Spis treści

1 Pojęcia podstawowe potrzebne do zrozumienia tematu **2**

2 pthread_create **2**
2.1 Argumenty wywołania . . . 3
2.2 Przykład wywołania . . . 3
2.3 Wartość zwracana . . . 3

3 pthread_join **4**
3.1 Argumenty wywołania . . . 4
3.2 Przykład wywołania . . . 4
3.3 Wartość zwracana . . . 4

4 Synchronizacja wątków - MUTEXY **5**

5 pthread_mutex_lock oraz pthread_mutex_unlock **5**
5.1 Argumenty wywołania . . . 5
5.2 Przykład wywołania . . . 6
5.3 Wartość zwracana . . . 6

6 Zmienne warunkowe **7**
6.1 pthread_cond_signal oraz pthread_cond_wait . . . 8
6.2 Argumenty wywołania . . . 8
6.3 Przykład wywołania . . . 8
6.4 Wartość zwracana . . . 9

7 Usuwanie wątków **9**
7.1 pthread_cancel . . . 9
7.2 pthread_testcancel . . . 10
7.3 Typ usunięcia wątku . . . 10

Na podstawie:

- *M.J.Rochkind - Programowanie w systemie UNIX dla zaawansowanych (Advanced UNIX Programming)*
- `https://ai.ia.agh.edu.pl/wiki/pl:dydaktyka:so:2017:labs:lab_intro`
- *Graham Glass, King Ables - Linux dla programistów i użytkowników*

1 Pojęcia podstawowe potrzebne do zrozumienia tematu

Wątek
Część programu wykonywana współbieżnie w obrębie jednego procesu; w jednym procesie może istnieć wiele wątków.
Każdy wątek **ma własny** licznik instrukcji, zegar oraz stos.
Wątki **współdzielą** dane procesu w którym są wykonywane, a więc dane globalne oraz zasoby takie jak otwarte pliki, obecny katalog itp.

Kolejność wykonywania wątków
Nie jest z góry określona, nie można przewidzieć który wątek zadziała najpierw, a który później (chyba, że używamy specjalnych funkcji np. czekających na wykonanie danego wątku).

Wątek startowy
Jest to wątek działający w wykonywanym procesie, np. uruchamiając program z funkcją main, wątek startowy to ten działający w funkcji main. Inne wątki tworzone są przez wątek startowy poprzez wywołania funkcji.

- Czym jest wątek ?
- Co współdzielą wątki ?
- Co wątki mają własne ?
- Ile wątków może mieć proces ?
- Jaka jest kolejność wykonywania wątków w procesie ?
- Czym jest wątek startowy ? Jak tworzone są wątki ?

2 pthread_create

Jest to funkcja, której użycie spowoduje stworzenie przez wątek startowy innego wątku.

```
// pthread_create - create thread
#include <pthread.h>
int pthread_create(
  pthread_t * thread_id, /* new threads ID */
  const pthread_attr_t * attr, /* attributes (or NULL) */
  void *(*start_fcn) (void *), /* starting function */
  void *arg /* arg to starting function */
); /* Returns 0 on success, error number on error *
```

2.1. Argumenty wywołania

- thread_id - wskaźnik na zmienną thread_id, musimy ją stworzyć zadeklarować w wątku, który tworzyć będzie nowy wątek
- attr - służy do ustawiania atrybutów wywołania przy pomocy innych funkcji, na nasze potrzeby wystarczy defaultowe wywołanie, czyli przekazanie NULL
- start_fcn - wskaźnik na funkcję, którą ma wykonywać nowo utworzony wątek
- arg - argumenty, które mają być przekazane jako wejście do funkcji nowego wątku, argument jest typu void aby można było przekazać zmienną jakiegokolwiek typu, często jednak należy sprawdzić czy zmienna ta się tam mieści np. wołając

```
assert(sizeof(long) <= sizeof(void *));
```

