Mutex-y

[**Mutex**](https://pl.wikipedia.org/wiki/Mutex) (***MUT****ual****EX****clusion*, wzajemne wykluczanie) jest blokadą, którą może uzyskać **tylko jeden wątek**. Mutexy służą głównie do realizacji sekcji krytycznych, czyli bezpiecznego w sensie wielowątkowym dostępu do zasobów współdzielonych.

Schemat działania na mutexach jest następujący:

1. pozyskanie blokady
2. modyfikacja lub odczyt współdzielonego obiektu
3. zwolnienie blokady

Mutex w **pthreads** jest opisywany przez strukturę typu pthread\_mutex\_t, zaś jego atrybuty pthread\_mutexattr\_t.

Obiekt pthread\_mutex\_t (wzajemnego wykluczania) musi być przed użyciem zainicjalizowany, co odbywa się za pomocą funkcji [pthread\_mutex\_init](http://www.linuxmanpages.com/man3/pthread_mutex_init.3thr.php).

**int** pthread\_mutex\_init (**pthread\_mutex\_t** \*mutex, **pthread\_mutexattr\_t** \*attr);

* mutex - wcześniej stworzony przez nas mutex
* attr - atrybuty tworzonego mutexu (domyślne ustawienia: NULL)

Zablokowanie obiektu przez wątek może zostać wykonane poprzez jedną z następujących funkcji:

* **int** pthread\_mutex\_lock(**pthread\_mutex\_t** \*mutex) - która jeśli obiektu mutex-a jest zablokowany przez inny wątek usypia obecny wątek, aż mutex zostanie odblokowany. Z kolei, jeśli obiekt mutex-a jest już zablokowany przez obecny wątek to albo:
  + usypia wywołujący ją wątek (jeśli jest to mutex typu "fast")
  + zwraca natychmiast kod błędu EDEADLK (jeśli jest to mutex typu "error checking")
  + normalnie kontynuuje prace, zwiększając licznik blokad mutex-a przez dany wątek (jeśli mutex jest typu "recursive"); odpowiednia liczba razy odblokowań musi nastąpić, aby mutex powrócił do stanu "unlocked"
* **int** pthread\_mutex\_trylock(**pthread\_mutex\_t** \*mutex) - która zachowuje się podobnie jak powyższa, z tym że obecny wątek nie jest blokowany jeśli mutex jest już zablokowany przez inny wątek,a jedynie ustawia flagę EBUSY.
* [pthread\_mutex\_timedlock](http://www.opengroup.org/onlinepubs/009695399/functions/pthread_mutex_timedlock.html)- jest rozwinięciem funkcji pthread\_mutex\_lock - podawany jest maksymalny czas czekania wątku na odblokowanie (zablokowanego przez inny wątek) mutex-a.

Odblokowanie mutex-a wykonywane jest za pomocą funkcji [pthread\_mutex\_unlock](http://www.linuxmanpages.com/man3/pthread_mutex_unlock.3thr.php).

|  |
| --- |
| **Ostrzeżenie:**Należy zwrócić uwagę, aby nie używać mutex-ów w funkcjach obsługujących  sygnały (signal handlers), gdyż może to doprowadzić do zablokowania się programu. |

Jednym z najprostszych zastosowań mutex-ów jest ochrona zmiennej przed równoczesnym dostępem z wielu wątków. Pokazuje to następujący przykład:

|  |
| --- |
| Chronienie dostępu do zmiennej |
| int x;  pthread\_mutex\_t x\_mutex = PTHREAD\_MUTEX\_INITIALIZER;  void my\_thread\_safe\_function(...) {  /\* Każdy dostęp do zmiennej x powinien się odbywać w następujący sposób: \*/  pthread\_mutex\_lock(&x\_mutex);  /\* operacje na x... \*/  pthread\_mutex\_unlock(&x\_mutex);  }  ... |

Zakleszczenia

Zakleszczenia są najczęściej spowodowane nieprawidłowym używaniem mechanizmu mutexów przez programistę. Może to wystąpić np. w sytuacji, gdy próbujemy zamknąć mutex w wątku, w którym już wcześniej to zrobiliśmy. Zachowanie programu jest wtedy uzależnione od typu używanego mutexu.

