Politechnika Świętokrzyska w Kielcach
Wydział Elektrotechniki, Automatyki i Informatyki

Programowanie Współbieżne

laboratorium

Temat:

Wątki i mutexy

(laboratorium nr 8)

Autor: **Mateusz Snoch**

Grupa: **3ID11A**

Data odbycia się laboratorium : **14.12.2017**

Data sporządzenia sprawozdania: **14.12.2017**

Przydatne dane:

pthread_create – tworzenie dodatkowych wątków
pthread_join – czekanie na koniec wątku
pthread_exit – kończenie
pthread_cancel – kasowanie wątków
pthread_cleanup_push – dodanie procedury obsługi porządkowania
pthread_cleanup_pop – usunięcie funkcji umieszczonej na szczycie stosu funkcji obsługi porządkowania

Zadanie 1

Napisać (lub przerobić jeden z poprzednich) program w którym proces macierzysty tworzy 100 lub więcej procesów, które nic nie wykonują i się kończą. Proces macierzysty czeka na zakończenie wszystkich swoich procesów, poczym zwraca zmierzony czas całej operacji (np. za pomocą clock_gettime (patrz. man -s3 clock_gettime)). Napisać analogiczny program bazujący na wątkach. Porównać wyniki. Czy wątki są szybsze?

Program bazujący na procesach (z1a.c):

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<unistd.h>
#include<sys/wait.h>
#include<time.h>

int main()
{
        int childpid, i, j;
        struct timespec przed, po;
        long int czas;

        clock_gettime(CLOCK_MONOTONIC, &przed);

        for (i=0; i<100; i++) {
                if ((childpid = fork()) == -1) {
                        perror("nie moge forknac");
                        exit(1);
                }
                else
                        if(childpid == 0) {
                                exit(0);
                        }
        }

        for (j=0;j<100;j++) {
                wait(NULL);
        }
```

```
        clock_gettime(CLOCK_MONOTONIC, &po);
        czas = po.tv_nsec - przed.tv_nsec;
        printf("Czas: %ld nanosekund \n", czas);

        return 0;
}
```

Program bazujący na wątkach (z1b.c):

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <time.h>
#include <pthread.h>

void *funkcja(void *arg){
        return NULL;
}

int main()
{
        int i, j;
        struct timespec przed, po;
        long int czas;
        pthread_t tab_tid[100];

        clock_gettime(CLOCK_MONOTONIC, &przed);

        for (i=0; i<100; i++) {
                if(pthread_create(&tab_tid[i], NULL, funkcja, NULL) > 0) {
                        perror("blad tworzenia watku");
                }
        }
        for(j=0; j<100; j++) {
                pthread_join(tab_tid[j], NULL);
        }

        clock_gettime(CLOCK_MONOTONIC, &po);
        czas  = po.tv_nsec - przed.tv_nsec;
        printf("Czas: %ld nanosekund\n", czas);
        return 0;
}
```

Wątki okazały się szybsze.

Zadanie 3

Napisać program tworzący dwa wątki.
W pierwszym wątku po 2s wywołać pthread_cancel z tidem wątku drugiego ponowić próbę po kolejnych 2s.
W drugim wątku na samym początku za pomocą pthread_set_cancelstate zabezpieczyć się przed jego usunięciem, po czym po 3s pozwolić na usunięcie.
Do obu wątków dodać procedurę porządkowania wyświetlającą jakiś komunikat.

Przetestować funkcję usuwającą procedurę porządkującą z parametrem 1 i 0.

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

unsigned long tid1,tid2;

void konczenie1(void *param)
{
        printf("Obsluga zakonczenia 1\n");
}

void konczenie2(void *param)
{
        printf("Obsluga zakonczenia 2\n");
}


void *watek1(void *arg){
        printf("Watek1: start, ide spac na 2s\n");
        pthread_cleanup_push(konczenie1, NULL);
        sleep(2);
        printf("Watek1: jestem po pierwszym sleep, usuwam watek2 i ide spac
na 2s\n");
        pthread_cancel(tid2);
        sleep(2);
        printf("Watek1: jestem po drugim sleep, znow próbuje usunąć
watek2\n");
        pthread_cancel(tid2);
        pthread_cleanup_pop(1);
        printf("Watek1: skonczyłem się normalnie\n");
        return NULL;
}

void *watek2(void *arg){
        printf("Watek2: start, bronie sie przed skasowanien i ide spac na
3s\n");
        pthread_cleanup_push(konczenie2, NULL);
        pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, NULL);
        sleep(3);
        printf("Watek2: jestem po pierwszym sleep, już pozwalam na
skasowanie\n");
        pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
        sleep(2);
        printf("Watek2: jestem po drugim sleep\n");
        pthread_cleanup_pop(1);
        printf("Watek2: skonczyłem się normalnie\n");
        return NULL;
}

int main()
{
        if (pthread_create(&tid1, NULL, watek1, NULL))
                perror("blad tworzenia watku1\n");
        else
                printf("stworzylismy watek1: %lX\n",(unsigned long int)
tid1);
```

```
        if (pthread_create(&tid2, NULL, watek2, NULL))
                perror("blad tworzenia watku2\n");
        else
                printf("stworzylismy watek2: %lX\n",(unsigned long
int)tid2);

        if(pthread_join(tid1, NULL) > 0)
                printf("blad oczekiwania na zakonczenie watku1\n");
        if(pthread_join(tid2, NULL) > 0)
                printf("blad oczekiwania na zakonczenie watku2\n");

        return 0;
}
```

Wnioski

Na ósmym laboratorium poznałem zasadę działania wątków i mutexów. Zauważyłem, że są bardzo podobne w działaniu do procesów macierzystych i dziedziczonych, lecz w znacznym stopniu ułatwiają pracę. Problemy sprawiło mi zadanie drugie, którego nie zdołałem ukończyć.