Tomasz Ligęza

Programowanie równoległe. Przetwarzanie równoległe i rozproszone.

Sprawozdanie z laboratorium 2.

Celem laboratorium było nabycie umiejętności tworzenia procesów i wątków przy użycia języka C, oraz wykorzystanie nabytej umiejętności w tworzeniu programów.

W ramach zajęć zrealizowałem następujące kroki:

- Przygotowałem katalog roboczy, w którym wypakowałem pliki potrzebne do wykonania laboratorium,
- Na poprzednim laboratorium wykonałem krok z tworzeniem biblioteki, więc mogłem pominąć ten krok,
- Uzupełniłem kod dodając procedury pomiaru czasu tworzenia wątków i procesów,

Kod źródłowy pliku fork.c:

```
inicjuj_czas();
for (i = 0; i < 1000; i++) {
    pid = fork();
    if (pid == 0) {
        zmienna_globalna++;
        exit(0);
    } else {
        wait(NULL);
    }
}
drukuj_czas();
```

Kod źródłowy pliku clone.c:

```
inicjuj_czas();
for (i = 0; i < 1000; i++) {
    pid = clone(&funkcja_watku, (void *) stos + ROZMIAR_STOSU,
            CLONE_FS | CLONE_FILES | CLONE_SIGHAND | CLONE_VM, 0);
    waitpid(pid, NULL, __WCLONE);
}
drukuj_czas();
```

Analiza wyników:

Średni czas tworzenia wątku wynosi: 4.30E-04s

Średni czas tworzenia procesu wynosi: 1.62E-05s

W czasie tworzenia wątku procesor może wykonać

4.30E-04s / 2.00896E-09s ~ 215 000 operacji arytmetycznych.

W czasie tworzenia wątku procesor może wykonać

4.30E-04s / 5.23365E-06s ~ 82 operacji wejścia wyjścia.

Po sprawdzeniu okazuje się, że optymalizacja nie wpływa na czas tworzenia procesów i wątków. Czas wykonania wynosi w przybliżeniu sumę czasu użytkownika i czasu systemowego.

- Stworzyłem program, w którym po kolei tworzę dwa wątki do działania równoległego,

```
int funkcja_watku(void *argument) {
   int zmienna_lokalna = 0;
   for(int i=0 ; i < 10000000 ; i++) {
      zmienna_lokalna++;
      zmienna_globalna++;
   }
   printf("Wartosc zmiennej globalnej w watku: %d\n", zmienna_globalna);
   printf("Wartosc zmiennej lokalnej w watku: %d\n", zmienna_lokalna);
   return 0;
}

main() {

   void *stos1;
   void *stos2;
   pid_t pid1;
   pid_t pid2;

   stos1 = malloc(ROZMIAR_STOSU);
   stos2 = malloc(ROZMIAR_STOSU);
   if (stos1 == 0 || stos2 == 0) {
      printf("Proces nadrzędny - blad alokacji stosu\n");
      exit(1);
   }

   inicjuj_czas();
   pid1 = clone(&funkcja_watku, (void *) stos1 + ROZMIAR_STOSU,
            CLONE_FS | CLONE_FILES | CLONE_SIGHAND | CLONE_VM, 0);

   pid2 = clone(&funkcja_watku, (void *) stos2 + ROZMIAR_STOSU,
            CLONE_FS | CLONE_FILES | CLONE_SIGHAND | CLONE_VM, 0);

   waitpid(pid1, NULL, __WCLONE);
   waitpid(pid2, NULL, __WCLONE);

   drukuj_czas();

   printf("Wartosc zmiennej globalnej w main: %d\n", zmienna_globalna);

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

Wypis wykonanego programu w terminalu:

Wartosc zmiennej globalnej w watku: 6202682
Wartosc zmiennej lokalnej w watku: 10000000
Wartosc zmiennej globalnej w watku: 11736365
Wartosc zmiennej lokalnej w watku: 10000000
czas standardowy = 0.000133
czas CPU = 0.000130
czas zegarowy = 0.080858
Wartosc zmiennej globalnej w main: 11736365

Jak widać wartości zmiennych lokalnych odpowiadają ilości operacji inkrementacji na nich wykonanych. Inaczej ma się sytuacja zmiennej globalnej, której wartość zawsze jest mniejsza bądź równa ilości operacji inkrementacji na niej wykonanych.

Dzieje się tak dlatego, że oba wątki chcą w tym samym momencie zinkrementować jedną zmienną.

Załóżmy, że wątek A pobiera wartość do rejestru (100) i ją inkrementuje (101). W tym samym momencie wątek B wykonuje identyczną sytuację. Po inkrementacji wątek A zapisuje zinkrementowaną wartość w zmiennej globalnej (101). Tą samą czynność wykonuje wątek B, tj. wapisuje wartość (101) w zmiennej globalnej. W ten sposób wartość zmiennej została zwiększona o 1, mimo iż oba wątki ją zinkrementowały.

- Wykonałem również polecenie na ocenę 4.0.

Kod zmienionej funkcji funkcja_watku() nowego programu poniżej:

```
int funkcja_watku(void *argument) {

    zmienna_globalna++;
    char* const arg[] = {"arg1", "arg2"};
    int wynik;
    wynik=execv("./program_clone", arg);
    if(wynik==-1)
      printf("Proces potomny nie wykonal programu\n");
   return 0;
}
```

Wnioski:

Tworzenie procesów i wątków daje ogrom możliwości i nie korzystanie z nich w dzisiejszych czasach jest ogromnym błędem i marnowaniem potencjału sprzętu. Niesie to ze sobą również problemy związane z pamięcią, do której w niektórych wypadkach mogą odnosić się dwa (bądź więcej) wątki naraz, co powoduje tzw. race condition.