Laboratorium nr. 2, laboratorium nr. 3	
Imię i nazwisko: Miłosz Dębowski	Kierunek: Informatyka Techniczna
Numer albumu: 415045	Gr. Lab.: 8

Laboratorium nr. 2

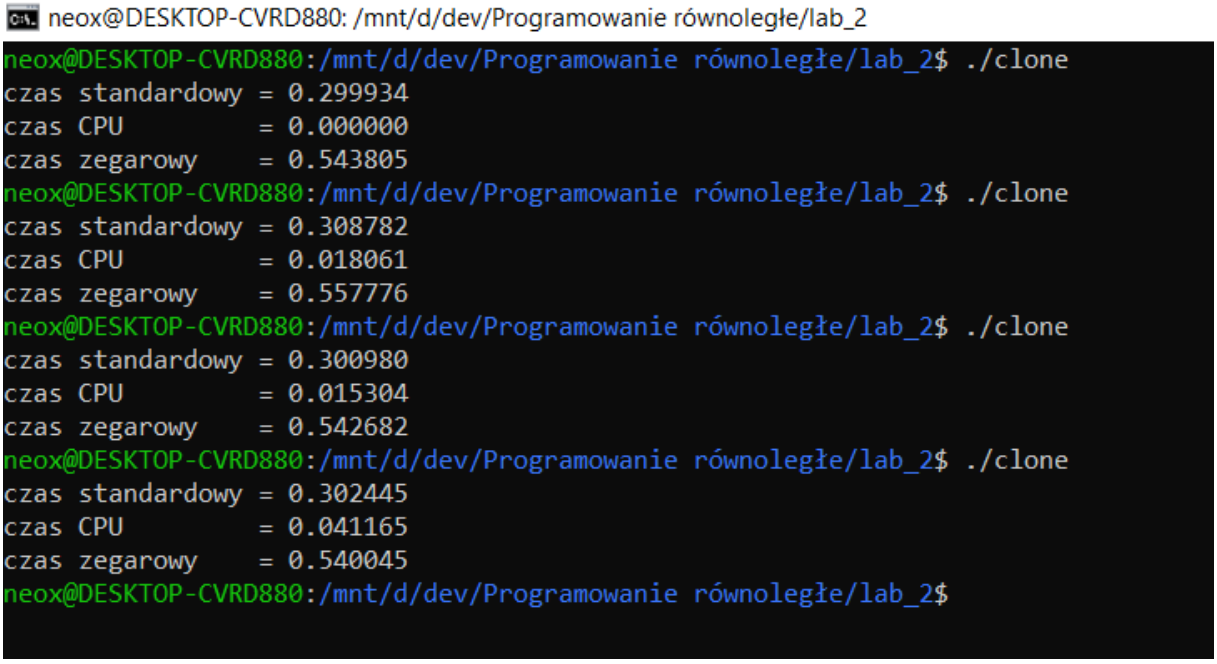
1. Cel:
    - przeprowadzenie pomiaru czasu CPU i zegarowego tworzenia procesów i wątków systemowych Linuxa
    - nabycie umiejętności pisania programów wykorzystujących tworzenie wątków i procesów

2. Wykonanie
    - Utworzenie folderów roboczych.
    - Pobranie i rozpakowanie plików potrzebnych do realizacji ćwiczenia.
    - Edycja kodu w pliku clone.c - załączenie biblioteki z folderu nadrzędnego, oraz dodanie wypisywania wartości zmiennej globalnej w funkcji funkcja_watku():

