

Sprawozdanie z przedmiotu Programowanie Równoległe i Rozproszone

Mechanizmy programowania współbieżnego w SO UNIX/LINUX

Wykonał:

Imię i nazwisko

Magdalena Paszko

Nr indeksu

76024

Grupa

L2

Prowadzący : dr inż. Krzysztof Szerszeń

1. Wprowadzanie

Funkcja systemowa `fork()` jest jedna z najważniejszych funkcji w systemach unixowych, także w Linuxie. Za jej sprawą, i tylko za jej sprawą, mogą się pojawiać w systemie nowe procesy. Jedynym procesem, który jest powoływany do życia w inny sposób jest proces o numerze (identyfikatorze) 1, czyli `init`.

Z punktu widzenia programisty, `fork()` jest bardzo prosta funkcja. Wywołuje się go tak:

```
pid = fork();
```

i w wyniku dostaje dwie różne wartości. Proces macierzysty, który wywołał `forka`, dostaje identyfikator dziecka. Dziecko dostaje w wyniku 0. Dzięki temu procesy mogą poznać "który jest który" bez uciekania się do sztuczek. Dziecko jest prawie dokładną kopią rodzica - różni się tylko tym, co zwraca `fork()`. Oczywiście w przypadku niepowodzenia dziecko nie jest tworzone a rodzic otrzymuje informacje o błędzie.

2. Definicja

Całkowanie numeryczne

Metoda polegająca na przybliżonym obliczaniu całek oznaczonych.

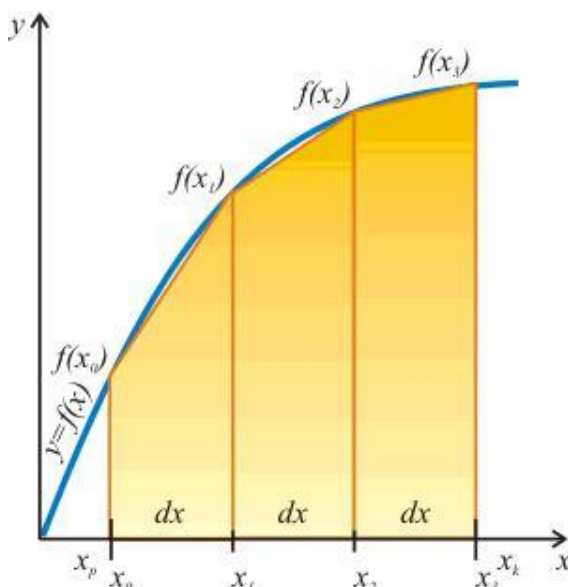
Proste metody całkowania numerycznego polegają na przybliżeniu całki za pomocą odpowiedniej sumy ważonej wartości całkowanej funkcji w kilku punktach.

Do uzyskania dokładniejszego przybliżenia dzieli się przedział całkowania na niewielkie fragmenty.

Ostateczny wynik to suma oszacowań całek w poszczególnych podprzedziałach.

Przedział zwykle dzieli się na równe podprzedziały.

Metoda trapezów



Przedział całkowania $\langle x_p, x_k \rangle$ dzielimy na $n+1$ równo odległych punktów $x_0, x_1, x_2, \dots, x_n$. Punkty te

wyznaczamy w prosty sposób wg wzoru:

dla $i = 0, 1, 2, \dots, n$

$$x_i = x_p + \frac{i}{n}(x_k - x_p)$$

Obliczamy odległość między dwoma sąsiednimi punktami - będzie to wysokość każdego trapezu:

$$dx = \frac{x_k - x_p}{n}$$

Dla każdego wyznaczonego w ten sposób punktu obliczamy wartość funkcji $f(x)$ w tym punkcie:

$$f_i = f(x_i), \text{ dla } i = 1, 2, \dots, n$$

Pole pod wykresem funkcji przybliżane jest polami n trapezów. Pole i -tego trapezu obliczamy wg wzoru:

dla $i = 1, 2, \dots, n$

$$f_i = f(x_i), \text{ dla } i = 1, 2, \dots, n$$

Przybliżona wartość całki jest sumą pól wszystkich otrzymanych w ten sposób trapezów:

$$s = P_1 + P_2 + \dots + P_n$$

czyli:

$$s = \frac{f_0 + f_1}{2} dx + \frac{f_1 + f_2}{2} dx + \frac{f_2 + f_3}{2} dx + \dots + \frac{f_{n-2} + f_{n-1}}{2} dx + \frac{f_{n-1} + f_n}{2} dx$$

$$s = \frac{dx}{2} (f_0 + f_1 + f_1 + f_2 + f_2 + f_3 + \dots + f_{n-2} + f_{n-1} + f_{n-1} + f_n)$$

$$s = \frac{dx}{2} (f_0 + 2f_1 + 2f_2 + \dots + 2f_{n-1} + f_n)$$

$$s = dx (f_1 + f_2 + \dots + f_{n-1} + \frac{f_0 + f_n}{2})$$

Wyprowadzony na końcu wzór jest podstawą przybliżonego wyliczania całki w metodzie trapezów.

