Procesy 2 - Procesy i wątki		
Dominik Wróbel	04 IV 2019	Czw. 17:00

Spis treści

1 Tworzenie wątku

1.1. Tworzenia wątku

Zadanie 1

Zmodyfikuj program tak by tworzył 10 wątków, z których każdy wypisze swój numer przesłany jako argument wywołania funkcji rozpoczęcia.

Listing 1: Zadania 1 - hello.c

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
// ------------------------------------------------------

void *PrintHello(void *arg);
// ------------------------------------------------------

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

      const int THREADS_NUM = 10;
      pthread_t threads[THREADS_NUM];
      int i;

      for(i = 0; i < THREADS_NUM ; i++){ // tworzenie 10 watkow

         int rc = pthread_create(&threads[i], NULL, PrintHello, (void*)i);
         if (rc){
             printf("Return code: %d\n", rc);
             exit(-1);
         }
         sleep(1);

      }


      return 0;
}
// ------------------------------------------------------

void *PrintHello(void *arg){
      printf("Next boring 'Hello World!' version! Thread number %d\n", ((int*) arg));
      return NULL;
}
```

1.2. Czekanie na zakończenie wątku

Zadanie 2

Zmodyfikuj program z pierwszego zadania w taki sposób, aby wątek główny (main) czekał na zakończenie pracy przez pozostałe wątki. Aby przetestować poprawność rozwiązania zmień kod poprzedniego programu w następujący sposób:

- Usuń wywołanie sleep(1); z funkcji main.
- Dodaj identyczne wywołanie sleep(1); na samym początku funkcji rozpoczęcia.
- Bezpośrednio przed instrukcją return w funkcji main wypisz informację o zakończeniu wątku głównego.
- Zmodyfikuj program w celu oczekiwania na zakończenie wątku. Poprawnie zmodyfikowany program powinien wyświetlać taki oto komunikat:
  Next boring 'Hello World!' version!
  End of the main thread!

Listing 2: Zadanie 2 - hello.c

```
#include <pthread.h>
   #include <stdio.h>
   #include <stdlib.h>
   // ------------------------------------------------------
   void *PrintHello(void *arg);
   // ------------------------------------------------------
   int main(int argc, char *argv[]){

        const int THREADS_NUM = 10;
        pthread_t threads[THREADS_NUM];
        int i;

        for(i = 0; i < THREADS_NUM ; i++){

           int rc = pthread_create(&threads[i], NULL, PrintHello, (void*)i);
           if (rc){
               printf("Return code: %d\n", rc);
               exit(-1);
           }

        // sleep(1); I - usun wywolanie sleep(1) z main
        }

       for(i = 0 ; i < THREADS_NUM; i++){ // IV - zmodyfikuj program aby czekal na
   koniec watkow
           pthread_join(threads[i], NULL); // wait for a thread to finish
       }

        printf("End of the main thread!\n"); // III - przed return wypisz info. o
   zakoncz. watku glow.

        return 0;
   }
   // ------------------------------------------------------
   void *PrintHello(void *arg){
```

```
37          sleep(1); // II -dodaj identyczne wywolanie sleep(1) na poczatku f. rozpoczecia
38           printf("Next boring 'Hello World!' version! Thread number %d\n", ((int*) arg));
39           return NULL;
40  }
```

W wyniku działania programu uzyskano następujący wynik:

Listing 3: Zadanie 2 - hello.c - wynik dzialania