Wyróżniamy trzy rodzaje mutexów:

* **Szybki mutex** (fast mutex) – domyślny typ, próba zamknięcia takiego mutexu z wątku, w którym wcześniej już to zrobiliśmy spowoduje zakleszczenie.
* **Rekursywny mutex** – nie powoduje zakleszczenia. Zapamiętuje, ile razy wątek zablokował danego mutexa i oczekuje na taką samą ilość odblokowań.
* **Mutex sprawdzający błędy** (error-checking mutex) – taka próba spowoduje błąd **EDEADLK** podczas wywoływania pthread\_mutex\_lock

By stworzyć niestandardowe mutexy używamy następujących typów i funkcji: **int** pthread\_mutexattr\_init(\*attr) - inicjalizuje zmienną z atrybutami  
**int** pthread\_mutexattr\_destroy(**pthread\_mutexattr\_t** \*attr) - zwalnianie  
**int** pthread\_mutexattr\_settype(**pthread\_mutexattr\_t** \*attr, int type) - ustawia typ mutexu  
**int** pthread\_mutexattr\_gettype(**const pthread\_mutexattr\_t** \*attr, int \*type) - pobiera typ mutexu

* **pthread\_mutexattr\_t attr** - zmienna przechowująca atrybuty mutexa
* **type** - typ mutexu - może przyjmować następujące wartości:

PTHREAD\_MUTEX\_NORMAL  
PTHREAD\_MUTEX\_ERRORCHECK  
PTHREAD\_MUTEX\_RECURSIVE

Współdzielenie mutexu z innymi procesami

Funkcje

* int pthread\_mutexattr\_setpshared(pthread\_mutexattr\_t \*attr, int pshared)
* int pthread\_mutexattr\_getpshared(const pthread\_mutexattr\_t \*attr, int \*pshared)

Zmiana priorytetu wątku posiadającego blokadę

Dostępne, gdy istnieje rozszerzenie [TPP](https://pl.wikibooks.org/wiki/POSIX_Threads/Wst%C4%99p#Opcje_standardu) (oraz [TPI](https://pl.wikibooks.org/wiki/POSIX_Threads/Wst%C4%99p#Opcje_standardu)).

Wartość atrybutu decyduje o strategii wykonywania programu, gdy wiele wątków o różnych priorytetach stara się o uzyskanie blokady. Atrybut może mieć wartości:

1. PTHREAD\_PRIO\_NONE,
2. PTHREAD\_PRIO\_PROTECT,
3. PTHREAD\_PRIO\_INHERIT (opcja [TPI](https://pl.wikibooks.org/wiki/POSIX_Threads/Wst%C4%99p#Opcje_standardu)).

W przypadku **PTHREAD\_PRIO\_NONE** priorytet wątku, który pozyskuje blokadę nie zmienia się.

W dwóch pozostałych przypadkach z mutexem powiązany zostaje pewien priorytet i gdy wątek uzyska blokadę, wówczas jego priorytet jest podbijany do wartość z mutexu (o ile oczywiście był wcześniej niższy). Innymi słowy w obrębie sekcji krytycznej wątek może działać z wyższym priorytetem.

Sposób ustalania priorytetu mutexu zależy od atrybutu:

* PTHREAD\_PRIO\_INHERIT - wybierany jest maksymalny priorytet spośród wątków oczekujących na uzyskanie danej blokady;
* PTHREAD\_PRIO\_PROTECT - priorytet jest ustalany przez programistę funkcją pthread\_mutexattr\_setprioceiling lub pthread\_mutex\_setprioceiling (opisane w następnej sekcji).

Dodatkowo, jeśli wybrano wartość PTHREAD\_PRIO\_PROTECT, wówczas wszelkie próby założenia blokady funkcjami pthread\_mutex\_XXXlock z poziomu wątków o priorytecie niższym niż ustawiony dla mutexa nie powiodą się - zostanie zwrócona wartość błędu EINVAL.

Funkcje

* int pthread\_mutexattr\_setprotocol(pthread\_mutexattr\_t \*attr, int protocol)  ([*doc*](http://www.opengroup.org/onlinepubs/000095399/functions/pthread_mutexattr_setprotocol.html))
* int pthread\_mutexattr\_getprotocol(const pthread\_mutexattr\_t \*attr, int \*protocol)  ([*doc*](http://www.opengroup.org/onlinepubs/000095399/functions/pthread_mutexattr_getprotocol.html))

Semafory nienazwane

* Nowy semafor jest tworzony przez funkcję [sem\_init](http://www.linuxmanpages.com/man3/sem_init.3thr.php)
* Operacja Sygnalizuj jest wykonywana za pomocą [sem\_post](http://www.linuxmanpages.com/man3/sem_post.3thr.php), która zwiększa licznik semafora. Funkcja ta nigdy nie powoduje blokady wywołującego ją wątku.
* Operacja Czekaj jest wykonywana za pomocą jednej z następujących funkcji:
  + [sem\_wait](http://www.linuxmanpages.com/man3/sem_wait.3thr.php) - proces jest usypiany aż dany semafor ma nie-zerową wartość, po czym następuje atomowa operacja zmniejszenia licznika semafora
  + [sem\_trywait](http://www.linuxmanpages.com/man3/sem_trywait.3thr.php) - jest nie blokującym wariantem sem\_wait - jeśli semafor ma zerowy licznik, funkcja nie usypia wątku, lecz zwraca -1 u stawia numer błędu na EAGAIN
* Semafor jest usuwany za pomocą [sem\_destroy](http://www.linuxmanpages.com/man3/sem_destroy.3thr.php)

Warunki Sprawdzające (Condition Variables)

Czasami konieczne jest monitorowanie przez wątek pewnych warunków. Implementacja ich sprawdzania z wykorzystaniem konwencjonalnych mechanizmów (ciągłego sprawdzania czy warunek jest spełniony) byłaby mocno nieefektywna, gdyż powodowałaby, że wątek byłby ciągle zajęty.