2.2. Przykład wywołania

```
static long x = 0; // wspoldzielona przez watki zmienna

static void *thread_func(void *arg) { // funkcja do przekazania dla nowego watku
  while (true) {
    printf("Thread 2 says %ld\n", ++x);
    sleep(1);
  }
}

int main(void) {

  pthread_t tid; // deklaracja id nowego watku

    ec_rv( pthread_create(&tid, NULL, thread_func, NULL) )
    // utworzenie nowego watku

    while (x < 10) {
      printf("Thread 1 says %ld\n", ++x); // dwa watki zmieniaja
      sleep(2);  // ta sama zmienna i ja wypisuja
    }

    return EXIT_SUCCESS;

  EC_CLEANUP_BGN
    return EXIT_FAILURE;
  EC_CLEANUP_END
}
```

2.3. Wartość zwracana

Funkcja zwraca 0 w przypadku sukcesu oraz kod błędu w przypadku błędu, błąd ten można sprawdzić przy pomocy makra ec_rv jak w powyższym programie.

- Do czego służy funkcja pthread_create ?
- Jakie są argumenty wywołania funkcji pthread_create ?
- Co zwraca funkcja pthread_create ?

3 pthread_join

Wątek może czekać na zakończenie innego wątku i przeczytać status, który on zwraca.

```
// pthread_join – wait for thread to terminate
#include <pthread.h>
int pthread_join(
  pthread_t thread_id, /* ID of thread to join */
  void **status_ptr /* returned exit status (if not NULL arg) */
); /* Returns 0 on success, error number on error */
```

3.1. Argumenty wywołania

- thread_id - id wątku na który oczekuje wątek
- status_ptr - zmienna przechowująca status wątku na który czekamy

3.2. Przykład wywołania

```
// pthread_join – wait for thread to terminate

static long x = 0; // wspoldzielona zmienna

static void *thread_func(void *arg) { // funkcja nowego watku
  while (x < (long)arg) {
    printf("Thread 2 says %ld\n", ++x);
    sleep(1);
  }

  return (void *)x;
}


int main(void) {
  pthread_t tid;  // id nowego watku
  void *status; // zmienna przechowujaca status zwracany przez watek

    assert(sizeof(long) <= sizeof(void *)); // sprawdzenie czy sie miesci

    ec_rv( pthread_create(&tid, NULL, thread_func, (void *)6) )

    while (x < 10) {
      printf("Thread 1 says %ld\n", ++x);
      sleep(2);
    }

    ec_rv( pthread_join(tid, &status) ) // czekaj na watek z id rownym tid

    printf("Thread 2's exit status is %ld\n", (long)status);

    return EXIT_SUCCESS;

  EC_CLEANUP_BGN
    return EXIT_FAILURE;
  EC_CLEANUP_END }
```

3.3. Wartość zwracana

Funkcja pthread_join zwraca 0 w przypadku sukcesu oraz kod błędu w przypadku błędu.

- Do czego służy funkcja pthread_join ?
- Jakie są argumenty wywołania funkcji pthread_join ?
- Co zwraca funkcja pthread_join ?

4 Synchronizacja wątków - MUTEXY

Sekcja krytyczna
Część programu która powinna być wykonywana tylko przez jeden wątek w danej chwili czasu ze względu na spójność danych i bezpieczeństwo programu. Np. jeden wątek nie powinien nadpisywać zmiennej, która w danej chwili jest czytana przez inny wątek. Program powinien dawać gwarancje, że operacja zapisywanie skończy się przed rozpoczęciem czytania.

Mutex (mutual-exclusion)
Jest to obiekt, który pozwala na ochronę sekcji krytycznych programu w taki sposób aby tylko jeden wątek mógł wejść w tą sekcję. Ochrona taka jest zapewniania przez funkcje systemowe pthread_mutex_lock oraz pthread_mutex_unlock.

- Czym jest sekcja krytyczna ?
- Czym jest Mutex (mutual exclusion) ?

