

**Міністерство Освіти І НАУКИ України**  
**Національний університет "Львівська політехніка"**

**Інститут ІКНІ**  
**Кафедра ПЗ**

**ЗВІТ**

До лабораторної роботи № 4

**На тему:** *“Створення та керування процесами засобами API в операційній системі LINUX ”*

**З дисципліни:** *“Операційні системи”*

**Лектор:**

Старший викладач ПЗ  
Грицай О.Д.

**Виконав:**

ст. гр. ПЗ-22  
Ясногородський Н.В.

**Прийняв:**

Старший викладач ПЗ  
Грицай О.Д.

« \_\_\_\_ » \_\_\_\_\_ 2022 р.

$\Sigma =$  \_\_\_\_

Львів – 2022

**Тема роботи:** створення та керування процесами засобами API в операційній системі Linux.

**Мета роботи:** ознайомитися з багатопоточністю в ОС Linux. Навчитися працювати з процесами, у ОС Linux.

### **Теоретичні відомості**

Процеси в ОС Linux створюються з допомогою системного виклику `fork()`. Цей виклик створює точну копію батьківського процесу. Після виконання `fork()` усі ресурси дочірнього процесу - це копія ресурсів батька. Копіювати процес з усіма виділеними сторінками пам'яті - справа дорога, тому в ядрі Linux використовується технологія Copy-On-Write. Всі сторінки пам'яті батька позначаються як read-only і стають доступні і батькові, і дитині. Як тільки один з процесів змінює дані на певній сторінці, ця сторінка не змінюється, а копіюється і змінюється вже копія. Оригінал при цьому «відв'язується» від даного процесу. Як тільки read-only оригінал залишається «прив'язаним» до одного процесу, сторінці знову призначається статус read-write.

Результат виклику `fork()` повертається і в батьківський і в дочірній процеси, які починають виконувати однакові інструкції. Відмінність між батьківським і дочірнім процесом полягає лише у :

- Дочірньому процесу присвоюється унікальний PID
- Ідентифікатори батьківського процесу PPID для цих процесів різні
- Дочірній процес вільний від сигналів, що очікують

Значення, що повертає `fork()` для батьківського це PID дочірнього, а для дочірнього 0.

## ЗАВДАННЯ

1. Виконати в окремому процесі табулювання функцій.
2. Реалізувати табулювання функцій у 2-ох, 4-ох, 8-ох процесорах. Виміряти час роботи процесів. Порівняти результати роботи в одному і в багатьох процесорах.
3. Реалізувати можливість зміни пріоритету виконання процесу.
4. Реалізувати можливість зупинки і відновлення роботи процесу.
5. Реалізувати можливість вбиття процесу.
6. Порівняти результати виконання програми під ОС Windows та Linux.
7. Результати роботи відобразити у звіті.

### Варіант 7:

Табулювати функцію  $\ln x$ , задану розкладом в ряд Тейлора, в області її визначення на відрізок від А до В (кількість кроків не менше 100 000 – задається користувачем).

### Хід виконання роботи

Спочатку створю підпрограму, яка рахуватиме ряди Тейлора на заданому проміжку та виводитиме PID / час виконання після завершення обрахунків:

```
#include <unistd.h>
```

```
#include <chrono>
```

```
#include <cmath>
```

```
#include <iostream>
```

```
#include <string>
```

```
void tabulate_lnx(double a, double b, double step, double  
iter_count) {
```

```
    pid_t pid = getpid();
```

```
    for (double n = a; n <= b; n += step) {
```

```
        double num, mul, cal, sum = 0;
```

```
        num = (n - 1) / (n + 1);
```

```
        for (int i = 1; i <= iter_count; i++) {
```

```
            mul = (2 * i) - 1;
```

```
            cal = pow(num, mul);
```

```
            cal = cal / mul;
```

```
            sum = sum + cal;
```

```
        }
```

```
        sum = 2 * sum;
```

Код програми:

```
#include <signal.h>
#include <unistd.h>

#include <chrono>
#include <iostream>

int ask_for_pid() {
    int pid;
    std::cout << "Enter PID of the process:" << std::endl;
    std::cin >> pid;
    return pid;
}

int main() {
    pid_t proc[8];
    int status[8];
    double A, B, step;
    int precision, countProc;
    std::cout << "Please, enter A (lower bound): " << std::e
    std::cin >> A;
    std::cout << "Please, enter B (upper bound): " << std::e
    std::cin >> B;
    std::cout << "Please, enter step for tabulation for each
process: "
        << std::endl;
    std::cin >> step;
    std::cout << "Please, enter iteration count for each pro
(precision): "
        << std::endl;
    std::cin >> precision;
    std::cout << "Please, enter the number of processes: " <
std::endl;
    std::cin >> countProc;
    double rangePerProcess = (B - A) / countProc;

    // creation of processes
    for (int i = 0; i < countProc; i++) {
        proc[i] = fork();
        if (proc[i] == -1) {
```