$$\int_{x_p}^{x_k} f(x) dx \approx \frac{x_k - x_p}{n} \left(\sum_{i=1}^{n-1} f\left(x_p + i \frac{x_k - x_p}{n}\right) + \frac{f(x_p) + f(x_k)}{2} \right)$$

Wzór Leibniza

Wzór pozwalający obliczyć n-tą pochodną iloczynu funkcji.

Został wprowadzony przez niemieckiego matematyka Gottfrieda Leibniza.

$$\pi = 4 \cdot \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{2 \cdot n - 1} = 4 \cdot \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots \right)$$

3. Implementacja algorytmów

a) Metoda trapezów

```
Linux Lite Terminal -
File Edit View Terminal Tabs Help
GNU nano 4.8 zadanie1.c Modified
#include <stdio.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <math.h>
#include <time.h>
#include "err.h"
#include "err.c"

float func(float x)
{
    return (4*x-6*x+5);
}

float met_trap(float xp, float xk, int n)
{
    float dx, calka = 0;
    dx = (xk - xp)/n;
    for (int i = 1; i < n; i++)
    {
        calka += func(xp+i*dx);
    }
    calka += (func(xp)+func(xk))/2;
    calka *= dx;
    printf("calka %f \n", calka);
}

float proces()
{
    srand(getpid()+getppid());
    float x = 5.0;
    float a = (float)rand()/(float)(RAND_MAX/x);
    float b = (float)rand()/(float)(RAND_MAX/x);
    int n = (rand()%1000+100);
    if(a<b){
        met_trap(a,b,n);
    }
    else{
        printf("a>b niestety nie mozna policzyc calki \n");
    }
}
```

Funkcja met_trap przyjmuje przedział <a,b> oraz n liczbę trapezów do obliczenia całki.

Oblicza całkę z wzoru metody trapezów którą opisałam powyżej.

```
float proces()
{
    srand(getpid()+getppid());
    float x = 5.0;
    float a = (float)rand()/(float)(RAND_MAX/x);
    float b = (float)rand()/(float)(RAND_MAX/x);
    int n = (rand()%1000+100);
    if(a<b){
        met_trap(a,b,n);
    }
    else{
        printf("a>b niestety nie mozna policzyc calki \n");
    }
}
```

Funkcja proces ustawia punkt startowy, który jest stosowany do generowania serii pseudo losowych liczb, jednak wstawiając NULL i wywołując wątki liczba zawsze była taka sama. Idealnym rozwiązaniem okazało się mieć inne ziarno dla każdego wywołania programu. Jednym ze sposobów jest użycie srand (getpid ()); gdzie getpid () zwraca numer identyfikacyjny bieżącego procesu. Lepsze ziarno można jednak uzyskać, łącząc getpid () czyli identyfikatorem rodzica:srand(getpid () + getppid ()). Następnie rand zwraca pseudolosową liczbę zmiennoprzecinkową a i b oraz całkowitą n oraz sprawdza czy a<b jeśli tak to wywołuje met_trap.

```
Linux Lite Terminal -
File Edit View Terminal Tabs Help
GNU nano 4.8 zadanie1.c

int main ()
{
printf("Prosze podac ilosc procesow: ");
int ilosc;
scanf("%d", &ilosc);

pid_t pid;
int i;
/* tworzy nowe procesy */
for (i = 1; i <= ilosc; i++)
switch (pid = fork()) {
case -1: /* blad */
syserr("Error in fork\n");
case 0: /* proces potomny */
printf(" Proces %d rozpoczyna prace \n ",i);
proces();
return 0;
}

/* czeka na zakonczenie procesow potomnych */
for (i = 1; i <= ilosc; i++) {
if (wait(0) == -1){
syserr("Error in wait\n");
}
}
return 0;
}

^G Get Help      ^O Write Out     ^W Where Is      ^K Cut Text      ^J Justify       ^C Cur Pos       M-U Undo         M-A Mark Text
^X Exit          ^R Read File     ^\ Replace       ^U Paste Text    ^T To Spell     ^_ Go To Line    M-E Redo         M-6 Copy Text
```

W main tak jak widać po komentarzach musimy na początku podać ilość procesów jaka ma wykonać.

Tworzymy nowe procesy.

