Imię i Nazwisko Anna Jasielec	**Kierunek** Informatyka Techniczna	**Rok studiów i grupa** rok 3, grupa 4
Data zajęć 18.10.2024	**Numer i temat sprawozdania** 2. Programowanie równoległe. Pomiar czasu procesów i wątków.	

1. **Cel laboratorium:**
   - Pomiar czasu CPU i zegarowego dla tworzenia procesów i wątków w systemie Linux.
   - Zdobycie umiejętności programowania z użyciem procesów i wątków.

2. **Przebieg zajęć:**
   - Pomiar czasu wykonania 1000 procesów i 1000 wątków z pustą funkcją wątku:
     - w wersji do debugowania:

```
OPT = -g -DDEBUG
```

	czas 1000 procesów (fork)	czas 1 procesu	czas 1000 wątków (clone)	czas 1 wątku
CPU	0,009319	0,000009319	0,007922	0,000007922
	0,008872	0,000008872	0,006636	0,000006636
	0,010998	0,000010998	0,009282	0,000009282
	0,007636	0,000007636	0,009614	0,000009614
średni CPU	0,00920625	9,20625E-06	0,0083635	8,3635E-06
zegarowy	0,249604	0,000249604	0,093225	0,000093225
	0,271953	0,000271953	0,092384	0,000092384
	0,259966	0,000259966	0,095334	0,000095334
	0,262445	0,000262445	0,103346	0,000103346
średni zegarowy	0,260992	0,000260992	0,09607225	9,60723E-05

     - w wersji z optymalizacją:

```
OPT = -O3
```

	czas 1000 procesów (fork)	czas 1 procesu	czas 10000 wątków (clone)	czas 1 wątku
CPU	0,008504	0,000008504	0,008621	8,621E-07
	0,022368	0,000022368	0,019773	1,9773E-06
	0,019218	0,000019218	0,008456	8,456E-07
	0,007419	0,000007419	0,015286	1,5286E-06
średni CPU	0,01437725	1,43773E-05	0,013034	1,3034E-06
zegarowy	0,257293	0,000257293	0,998032	9,98032E-05
	0,273031	0,000273031	1,023479	0,000102348
	0,249315	0,000249315	0,990762	9,90762E-05
	0,25616	0,00025616	0,093284	9,3284E-06
średni zegarowy	0,25894975	0,00025895	0,77638925	7,76389E-05

- Analiza różnic w działaniu clone() i fork().

	O3		DDBUG	
	fork	clone	fork	clone
czas cpu	1,43773E-05	1,3034E-06	9,20625E-06	8,3635E-06
czas zegarowy	0,00025895	7,76389E-05	0,000260992	9,60723E-05

➔ ***Jaka jest relatywna wielkość narzutu na tworzenie wątków, w porównaniu do narzutu na tworzenie procesów?***
Analiza wyników pokazuje, że narzut na tworzenie wątków za pomocą clone() jest znacznie mniejszy niż na tworzenie procesów za pomocą fork(). Wynika to z faktu, że wątki dzielą przestrzeń adresową i zasoby pamięci, co pozwala na szybsze ich tworzenie. Procesy, natomiast, kopiują całą przestrzeń adresową, co generuje większy narzut systemowy.

➔ ***Ile operacji arytmetycznych (szacunkowo) może wykonać procesor w czasie, który zajmuje tworzenie wątków?***

	czas 100000 operacji wejścia/wyjścia	czas 1 operacji wejścia/wyjścia	czas 100000 operacji arytmetycznych	czas 1 operacji arytmetycznej
zegarowy	0,021169	2,1169E-07	0,000071	7,1E-10
liczba operacji, które wykonają się w czasie tworzenia 1 wątku		366,7576409		109350,5986

Wykorzystując wyniki otrzymane w poprzednich laboratoriach i porównując czas tworzenia jednego wątku (ok. 1,3E-06 sekundy) z czasem potrzebnym do wykonania jednej operacji arytmetycznej (ok. 7,1E-10 sekundy) można stwierdzić, że procesor może wykonać około 110 000 operacji arytmetycznych w czasie, który zajmuje utworzenie jednego wątku.

➔ ***Ile operacji wejścia/wyjścia?***
Jedna operacja wejścia/wyjścia zajmuje około 2,12E-07 sekundy, a czas potrzebny na stworzenie jednego wątku wynosi 1,3E-06 sekundy. Zatem w czasie tworzenia jednego wątku procesor może wykonać około 370 operacji wejścia wyjścia.