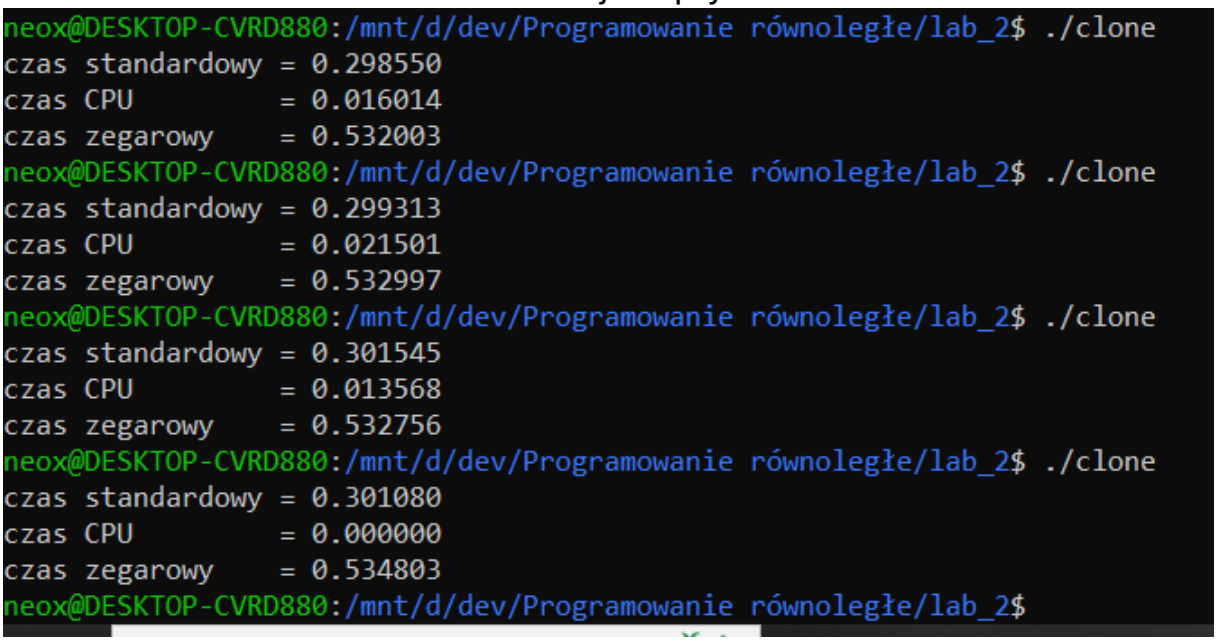
        ```
        #include  "pomiar_czasu/pomiar_czasu.h"
        ...
        printf(„Zmienna globalna = %d\n”, zmienna_globalna);
        ```
    - Dodanie funkcji inicjuj_czas() oraz drukuj_czas(), kolejno przed i po pętli wykonującej obliczenia.
    - Analogiczne wykonanie zmian w kodzie programu fork.c
    - Napisanie nowego programu na podstawie programu clone.c w którym, bezpośrednio po sobie tworzone są dwa wątki do działania równoległego. Wątki zwiększają w odpowiednio długiej pętli (co najmniej 100000 iteracji) wartości dwóch zmiennych - jedną globalną a drugą lokalną.

3. Otrzymane wyniki

Clone – wersja do debugowania



Clone – wersja zoptymalizowana



Fork – wersja do debugowania

```
neox@DESKTOP-CVRD880:/mnt/d/dev/Programowanie równoległe/lab_2$ make
gcc -c -g -DDEBUG -O0 fork.c -Ipomiar_czasu
gcc -g -DDEBUG -O0 fork.o -o fork -Lpomiar_czasu -lpomiar_czasu -lm
gcc -c -g -DDEBUG -O0 clone.c -Ipomiar_czasu
gcc -g -DDEBUG -O0 clone.o -o clone -Lpomiar_czasu -lpomiar_czasu -lm
gcc -c -g -DDEBUG -O0 program.c -Ipomiar_czasu
gcc -g -DDEBUG -O0 program.o -o program -Lpomiar_czasu -lpomiar_czasu -lm
neox@DESKTOP-CVRD880:/mnt/d/dev/Programowanie równoległe/lab_2$ ./fork
czas standardowy = 0.705018
czas CPU         = 0.007350
czas zegarowy    = 1.698839
neox@DESKTOP-CVRD880:/mnt/d/dev/Programowanie równoległe/lab_2$ ./fork
czas standardowy = 0.688919
czas CPU         = 0.042006
czas zegarowy    = 1.620512
neox@DESKTOP-CVRD880:/mnt/d/dev/Programowanie równoległe/lab_2$ ./fork
czas standardowy = 0.666140
czas CPU         = 0.016274
czas zegarowy    = 1.649433
neox@DESKTOP-CVRD880:/mnt/d/dev/Programowanie równoległe/lab_2$ ./fork
czas standardowy = 0.667063
czas CPU         = 0.043192
czas zegarowy    = 1.623777
neox@DESKTOP-CVRD880:/mnt/d/dev/Programowanie równoległe/lab_2$ ./fork
czas standardowy = 0.666864
czas CPU         = 0.031902
czas zegarowy    = 1.653461
neox@DESKTOP-CVRD880:/mnt/d/dev/Programowanie równoległe/lab_2$
```

Fork – wersja zoptymalizowana

```
neox@DESKTOP-CVRD880:/mnt/d/dev/Programowanie równoległe/lab_2$ ./fork
czas standardowy = 0.706186
czas CPU         = 0.008234
czas zegarowy    = 1.681918
neox@DESKTOP-CVRD880:/mnt/d/dev/Programowanie równoległe/lab_2$ ./fork
czas standardowy = 0.701180
czas CPU         = 0.018540
czas zegarowy    = 1.640420
neox@DESKTOP-CVRD880:/mnt/d/dev/Programowanie równoległe/lab_2$ make
make: Nothing to be done for 'all'.
neox@DESKTOP-CVRD880:/mnt/d/dev/Programowanie równoległe/lab_2$ ./fork
czas standardowy = 0.692806
czas CPU         = 0.018458
czas zegarowy    = 1.702015
neox@DESKTOP-CVRD880:/mnt/d/dev/Programowanie równoległe/lab_2$ ./fork
czas standardowy = 0.679879
czas CPU         = 0.010244
czas zegarowy    = 1.629324
neox@DESKTOP-CVRD880:/mnt/d/dev/Programowanie równoległe/lab_2$
```

