 1-4 Specyfikacja akcji przy zmianie statusu kolejki z pusty na niepusty

### 1.11 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)
```

### 1.12 Interfejs /proc

W katalogu /proc znajdują się pliki za pomocą których można testować parametry kolejek komunikatów

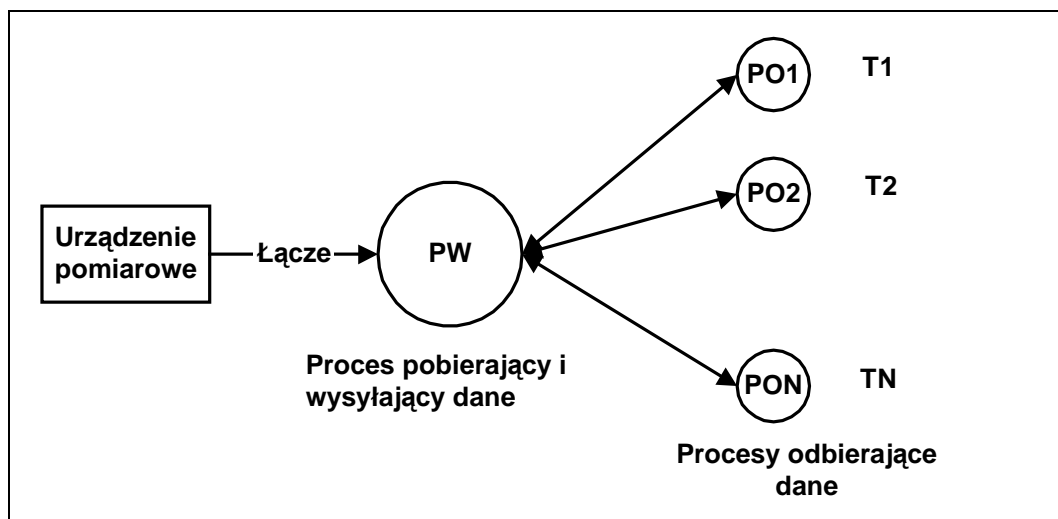
–

<code>/proc/sys/fs/mqueue/msg_max</code>	maksymalna długość kolejki – domyślnie 10
<code>/proc/sys/fs/mqueue/msgsize_max</code>	maksymalna długość komunikatu – domyślnie 8192
<code>/proc/sys/fs/mqueue/queues_max</code>	maksymalna liczba kolejek – domyślnie 256

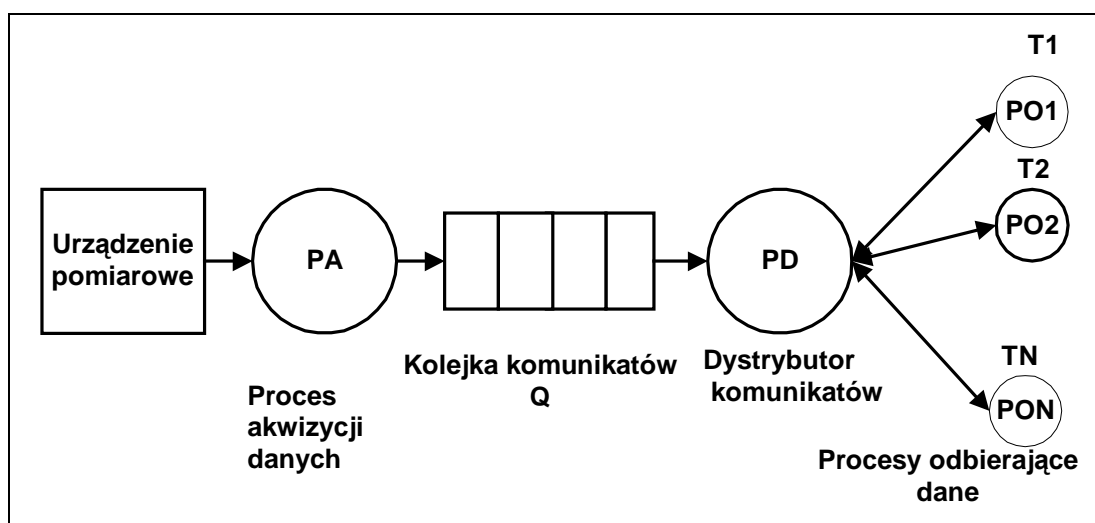
### 1.13 Montowanie kolejek komunikatów w systemie plików

```
# mkdir /dev/mqueue
# mount -t mqueue none /dev/mqueue
```

### 1.14 Przykład zastosowanie kolejki komunikatów w systemie akwizycji danych.



Rys. 1-4 Proces akwizycji danych PW przesyła wyniki do N procesów PO odbierających dane za pomocą komunikatów



Rys. 1-5 Akwizycja i dystrybucja danych odbywa się poprzez dwa procesy PA i PD połączone kolejką komunikatów Q.