5 pthread_mutex_lock oraz pthread_mutex_unlock

Wzajemne wykluczenia działa na zasadzie:
Jeśli mutex jest zamknięty, to sekcja krytyczna jest blokowana do czasu odblokowania mutexa.

```
// pthread_mutex_lock - lock mutex
#include <pthread.h>
int pthread_mutex_lock(
  pthread_mutex_t *mutex /* mutex to lock */
); /* Returns 0 on success, error number on error */
```

```
// pthread_mutex_unlock - unlock mutex
#include <pthread.h>
int pthread_mutex_unlock(
  pthread_mutex_t *mutex /* mutex to unlock */
); /* Returns 0 on success, error number on error */
```

5.1. Argumenty wywołania

Argumentem wywołania tych funkcji jest zmienna, która reprezentuje Mutex. Do inicjalizacji tej zmiennej najlepiej wykorzystać gotowy inicjalizator:

```
static pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;
```

5.2. Przykład wywołania

```
static pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER; // inicjalizacja mutexa

static long x = 0; // zmienna wspoldzielona

static void *thread_func(void *arg) { // funkcja dla nowego watku
  bool done;
    while (true) {
      ec_rv( pthread_mutex_lock(&mtx) ) // zamkniecie mutexa
      done = x >= (long)arg;
      ec_rv( pthread_mutex_unlock(&mtx)) // otwarcie mutexa


      if (done)
        break;
      ec_rv( pthread_mutex_lock(&mtx) )
      printf("Thread 2 says %ld\n", ++x);
      ec_rv( pthread_mutex_unlock(&mtx) )
      sleep(1);
      }

      return (void *)x;
  EC_CLEANUP_BGN
    EC_FLUSH("thread_func")
    return NULL;
  EC_CLEANUP_END
}


int main(void) {
  pthread_t tid;
  void *status;
  bool done;

    assert(sizeof(long) <= sizeof(void *));

    ec_rv( pthread_create(&tid, NULL, thread_func, (void *)6) )
    while (true) {
      ec_rv( pthread_mutex_lock(&mtx) )
      done = x >= 10;
      ec_rv( pthread_mutex_unlock(&mtx) )
      if (done)
        break;
      ec_rv( pthread_mutex_lock(&mtx) )
      printf("Thread 1 says %ld\n", ++x);
      ec_rv( pthread_mutex_unlock(&mtx) )
      sleep(2);
    }
    ec_rv( pthread_join(tid, &status) )
    printf("Thread 2's exit status is %ld\n", (long)status);
    return EXIT_SUCCESS;

  EC_CLEANUP_BGN
    return EXIT_FAILURE;
  EC_CLEANUP_END
}
```

5.3. Wartość zwracana

Funkcje pthread_mutex_lock oraz pthread_mutex_unlock zwracają 0 w przypadku sukcesu oraz kod błędu w przypadku błędu.

- Do czego służą funkcje pthread_mutex_lock oraz pthread_mutex_unlock ?
- Jakie są argumenty wywołania funkcji pthread_mutex_lock oraz pthread_mutex_unlock ?
- Co zwracają funkcje pthread_mutex_lock oraz pthread_mutex_unlock ?

6 Zmienne warunkowe

Problem z wątkami, czyli dlaczego potrzebujemy zmiennych warunkowych

Rozważamy program w którym jeden proces czyta dane i zapisuje je do kolejki, drugi proces odbiera dane z kolejki i zapisuje je do bazy danych. Przykładowa implementacja takiego programu to:

Thread A	*Thread B*
1. Read data [B]	1. Lock *M* [b]
2. Lock *M* [b]	2. If item on queue, remove it and update database
3. Put item on queue	3. Unlock *M*
4. Unlock *M*	4. Goto step 1
5. Goto step 1	

Rysunek 1: Przykładowa implementacja

Problem z tym programem jest taki, że wątek B działa w nieskończonej pętli i traci duża zasobów CPU, lepiej byłoby gdyby wątek A poinformował wątek B o tym, że dodał dane, a wtedy wątek B mógłby je zapisać do bazy danych. Problem ten może być rozwiązany przy użyciu zmiennych warunkowych.

