

WOJSKOWA AKADEMIA TECHNICZNA

Wydział Cybernetyki




Systemy Operacyjne

„Wątki w systemach UNIX/LINUX”.

Wykonał: kpr. pchor. Damian Krata
Grupa: I4X3S1 numer na zajęciach 6
Data wykonania ćwiczenia: 10.11.2015r.
Prowadzący: mgr inż. Łukasz Laszko

Treść zadania:

1
Punkty: --/1,00


Napisać program wielowątkowy, który dokona konwersji liczb dziesiętnych, zadanych jako argument wywołania programu, do postaci binarnej. Wątek główny zleca konwersję poszczególnych liczb wątkom roboczym. Wątek główny oczekuje na zakończenie wszystkich wątków roboczych, wypisuje na standardowe wyjście identyfikatory zakończonych wątków i uzyskane przez nie wyniki.

Odpowiedź: ☐ Prawda
☐ Fałsz

Zatwierdź

Rozwiązanie zadania:

Na początku dodałem potrzebne biblioteki. Do obsługi wątków potrzeba skorzystać z pliku nagłówkowego biblioteki POSIX threads - pthread.h. Została tam zawarta implementacja wątków oraz zestaw procedur, wywołań w oparciu o język C. Należy podkreślić, że kolejność argumentów funkcji jest bardzo konsekwentna, podobnie nazwy owych funkcji i typów - poprzedzone „pthread”, np.:

```
pthread_create(&oczekiwanie,NULL,(void*)kropka,NULL);
```

Ponadto pthreads jest zewnętrzną biblioteką, dlatego też należy linkerowi podać ścieżkę do pliku bibliotecznego, np. przy kompilacji gcc dodałem opcję -pthread:

Co to jest wątek?

Technicznie, jest to niezależny strumień instrukcji, który może być wykonywany jednocześnie z innym strumieniem instrukcji danego procesu.

Kod programu:

```
#include <sys/syscall.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
```

```

void zamien(int liczba)
{
    if(liczba>0)
    {
        zamien (liczba/2);
        printf("%d",liczba%2);
    }
}

```

```

void* wyswietl(void *z)
{
    pid_t tid = (pid_t) syscall (SYS_gettid);
    printf ("\nliczba przed konwersja: %d , TID: %d \n", (int) z,      tid);
    printf ("liczba po konwersji\n");
    zamien((int) z);
    pthread_exit(z);
}

```

```

int main (int argc, char *argv[])
{
    int ile = atoi(argv[1]);
    pthread_t id_watek[ile];
    int m=0;
    while (m<ile)
    {
        int s = rand()%5;
        pthread_create(&id_watek[m], 0, &wyswietl, (void *) atoi(argv[m+2]));
        sleep(s);
        m++;
    }
}

```

Funkcja `zamien` pobiera jako argument liczbę całkowitą a zwraca do strumienia wyjściowego tę samą liczbą zapisaną binarnie. Funkcja `wyswietl` odpowiedzialna jest za wypisanie identyfikatora wątku oraz wypisanie liczby przed i po konwersji. Najciekawszą rzeczą w tym kodzie jest fakt działania pętli `while`. Najpierw odczytujemy z argumentów programu ile liczb dziesiętnych zostanie podanych (`atoi(argv[1])`) a następnie tak długo jak te liczby istnieją tworzymy wątki robocze wykonujące zamianę.

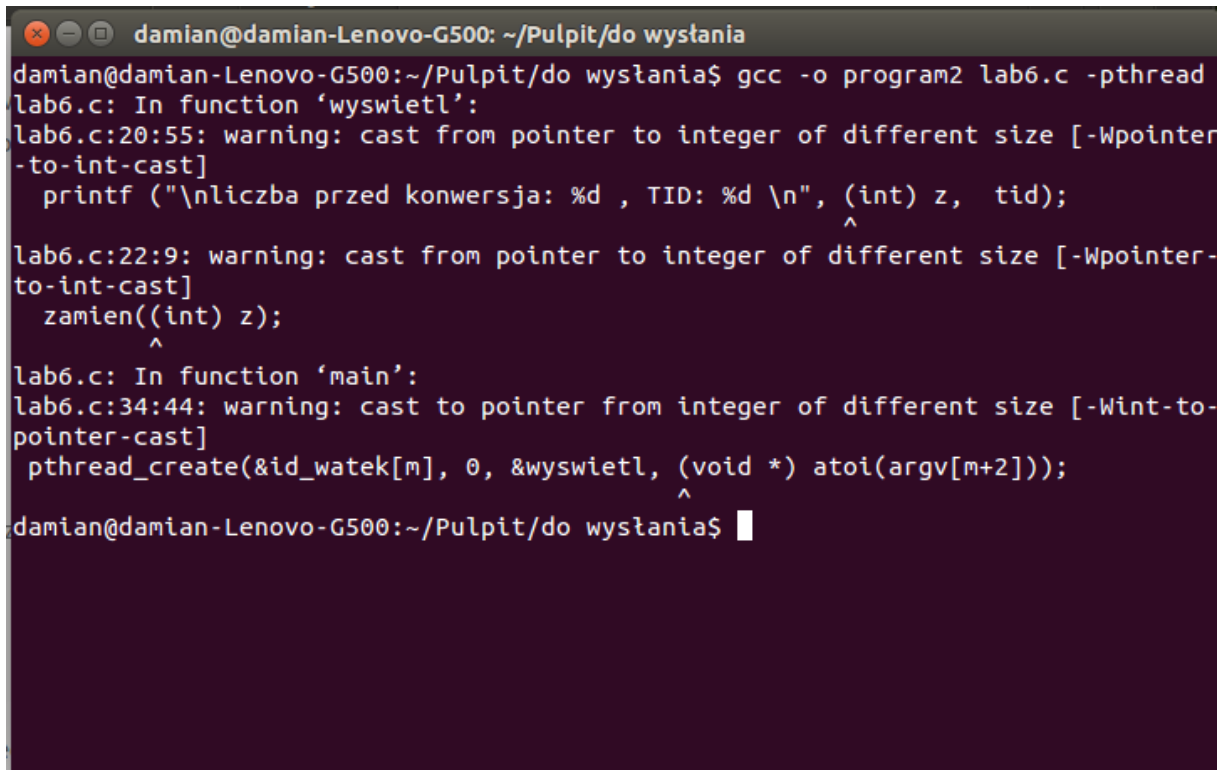
Definicja z Wikipedii:

- `int pthread_create(pthread_t *id, const pthread_attr_t *attr, void* (*fun)(void*), void* arg)`
 - `id` - identyfikator wątku;
 - `attr` - wskaźnik na atrybuty wątku, określające szczegóły dotyczące wątku; można podać `NULL`, wówczas zostaną użyte domyślne wartości;
 - `fun` - funkcja wykonywana w wątku; przyjmuje argument typu `void*` i zwraca wartość tego samego typu;
 - `arg` - przekazywany do funkcji.

U nas jest tyle wątków roboczych ile jest liczb dziesiętnych dlatego identyfikator wątku ma wartość identyfikatora pętli, atrybuty są domyślne, funkcja wykonywana w wątku to funkcja `wyswietl`, natomiast przekazywany do niej argument to (`atoi(argv[m+2])`). Dlaczego `m+2`? Dlatego, że iteracje pętli zaczynamy od `m=0`, skoro `argv[1]` to była liczba liczb dziesiętnych to liczby dziesiętne zaczynamy dopiero od indeksu 2, zatem `m+2`.

Wynik uruchomienia programu:

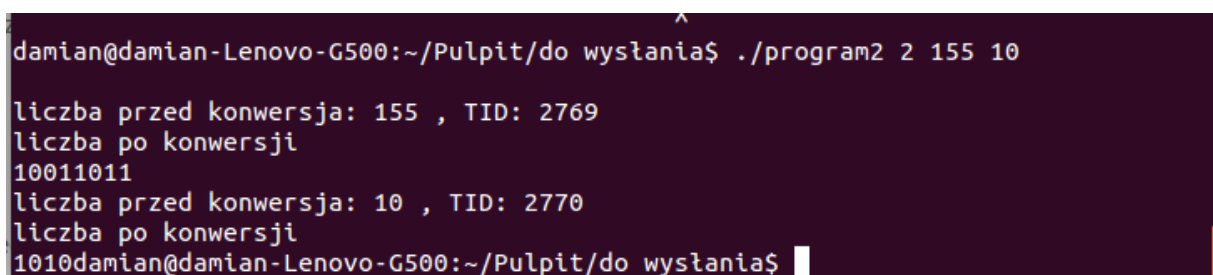
Podczas kompilacji nie obyło się bez warning'ów.



```
damian@damian-Lenovo-G500: ~/Pulpit/do wysłania
damian@damian-Lenovo-G500:~/Pulpit/do wysłania$ gcc -o program2 lab6.c -pthread
lab6.c: In function 'wyswietl':
lab6.c:20:55: warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]
    printf ("\nliczba przed konwersja: %d , TID: %d \n", (int) z, tid);
                                                    ^
lab6.c:22:9: warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]
    zamien((int) z);
          ^
lab6.c: In function 'main':
lab6.c:34:44: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
    pthread_create(&id_watek[m], 0, &wyswietl, (void *) atoi(argv[m+2]));
                                                    ^
damian@damian-Lenovo-G500:~/Pulpit/do wysłania$
```

Wywołujemy program z parametrami 2, 155, 10

Oznacza to że podamy dwie liczby dziesiętne. Pierwszą z nich będzie 155 a drugą 10.



```
damian@damian-Lenovo-G500:~/Pulpit/do wysłania$ ./program2 2 155 10
liczba przed konwersja: 155 , TID: 2769
liczba po konwersji
10011011
liczba przed konwersja: 10 , TID: 2770
liczba po konwersji
1010damian@damian-Lenovo-G500:~/Pulpit/do wysłania$
```

Tę samą czynność powtarzamy dla większej liczby parametrów. Specjalnie dobrałem kolejne dwa razy większe od poprzednich, ponieważ mnożenie przez dwa binarnie odpowiada dodaniu zera, co łatwo można zaobserwować na poniższym obrazku.

```
damian@damian-Lenovo-G500:~/Pulpit/do wysłania$ ./program2 5 155 10 20 40 80  
liczba przed konwersja: 155 , TID: 2787  
liczba po konwersji  
10011011  
liczba przed konwersja: 10 , TID: 2788  
liczba po konwersji  
1010  
liczba przed konwersja: 20 , TID: 2789  
liczba po konwersji  
10100  
liczba przed konwersja: 40 , TID: 2790  
liczba po konwersji  
101000  
liczba przed konwersja: 80 , TID: 2791  
liczba po konwersji  
1010000damian@damian-Lenovo-G500:~/Pulpit/do wysłania$
```

Wnioski:

Cel ćwiczenia uważam za zrealizowany w stopniu bardzo dobrym. Praca nad programem umożliwiła mi poszerzenie wiedzy o wątkach zdobytej na laboratoriach, a przede wszystkim wykorzystanie jej w praktyce. Zauważyłem, że użycie wątków zwiększa potencjalną efektywność programu. Ponadto wątki mają mniejsze wymagania związane z ich obsługą niż procesy, przez co są szybsze i wydajniejsze.