Program – wersja do debugowania

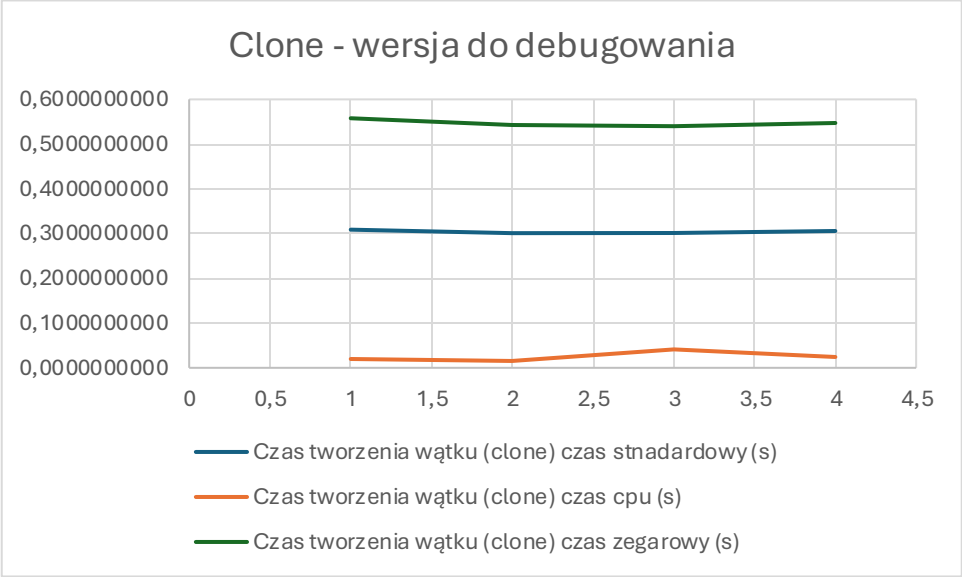
```
neox@DESKTOP-CVRD880:/mnt/d/dev/Programowanie równoległe/lab_2$ ./program
Wewnatrz funkcji:
Zmienna globalna: 830187
Zmienna lokalna: 1000000
Wewnatrz funkcji:
Zmienna globalna: 1055699
Zmienna lokalna: 1000000
Zmienna globalna: 1055699
Zmienna lokalna: 0
neox@DESKTOP-CVRD880:/mnt/d/dev/Programowanie równoległe/lab_2$ ./program
Wewnatrz funkcji:
Zmienna globalna: 1022202
Zmienna lokalna: 1000000
Wewnatrz funkcji:
Zmienna globalna: 1143249
Zmienna lokalna: 1000000
Zmienna globalna: 1143249
Zmienna lokalna: 0
neox@DESKTOP-CVRD880:/mnt/d/dev/Programowanie równoległe/lab_2$ ./program
Wewnatrz funkcji:
Zmienna globalna: 985988
Zmienna lokalna: 1000000
Wewnatrz funkcji:
Zmienna globalna: 1052138
Zmienna lokalna: 1000000
Zmienna globalna: 1052138
Zmienna lokalna: 0
```

