**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Параллельные алгоритмы»**

Тема: Процессы в UNIX

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Ковынев М.В. |
| Преподаватель |  | Митяков А.В. |

Санкт-Петербург

2018

**Цель работы.**

Знакомство с UNIX API для управления процессами в ОС.

**Задание.**

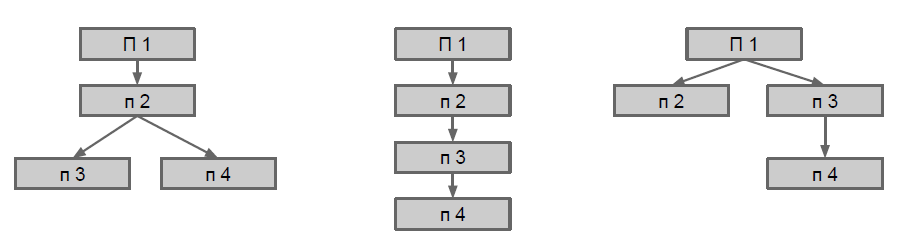
Реализовать следующее генеалогическое древо процессов, изображенное на рис. 1:

Рисунок 1 – Требуемая структура процессов

**Основные теоретические положения.**

Программа - бинарный файл. Процесс - программа в стадии исполнения. Процесс в UNIX:

* работает в пользовательском пространстве
* имеет определенные атрибуты в ядре

Каждый процесс исполняется в собственном виртуальном адресном пространстве. Для взаимодействия нескольких процессов необходимо использовать IPC. Устройство процесса схематично показано на рис. 2.

Рисунок 2 – Устройство процесса

**Ход работы.**

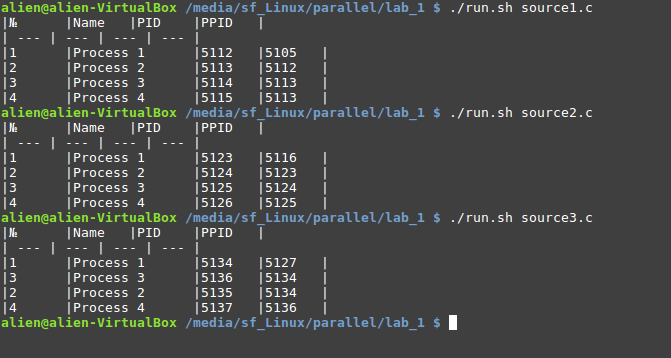
1. Реализуем функцию *void printProc(int proc\_id),* которая будет выводить информацию о PID и PPIDпроцесса, используя системные вызовы *getpid()* и *getppid()*.
2. В функции *int main(int argc, char\* argv[])* будем последовательно создавать процессы согласно заданию используя системный вызов *pid\_t fork()*.
3. Возвращаемое значение типа *pid\_t* равно нулю для дочернего процесса, **PID** созданного дочернего процесса для родительского и отрицательное значение в случае ошибки.
4. Используя это возвращаемое значения реализуем ветвление в программе для построения требуемой иерархии процессов.
5. Далее приведём примеры работы написанных программ.

Рисунок 3 – Результаты работы программ

1. Как видно из вывода каждой программы реализовано требуемое древо процессов. Исходный код находится в приложениях А, Б, В.

**Выводы**

Процессы - в целом надежнее чем потоки, т.к. работают в изолированном виртуальном пространстве. Процессы подходят для решения следующих задач:

* процессы, взаимодействующие в разные время
* распределенные приложения
* требуется безопасность по данным

Однако создание процессов требует большого количества времени. Также процессы требуют более сложного IPC, чем потоки.

**ПРИЛОЖЕНИЕ А.**

**SOURCE1.C**

#include "stdio.h"

#include "unistd.h"

#include "sys/types.h"

#include "sys/wait.h"

void printHead(){

printf("|№\t|Name\t|PID\t|PPID\t|\n");

printf("| --- | --- | --- | --- |\n");

}

void printProc(int proc\_id){

printf("|%d\t|Process %d\t|%d\t|%d\t|\n", proc\_id, proc\_id, getpid(), getppid());

}

void printError(int id){

printf("Error to create process %d\n", id);

}

int main(){

printHead();

printProc(1);

pid\_t pid = fork();

if (pid == 0) {

// P2 from P1

printProc(2);

pid = fork();

if (pid == 0) {

// P3 from P2

printProc(3);

}

else if (pid > 0) {

// P2 creates P4

pid = fork();

if (pid == 0) {

// P4 from P2

printProc(4);

}

else if (pid > 0) {

// End P2

wait(NULL);

}

else {

printError(4);

}

}

else {

printError(3);

}

}

else if (pid > 0) {

// End P1

wait(NULL);

}

else {

printError(2);

}

}

**ПРИЛОЖЕНИЕ Б.**

**SOURCE2.C**

#include "stdio.h"

#include "unistd.h"

#include "sys/types.h"

#include "sys/wait.h"

void printHead(){

printf("|№\t|Name\t|PID\t|PPID\t|\n");

printf("| --- | --- | --- | --- |\n");

}

void printProc(int proc\_id){

printf("|%d\t|Process %d\t|%d\t|%d\t|\n", proc\_id, proc\_id, getpid(), getppid());

}

void printError(int id){

printf("Error to create process %d\n", id);

}

int main(){

printHead();

printProc(1);

pid\_t pid = fork();

if (pid == 0) {

// P2 from P1

printProc(2);

pid = fork();

if (pid == 0) {

// P3 from P2

printProc(3);

pid = fork();

if (pid == 0) {

// P4 from P3

printProc(4);

}

else if (pid > 0) {

// End P3

wait(NULL);

}

else {

printError(4);

}

}

else if (pid > 0) {

// End P2

wait(NULL);

}

else {

printError(2);

}

}

else if (pid > 0) {

// End P1

wait(NULL);

}

else {

printError(2);

}

}

**ПРИЛОЖЕНИЕ В.**

**SOURCE3.C**

#include "stdio.h"

#include "unistd.h"

#include "sys/types.h"

#include "sys/wait.h"

void printHead(){

printf("|№\t|Name\t|PID\t|PPID\t|\n");

printf("| --- | --- | --- | --- |\n");

}

void printProc(int proc\_id){

printf("|%d\t|Process %d\t|%d\t|%d\t|\n", proc\_id, proc\_id, getpid(), getppid());

}

void printError(int id){

printf("Error to create process %d\n", id);

}

int main(){

printHead();

printProc(1);

pid\_t pid = fork();

if (pid == 0) {

// P2 from P1

printProc(2);

}

else if (pid > 0) {

// P1 again

pid = fork();

if (pid == 0) {

// P3 from P1

printProc(3);

pid = fork();

if (pid == 0) {

// P4 from P3

printProc(4);

}

else if (pid > 0) {

// End P3

wait(NULL);

}

else {

printError(4);

}

}

else if (pid > 0) {

// End P1

wait(NULL);

}

else {

printError(3);

printError(4);

}

}

else {

printError(2);

}

}