Przetwarzanie wielowątkowe

Wątki pthreads

- Specyfikacja POSIX (IEEE 1003.1, 1995, 2001)
- Operacje na wątkach (*man pthreads* i inne):
  - określanie atrybutów („odłączalność”, adres i rozmiar stosu, itp.)
  - tworzenie i uruchamianie (*pthread_create*)
  - porównywanie identyfikatorów (*pthread_equal*, *pthread_self*)
  - zabijanie i przesyłanie sygnałów (*pthread_cancel*, *pthread_kill*)
  - odłączanie (*pthread_detach*)
  - oczekiwanie na zakończenie (*pthread_join*)

```
int pthread_create(pthread_t *thread, pthread_attr_t *attr,
                   void * (*start_routine)(void *), void * arg)

int pthread_join(pthread_t th, void **thread_return)
```

Tworzenie wątków

```
#include<pthread.h>
void * f_watek (void * arg_wsk);

int main( int argc, char *argv[] ){
        pthread_t tid_1, tid_2;
        int arg_1 = 1, arg_2 = 2;
        int wyn_1 = pthread_create(& tid_1, NULL, f_watek, &arg_1);
        int wyn_2 = pthread_create(& tid_2, NULL, f_watek, &arg_2);
        wyn_1 = pthread_join( tid_1, NULL);
        wyn_2 = pthread_join( tid_2, NULL);
}

void * f_watek (void * arg_wsk){
        int moj_arg = *( (int *) arg_wsk );
        // ...
}
```

Przesyłanie argumentów do wątków

```
typedef struct {  int a;  double b;} struktura;
int main(){
        struktura s;  s.a = 1; s.b = 3.14;
        pthread_t tid;
        int info = pthread_create(& tid, NULL, funkcja_w, (void *) &s);
        // .......
        }

void * funkcja_w (void * arg_wsk){
     struktura s_lok = *((struktura *)wsk); // struktura lokalna! - kopiowanie zawartości
     // dostęp do zmiennych lokalnych:
     printf("watek: s_lok.a = %d, s_lok.b = %lf\n", s_lok.a, s_lok.b);
     // dostęp do zmiennych nie-lokalnych - kopiowanie wskaźnika z rzutowaniem typu
    struktura * s_wsk = (struktura *) arg_wsk;
     printf("zewn.: "s_zewn.a = %d, s_zewn.b = %lf\n", s_wsk->a, s_wsk->b);
     // ........
}
```

Metodologia programowania równoległego

- Klasyfikacja ze względu na mechanizmy programowe:
  - MPMD *(**Multiple Program Multiple Data**)*
    - każdy proces realizuje różny kod, związany z różnymi plikami binarnymi (np. poprzez fork/exec)
    - architekturą sprzętu jest zawsze MIMD
    - odpowiada dekompozycji funkcjonalnej, ewentualnie dekompozycji złożonej
    - koniecznym elementem jest komunikacja, dzięki której różne procesy uzyskują rozwiązanie pojedynczego zadania obliczeniowego
    - implementacja jest dokonywana najczęściej w modelu z pamięcią rozproszoną za pomocą przesyłania komunikatów
    - programowanie równoległe zbliża się w tym przypadku do programowania rozproszonego

Metodologia programowania równoległego

- Klasyfikacja ze względu na mechanizmy programowe:
  - SPMD ***(Single Program Multiple Data)***
    - każdy proces (wątek) realizuje ten sam program (związany z tym samym plikiem binarnym)
    - model bardzo uniwersalny, częściej wykorzystywany niż MPMD
      - dla architektur SIMD oznacza wykonywanie tego samego kodu z synchronizacja sprzętową (w wariancie SIMT (karty graficzne) możliwe jest zróżnicowanie wykonywanego kodu realizowane poprzez serializację)
      - dla architektur MIMD:
        - wątki mogą realizować ten sam kod, każdy wątek otrzymuje przydzieloną porcję danych na których operuje
        - wątki mogą realizować różne funkcje tego samego kodu lub różne ścieżki przechodzenia przez ten sam kod

SPMD w praktyce

- Przekazywanie wątkom identyfikatorów w zakresie od 0 do p-1

```
#include<pthread.h>

#define LICZBA_W_MAX 4  // maksymalna liczba wątków

void *funkcja_w( void *arg_wsk);

int main( int argc, char *argv[] ){
      int i; pthread_t watki[LICZBA_W_MAX]; int indeksy[LICZBA_W_MAX];
     int p=LICZBA_W_MAX;  // rzeczywista liczba wątków p – jako parametr
      for(i=0;i<p;i++) indeksy[i]=i;
      for(i=0; i<p; i++ )                    // błąd synchronizacji, kiedy przekazujemy &i !
                        pthread_create( &watki[i], NULL, funkcja_w, (void *) &indeksy[i] );
     for(i=0; i<LICZBA_W; i++ )  pthread_join( watki[i], NULL );
}
```

Programowanie wielowątkowe

- W celu realizacji modelu SPMD można do wielu wątków wykonujących ten sam kod przesłać różne dane jako argumenty pierwotnie wywoływanej procedury
  - częstym przypadkiem jest przesłanie identyfikatora wątku – każdy wątek posiada indywidualny identyfikator i na jego podstawie może:
    - zlokalizować dane, na których dokonuje przetwarzania
      - `dane[ f(my_id) ]`
    - wybrać ścieżkę wykonania programu
      - `if(my_id == ...){.....}`
    - określić iteracje pętli, które ma wykonać
      - `for( i=f1(my_id); i< f2(my_id); i += f3(my_id) ){ ..... }`
- Komunikacja między wątkami odbywa się głównie za pomocą pamięci wspólnej