Rozwiązaniem tego problemu jest mechanizm warunków sprawdzających ([Condition Variables](http://www.linuxmanpages.com/man3/pthread_cond_init.3thr.php)). Pozwala on na uśpienie wątku aż do momentu, gdy pewne warunki na dzielonych danych zostaną spełnione.

Za pomocą zmiennych warunkowych możemy wstrzymywać wykonywanie wątku, aż do momentu, gdy zajdzie określony przez nas warunek. Zmiennej warunkowej towarzyszy mutex, który zapewnia wyłączność w trakcie odczytu/zmiany wartości flagi.  
  
Gdy wątek dochodzi do sekcji zależnej od pewnego warunku (np. flagi), wykonywana jest sekwencja:

* Wątek zajmuje mutexa następnie sprawdza warunek.
* Jeżeli warunek jest spełniony – wtedy wątek wykonuje kolejne instrukcje.
* Jeśli warunek nie jest spełniony – wtedy wątek jednocześnie odblokowuje mutex i wstrzymuje działanie aż do spełnienia warunku (poinformowania o zmianie warunku przez inny watek).

Przy zmianie warunku muszą być podjęte następujące kroki:

* Zajęcie mutexa towarzyszącego zmiennej warunkowej
* Podjęcie akcji, która może zmienić warunek
* Zasygnalizowanie oczekującym wątkom zmiany warunku.
* Odblokowanie mutexa.

Zmienne warunkowe przechowywane są w typie pthread\_cond\_t.  
Zmienne warunkowe obsługują następujące funkcje:

**int** pthread\_cond\_init(**pthread\_cond\_t** \*cond, pthread\_condattr\_t \*attr) - inicjalizacja zmiennej  
**int** pthread\_cond\_destroy(**pthread\_cond\_t** \*cond) - usunięcie zmiennej  
**int** pthread\_cond\_wait(**pthread\_cond\_t** \*cond, pthread\_mutex\_t \*mutex) - ustawia wątek w tryb oczekiwania w czasie, którego Mutex jest odblokowany  
**int** pthread\_cond\_timedwait(**pthread\_cond\_t** \*cond, pthread\_mutex\_t \*mutex, const struct timespec \*timeout) - usypia wątek na określoną ilość czasu  
**int**pthread\_cond\_broadcast(**pthread\_cond\_t** \*cond) - powiadamia wszystkie oczekujące wątki  
**int** pthread\_cond\_signal(**pthread\_cond\_t** \*cond) - powiadamia tylko jeden wątek

Zmienna warunkowa może być również inicjalizowana makrem **PTHREAD\_COND\_INITIALIZER**, które zainicjalizuje ją standardowymi atrybutami.  
  
Do obsługi atrybutów służa funkcje: **int** pthread\_condattr\_init(**pthread\_condattr\_t** \*attr) - inicjalizacja  
**int** pthread\_condattr\_destroy(**pthread\_condattr\_t** \*attr) - zwolnianie  
**int** pthread\_condattr\_getpshared(**const pthread\_condattr\_t** \*attr, **int** \*pshared) - pobiera atrybut process-shared  
**int** pthread\_condattr\_setpshared(**pthread\_condattr\_t** \*attr, **int** pshared) - ustawia atrybut process-shared

Poniżej znajduje się "urywek" kodu wykorzystujący warunki sprawdzające. Dwie zmienne dzielone (x i y) są sprawdzana pod kątem większości x od y. Każda zmiana dowolnej ze zmiennych powoduje obudzenie "zainteresowanych" wątków.

|  |
| --- |
| Przykład użycia warunków sprawdzających |
| int x,y;  pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;  pthread\_cond\_t cond = PTHREAD\_COND\_INITIALIZER;  ...  // (Watek 1)  // Czekanie aż x jest większe od y jest  // przeprowadzane następująco:  pthread\_mutex\_lock(&mutex);  while (x <= y) {  pthread\_cond\_wait(&cond, &mutex);  }  ...  pthread\_mutex\_unlock(&mutex);  ...  // (Watek 2)  // Kazda modyfikacja x lub y może  // powodować zmianę warunków. Należy  // obudzić pozostałe wątki, które korzystają  // z tego warunku sprawdzającego.  pthread\_mutex\_lock(&mutex);  /\* zmiana x oraz y \*/  if (x > y)  pthread\_cond\_broadcast(&cond);  pthread\_mutex\_unlock(&mutex); |