```
1  Next boring 'Hello World!' version! Thread number 6
2  Next boring 'Hello World!' version! Thread number 7
3  Next boring 'Hello World!' version! Thread number 5
4  Next boring 'Hello World!' version! Thread number 8
5  Next boring 'Hello World!' version! Thread number 9
6  Next boring 'Hello World!' version! Thread number 4
7  Next boring 'Hello World!' version! Thread number 3
8  Next boring 'Hello World!' version! Thread number 2
9  Next boring 'Hello World!' version! Thread number 1
10 Next boring 'Hello World!' version! Thread number 0
11 End of the main thread!
```

1.3. Synchronizacja wątków

Zadanie 3

...
Uzupełnij poniższy szkielet programu, tak aby realizował powyższą funkcjonalność:

Listing 4: Zadanie 3 - func.c

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

   #define NUM 4
   #define LENGTH 100
   // ---------------------------------------------------------

   typedef struct {
        long* a;
        long sum;
        int veclen;
   } CommonData;
   // ---------------------------------------------------------

   CommonData data;
   pthread_t threads[NUM];
   pthread_mutex_t mutex;

   void* calc(void* arg); // Funkcja rozpoczecia
   // ---------------------------------------------------------

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

        int threadCreated;
        long i, sum = 0;
        void* status;
        long* a = (long*) malloc (NUM*LENGTH*sizeof(long));
        pthread_attr_t attr;

        //Prepare data structure
        for (i=0; i<LENGTH*NUM; i++) {
             a[i] = i;
             sum += i;
        }

        data.veclen = LENGTH;
        data.a = a;
        data.sum = 0;

        //mutex initialization
        pthread_mutex_init(&mutex, NULL);

        //[1] setting thread attribute
        pthread_attr_init(&attr);
        pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

        for(i=0;i<NUM;i++){ // stworzeniu NUM liczby watkow
           threadCreated = pthread_create(&threads[i], &attr, calc, (void*) i); //
   tworzenie watkow
        }

        //join
```

```
53          for(i=0;i<NUM;i++) {
54               pthread_join(threads[i], &status);
55          }
56
57         //[1] destroy - not needed anymore
58          pthread_attr_destroy(&attr);
59
60          //Print
61          printf ("Correct result is: %ld \n", sum);
62          printf ("Function result is: %ld \n", data.sum);
63
64          //Clean
65          free (a);
66          pthread_mutex_destroy(&mutex);
67          return 0;
68     }
69     // ------------------------------------------------------------
70
71     void* calc(void* arg)
72     {
73          long* x = data.a;
74          long mysum = 0;
75          int i;
76
77          int partitionNumber = (int) arg;
78
79          printf("%d  ", partitionNumber);
80
81          int startIndex = partitionNumber * LENGTH;
82          int endIndex = startIndex + LENGTH;
83
84         printf("s: %d, e: %d", startIndex, endIndex);
85
86          for (i=startIndex; i<endIndex; i++){
87               mysum += x[i];
88          }
89
90         printf("%ld\n", mysum);
91         // sekcja krytyczna
92          pthread_mutex_lock(&mutex);
93          data.sum += mysum;
94          pthread_mutex_unlock(&mutex);
95         // koniec sekcji krytycznej
96
97          pthread_exit((void*) 0);
98     }
99 // ------------------------------------------------------------
```

W wyniku działania programu uzyskano:

Listing 5: Zadanie 3 - func.c - wynik dzialania

```
3  s: 300, e: 40034950
2  s: 200, e: 30024950
1  s: 100, e: 20014950
0  s: 0, e: 1004950
Correct result is: 79800
Function result is: 79800
```

1.4. Zmienne warunkowe

Zadanie 4

Rozbuduj poniższy program, który ma realizować prostą funkcjonalność:

- 2 wątki inkrementują (funkcja increment) wartość zmiennej globalnej globalvariable.
- Trzeci wątek (funkcja printinfo) oczekuje na sygnał, aby oznajmić, że osiągnięto żądaną wartość MAXVAL.
- Po wypisaniu informacji wszystkie wątki i cały program kończą działanie.

Listing 6: Zadanie 4 - cond.c

```c
#include <pthread.h>
    #include <stdio.h>
    #include <stdlib.h>
    // ---------------------------------------------------------

    #define MAXVAL 100

    int globalvariable = 0;
    pthread_mutex_t mutex;
    pthread_cond_t cond;

    void* increment(void*);
    void* printinfo(void*);
    // ---------------------------------------------------------

    int main(){

          pthread_t t1, t2, t3;
          pthread_attr_t attr;

          // mutex initialization
          pthread_mutex_init(&mutex, NULL);

          // conditional initialization
          pthread_cond_init(&cond, NULL);

          pthread_attr_init(&attr);
          pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

          pthread_create(&t1, &attr, increment, NULL);
          pthread_create(&t2, &attr, increment, NULL);
          pthread_create(&t3, &attr, printinfo, NULL);

          pthread_join(t1, NULL);
          printf("t1 finished!\n");
          pthread_join(t2, NULL);
          printf("t2 finished!\n");
          pthread_join(t3, NULL);
          printf("t3 finished!\n");

          printf("Finishing...\n");
          return 0;
    }
    // ---------------------------------------------------------

    void* increment(void* arg) {
```

```
47
48          while(1){
49              pthread_mutex_lock(&mutex); // zamknij Mutex
50
51              if(globalvariable == MAXVAL){ // jesli osiagnieta wartosc max
52                  pthread_cond_signal(&cond); // poinformuj watek wypisujacy
53                  pthread_mutex_unlock(&mutex); // otworz mutex
54                  break;
55              }
56
57              globalvariable++; // jesli nie osiagnieto max to inkrementuj
58              pthread_mutex_unlock(&mutex); // otworz mutex po inkrementacji
59          }
60
61
62
63           pthread_exit((void*) 0);
64      }
65      // ------------------------------------------------------------
66
67      void* printinfo(void* arg) {
68          int val;
69          int i;
70
71          pthread_mutex_lock(&mutex); // zamknij mutex
72
73          while(globalvariable < MAXVAL)
74              pthread_cond_wait(&cond, &mutex); // czekaj az dwa watki skoncza
75
76          pthread_mutex_unlock(&mutex); // otworz mutex
77
78          printf("Adding values completed, globalvariable is %d \n", globalvariable);
79
80          pthread_exit((void*) 0);
81      }
82  // ------------------------------------------------------------
```

