

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина «Операционные среды и системное программирование»

ОТЧЕТ

К лабораторной работе № 4
на тему

**УПРАВЛЕНИЕ ПРОЦЕССАМИ И ВЗАИМОДЕЙСТВИЕ
ПРОЦЕССОВ**

Выполнил

К. А. Тимофеев

Проверил

Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

1 Постановка задачи.....	3
2 Краткие теоретические сведения.....	4
3 Результаты выполнения лабораторной работы.....	6
Выводы	7
Список использованных источников	8
Приложение А (обязательное) Листинг исходного кода	9

1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения данной лабораторной работы является изучение основных особенностей подсистемы управления процессами и средств взаимодействия процессов в Unix. Кроме этого, необходимо реализовать программу на языке программирования C, которая будет реализовывать параллельную обработку блока данных различными процессами с использованием семафоров.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

По терминологии Unix выполняющаяся программа называется процессом.

Ядро Unix является многозадачным, оно может распределять время между несколькими процессами, создавая впечатление, будто все они выполняются одновременно. При этом ядро регулирует доступ к ресурсам. В данном случае ресурсом является время центрального процессора.[1]

В ходе процесса одновременно выполняется только одно действие. То есть в любой момент времени можно точно определить, какая часть кода выполняется.

Регулирование доступа к времени центрального процессора называется планированием. Планирование делит время на кванты, которые называются временными квантами.

Процессы в Unix имеют три различных вида приоритетов:

- статический;
- динамический;
- реального времени.

Приоритет – это целочисленное значение, которое присваивает процессу при определении того, какому процессу должно быть выделено определенное время.

Идентификация процесса – присвоение ему целочисленного идентификатора, уникального в пределах системы.

Каждый процесс в Unix системе имеет свой PID процесса, который используется для идентификации процесса в системе и назначается ядром операционной системы. В операционной системе Windows концепция идентификации процесса существует, но является не такой выраженной, как в Unix-системах. Вместо этого, в Windows каждый процесс имеет свой уникальный дескриптор процесса.[2]

IPC – это механизм обмена данными и синхронизации между процессами в операционной системе.[3]

В Unix-подобных системах существует несколько механизмов IPC, а именно:

1 Каналы: представляют собой однонаправленный поток данных между двумя связанными процессами.

2 Сигналы: базовые и простейшие IPC для управления и взаимодействия процессами.

3 Семафоры: примитивы синхронизации, которые используются для доступа к ресурсам, которые могут быть разделяемыми между несколькими процессами. Представляют собой счетчик, который может быть уменьшен или увеличен, и используется для ограничения доступа процессов к критическим секциям или ресурсам.

4 Разделяемая память: область памяти, которая может быть доступна нескольким процессам.

5 Очереди сообщений: передача данных осуществляется в виде законченных фрагментов с определенной структурой и в определенном порядке.

Для выполнения данной лабораторной работы были использованы следующие сведения и концепции:

1 Разделяемая память: для работы с разделяемой памятью была использована структура данных, к которой обращались процессы, а также функции `ftok` для создания ключа, `shmget` для создания нового сегмента разделяемой памяти, `shmat` для присоединения сегмента разделяемой памяти, `shmdt` для отсоединения сегмента разделяемой памяти, `shmctl` для удаления сегмента разделяемой памяти.

2 Семафоры: для контроля доступа к разделяемой памяти каждого процесса были использованы семафоры, а также функции `malloc` для выделения определенного размера памяти под семафор, `sem_init` для инициализации семафора, `sem_wait` для уменьшения значения семафора на единицу, `sem_post` для увеличения значения семафора на единицу, `sem_destroy` для уничтожения семафора, `free` для освобождения выделенной памяти под семафор.

3 Управление процессами: при помощи функции `fork` создавались дочерние процессы, в которых происходила инициализация значениями блока данных. После при помощи функции `wait` родительский процесс ждет завершения дочерних процессов. Подсчет суммы значений блока данных осуществляется уже в основном родительском процессе.

