МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В. Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №8

«Многопоточное программирование и средства синхронизациии процессов в ОС Linux»

по дисциплине

«Операционные системы»

|  |  |
| --- | --- |
|  | Выполнил:  студент 3 курса  Направления подготовки  09.03.01 – Информатика и вычислительная техника  группы ВТ-32  Воскобойников И. С. |
|  | Проверил:  Дмитренко П. С.  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.  Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Белгород, 2020

**Цель работы**: познакомиться с принципами создания многопоточных приложений и основными элементами синхронизации процессов и потоков

**Задания**

1. Разработайте программу, реализующую механизм управлением процессами   
2. Создайте многопоточное приложение   
3. Примените mutex для синхронизации потоков

**Выполнение**

1. Разработайте программу, реализующую механизм управлением процессами   
В программе должны быть реализованы следующие функции:

(a) Создать процесс   
(b) Завершить созданный процесс, при нажатии на клавишу Enter   
(c) Дождаться завершения процесса

*Код программы*

void task1(){

int pid = 0;

int change;

int flag = true;

while (flag)

{

std::cout << "Please, change command:" << std::endl;

std::cout << "1 - FORK" << std::endl;

std::cout << "2 - WAIT" << std::endl;

std::cout << "3 - WAIT\_PID" << std::endl;

std::cout << "4 - KILL" << std::endl;

std::cout << "Others - Exit" << std::endl;

std::cin >> change;

switch (change)

{

case 1:

{

pid = fork();

if (pid)

std::cout << "Create fork, pid: " << pid << std::endl;

break;

}

case 2:

{

std::cout << "Process waiting..." << std::endl;

wait(NULL);

break;

}

case 3:

{

int buf\_pid;

std::cout << "Please, input PID: ";

std::cin >> buf\_pid;

std::cout << "Process waiting exit PID " << buf\_pid << std::endl;

waitpid(buf\_pid, NULL, 0);

break;

}

case 4:

{

int buf\_pid;

std::cout << "Please, input PID: ";

std::cin >> buf\_pid;

std::cout << "Killed process PID" << buf\_pid << std::endl;

kill(buf\_pid, 9);

break;

}

default: // Љ®­Ґж Їа®Ја ¬¬л, Ґб«Ё ўлЎа ­® ­Ґ в®

{

flag = false;

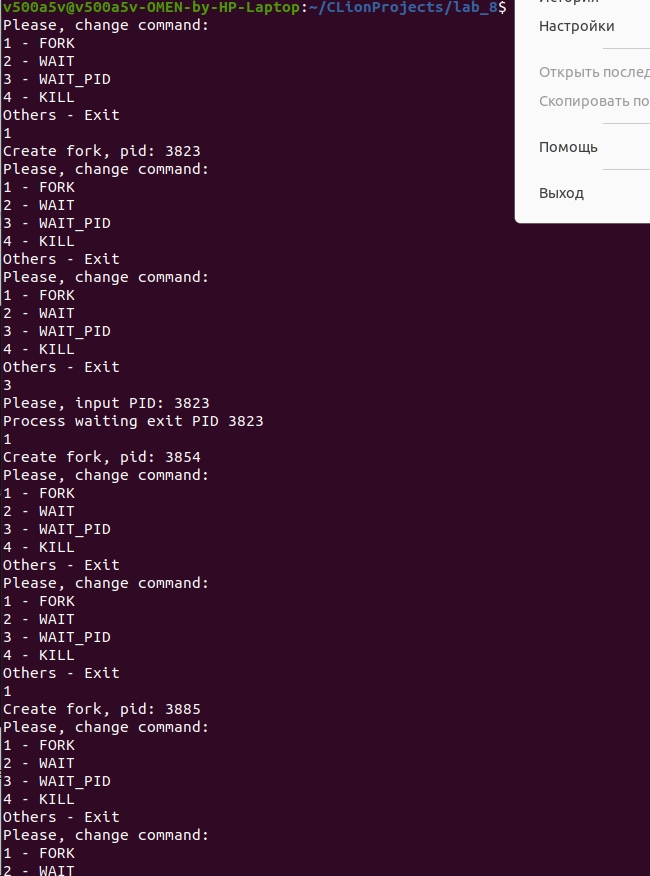
break;

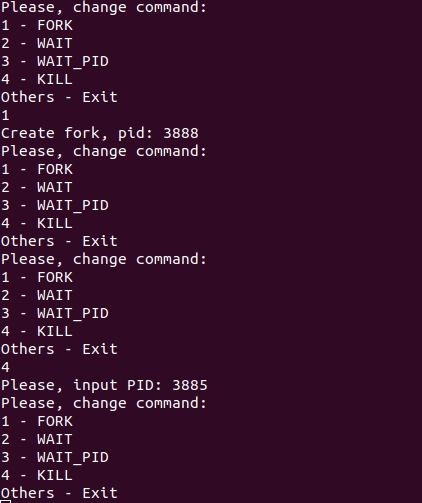
}

}

}

}





C:\Users\500a5\Desktop\лаб 8\05-12-2020 14-22-12.jpg

2. Создайте многопоточное приложение по следующему описанию:

(a) Запустить 5 параллельных потоков   
Каждый поток должен вывести на экран свой номер   
(b) Программа должна ожидать завершения всех потоков   
(c) Реализуйте внутри потоков алгоритм суммирования единиц:   
Каждый поток должен прибавлять в цикле единицу к двум переменным