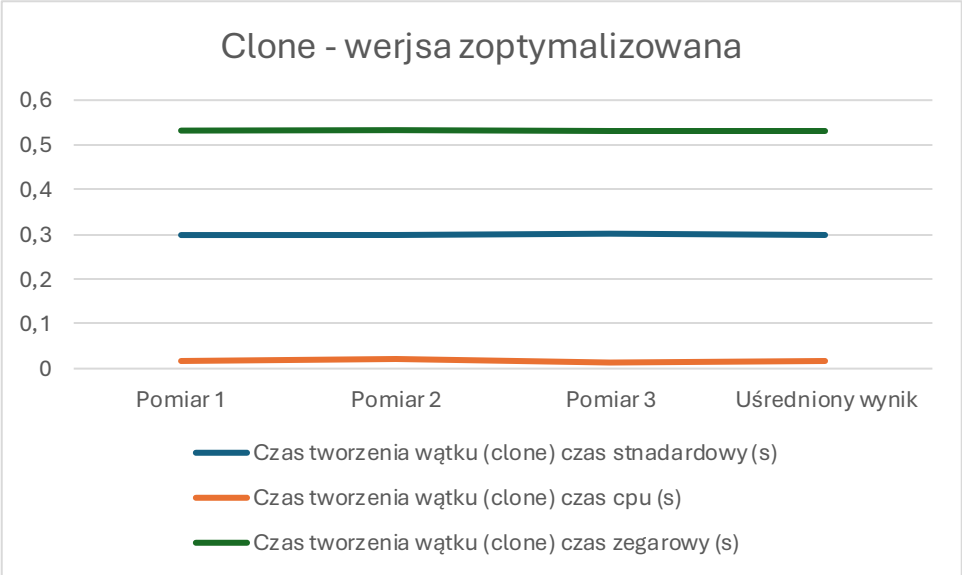
Program – wersja zoptymalizowana

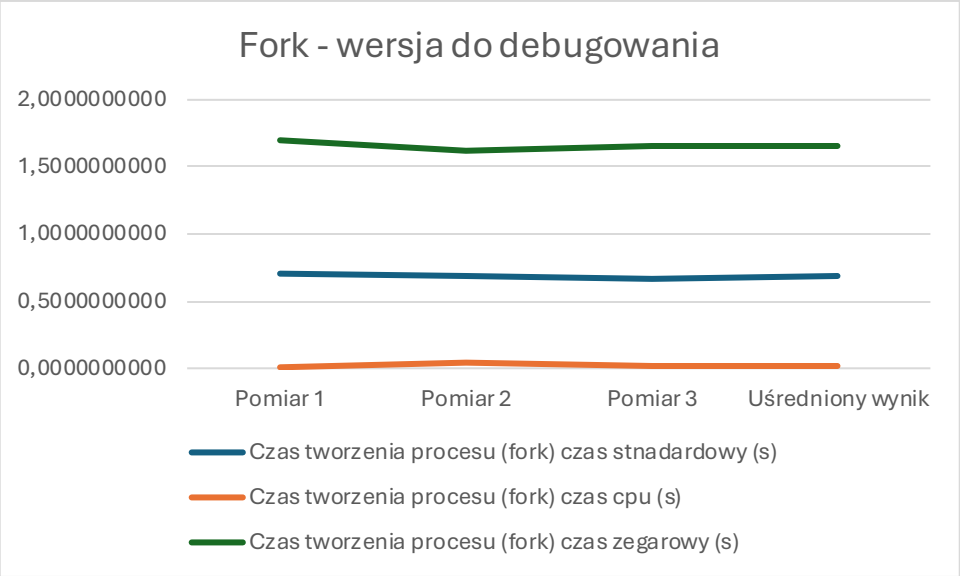
```
neox@DESKTOP-CVRD880:/mnt/d/dev/Programowanie równoległe/lab_2$ ./program
Wewnatrz funkcji:
Zmienna globalna: 1000000
Zmienna lokalna: 1000000
Wewnatrz funkcji:
Zmienna globalna: 2000000
Zmienna lokalna: 1000000
Zmienna globalna: 2000000
Zmienna lokalna: 0
neox@DESKTOP-CVRD880:/mnt/d/dev/Programowanie równoległe/lab_2$ ./program
Wewnatrz funkcji:
Wewnatrz funkcji:
Wewnatrz funkcji:
Zmienna globalna: 2000000
Zmienna globalna: 2000000
Zmienna lokalna: 1000000
Zmienna lokalna: 1000000
Zmienna globalna: 2000000
Zmienna lokalna: 0
neox@DESKTOP-CVRD880:/mnt/d/dev/Programowanie równoległe/lab_2$ ./program
Wewnatrz funkcji:
Wewnatrz funkcji:
Zmienna globalna: 1000000
Wewnatrz funkcji:
Zmienna lokalna: 1000000
Zmienna globalna: 2000000
Zmienna lokalna: 1000000
Zmienna globalna: 2000000
Zmienna lokalna: 0
```