Jeśli wartość zwracana wynosi -1 oznacza to, że proces sprawdzający wartość zwracaną to proces-rodzic, ale nie udało się stworzyć procesu potomnego (dziecka) – błąd.

Jeśli wartość zwracana wynosi 0 oznacza to, że proces sprawdzający wartość zwracaną to proces-dziecko w którym funkcje proces a one met_trapezow – proces potomny.

Na koniec mamy funkcje wait co oznacza oczekiwanie na zakończenie procesu potomnego. Funkcja zwraca identyfikator (pid) procesu, który się zakończył. Gdy zwraca -1 oznacza że jakiś proces nie zakończył się.

Wynik programu:

```
linux ~ > Documents gcc zadanie1.c -o zadanie1 -lm
linux ~ > Documents ./zadanie1
Prosze podac ilosc procesow: 10
Proces 4 rozpoczyna prace
Proces 7 rozpoczyna prace
Proces 3 rozpoczyna prace
calka 0.792679
Proces 5 rozpoczyna prace
Proces 2 rozpoczyna prace
Proces 6 rozpoczyna prace
calka 1.247457
calka -0.471698
Proces 8 rozpoczyna prace
a>b niestety nie mozna policzyc calki
a>b niestety nie mozna policzyc calki
calka -3.239468
Proces 9 rozpoczyna prace
Proces 10 rozpoczyna prace
calka 1.652073
Proces 1 rozpoczyna prace
a>b niestety nie mozna policzyc calki
calka -3.844780
calka 4.108016
linux ~ > Documents
```

b) Wzór Leibniza

```
Linux Lite Terminal -
File Edit View Terminal Tabs Help
GNU nano 4.8 program2.c Modified
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include "err.h"
#include "err.c"
#include <math.h>
#include <time.h>
#include <sys/mman.h>

float lib()
{
    srand(getpid()+getppid());
    int n=0;
    n= (rand()%4000+100);
    float wynik = 0, liczba = 0, zmienna = 0;
    for(int i=0; i<=n; i++){
        liczba =(pow((- 1.0), i ) ) / ( 2 * i + 1 );
        zmienna +=liczba;
    }
    wynik = 4*zmienna;
    return wynik;
}

int proces()
{
    printf("Liczba PI wynosi: %f \n", lib());
}

int main ()
{
    // ...
}
```

Funkcja lib ustawia punkt startowy, który jest stosowany do generowania serii pseudo losowych liczb analogicznie jak w pierwszym przykładzie. Oblicza przybliżenie liczby PI wzorem Leibniza do liczby która jest generowana pseudolosowo zwracając na koniec wynik przybliżenia.

```
Linux Lite Terminal -
File Edit View Terminal Tabs Help
GNU nano 4.8 program2.c Modified
}

int main ()
{
    printf("Prosze podac ilosc procesow: ");
    int ilosc;
    scanf("%d", &ilosc);

    pid_t pid;
    int i;
    /* tworzy nowe procesy */
    for (i = 1; i <= ilosc; i++)
        switch (pid = fork()) {
            case -1: /* blad */
                syserr("Error in fork\n");

            case 0: /* proces potomny */
                printf(" Proces %d rozpoczyna prace \n ",i);
                proces();
                return 0;
        }

    /* czeka na zakonczenie procesow potomnych */
    for (i = 1; i <= ilosc; i++) {
        if (wait(0) == -1){
            syserr("Error in wait\n");
        }
    }
    return 0;
}
```

Main jest analogiczny jak w przykładzie pierwszym funkcja proces wywołuje funkcje lib która oblicza liczbę PI

Wynik programu:

```
linux ~ > Documents nano program2.c
linux ~ > Documents gcc program2.c -o program2 -lm
linux ~ > Documents ./program2
Prosze podac ilosc procesow: 11
Proces 3 rozpoczyna prace
Proces 4 rozpoczyna prace
Liczba PI wynosi: 3.144426
Proces 2 rozpoczyna prace
Proces 5 rozpoczyna prace
Liczba PI wynosi: 3.141004
Liczba PI wynosi: 3.141116
Proces 6 rozpoczyna prace
Liczba PI wynosi: 3.141204
Proces 7 rozpoczyna prace
Liczba PI wynosi: 3.141850
Proces 8 rozpoczyna prace
Proces 9 rozpoczyna prace
Proces 10 rozpoczyna prace
Liczba PI wynosi: 3.140181
Proces 1 rozpoczyna prace
Proces 11 rozpoczyna prace
Liczba PI wynosi: 3.142068
Liczba PI wynosi: 3.141991
Liczba PI wynosi: 3.141978
Liczba PI wynosi: 3.140997
Liczba PI wynosi: 3.141329
linux ~ > Documents
```