Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра интеллектуальных информационных технологий

**Лабораторная работа №2 по курсу «OC» на тему:**

**«Создание процессов в ОС Linux»**

Вариант 1

Выполнила студентка Аблакулов А.Д.

группы 021731:

Проверил: Цирук. В.А.

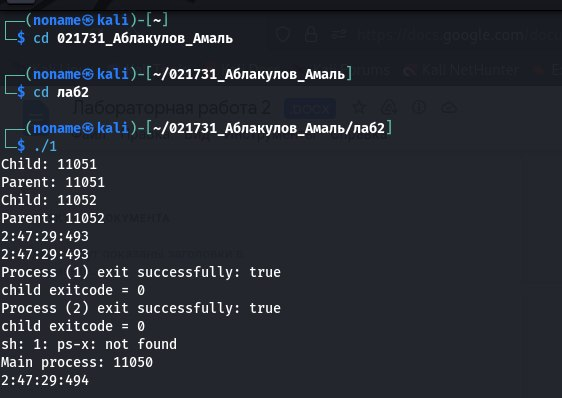
**МИНСК**

2022

**Цель работы:** научиться создавать процессы и потоки, а также управлять ими

**Задание для выполнения:**

1. Написать программу, создающую два дочерних процесса с использованием двух вызовов fork(). Родительский и два дочерних процесса должны выводить на экран



#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/timeb.h>

#include <time.h>

int main()

{

int status;

pid\_t child1, child2;

if (child1 = fork() == 0)

{

printf("Child: %d\n", getpid());

printf("Parent: %d\n", getpid());

}

else

{

if (child2 = fork() == 0)

{

printf("Child: %d\n", getpid());

printf("Parent: %d\n", getpid());

}

else

{

waitpid(child1, &status, 0);

printf("Process (1) exit successfully: %s\n", (WIFEXITED(status) ? "true" : "false"));

printf("child exitcode = %i\n", WEXITSTATUS(status));

waitpid(child2, &status, 0);

printf("Process (2) exit successfully: %s\n", (WIFEXITED(status) ? "true" : "false"));

printf("child exitcode = %i\n", WEXITSTATUS(status));

system("ps-x");

printf("Main process: %d\n", getpid());

}

}

struct timeb sys\_time;

struct tm \*loctime;

ftime(&sys\_time);

loctime = localtime(&sys\_time.time);

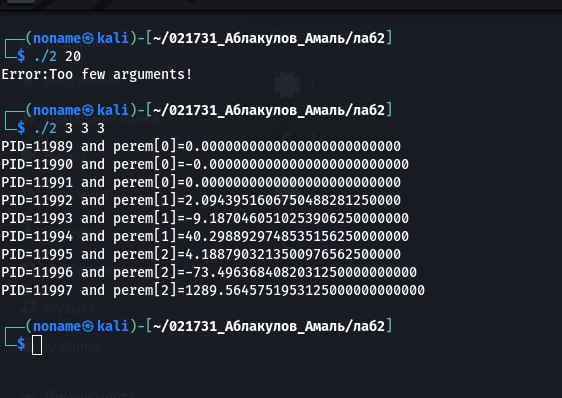
printf("%d:%d:%d:%d\n", loctime->tm\_hour, loctime->tm\_min, loctime->tm\_sec, sys\_time.millitm);

return 0;

}

Индивидуальное задание 1

1. Написать программу нахождения массива K последовательных значений функции y[i]=sin(2\*PI\*i/N) (где i=0, 1, 2...K-1) с использованием ряда Тейлора. Пользователь задаёт значения K, N и количество n членов ряда Тейлора. Для расчета каждого члена ряда Тейлора запускается отдельный поток. Каждый поток выводит на экран свой pid и рассчитанное значение ряда. Головной процесс суммирует все члены ряда Тейлора, и полученное значение y[i] записывает в файл.



#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/wait.h>

#include <fcntl.h>

#include <unistd.h>

#include <errno.h>

#define PI 3.14159265358979323846

extern int errno;

//deleting string from substring

void DeletePartPath(char \*str, char \*fnd, char \*buf)

{

char \*p=strstr(str, fnd);

if(p==NULL)

{strcpy(buf, str); return;}

char \*fnd1=fnd;

while(str<p)

{\*buf=\*str; buf++; str++; \*buf='\0';}

while(\*fnd1)

{str++; fnd1++;}

strcpy(buf, str);

return;

}

//type of error

void mist(char err[])

{

switch(errno)