➔ ***Czy optymalizacja wpływa na czas tworzenia procesów i wątków? Co jest tego przyczyną?***
Na podstawie przedstawionych wyników można zauważyć, że optymalizacja wpływa na czas tworzenia procesów i wątków. Przyczyną może być redukowanie zbędnych operacji i poprawianie zarządzania zasobami w wersji zoptymalizowanej.

- Utworzenie programu z dwoma wątkami działającymi w sposób równoległy, wykorzystując clone(). Praca na zmiennych globalnych i lokalnych.
Funkcja wątku utworzonego programu:

```
int funkcja_wątku(void* argument) {
    int lokalna_zmienna = *((int*) argument);
    for(int i = 0; i < 100000; i++) {
```

```
        zmienna_globalna ++;
        lokalna_zmienna ++;
    }

    printf ("Wątek zakończony: zmienna_globalna = %d, lokalna_zmienna = %d\n",
            zmienna_globalna, lokalna_zmienna);
    return 0;
}
```

Funkcja main w utworzonym programie:

```
int main() {
    void* stos1;
    void* stos2;
    pid_t pid1, pid2;

    // alokacja stosu dla wątku
    stos1 = malloc(ROZMIAR_STOSU);
    stos2 = malloc(ROZMIAR_STOSU);
    if (stos1 == 0 || stos2 == 0) {
        printf("Błąd alokacji stosu");
        exit(1);
    }

    int lokalna_zmienna1 = 0;
    int lokalna_zmienna2 = 0;

    // tworzenie pierwszego wątku
    pid1 = clone(&funkcja_wątku, (void*)stos1 + ROZMIAR_STOSU,
                 CLONE_FS | CLONE_FILES | CLONE_SIGHAND | CLONE_VM,
                 &lokalna_zmienna1);

    // tworzenie drugiego wątku
    pid2 = clone(&funkcja_wątku, (void*)stos2 + ROZMIAR_STOSU,
                 CLONE_FS | CLONE_FILES | CLONE_SIGHAND | CLONE_VM,
                 &lokalna_zmienna2);

    // oczekiwanie na zakończenie
    waitpid(pid1, NULL, __WCLONE);
    waitpid(pid2, NULL, __WCLONE);

    // wypisanie wartości zmiennych
    printf("Wszystkie wątki zakończone: zmienna_globalna = %d, lokalna_zmienna1
= %d, lokalna_zmienna2 = %d\n",
            zmienna_globalna, lokalna_zmienna1, lokalna_zmienna2);

    free(stos1);
    free(stos2);
    return 0;}
```

- Analiza zachowania zmiennych modyfikowanych przez równoległe wątki poprzez wypisanie ich wartości.
    - W wersji do debugowania:

```
ajasielec@DESKTOP-J8MMMJM:~/PR/Parallel-Programming/lab-2$ ./multi_thread_clone
Wątek zakończony: zmienna_globalna = 75941, lokalna_zmienna = 100000
Wątek zakończony: zmienna_globalna = 117833, lokalna_zmienna = 100000
Wszystkie wątki zakończone: zmienna_globalna = 117833, lokalna_zmienna1 = 0, lokalna_zmienna2 = 0
```

➔ ***Czy wartości zmiennych odpowiadają liczbie operacji wykonanych przez wątki?***

Wartości zmiennych lokalnych są zgodne z oczekiwaniami (100 000). Każdy wątek wykorzystywał swoją unikalną zmienną w funkcji wątku. Po zakończeniu wszystkich wątków lokalna_zmienna1 i lokalna_zmienna2 równają się 0, ponieważ one są lokalne dla funkcji main() i są przekazywane do funkcji wątku, gdzie jedynie odczytujemy ich wartość, nie modyfikujemy ich wartości w funkcji main().

Zmienna globalna natomiast jest mniejsza od oczekiwanego wyniku (200 000). Oznacza to, że niektóre zwiększenia zostały utracone z powodu współbieżnego dostępu.

➔ ***Jak zachowują się zmienne globalne (statyczne), a jak zmienne lokalne w funkcjach wykonywanych przez wątki?***

Zmienna globalne są współdzielone przez równoległe wątki, co powoduje ryzyko kolizji, jeśli wiele wątków próbuje je modyfikować równocześnie.

Zmienne lokalne są unikalne dla każdego wątku, co oznacza, że każdy wątek ma swoją własną kopię zmiennych lokalnych, co eliminuje problemy związane ze współbieżnością.