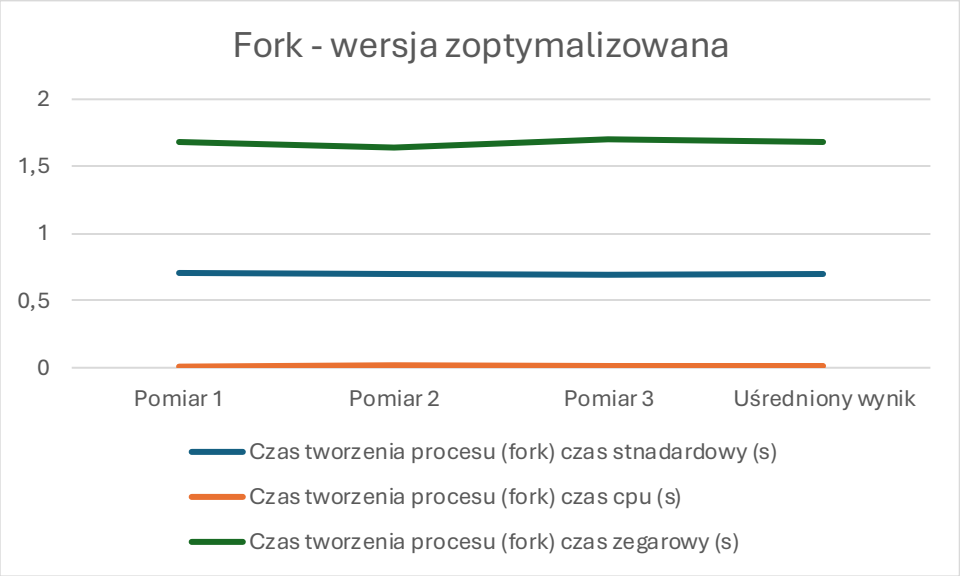
Wersja DDEBUG	Czas tworzenia wątku (clone)			Czas tworzenia procesu (fork)		
	czas stnadardowy (s)	czas cpu (s)	czas zegarowy (s)	czas stnadardowy (s)	czas cpu (s)	czas zegarowy (s)
Pomiar 1	0,3087820000	0,0180610000	0,5577760000	0,7050180000	0,00735	1,698839
Pomiar 2	0,3009800000	0,0153040000	0,5426820000	0,6889190000	0,042006	1,620512
Pomiar 3	0,3024450000	0,0411650000	0,5400450000	0,6661400000	0,016274	1,649433
Uśredniony wynik	0,3040690000	0,0248433333	0,5468343333	0,6866923333	0,02187667	1,656261333

Wersja -O3	Czas tworzenia wątku (clone)			Czas tworzenia procesu (fork)		
	czas stnadardowy (s)	czas cpu (s)	czas zegarowy (s)	czas stnadardowy (s)	czas cpu (s)	czas zegarowy (s)
Pomiar 1	0,29855	0,016014	0,532003	0,706186	0,008234	1,681918
Pomiar 2	0,299313	0,021501	0,532997	0,70118	0,01854	1,64042
Pomiar 3	0,301545	0,013568	0,532756	0,692806	0,018458	1,702015
Uśredniony wynik	0,299802667	0,017027667	0,532585333	0,700057333	0,01507733	1,674784333






Fork - wersja do debugowania
2,0000000000
1,5000000000
1,0000000000
0,5000000000
0,0000000000
Pomiar 1
Pomiar 2
Pomiar 3
Uśredniony wynik
Czas tworzenia procesu (fork) czas stnadardowy (s)
Czas tworzenia procesu (fork) czas cpu (s)
Czas tworzenia procesu (fork) czas zegarowy (s)


Fork - wersja zoptymalizowana
2
1,5
1
0,5
0
Pomiar 1
Pomiar 2
Pomiar 3
Uśredniony wynik
Czas tworzenia procesu (fork) czas stnadardowy (s)
Czas tworzenia procesu (fork) czas cpu (s)
Czas tworzenia procesu (fork) czas zegarowy (s)

4. Kod programów

clone.c

```c
#define _GNU_SOURCE
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>
#include "pomiar_czasu/pomiar_czasu.h"
#include <sys/types.h>
#include <sys/wait.h>
#include <sched.h>
#include <linux/sched.h>

int zmienna_globalna=0;

#define ROZMIAR_STOSU 1024*64

int funkcja_watku( void* argument )
{

zmienna_globalna++;

return 0;
}

int main()
{

void *stos;
pid_t pid;
int i;
inicjuj_czas();
stos = malloc( ROZMIAR_STOSU );
if (stos == 0) {
printf("Proces nadrzędny - blad alokacji stosu\n");
exit( 1 );
}

for(i=0;i<10000;i++){

pid = clone( &funkcja_watku, (void *) stos+ROZMIAR_STOSU,
CLONE_FS | CLONE_FILES | CLONE_SIGHAND | CLONE_VM, 0 );

waitpid(pid, NULL, __WCLONE);

}
drukuj_czas();
free( stos );
}
```

fork.c

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

#include"pomiar_czasu/pomiar_czasu.h"

int zmienna_globalna=0;

int main(){

int pid, wynik, i;

inicjuj_czas();

for(i=0;i<10000;i++){

pid = fork();

if(pid==0){

zmienna_globalna++;

// char arg1[] = "/bin/ls";
// char arg2[] = ".";
// char* arg[] = {arg1,arg2,NULL};
// char* arg[] = {"/bin/ls",".",NULL};
// wynik=execv("/bin/ls",arg);
// wynik=execv("./program",NULL);
// if(wynik==-1)
//      printf("Proces potomny nie wykonal programu\n");

exit(0);