Zmienne warunkowe

Jest to mechanizm umożliwiający komunikację pomiędzy wątkami. Przy użyciu zmiennych warunkowych jeden wątek może poinformować drugi o tym, że wykonał już jakieś zadanie.

Thread A	*Thread B*
1. Read data [B]	1. Lock *M* [b]
2. Lock *M* [b]	2. while (queue is empty) {
3. Put item on queue	cond_wait(*C*, *M*) [B]
4. cond_signal(*C*)	}
5. Unlock *M*	3. Remove item; update database
6. Goto step 1	4. Unlock *M*
	5. Goto step 1

Rysunek 2: Przykładowa implementacja

Ważne: przy wywołanie cond_wait mutex M musi być zamknięty, a gdy funkcja ta zwraca (czyli po tym jak wątek A otworzył mutex M) jest on ponownie zamykany. W linuxie do obsługi zmiennych warunkowych używamy funkcji opisanych w następnym podrozdziale.

- Jaki problem rozwiązują zmienne warunkowe ?
- Czym są zmienne warunkowe ?

6.1. pthread_cond_signal oraz pthread_cond_wait

Są to funkcje, które służą do obsługi zmiennych warunkowych.

```
// pthread_cond_signal - signal condition
#include <pthread.h>
int pthread_cond_signal(
  pthread_cond_t *cond /* condition variable */
); /* Returns 0 on success, error number on error */
```

```
// pthread_cond_wait - wait for condition
#include <pthread.h>
int pthread_cond_wait(
  pthread_cond_t *cond, /* condition variable */
  pthread_mutex_t *mutex /* mutex */
); /* Returns 0 on success, error number on error */
```

6.2. Argumenty wywołania

Funkcje pthread_cond_signal oraz pthread_cond_wait przyjmują jako argumenty mutex oraz zmienną warunkową. Mutex został opisany już wcześniej, a zmienna warunkowa może być utworzona podobnie jak mutex przez makro, które automatycznie tworzy taką zmienną:

```
static pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

6.3. Przykład wywołania

```
static pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER; // stworzenie mutexa
static pthread_cond_t cond = PTHREAD_COND_INITIALIZER; // stworzenie zmiennej warunkowej

struct node {  // wezly ktore sa wkladane do kolejki
  int n_number;
  struct node *n_next;
} *head = NULL;


static void *thread_func(void *arg) {      // funkcja watku
  struct node *p;

    while (true) {
      ec_rv( pthread_mutex_lock(&mtx) )  // zamknij mutex
      while (head == NULL)
        ec_rv( pthread_cond_wait(&cond, &mtx) )  // czekaj na zmienna warunkowa
      p = head;
      head = head->n_next;
      printf("Got %d from front of queue\n", p->n_number);
      free(p);
      ec_rv( pthread_mutex_unlock(&mtx) ) // otworz mutex
    }

```

```
24      return (void *)true;
25
26    EC_CLEANUP_BGN
27      (void)pthread_mutex_unlock(&mtx);
28      EC_FLUSH("thread_func")
29      return (void *)false;
30    EC_CLEANUP_END
31    }
32
33    int main(void) {      // watek startowy
34      pthread_t tid;
35      int i;
36      struct node *p;
37
38        ec_rv( pthread_create(&tid, NULL, thread_func, NULL) )   // stworz watek
39        for (i = 0; i < 10; i++) {
40          ec_null( p = malloc(sizeof(struct node)) )
41          p->n_number = i;
42          ec_rv( pthread_mutex_lock(&mtx) )     // zamknij mutex
43          p->n_next = head;
44          head = p;
45          ec_rv( pthread_cond_signal(&cond) )      // poinformuj drugi watek ze skonczylem
46          ec_rv( pthread_mutex_unlock(&mtx) )      // otworz mutex
47          sleep(1);
48        }
49
50        ec_rv( pthread_join(tid, NULL) )     // czekaj na watek
51        printf("All done — exiting\n");
52        return EXIT_SUCCESS;
53
54      EC_CLEANUP_BGN
55        return EXIT_FAILURE;
56      EC_CLEANUP_END }
```

6.4. Wartość zwracana

Funkcje pthread_cond_wait oraz pthread_cond_signal zwracają 0 w przypadku sukcesu oraz kod błędu w przypadku błędu.

- Do czego służą funkjce pthread_cond_signal oraz pthread_cond_wait ?
- Jakie argumenty przyjmują funkcje pthread_cond_signal oraz pthread_cond_wait ?
- Co zwracają funkcje pthread_cond_signal oraz pthread_cond_wait ?

7 Usuwanie wątków

7.1. pthread_cancel

Jeden wątek może usunąć inny przez wywołanie funkcji pthread_cancel.

