Programowanie Współbieżne

Komunikacja między procesowa Posix IPC
Kolejki komunikatów

Posix IPC

W standardzie Posix są 3 rodzaje komunikacji między procesowej.

- Posiksowe kolejki komunikatów
- Posiksowe semafory
- Posiksowa pamięć wspólna

Podczas linkowania trzeba dodać -lrt

Posix IPC

	Kolejka komunikatów	Semafor	Pamięć wspólna
plik nagłówkowy	<mqueue.h>	<semaphore.h>	<sys/mman.h>
funkcja do tworzenia, otwierania lub usuwania	mq_open mq_close mq_unlink	sem_open sem_close sem_unlink sem_init semdestroy	shm_open shm_unlink
funkcje operacji sterujących	mq_getattr mq_setattr		ftruncate fstat
funkcje komunikacji	mq_send mq_receive mq_notify	sem_wait sem_trywait sem_post sem_getvalue	mmap munmap

Posix IPC

Nazwy odnoszące się do komunikacji międzyprocesowej

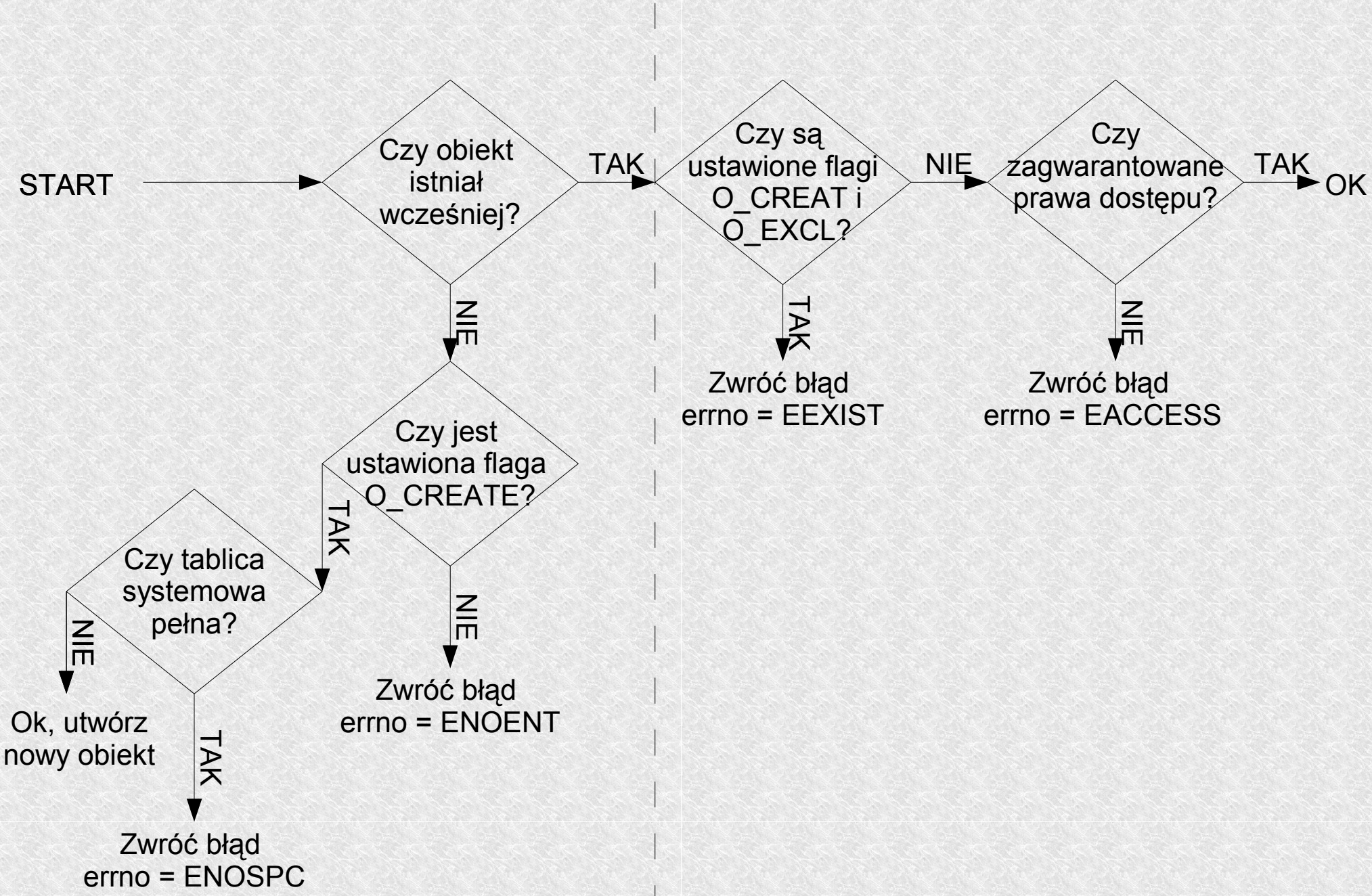
- Nazwa zgodna z nazwami ścieżek (kończy się '\0' i ma co najwyżej PATH_MAX
- Jeżeli zaczyna się od '/' wszystkie wywołania odnoszą się do tej samej kolejki jeśli nie to zależy od implementacji
- Interpretacja pozostałych znaków '/' również zależy od implementacji