} else {

wait(NULL);

}

}

drukuj_czas();

}
```

program.c

```c
#define _GNU_SOURCE
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>
#include "pomiar_czasu/pomiar_czasu.h"
#include <sys/types.h>
#include <sys/wait.h>
#include <sched.h>
#include <linux/sched.h>

int zmienna_globalna=0;
#define ROZMIAR_STOSU 1024*64
int funkcja_watku( void* argument )
{
int zmienna_lokalna = *(int *)argument;
for(int i = 0; i < 1e+8; i++){
zmienna_globalna++;
zmienna_lokalna++;
}
printf("Wewnatrz funkcji:\n");
printf("Zmienna globalna: %d\n",zmienna_globalna);
printf("Zmienna lokalna: %d\n",zmienna_lokalna);
return 0;
}

int main()
{
void *stos,*stos1;
pid_t pid,pid1;
int i;
int zmienna_lokalna = 0;
stos = malloc( ROZMIAR_STOSU );
stos1 = malloc( ROZMIAR_STOSU );
if (stos == 0) {
printf("Proces nadrzędny - blad alokacji stosu\n");
exit( 1 );
}
pid = clone( &funkcja_watku, (void *) stos+ROZMIAR_STOSU,CLONE_FS | CLONE_FILES | CLONE_SIGHAND | CLONE_VM, &zmienna_lokalna );
pid1 = clone( &funkcja_watku, (void *) stos1+ROZMIAR_STOSU, CLONE_FS | CLONE_FILES | CLONE_SIGHAND | CLONE_VM, &zmienna_lokalna );
waitpid(pid, NULL, __WCLONE);
waitpid(pid1, NULL, __WCLONE);
printf("Zmienna globalna: %d\n",zmienna_globalna);
printf("Zmienna lokalna: %d\n",zmienna_lokalna);
free( stos );
free(stos1);
}
```

5. Wnioski
    - Wykonywanie zadań przez wątki jest około dwa razy szybsze niż w przypadku procesów, gdy zadania są identyczne.
    - Funkcja fork() to podstawowa metoda tworzenia procesów w systemach Unix, która dzieli proces wywołujący na dwa: proces "rodzica" i proces "dziecka". Różnica między nimi polega na wartości zwracanej przez fork() - proces rodzica otrzymuje PID procesu dziecka, proces dziecka - wartość 0, a w przypadku błędu proces potomny nie powstaje, a funkcja zwraca -1.
    - Funkcja clone() jest rozbudowaną wersją funkcji fork(), która umożliwia bardziej szczegółowe kontrolowanie procesu tworzenia nowych procesów i pozwala na współdzielenie zasobów między procesami.
    - Zoptymalizowana wersja programu.c zwraca wartość zmiennej globalnej dwa razy większą niż lokalnej. Wynika to z optymalizacji polegającej na jednorazowym dodaniu 10000 do zmiennej lokalnej i 20000 do globalnej zamiast dodawania +1 w każdej z 10000 operacji. Takie działanie uznaje się za nadmiernie agresywną optymalizację.

Laboratorium nr. 3

1. Cel:
    - nabycie praktycznej umiejętności manipulowania wątkami Pthreads – tworzenia, niszczenia, elementarnej synchronizacji
    - przetestowanie mechanizmu przesyłania argumentów do wątku
    - poznanie funkcjonowania obiektów określających atrybuty wątków.

2. Wykonanie
    - Utworzenie folderów roboczych.
    - Pobranie i rozpakowanie plików potrzebnych do realizacji ćwiczenia.
    - Edycja kodu w pliku pthreads_detach_kill.c - uzupełnienie kodu programu w opraciu o skrypt podany przez profesora
    - Utworzenie pliku Makefile, uruchomienie i przetestowanie programu
    - Zaprojektowanie, utworzenie i przetestowanie nowej procedury wątków, do której jako argument przesyłany jest identyfikator każdego wątku, z zakresu 0..liczba_wątków-1

3. Otrzymane wyniki

    pthreads_detach_kill.c – zadanie 1