{

case ENOTTY:{fprintf(stderr,"%s Error: Uknown operation I/O control!\n",err);break;}

case EACCES:{fprintf(stderr,"%s Error: Access is denied!\n",err);break;}

case EBADF :{fprintf(stderr,"%s Error: Invalid file descriptor!\n",err);break;}

case EBUSY :{fprintf(stderr,"%s Error: Resourse is busy!\n",err);break;}

case EMFILE:{fprintf(stderr,"%s Error: Too many open files!\n",err);break;}

case EISDIR:{fprintf(stderr,"%s Error: It is directory!\n",err);break;}

default: {fprintf(stderr,"%s Error:!\n",err);break;}

}

}

float PowFloat(float value,int in)

{

int i;

float result=1;

for(i=1;i<=abs(in);i++)

result\*=value;

return result;

}

int PowMinusOne(int in)

{

if((in%2)==0) return 1;

else return -1;

}

int TailorFunction(char err[], int K, int N, int FF)

{

FILE \*input,\*output;

char InPath[]="/tmp/math.txt";

char OuPath[]="/tmp/result.txt";

if((input=fopen(InPath,"w+"))==NULL)

{

DeletePartPath(err,"./",err);

mist(err);

return 1;

}

long int factorial=1;

int cnt=1; //counter

int sign;

float degree; //count X^i

float perem=0;

pid\_t pid;

int processes = 0;

for (int i=0;i<N;i++)

{

for (int j=0;j<K;j++)

{

if (processes == (FF+1))

{

wait(NULL);

processes--;

}

pid = fork();

if (pid == 0)

{

//elements

degree=PowFloat((2\*PI\*i/N),2\*j+1);//counting X^(2i+1)

sign=PowMinusOne(j+2); //change sign

perem=sign\*degree/factorial;

cnt=cnt+2; //next element of factorial

factorial=factorial\*(cnt-1)\*cnt; //counting of next factorial element

//counting and ID of process

fprintf(stdout,"PID=%d and perem[%d]=%.25f\n",getpid(),i,perem);

//forming of string

if(fprintf(input, "%d %d %.8lf\n", getpid(), i, perem) ==-1)

{

DeletePartPath(err,"./",err);

mist(err);

return 1;

}

exit(0);

}

processes++;

}

}

while (wait(NULL) > 0) { };

if((output=fopen(OuPath,"w"))==NULL)

{

DeletePartPath(err,"./",err);

mist(err);

return 1;

}

double \*result = alloca(sizeof(double)\*N);

memset(result, 0, sizeof(double)\*N);

rewind(input);

int pidd, f;

double member\_value;

for (int i = 0; i < N\*K; i++) {

if (fscanf(input, "%d %d %lf", &pidd, &f, &member\_value) == -1){

DeletePartPath(err,"./",err);

mist(err);

return 1;

}

result[f] += member\_value;

}

for (int i = 0; i < N; i++)

{

if (fprintf(output, "y[%d]=%.8lf\n", i, result[i]) == -1)

{

DeletePartPath(err,"./",err);

mist(err);

return 1;

}

}

//trying to close

if(fclose(input)==-1)

{

DeletePartPath(err,"./",err);

mist(err);

return 1;

}

//trying to close

if(fclose(output)==-1)

{

DeletePartPath(err,"./",err);

mist(err);

return 1;

}

return 0;

}

int main(int argc,char \*argv[])

{

int K,N,FF;

//number of arguments

if(argc<4)

{

fprintf(stderr,"Error:Too few arguments!\n");

return 1;

}

K=atoi(argv[1]);

N=atoi(argv[2]);

FF=atoi(argv[3]);

TailorFunction(argv[0],K,N,FF);

return 0;

}

Индивидуальное задание 2

Написать программу, которая будет реализовывать следующие функции:

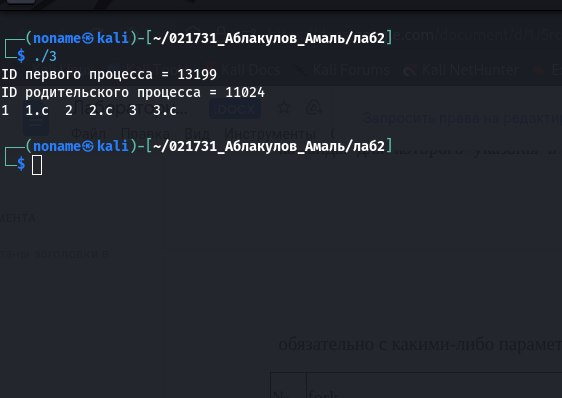
* сразу после запуска получает и сообщает свой ID и ID родительского процесса;
* перед каждым выводом сообщения об ID процесса и родительского процесса эта информация получается заново;
* порождает процессы, формируя генеалогическое дерево согласно варианту, сообщая, что "процесс с ID таким-то породил процесс с таким-то ID";
* перед завершением процесса сообщить, что "процесс с таким-то ID и таким- то ID родителя завершает работу";
* один из процессов должен вместо себя запустить программу, указанную в варианте задания.