Posix IPC

Tworzenie i otwieranie kanałów Posixowych, wartości ***flag***

	mq_open	sem_open	shm_open
Tylko do odczytu	O_RDONLY		O_RDONLY
Tylko do zapisu	O_WRONLY		
Do odczytu i zapisu	O_RDWR		O_RDWR
Utworzenie, gdy wcześniej nie istniał	O_CREATE	O_CREAT	O_CREAT
Utworzenie na wyłączność	O_EXCL	O_EXCL	O_EXCL
Tryb bez blokowania	O_NBLOCK		
Skrócenie, gdy już istniał			O_TRUNC

O_TRUNC – gdy ustawiony, to gdy istniejący obiekt pamięci wspólnej jest otwierany w trybie RW, to wskazany obiekt ma mieć nadany rozmiar 0



Tworzymy nowy obiekt | Korzystamy z istniejącego obiektu

Posix IPC

- Gdy korzystamy z istniejącego obiektu sprawdzanie praw dostępu odbywa się na podstawie:
  - Bitów praw dostpępu przypisanych do obiektu IPC podczas jego tworzenia
  - Wymaganego trybu dostępu (O_RDONLY, O_WRONLY lub O_RDWR)
  - Obowiązującego id użytkownika oraz obowiązującego id grupy procesu.

Posix IPC

- Jądro systemu sprawdza:
  - Czy obowiązujący id użytkownika danego procesu to 0 (root)
  - Czy obowiązujący id użytkownika danego procesu == id właściciela obiektu IPC oraz odpowiednie bity dostępu pozwalają np na zapis, odczyt
  - jw. tylko dotyczy się grupy
  - czy ustawiony odpowiedni bit prawa dostępu dla innych.
- Jeżeli któryś z powyższych będzie spełniony to uzyskamy dostęp do obiektu IPC na podanych warunkach.

Kolejki Komunikatów Posix

- Podstawowe różnice do komunikatów IPC
  - Odczyt przekazuje zawsze najstarszy komunikat o najwyższym priorytecie, z systemu V można odczytać najstarszą wiadomość dowolnego typu
  - Kolejki komunikatów w posiksie mogą generować sygnał lub tworzyć wątek w chwili umieszczenia komunikatu

Kolejki Komunikatów Posix

- Każdy komunikat w kolejce ma następujące parametry
  - Priorytet - liczba całkowita bez znaku (Posix) lub liczbą całkowitą typu long
  - Rozmiar obszaru danych w komunikacie >=0
  - Dane (jeśli rozmiar > 0)

Kolejki Komunikatów Posix

- Do tworzenia kolejki komunikatów służy funkcja:

```
#include <mqueue.h>

mqd_t mq_open(const char *name, int oflag,...
   /* mode_t mode, struct mq_attr *atr */);
```

- **name** – nazwa zaczynająca się od '/'
- **oflag** – flagi, przy tworzeniu musi być ustawione O_CREATE)
- **mode** –słowo trybu dostępu
- **atr** - dodatkowe parametry, gdy NULL brana wartość domyślna

Kolejki Komunikatów Posix

- Zamknięcie kolejki:

```
#include <mqueue.h>

int mq_close(mqd_t *mqdes);
```

- ***mqdes*** **–** identyfikator kolejki

Jeżeli proces się kończy, to wszystkie otwarte przez niego kolejki są zamykane, nie zostają one jednak usunięte!