3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе лабораторной работы была реализована программа, в которой реализуется заполнение родительским процессом матрицы со случайными элементами, после чего дочерние процессы получают матрицу квадратных корней значений оригинальной матрицы. Результат работы программы представлен на рисунке 3.1.

```
user@Honorable:~/SP/Lab4$ bin/program
before
1837484288.000 700176448.000 1614934272.000 101103912.000 115351768.000 2120248704.000 1523773312.000 314371200.000 396610720.000 1007772160.000 1538853888.000 246772848.000
2023425024.000 1633098112.000 729582080.000 644621248.000 603269376.000 1488752128.000 1335996416.000 1989488512.000 1331416448.000 988992256.000 2043183232.000 653747072.000
2146906496.000 530212640.000 1737950720.000 1130792192.000 382976960.000 551617536.000 1832314112.000 72977584.000 1251794048.000 1299764736.000 174081488.000 1367145728.000
1272529920.000 1697854848.000 1681516928.000 1669140608.000 558143296.000 1072887296.000 1915913472.000 434084672.000 558581696.000 498011936.000 1078705920.000 1161771136.000
1908764832.000 267210704.000 1083776000.000 1170896832.000 1256210044.000 899475520.000 1824443904.000 1255633792.000 1429688192.000 1414910976.000 238942416.000 1812665088.000
1966528384.000 2871256576.000 1885642752.000 1070838784.000 1223537664.000 2059724160.000 299500896.000 348583968.000 1610095360.000 1972017920.000 2817724544.000 20755008.000
897421584.000 1786154368.000 454839680.000 1455923200.000 136882672.000 1533545600.000 478210624.000 2123446656.000 1800764288.000 1473986560.000 1146659840.000 909491584.000
225978480.000 823620160.000 17641736.000 165566608.000 91047440.000 256584160.000 1320848128.000 2057575936.000 180357088.000 1059007168.000 980931008.000 1403894784.000
971247680.000 1271431936.000 1752478720.000 433859392.000 1095966208.000 1622719744.000 454614400.000 1993387648.000 1261390464.000 909454080.000 1301827200.000 1398073088.000
295516032.000 1772637888.000 1374036096.000 2096280320.000 1098540800.000 373212384.000 858288256.000 1324519296.000 1196832512.000 875929984.000 832702272.000 1287879936.000
1132514176.000 6066716.000 1197972224.000 1312871168.000 1065073856.000 31419608.000 509282368.000 2036321536.000 1302851584.000 174277472.000 322697312.000 251334880.000
1796997120.000 777311680.000 97238120.000 910903936.000 1686765824.000 1399065344.000 161493392.000 1982281856.000 1023619520.000 1535529472.000 1931078528.000 2122160384.000
begin fork
Output from main process
42865.887 26460.848 40186.246 10055.044 10740.194 46046.160 39035.539 17730.516 19915.088 31745.428 39228.227 15709.005
44982.496 40411.609 27010.777 25389.393 24561.543 38584.352 36551.285 44603.684 36488.578 31448.248 45201.586 25568.479
46334.723 23026.346 41688.734 33627.254 19569.797 23486.539 42805.539 8542.691 35380.703 36052.250 13193.994 36974.934
35672.539 41205.035 41006.305 40855.117 23625.057 32754.959 43771.148 20834.699 23632.641 22316.182 32843.660 34084.766
44573.133 16346.825 31682.424 34215.449 35443.066 29991.258 42713.508 35434.922 37811.219 37615.301 15457.763 42575.406
44345.555 45511.059 43423.988 32723.674 34979.102 45384.184 17044.086 18670.404 40125.992 44407.410 44919.090 4555.767
29956.994 42262.918 21326.971 38156.562 11691.137 39160.512 21684.340 46080.871 42435.414 38392.531 33862.367 30157.777
15032.581 28698.783 4200.207 40689.883 9541.878 16018.244 36343.473 45360.512 13429.709 32542.391 31319.818 37468.582
31164.848 35657.145 41862.617 20829.291 33105.379 40282.996 21321.689 44647.371 35516.059 30157.156 36080.844 37390.816
17190.580 42095.582 37067.992 45785.152 33144.242 19318.705 29296.557 36393.945 34595.266 29596.115 28856.582 35887.043
33652.848 2463.071 34611.734 36233.562 32635.469 5605.320 23859.639 45125.621 36095.035 13201.419 17963.777 15853.520
42391.004 27880.311 9860.939 30181.186 41070.254 37404.082 12708.005 44522.824 31994.055 39185.832 43944.039 46066.910
user@Honorable:~/SP/Lab4$
```

Рисунок 3.1 – Результат работы программы

Таким образом, в ходе лабораторной работы была реализована программа, реализующая заполнение блока данных случайными значениями и подсчет квадратных корней этих значений.