- W wersji z optymalizacją:

```
ajasielec@DESKTOP-J8MMMJM:~/PR/Parallel-Programming/lab-2$ ./multi_thread_clone
Wątek zakończony: zmienna_globalna = 100000, lokalna_zmienna = 100000
Wątek zakończony: zmienna_globalna = 200000, lokalna_zmienna = 100000
Wszystkie wątki zakończone: zmienna_globalna = 200000, lokalna_zmienna1 = 0, lokalna_zmienna2 = 0
```

Po uruchomieniu programu w wersji z optymalizacją widzimy, że wartość zmiennej globalnej są zgodne z oczekiwaniami (200 000), co potwierdza skuteczność zastosowanych algorytmów optymalizacyjnych. Kompilator zdołał zredukować dodawanie do zmiennej globalnej w pętli, przekształcając je w jednorazowe dodanie sumy wszystkich iteracji. Dzięki temu operacje stały się bardziej wydajne i zgodne z oczekiwaniami.

- Utworzenie nowego programu zajecia_2.c wypisującego na ekranie imię, nazwisko i numer procesu, wykorzystującego argumenty przesłane do funkcji main.
  Funkcja main:

```c
int main(int argc, char *argv[]) {
    pid_t pid = getpid(); // id procesu
    printf("Imię i nazwisko: Anna Jasielec, numer procesu: %d\n", pid);
    printf("%s", argv[0]);
    return 0;
}
```

- Uruchomienie programu ./zajecia_2 za pomocą funkcji exec() z przesłaniem odpowiednich danych:
  Fragment kodu w programie clone.c:

```
  char* args[] = {"./zajecia2", NULL}; // uruchomienie programu

   int wynik;
   wynik = execv(args[0], args);
   if(wynik == -1)
     printf("Proces potomny nie wykonal programu\n");
```

Wynik działania:

```
ajasielec@DESKTOP-J8MMMJM:~/PR/Parallel-Programming/lab-2$ ./clone
Imię i nazwisko: Anna Jasielec, numer procesu: 72623
./zajecia2Imię i nazwisko: Anna Jasielec, numer procesu: 72624
./zajecia2Imię i nazwisko: Anna Jasielec, numer procesu: 72625
./zajecia2Imię i nazwisko: Anna Jasielec, numer procesu: 72626
./zajecia2Imię i nazwisko: Anna Jasielec, numer procesu: 72627
./zajecia2Imię i nazwisko: Anna Jasielec, numer procesu: 72628
./zajecia2Imię i nazwisko: Anna Jasielec, numer procesu: 72629
./zajecia2Imię i nazwisko: Anna Jasielec, numer procesu: 72630
```

Fragment kodu w programie fork.c:

```
// Tablica argumentów do execv
      char* args1[] = {"/bin/ls", ".", NULL}; // // uruchomienie komedy ls
      char* args2[] = {"./zajecia2", NULL}; // uruchomienie programu

      wynik = execv(args2[0], args2);

       if(wynik==-1)
           printf("Proces potomny nie wykonal programu\n");
```

Wynik działania:

```
ajasielec@DESKTOP-J8MMMJM:~/PR/Parallel-Programming/lab-2$ ./fork
Imię i nazwisko: Anna Jasielec, numer procesu: 74020
./zajecia2Imię i nazwisko: Anna Jasielec, numer procesu: 74021
./zajecia2Imię i nazwisko: Anna Jasielec, numer procesu: 74022
./zajecia2Imię i nazwisko: Anna Jasielec, numer procesu: 74023
./zajecia2Imię i nazwisko: Anna Jasielec, numer procesu: 74024
./zajecia2Imię i nazwisko: Anna Jasielec, numer procesu: 74025
./zajecia2Imię i nazwisko: Anna Jasielec, numer procesu: 74026
```

3. **Wnioski:**
   - Czas tworzenia wątków (clone) jest znacznie krótszy niż czas tworzenia procesów (fork), co wynika z faktu, że wątki dzielą przestrzeń adresową, co ogranicza narzut systemowy.
   - Wersja zoptymalizowana (-O3) wykazuje krótszy czas tworzenia procesów i wątków w porównaniu do wersji debug (-g -DDEBUG), co jest rezultatem redukcji zbędnych operacji.

- Zmienne lokalne są unikalne dla każdego wątku, eliminując problemy ze współbieżnością, podczas gdy zmienne globalne mogą prowadzić do kolizji, co wpływa na ich wartość w wyniku działania równoległych wątków.
- Procesor jest wstanie wykonać około 110 000 operacji arytmetycznych lub 370 operacji wejścia/wyjścia w czasie tworzenia jednego wątku.
- Wartości zmiennych globalnych są zgodne z oczekiwaniami w wersji zoptymalizowanej, co potwierdza skuteczność zastosowanych algorytmów optymalizacyjnych.
- Funkcja exec() umożliwia efektywne zarządzanie procesami oraz uruchamianie nowych programów z przekazywanymi argumentami.