```
neox@NEOX:/mnt/d/dev/Programowanie równoległe/lab_3/zad_1$ make
gcc -c -g -DDEBUG pthreads_detach_kill.c
gcc -g -DDEBUG pthreads_detach_kill.o -o zad1 -lpthread
neox@NEOX:/mnt/d/dev/Programowanie równoległe/lab_3/zad_1$ ./zad1
watek glowny: tworzenie watku potomnego nr 1
        watek potomny: uniemozliwione zabicie
        watek glowny: wyslanie sygnalu zabicia watku
        watek potomny: umozliwienie zabicia
        watek glowny: watek potomny zostal zabity
watek glowny: tworzenie watku potomnego nr 2
        watek potomny: uniemozliwione zabicie
        watek glowny: odlaczenie watku potomnego
        watek glowny: wyslanie sygnalu zabicia watku odlaczonego
        watek glowny: czy watek potomny zostal zabity
        watek glowny: sprawdzanie wartosci zmiennej wspolnej
        watek potomny: umozliwienie zabicia
        watek glowny: odlaczony watek potomny PRAWDOPODOBNIE zostal zabity
watek glowny: tworzenie odlaczonego watku potomnego nr 3
        watek glowny: koniec pracy, watek odlaczony pracuje dalej
        watek potomny: uniemozliwione zabicie
        watek potomny: umozliwienie zabicia
        watek potomny: zmiana wartosci zmiennej wspolnej
neox@NEOX:/mnt/d/dev/Programowanie równoległe/lab_3/zad_1$
```

zadanie 2

```
neox@NEOX:/mnt/d/dev/Programowanie równoległe/lab_3/zad_2$ make
gcc -c -g -DDEBUG zad2.c
gcc -g -DDEBUG zad2.o -o zad2 -lpthread
neox@NEOX:/mnt/d/dev/Programowanie równoległe/lab_3/zad_2$ ./zad2
SYSTEM ID: 0 & SELF ID = 1404172185084800
SYSTEM ID: 1 & SELF ID = 1404172101157760
SYSTEM ID: 2 & SELF ID = 1404170675053440
SYSTEM ID: 4 & SELF ID = 1404171933303680
SYSTEM ID: 3 & SELF ID = 1404172017230720
SYSTEM ID: 5 & SELF ID = 1404171106362240
neox@NEOX:/mnt/d/dev/Programowanie równoległe/lab_3/zad_2$
```

4. Kod programów

Makefile – zadanie 1

```
1  # kompilator c
2  CCOMP = gcc
3  # konsolidator
4  LOADER = gcc
5  # opcje optymalizacji:
6  # wersja do debugowania
7  OPT = -g -DDEBUG
8  # wersja zoptymalizowana do mierzenia czasu
9  # OPT = -O3
10 # zaleznosci i komendy
11 zad1: pthreads_detach_kill.o
12 $(LOADER) $(OPT) pthreads_detach_kill.o -o zad1 -lpthread $(LIB)
13
14 # jak uzyskac plik pthreads_kill_detach.o ?
15 pthreads_detach_kill.o: pthreads_detach_kill.c
16 $(CCOMP) -c $(OPT) pthreads_detach_kill.c
17
18 clean:
19 rm -f *.o zad1
```

Makefile – zadanie 2

```
# kompilator c
CCOMP = gcc

# konsolidator
LOADER = gcc

# opcje optymalizacji:
# wersja do debugowania
OPT = -g -DDEBUG
# wersja zoptymalizowana do mierzenia czasu
# OPT = -O3

# zaleznosci i komendy
zad2: zad2.o
$(LOADER) $(OPT) zad2.o -o zad2 -lpthread $(LIB)

# jak uzyskac plik pthreads_kill_detach.o ?
zad2.o: zad2.c
$(CCOMP) -c $(OPT) zad2.c

clean:
rm -f *.o zad2
```

pthreads_detach_kill.c

```
#include<stdlib.h>
#include<stdio.h>
#include<pthread.h>
#include <unistd.h>
int zmienna_wspolna=0;
#define WYMIAR 1000
#define ROZMIAR WYMIAR*WYMIAR
double a[ROZMIAR],b[ROZMIAR],c[ROZMIAR];
double czasozajmowacz(){
int i, j, k;
int n=WYMIAR;
for(i=0;i<ROZMIAR;i++) a[i]=1.0*i;
for(i=0;i<ROZMIAR;i++) b[i]=1.0*(ROZMIAR-i);
for(i=0;i<n;i++){
for(j=0;j<n;j++){
c[i+n*j]=0.0;
for(k=0;k<n;k++){
c[i+n*j] += a[i+n*k]*b[k+n*j];
}
}
}
return(c[ROZMIAR-1]);
```