## Протокол роботи програми

Виконання в 4-х процессах:

```
Please, enter A (lower bound):
1
Please, enter B (upper bound):
10
Please, enter step for tabulation for each process:
0.001
Please, enter iteration count for each process (precision):
100
Please, enter the number of processes:
4
*****0
*****0
Launch with args:0 2.25 0.001 100
tab 1 of window id 4354
*****1
*****1
Launch with args:2.25 4.5 0.001 100
tab 1 of window id 4355
*****2
*****2
Launch with args:4.5 6.75 0.001 100
tab 1 of window id 4356
*****3
*****3
Launch with args:6.75 9 0.001 100
tab 1 of window id 4357

Please, choose the action:
1. Change priority
2. Suspend process
3. Resume process
4. Kill process
5. Quit
█
```

```
nickyasnorodskiy — program_15 6.750000 9.000000 0.001000 100 —...
ln(8.99)=2.19611
PID: 61280
ln(8.991)=2.19622
PID: 61280
ln(8.992)=2.19634
PID: 61280
ln(8.993)=2.19645
PID: 61280
ln(8.994)=2.19656
PID: 61280
ln(8.995)=2.19667
PID: 61280
ln(8.996)=2.19678
PID: 61280
ln(8.997)=2.19689
PID: 61280
ln(8.998)=2.197
PID: 61280
ln(8.999)=2.19711
PID: 61280
ln(9)=2.19722
PID: 61280
Duration: 8.07371 ms

nickyasnorodskiy — program_15 0.000000 2.250000 0.001000 100 —...
ln(2.24)=0.806476
PID: 60929
ln(2.241)=0.806922
PID: 60929
ln(2.242)=0.807368
PID: 60929
ln(2.243)=0.807814
PID: 60929
ln(2.244)=0.80826
PID: 60929
ln(2.245)=0.808706
PID: 60929
ln(2.246)=0.809151
PID: 60929
ln(2.247)=0.809596
PID: 60929
ln(2.248)=0.810041
PID: 60929
ln(2.249)=0.810486
PID: 60929
ln(2.25)=0.81093
PID: 60929
Duration: 9.48713 ms

nickyasnorodskiy — program_15 4.500000 6.750000 0.001000 100 —...
ln(6.739)=1.90791
PID: 61220
ln(6.74)=1.90806
PID: 61220
ln(6.741)=1.90821
PID: 61220
ln(6.742)=1.90836
PID: 61220
ln(6.743)=1.9085
PID: 61220
ln(6.744)=1.90865
PID: 61220
ln(6.745)=1.9088
PID: 61220
ln(6.746)=1.90895
PID: 61220
ln(6.747)=1.9091
PID: 61220
ln(6.748)=1.90925
PID: 61220
ln(6.749)=1.90939
PID: 61220
Duration: 8.70712 ms

nickyasnorodskiy — program_15 2.250000 4.500000 0.001000 100 —...
ln(4.49)=1.50185
PID: 61119
ln(4.491)=1.50208
PID: 61119
ln(4.492)=1.5023
PID: 61119
ln(4.493)=1.50252
PID: 61119
ln(4.494)=1.50274
PID: 61119
ln(4.495)=1.50297
PID: 61119
ln(4.496)=1.50319
PID: 61119
ln(4.497)=1.50341
PID: 61119
ln(4.498)=1.50363
PID: 61119
ln(4.499)=1.50386
PID: 61119
ln(4.5)=1.50408
PID: 61119
Duration: 9.71729 ms
```

Вимірявши час виконання програми на різних кількості процесів, ми бачимо, що процес пришвидшується зі збільшенням процесів

### Зупинка процесу

```
Please, choose the action:
1. Change priority
2. Suspend process
3. Resume process
4. Kill process
5. Quit
2
Enter PID of the process:
26796
```

### Відновлення процесу

```
Please, choose the action:
1. Change priority
2. Suspend process
3. Resume process
4. Kill process
5. Quit
3
Enter PID of the process:
26796
```

### Завершення процесу

```
Please, choose the action:
1. Change priority
2. Suspend process
3. Resume process
4. Kill process
5. Quit
4
Enter PID of the process:
26796
```

### Зміна пріоритету

```
Please, choose the action:
1. Change priority
2. Suspend process
3. Resume process
4. Kill process
5. Quit
1
Enter PID of the process:
26786
Enter priority
16
priority change result: 0 errno: 10
```

### Висновки

Виконуючи цю лабораторну роботу, я навчився працювати з Linux API та став краще розуміти, як відбувається взаємодія із процесами на рівні програмного забезпечення на цій операційній системі.