W wyniku działania programu uzyskano wynik:

Listing 7: Zadanie 4 - cond.c - wynik dzialania

```
1 Adding values completed, globalvariable is 100
2 t1 finished!
3 t2 finished!
4 t3 finished!
5 Finishing...
```

1.5. Kasowanie wątku

Zadanie 5

Uzupełnij poniższy program o:

- Przesłanie do tworzonych wątków argumentów (wykorzystaj strukturę thread_params) zawierających informację o: Indeksie wątku, który jest równy wartości ti w pętli tworzącej wątki. Wartości szukanej.
- Informację o liczbie iteracji jakie wykonał każdy kończący się wątek w celu odnalezienia wartości (wykorzystaj mechanizmy czyszczące pthread_ cleanup_ push, pthread_ cleanup_ pop).

Listing 8: Zadanie 5 - randomsearch.c

```
#include <stdio.h>
    #include <unistd.h>
    #include <stdlib.h>
    #include <sys/types.h>
    #include <pthread.h>
    #include <errno.h>
    // ---------------------------------------------------------

    #define NUM_THREADS 5
    #define TARGET 100
    // ---------------------------------------------------------




    struct thread_params {
         int target; // I. wartosc szukana
         int thread_idx; // I. indeks watku
         int threadNumber;
         int *tries; // liczba iteracji
    };
    // ---------------------------------------------------------


   void thread_cancel_handler(void * arg){ // funkcja wolana gdy watek ginie
        struct thread_params * tp = (struct thread_params*) arg;
        printf("Thread %d, iteration %d\n" , tp->threadNumber, *(tp->tries));
    }


    struct thread_params tp[NUM_THREADS];
    pthread_t threads[NUM_THREADS];
    pthread_mutex_t mutex;
    int tries;

    void *search(void *arg);
    void cleanup(void *args);
    // ---------------------------------------------------------

    int main (int argc, char *argv[]){
         int ti;
         int target=TARGET;

         tries = 0;
```

```
46          pthread_mutex_init(&mutex, NULL);
47
48          printf("Searching for: %d\n", target);
49
50          for (ti=0;ti<NUM_THREADS;ti++){
51               tp[ti].target = target;
52               tp[ti].thread_idx = ti;
53               tp[ti].tries = &tries;
54               pthread_create(&threads[ti], NULL, search, &(tp[ti]));
55          }
56
57          for (ti=0;ti<NUM_THREADS;ti++){
58               pthread_join(threads[ti], NULL);
59          }
60
61          printf("Number of all iterations: %d.\n", tries);
62          pthread_mutex_destroy(&mutex);
63          return 0;
64     }
65     // ------------------------------------------------------------
66
67
68
69     void *search(void *arg){
70
71     struct thread_params * tp = (struct thread_params*) arg;
72
73     pthread_cleanup_push(thread_cancel_handler, &tp); // wykorzystanie mechanizmow
       czyszczacych
74     pthread_cleanup_pop(1);
75
76      int threadIdx = tp->thread_idx; // indeks watku
77      int toFind = tp->target;   // wartosc do odszukania
78      int ti = 0;
79      int rnd;
80
81      pthread_t tid = pthread_self();
82      srand(tid);
83
84      pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
85      pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED, NULL);
86
87
88      while(1){
89           while(pthread_mutex_trylock(&mutex) == EBUSY){
90                pthread_testcancel();
91           }
92
93           *(tp->tries)++;
94
95           pthread_mutex_unlock(&mutex);
96
97           rnd = (int)(rand()%1000);
98
99           if (toFind == rnd) {
100               while(pthread_mutex_trylock(&mutex) == EBUSY){
101                    pthread_testcancel();
102               }
103               printf("Number found by %d!\n", threadIdx);
104               for(ti=0;ti<NUM_THREADS;ti++){
105                    if(ti == threadIdx)      // kasowanie innych watkow - ten watek
       zakoczy sie normalnie
106                         continue;
```

```
107                     pthread_cancel(threads[ti]);
108                 }
109                 sleep(1);
110                 pthread_mutex_unlock(&mutex);
111                 break;
112             }
113     }
114
115     return((void *)0);
116 }
```