```c
}
void * zadanie_watku (void * arg_wsk)
{
pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, NULL);
printf("\twatek potomny: uniemozliwione zabicie\n");
czasozajmowacz();
printf("\twatek potomny: umozliwienie zabicia\n");
pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
pthread_testcancel()
zmienna_wspolna++;
printf("\twatek potomny: zmiana wartosci zmiennej wspolnej\n");
return(NULL);
}
int main()
{
pthread_t tid;
pthread_attr_t attr;
void *wynik;
int i;
//Wątek przyłączalny
printf("watek glowny: tworzenie watku potomnego nr 1\n");
/*Tu wstaw kod tworzenia wątku z domyślnymi własnościami*/
pthread_create(&tid, NULL, zadanie_watku, NULL);//dopisane
sleep(2); // czas na uruchomienie watku
printf("\twatek glowny: wyslanie sygnalu zabicia watku\n");
pthread_cancel(tid);
//Co nalezy zrobić przed sprawdzeniem czy wątki się skonczyły?
pthread_join(tid, &wynik);//dopisane
if (wynik == PTHREAD_CANCELED)
printf("\twatek glowny: watek potomny zostal zabity\n");
else
printf("\twatek glowny: watek potomny NIE zostal zabity - blad\n");
//Odłączanie wątku
zmienna_wspolna = 0;
printf("watek glowny: tworzenie watku potomnego nr 2\n");
/*Tu wstaw kod tworzenia wątku z domyślnymi własnościami*/
pthread_create(&tid, NULL, zadanie_watku, NULL);//dopisane
sleep(2); // czas na uruchomienie watku
printf("\twatek glowny: odlaczenie watku potomnego\n");
//Instrukcja odłączenia?
pthread_detach(tid);//dopisane
printf("\twatek glowny: wyslanie sygnalu zabicia watku odlaczonego\n");
pthread_cancel(tid);
//Czy watek został zabity? Jak to sprawdzić?
printf("\twatek glowny: czy watek potomny zostal zabity \n");//całe
   dopisane
printf("\twatek glowny: sprawdzanie wartosci zmiennej wspolnej\n");
for(i=0;i<10;i++)
{
```

```
71 sleep(1);
72 if(zmienna_wspolna!=0) break;
73 }
74 if (zmienna_wspolna==0) printf("\twatek glowny: odlaczony watek potomny
   PRAWDOPODOBNIE zostal zabity\n");
75 else printf("\twatek glowny: odlaczony watek potomny PRAWDOPODOBNIE NIE
   zostal zabity\n");
76 //Wątek odłączony
77 //Inicjacja atrybutów?
78 pthread_attr_init(&attr);//dopisane
79 //Ustawianie typu watku na odłaczony
80 pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);//dopisane
81 printf("watek glowny: tworzenie odlaczonego watku potomnego nr 3\n");
82 pthread_create(&tid, &attr, zadanie_watku, NULL);
83
84 //Niszczenie atrybutów
85 pthread_attr_destroy(&attr);//dopisane
86
87 printf("\twatek glowny: koniec pracy, watek odlaczony pracuje dalej\n");
88 pthread_exit(NULL); // co stanie sie gdy uzyjemy exit(0)?
89 }
```

zad2.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#define THREADS_COUNT 6
void* zadanie_watku(void* arg_wsk) {
    int moj_arg = *((int*)arg_wsk);
    printf("SYSTEM ID: %d & SELF ID = %ld\n", moj_arg, pthread_self());
    return NULL;
}
int main() {
    pthread_t threads[THREADS_COUNT];
    int index[THREADS_COUNT];

    for (int i = 0; i < THREADS_COUNT; i++) {
        index[i] = i;
        pthread_create(&threads[i], NULL, zadanie_watku, (void*)&index[i]);
    }

    for (int i = 0; i < THREADS_COUNT; i++) {
        pthread_join(threads[i], NULL);
    }
    pthread_exit(NULL);
}
```

5. Wnioski
    - Program pthreads_detach_kill.c demonstruje różne metody zarządzania wątkami w języku C, w tym tworzenie, anulowanie i odłączanie wątków.
    - Możliwość anulacji wątku można kontrolować, co jest istotne, gdy wątek wykonuje długotrwałe operacje i nie powinien być przerywany w trakcie ich wykonywania.
    - Wątki mogą być przyłączalne lub odłączalne, co wpływa na sposób ich zarządzania oraz na to, jak i kiedy można sprawdzić ich zakończenie.
    - Użycie funkcji sleep() w programie ilustruje sposób, w jaki można wprowadzić opóźnienia w działaniu wątków, co może być istotne w kontekście synchronizacji lub testowania.
    - Oto wnioski płynące z podanego kodu i wyników:
    - Każdy wątek ma swój unikalny SELF ID, co wskazuje, że są to niezależne wątki systemowe. SELF ID to identyfikator przypisywany przez system do każdego wątku, co potwierdza, że wątki są traktowane jako oddzielne jednostki wykonawcze.
    - Mimo że wątki są tworzone w określonej kolejności, system operacyjny decyduje o ich rzeczywistym czasie rozpoczęcia i zakończenia, co może skutkować różnymi kolejnościami wypisywania wyników przy każdym uruchomieniu programu.