```
1  // pthread_cancel - cancel thread
2  #include <pthread.h>
3  int pthread_cancel(
4    pthread_t thread_id /* ID of thread to cancel */
5  ); /* Returns 0 on success, error number on error */
```

Funkcja ta przyjmuje jako argument id wątku, który chcemy usunąć.
Ważne: funkcja ta nie usuwa wątku od razu, ale czeka na jedno z wywołań systemowych takich jak np. read, waitpid, pthread_cond_wait w wątku który usuwamy. Tak wiec punktem usunięcia wątku jest potencjalnie każde wywołanie funkcji, ale ostatecznie najlepiej sprawdzić w dokumentacji.
Funkcja zwraca 0 w przypadku sukcesu oraz kod błędu dla błędu.
Jeśli w wątku nie ma żadnego punktu usunięcia wówczas wątek pozostaje żywy, dlatego właśnie potrzebna jest funkcji pthread_textcancel .

7.2. pthread_testcancel

Funkcja ta służy do ustawienia punktu usunięcia wątku poprzez jej wywołanie. Jeśli wątek nie jej przeznaczony do usunięcia wówczas ta funkcja nie ma żadnego działania.

```
\\ pthread_testcancel - test for cancellation
#include <pthread.h>
void pthread_testcancel(void);
```

7.3. Typ usunięcia wątku

Normalnie wątek usuwany jest tylko w punkcie usunięcia, ale tak naprawdę zależy to od ustawionego typu usunięcia wątku.

- PTHREAD_CANCEL_DEFERRED - to defaultowy typ, jego ustawienie powoduje, że wątek jest usuwany tylko w punkcie usunięcia wątku
- PTHREAD_CANCEL_ASYNCHRONOUS - to typ którego ustawienie powoduje natychmiastowe usunięcie wątku bez oczekiwania na punkt usunięcia

Tryby te mogą zostać ustawione przy użyciu funkcji *pthread_setcanceltype*.

- Co robi funkcja pthread_cancel ?
- Corobi funkcja pthread_testcancel ?
- Czym są typy usunięcia wątku ?

7.4. pthread_cleanup_push oraz pthread_cleanup_pop

Czasami potrzebne jest wykonanie jakiejś akcji przed zamknięciem wątku.
Np. pthread_cond_wait jest punktem usunięcia wątku, a przy tym wejście do tej funkcji powoduje zamknięcie Mutexa, nie chcemy aby po usunięciu wątku Mutex był zamknięty.
Problem ten rozwiązują funkcje pthread_cleanup_push oraz pthread_cleanup_pop.

```
pthread_cleanup_push-install cleanup handler
#include <pthread.h>
void pthread_cleanup_push(
  void (*handler)(void*), /* pointer to cleanup-handler function */
  void *arg /* data to pass to function */
);
```

```
pthread_cleanup_pop-uninstall cleanup handler
#include <pthread.h>
void pthread_cleanup_pop(
  int execute /* execute handler? */
);
```

Funkcja _push przyjmuje jako argument funkcję, która ma być wykonana gdy wątek zostanie usunięty, a funkcja _pop określa czy handler ma być wywołany również w przypadku gdy wątek normalnie się zakończył (true) czy nie (false).