* Общей переменной для всех потоков
* Частной переменной, которая у каждого потока своя

После выполнения всего цикла суммирования значения всех суммируемых переменных необходимо вывести на экран   
Охарактеризуйте полученные результаты

*Код программы*

const int count\_thread = 5;

const int max\_sum = 10000000;

int common\_sum = 0;

void\* sum(void\* args) // ”г­ЄжЁп бг¬¬Ёа®ў ­Ёп ¤«п Ї®в®Є

{

pthread\_t id = pthread\_self();

int sum = 0;

std::cout << "Created thread, id: " << id << std::endl;

for (int i = 0; i < max\_sum; i++)

{

sum++;

common\_sum++;

}

std::cout << "Thread-id: " << id << ", sum = " << sum << ", common\_sum = " << common\_sum << std::endl;

return NULL;

}

void task2(){

pthread\_t ts[count\_thread]; // Џ®в®ЄЁ

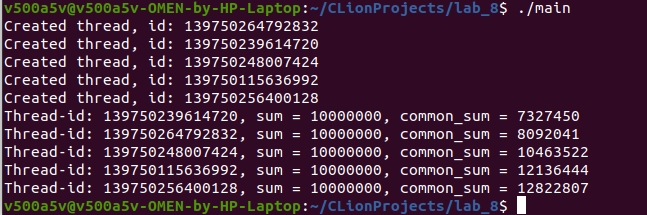
for (int i = 0; i < count\_thread; i++)

pthread\_create(ts + i, NULL, sum, NULL);

for (auto i : ts) // †¤Ґ¬ § ўҐаиҐ­Ёп ўбҐе Ї®в®Є®ў

pthread\_join(i, NULL);

}



Т.к. потоки пытались получить неконтролируемый доступ к одному и тому же ресурсу, в ряде случаев это приводило к помехам в их работе из-за того, что ресурс уже был занят другим потоком. Поэтому вместо правильной суммы в 50 млн мы получили неправильную.

3. Примените mutex для синхронизации потоков   
(a) Реализуйте в программе механизм запуска только одной копии процесса   
(b) Реализуйте идею критических секций для алгоритма суммирования из предыдущего этапа лабораторной работы, таким образом чтобы программа получала верные результаты   
Охарактеризуйте эффективность данного метода синхронизации   
(c) Продемонстрируйте пример возникновения взаимной блокировки в вашей программе 

*Код программы*

pthread\_mutex\_t mutex;

void\* sum2(void\* args) // ”г­ЄжЁп бг¬¬Ёа®ў ­Ёп ¤«п Ї®в®Є

{

pthread\_t id = pthread\_self();

int sum = 0;

//std::cout << "Created thread, id: " << id << std::endl;

for (int i = 0; i < max\_sum; i++)

{

sum++;

pthread\_mutex\_lock(&mutex);

common\_sum++;

pthread\_mutex\_unlock(&mutex);

}

std::cout << "Thread-id: " << id << ", sum = " << sum << ", common\_sum = " << common\_sum << std::endl;

return NULL;

}

void task3(){

pthread\_t ts[count\_thread]; // Џ®в®ЄЁ

pthread\_mutex\_init(&mutex, NULL); // €­ЁжЁ «Ё§ жЁп ¬мовҐЄб

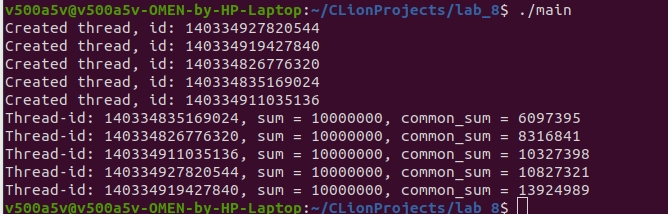
for (int i = 0; i < count\_thread; i++)

pthread\_create(ts + i, NULL, sum, NULL);

for (auto i : ts) // †¤Ґ¬ § ўҐаиҐ­Ёп ўбҐе Ї®в®Є®ў

pthread\_join(i, NULL);

}



При использовании синхронизации и разграничении доступа к общей переменной мы получаем верную сумму в 50 млн. Однако, использование мьютексов неэффективно, поскольку вынуждает потоки простаивать в ожидании того, когда мьютекс будет разблокирован. Возможны ситуации, когда мьютекс будет заблокирован навечно и процессы не смогут работать в принципе.   
  
Например, если убрать разблокирование мьютекса, то ни один поток не сможет завершить свою работу

*Контрольные вопросы*

1. Что такое взаимная блокировка? При каких условиях она возникает?

Взаимная блокировка – ситуация, когда несколько потоков или процессов ожидают доступ к ресурсу, который уже был занят одним из них. Так может быть, например, если поток закроет доступ к ресурсу, но так и не откроет его или попытается закрыть еще раз

2. Что такое мьютекс/семафор?

Семафор - это счетчик, для которого реализованы операции инкремента и декремента, причем декремент  при нулевом значении блокирует процесс, его вызвавший, пока декремент не станет возможным без уменьшения до -1.  
  
Мьютекс - это двоичный семафор, имеющий   
уникального владельца: разблокировать его может только тот поток, что заблокировал

3. Для чего применяется функция pthread\_join?

Данная функция позволяет приостановить работу процесса, пока не закончит свою работу созданный им поток.