На основании выходной информации программы предыдущего пункта изобразить генеалогическое дерево процессов (с указанием идентификаторов процессов). Объяснить каждое выведенное сообщение и их порядок в предыдущем пункте.

В столбце **fork** описано генеалогическое древо процессов: каждая цифра указывает на относительный номер (не путать с pid) процесса, являющегося родителем для данного процесса. Например, строка 0 1 1 1 3 означает, что первый процесс не имеет родителя среди ваших процессов (порождается и запускается извне), второй, третий и четвертый - порождены первым, пятый - третьим.

В столбце exec указан номер процесса, выполняющего вызов **exec**, команды для которого указаны в последнем столбце. Запускайте команду обязательно с какими-либо параметрами.





//0111335 1 ls

#include <unistd.h>

#include <sys/types.h>

#include <stdio.h>

#include <stdlib.h>

int main(){

pid\_t pid;

char \*arg[] = {"/usr/bin/ls", 0};

printf("ID первого процесса = %d\nID родительского процесса = %d\n", getpid(), getppid());

execv(arg[0], arg); // exec() вызов команды ls

// Порождение второго процесса

printf("\n\tПорождение процесса №2\n");

if ((pid = fork()) == -1)

printf("Ошибка!\n");

else if (pid == 0)

{

//printf("\t\t\tДочерний процесс\n");

printf("ID этого процесса = %d\nID родительского процесса = %d\n", getpid(), getppid());

// Порождение четвертого процесса

{

printf("\n\tПорождение процесса №4\n");

if ((pid = fork()) == -1)

printf("Ошибка!\n");

else if (pid == 0){

printf("\t\t\tДочерний процесс\n");

printf("ID этого процесса = %d\nID родительского процесса = %d\n", getpid(), getppid());

printf("Завершился процесс: PID = %d, PPID = %d\n", getpid(), getppid());

exit(0);

}

else

sleep(2);

}

// Порождение пятого процесса

{

printf("\n\tПорождение процесса №5\t\n");

if ((pid = fork()) == -1)

printf("Ошибка!\n");

else if (pid == 0)

{

printf("\t\t\tДочерний процесс\n");

printf("ID этого процесса = %d\nID родительского процесса = %d\n", getpid(), getppid());

// Порождение шестого процесса

{

printf("\n\t\tПорождение процесса №6\t\n");

if ((pid = fork()) == -1)

printf("Ошибка!\n");

else if (pid == 0)

{

printf("\t\t\tДочерний процесс\n");

printf("ID этого процесса = %d\nID родительского процесса = %d\n",

getpid(), getppid());

printf("Завершился процесс: PID = %d, PPID = %d\n", getpid(), getppid());

exit(0);

}

else

sleep(2);

}

// Порождение седьмого процесса

{

printf("\n\t\tПорождение процесса №7\t\n");

if ((pid = fork()) == -1)

printf("Ошибка!\n");

else if (pid == 0)

{

printf("\t\t\tДочерний процесс\n");

printf("ID этого процесса = %d\nID родительского процесса = %d\n",

getpid(), getppid());

printf("Завершился процесс: PID = %d, PPID = %d\n", getpid(), getppid());

exit(0);

}

else

sleep(2);

}

printf("Завершился процесс: PID = %d, PPID = %d\n", getpid(), getppid());

exit(0);

}

else sleep(4);

}

printf("Завершился процесс: PID = %d, PPID = %d\n", getpid(), getppid());

exit(0);

}

else sleep(4);

// Порождение третьего процесса

printf("\nПорождение процесса №3\n");

if ((pid = fork()) == -1)

printf("Ошибка!\n");

else if (pid == 0)

{

printf("\t\t\tДочерний процесс\n");

printf("ID этого процесса = %d\nID родительского процесса = %d\n", getpid(), getppid());

printf("Завершился процесс: PID = %d, PPID = %d\n", getpid(), getppid());

exit(0);

}

else

sleep(4);

sleep(4);

return 0;

}

**Вывод:**

Ознакомился с основными командами OS Linux , научился запускать скрипты и работать с командной строкой изучил утилиту gcc чтоб запускать .c файлы