Kolejki Komunikatów Posix

- Usunięcie nazwy kolejki:

```
#include <mqueue.h>

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

- ***name*** **–** nazwa kolejki

Funkcja ta usuwa nazwę ***name*** wtedy gdy ilość otwarć jest > 0
kolejka nie zostanie jednak usunięta z systemu do czasu ostatniego wywołania ***mq_close***

Kolejki Komunikatów Posix

Usunięcie nazwy kolejki mq_unlink.c:

```
#include <stdlib.h>
#include <stdio.h>
#include <mqueue.h>

int main (int argc, char **argv)
{

if (0 > mq_unlink("/blabla"))
    {
    perror("blad usuwania kolejki");
    exit(1);
    }
exit(0);
}
```

Kolejki Komunikatów Posix

- Odczytanie i ustawienie atrybutów kolejki:

```
#include <mqueue.h>

int mq_getattr(mqd_t mqdes, struct mq_attr
  *attr);

int mq_setattr(mqd_t mqdes, const struct mq_attr
  *attr, struct mq_attr *old_attr);
```

- **mqdes –** identyfikator kolejki
- **attr –** atrybuty kolejki
- **old_attr –** jeśli != NULL zapisane zostaną stare atrybuty kolejki

Kolejki Komunikatów Posix

- Odczytanie i ustawienie atrybutów kolejki:

```
struct mq_attr {
long mq_flags; /*znaczniki kolejki np. O_NONBLOCK*/
long mq_maxmsg; /* maksymalna liczba komunikatów */
long mq_msgsize; /* maksymalny rozmiar komunikatu w bajtach */
long mq_curmsgs; /* bieżaca liczba kokmunikatów */
};
```

- Wskaźnik do tej struktury można przekazać jako czwarty parametr ***mq_open*** i tam ustawić ***mq_maxmsg*** i ***mq_msgsize***.
- ***mq_setattr*** może jedynie ustawić mq_flags
- liczby komunikatów nie możemy zmienić za pośrednictwem tych funkcji

Kolejki Komunikatów Posix

Przykład mq_create.c:

```
#include <stdlib.h>
#include <stdio.h>
#include <mqueue.h>

int main (int argc, char **argv)
{
int flags;
mqd_t mqd;
struct mq_attr mqat;

flags = O_RDWR | O_CREAT ;
/* flags |= O_EXCL; */

mqat.mq_maxmsg = 8;
mqat.mq_msgsize = 4096;

if (0 > (mqd = mq_open("/blabla",flags,0666,&mqat)))
    {
    perror("blad tworzenia kolejki");
    exit(1);
    }
mq_getattr(mqd,&mqat);
printf("mq_maxmsg: %lu\nmq_msgsize: %lu\nmq_curmsgs: %lu\n", mqat.mq_maxmsg,
mqat.mq_msgsize, mqat.mq_curmsgs);
mq_close(mqd);
exit(0);
}
```

Kolejki Komunikatów Posix

- Wysłanie komunikatu:

```
#include <mqueue.h>

int mq_send(mqd_t mqdes, const char *ptr, size_t
  len, unsigned int prio);
```

- **mqdes –** identyfikator kolejki
- **ptr** - początek danych
- **len** - rozmiar danych
- **prio** – priorytet, liczba całkowita bez znaku. Posix wymaga by można było ustawiać conajmniej 32 priorytety. Wartość ta to MQ_PRIO_MAX w pliku limits.h w przypadku linuxa 32768.

```
/* Wysłanie komunikatu mq_send.c */
#include <stdlib.h>
#include <stdio.h>
#include <mqueue.h>
#include <string.h>

int main (int argc, char **argv)
{
int flags;
mqd_t mqd;
struct mq_attr mqat;
char msg[256];
strcpy(msg,"ala ma kota1");

flags = O_RDWR | O_CREAT ;

if (0 > (mqd = mq_open("/blabla",flags,0666,NULL)))
    {
    perror("blad tworzenia kolejki");
    exit(1);
    }

if (0 > mq_send(mqd,msg,sizeof(msg),10))
    {
    perror("blad wyslania komunikatu");
    exit(1);
    }
mq_getattr(mqd,&mqat);
printf("mq_maxmsg: %lu\nmq_msgsize: %lu\nmq_curmsgs: %lu\n", mqat.mq_maxmsg,
mqat.mq_msgsize, mqat.mq_curmsgs);
mq_close(mqd);
exit(0);
}
```

Kolejki Komunikatów Posix

- Odbiór komunikatu:

```
#include <mqueue.h>

int mq_receive(mqd_t mqdes, char *ptr, size_t
  len, unsigned int *prio);
```

- ***mqdes*** – identyfikator kolejki
- ***ptr*** – adres bufora
- ***len*** - rozmiar danych, musi być conajmniej o wielkości maksymalnej wielkości komunikatu. Dla tego często przed odbiorem komunikatu powinniśmy odczytać pole mq_msgsize struktury mq_attr.
- ***prio*** – priorytet.

Z kolejki Posix zawsze odbierane są komunikaty o najwyższym priorytecie i najstarsze. Dopuszcza się by przekazać 0 bajtów. Funkcja ta zwróci -1 w przypadku błędu 0 jest akceptowalne.

```
/* Odbiór komunikatu mq_receive.c */
#include <stdlib.h>
#include <stdio.h>
#include <mqueue.h>
#include <string.h>

int main (int argc, char **argv)
{
int flags;
mqd_t mqd;
struct mq_attr mqat;
char msg[4096];
unsigned int prio = 0;

flags = O_RDWR | O_CREAT ;

if (0 > (mqd = mq_open("/blabla",flags,0666,NULL)))
    {
    perror("blad tworzenia kolejki");
    exit(1);
    }
if (0 > mq_receive(mqd,msg,4096,&prio))
    {
    perror("blad odbioru komunikatu");
    exit(1);
    }
printf("Otrzymalem komunikat o priorytecie %u\n",prio);
printf("[%s]\n",msg);
mq_getattr(mqd,&mqat);
printf("mq_maxmsg: %lu\nmq_msgsize: %lu\nmq_curmsgs: %lu\n", mqat.mq_maxmsg,
mqat.mq_msgsize, mqat.mq_curmsgs);
mq_close(mqd);
exit(0);
}
```

Kolejki Komunikatów Posix

Informowanie o komunikacie:

```
#include <mqueue.h>
int mq_notify(mqd_t mqdes, const stuct sigevent
   *notification);
```

- *notification* – struktura związana z sygnałami czasu rzeczywistego Posix.1

```
struct sigevent {
int sigev_notify; //SIGEV_{NONE, SIGNAL, THREAD,
THREAD_ID}
int sigev_signo; // jeżeli SIGEV_SIGNAL to nr
sygnału
union sigval sigev_value; //przekazywane
procedurze obsługi albo wątkowi
//jeżeli SIGEV_THREAD to
void (*sigev_notify_function)(union sigval);
pthreaed_attr_t *sigev_notify_attributes;
}
```

Kolejki Komunikatów Posix

Informowanie o komunikacie:

```
#include <mqueue.h>
union sigval {
   int sival_int;
   void *sival_ptr;
};
```

Kolejki Komunikatów Posix

- Gdy ***notification***!= 0 to proces chce być powiadamiany kiedy w określonej kolejce pojawia się komunikat, a kolejka ta była pusta.
- Gdy ***notification***== 0 a proces był zarejestrowany do powiadamiania to informacja ta jest kasowana.
- Dla jednej kolejki, w danej chwili tylko jeden proces może być zarejestrowany do powiadamiania.
- Powiadomienie nie jest wysyłane gdy proces został zablokowany na mq_receive czekając na odbiór z danej kolejki komunikatów.
- gdy zostanie wysłane powiadomienie to rejestracja jest kasowana i proces powinien znowu wywołać mq_notify.
- informowanie po raz drugi nie wystąpi zanim kolejka nie będzie pusta dla tego ponowne mq_notify powinno być przed mq_receive.

Kolejki Komunikatów Posix

Przykład mq_notify1.c:

```
#include <stdlib.h>
#include <stdio.h>
#include <mqueue.h>
#include <string.h>
#include <signal.h>

mqd_t mqd;
int flags;
char *msg;
struct sigevent sigev;
struct mq_attr mqat;

static void sig_usr1(int signo)
{
ssize_t n;
mq_notify(mqd,&sigev);
n = mq_receive(mqd,msg, mqat.mq_msgsize, NULL);
printf("SIGUSR! odebrał %ld bajtów\n", (long) n);
printf("[%s]\n",msg);
return;
}
```

Kolejki Komunikatów Posix

Przykład mq_notify1.c:

```
int main (int argc, char **argv)
{
flags = O_RDWR | O_CREAT ;

if (0 > (mqd = mq_open("/blabla",flags,0666,NULL)))
    {
    perror("blad tworzenia kolejki");
    exit(1);
    }
mq_getattr(mqd,&mqat);
printf("mq_maxmsg: %lu\nmq_msgsize: %lu\nmq_curmsgs: %lu\n",
    mqat.mq_maxmsg, mqat.mq_msgsize, mqat.mq_curmsgs);
msg = malloc(mqat.mq_msgsize);
signal(SIGUSR1, sig_usr1);
sigev.sigev_notify = SIGEV_SIGNAL;
sigev.sigev_signo = SIGUSR1;
mq_notify(mqd,&sigev);
for (;;)
    {
    sleep(1);
    printf("czekam\n");
    }
mq_close(mqd);
exit(0);
}
```

Kolejki Komunikatów Posix

- Nieprawidłowości:
- W procedurze obsługi sygnału w standardzie Posix.1 powinny znaleźć się jedynie funkcje określane jako (async-signal-safe).

access	execle	getuid	setpgid	Sleep	umask
aio_return	execve	kill	setsid	stat	uname
aio_suspend	_exit	link	setuid	sysconf	unlink
alarm	fcntl	lseek	sigaction	tcdrain	utime
cfgetispeed	fdatasync	mkdir	sigaddset	tcflow	wait
cfgetospeed	fork	mkfifo	sigdelset	tcflush	waitpid
cfsetispeed	fpathcconf	open	sigemptyset	tcgetattr	wirte
cfsetospeed	fstat	pathconf	sigfillset	tcgetpgrp	
chdir	fsync	pause	sigismember	tcsendbreak	
chmod	getegid	pipe	signal	tcsetattr	
chown	geteuid	raise	sigpause	tcsetpgrp	
clock_gettime	getgid	read	sigpending	time	
close	getgroups	rename	sigprocmask	timer_getoverrun	
creat	getpgrp	rmdir	sigqueue	timer_gettime	
dup	getpid	sem_post	sigset	timer_settime	
dup2	getppid	setgid	sigsuspend	times	

Kolejki Komunikatów Posix

Można przerobić tak program by funkcja obsługi sygnału jedynie zmieniała jakąś zmienną globalną która by była odpowiedzialna za pobranie komunikatu. Pamiętać należy że sygnał jest jedynie generowany w momencie pojawienia się pierwszego komunikatu w pustej kolejce.

Dobrym pomysłem jest uruchomienie podczas nadejścia komunikatu wątku który by go odebrał.

W tym celu trzeba sigev_notify ustawić na SIGEV_THREAD

```
/*Przykład mq_notify2.c:*/
#include <stdlib.h>
#include <stdio.h>
#include <mqueue.h>
#include <errno.h>
#include <pthread.h>
#include <unistd.h>
extern int errno;
mqd_t mqd;
int flags;
struct sigevent sigev;
struct mq_attr mqat;

static void notify_thread(union sigval arg)
{
ssize_t n;
char *msg;
printf("notify_thread START\n");
msg = malloc(mqat.mq_msgsize);
mq_notify(mqd,&sigev);
while (( n = mq_receive(mqd,msg, mqat.mq_msgsize, NULL)) >= 0)
    {
    printf("THREAD odebrał %ld bajtów\n", (long) n);
    printf("[%s]\n",msg);
    }
if (errno != EAGAIN)
    perror("błąd odbioru");
free(msg);
pthread_exit(NULL);
printf("notify_thread STOP\n");
return;
}
```

```
int main (int argc, char **argv)
{
flags = O_RDWR | O_CREAT ;

if (0 > (mqd = mq_open("/blabla",flags,0666,NULL)))
    {
    perror("blad tworzenia kolejki");
    exit(1);
    }
mq_getattr(mqd,&mqat);
printf("mq_maxmsg: %lu\nmq_msgsize: %lu\nmq_curmsgs:
%lu\n",mqat.mq_maxmsg,mqat.mq_msgsize,mqat.mq_curmsgs);
sigev.sigev_notify = SIGEV_THREAD;
sigev.sigev_value.sival_ptr = NULL;
sigev.sigev_notify_function = notify_thread;
sigev.sigev_notify_attributes = NULL;
mq_notify(mqd,&sigev);
for (;;)
    {
    sleep(1);
    printf("czekam\n");
    }
mq_close(mqd);
exit(0);
}
```

Kolejki Komunikatów Posix

Gdy sigev_notify = SIGEV_THREAD_ID to określony wątek dostanie sygnał.