ВЫВОДЫ

В ходе лабораторной работы были изучены основные особенности подсистемы управления процессами и средства взаимодействия процессов в Unix. Кроме этого, была реализована программа на языке программирования C, которая параллельно обрабатывает блок данных различными процессами, которые считают квадратные корни значений оригинального блока данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Процессы и потоки [Электронный ресурс]. – Режим доступа: <https://acm.bsu.by/wiki/Unix2019b/>. – Дата доступа: 11.02.2024.

[2] Архитектура Unix. Процессы [Электронный ресурс]. – Режим доступа: http://heap.altlinux.org/tmp/unix_base/ch01s03.html. – Дата доступа: 13.02.2024.

[3] Разделяемая память и семафоры [Электронный ресурс]. – Режим доступа: <https://debianinstall.ru/razdelyaemaya-pamyat-semafor-y-i-ocheredi-soobshhenij-v-os-linux/>. – Дата доступа: 13.02.2024.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг исходного кода

Листинг 1 – Программный код lab4.c

```
// semaphores processes shared memory
# include <sys/types.h>
# include <sys/ipc.h>
#include <sys/shm.h>
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <time.h>
#include <errno.h>
#include <math.h>

#define ROW_PROCESS 6
#define COLUMNS_PROCESS 6

void process(int iBeg, int jBeg, int iEnd, int jEnd, int shmid, int rows, int columns)
{
    void* mem = (void*)shmat(shmid, NULL, 0);

    sem_t* sem = (sem_t*)mem;

    float* arr = (float*)(sem + 1);

    for(int i = iBeg; i < iEnd && i < rows; ++i)
    {
        for(int j = jBeg; j < jEnd && j < columns; ++j)
        {
            arr[i*columns + j] = sqrtf(arr[i*columns + j]);
        }
        sem_wait(sem);
        printf("row %i from %i to %i is processed\n", i, jBeg, jEnd<columns?jEnd:columns);
        sem_post(sem);
    }

    shmdt(mem);
}

int main()
{
    // buffer for shm operations
    int rows = 12;
    int columns = 12;

    int pids[ROW_PROCESS * COLUMNS_PROCESS];

    struct shmid_ds buf;
    // key
    key_t key = ftok(".", 'R');
    if (key == -1)
    {
        perror("ftok error");
        exit(1);
    }
    // semaphore
    int shmid = shmget(key, sizeof(sem_t) + sizeof(float)*rows*columns, IPC_CREAT|IPC_EXCL| 00666);
    if(shmid == -1)
    {
        if(errno == EEXIST)
        {
            perror("Error eexist");
        }
        else
        {
            perror("Not eexist");
        }
        exit(1);
    }
    void* mem = (void*)shmat(shmid, NULL, 0);
    if((long)mem == -1)
    {
        perror("memory at error");
        exit(1);
    }

    sem_t* sem = (sem_t*)mem;
```

```

if(sem_init(&sem, 1, 0) == -1)
{
    perror("sem_init error");
    exit(1);
}

// array init
float* arr = (float*) (sem + 1);

srand(time(NULL));
for(int i = 0; i < rows; ++i)
{
    for(int j = 0; j < columns; ++j)
    {
        arr[i*columns + j] = rand();
    }
}

printf("before\n");
for(int i = 0; i < rows; ++i)
{
    for(int j = 0; j < columns; ++j)
    {
        printf("%.3f ", arr[i*columns + j]);
    }
    printf("\n");
}
printf("begin fork\n");
for(int i = 0; i < ROW_PROCESS; ++i)
{
    for (int j = 0; j < COLUMNS_PROCESS; ++j)
    {
        int res = fork();
        if(res == -1)
        {
            perror("fork error");
            exit(1);
        }
        else if(res == 0)
        {
            process(i * (rows / ROW_PROCESS), (j) * (columns / COLUMNS_PROCESS), (i + 1) *
(rows/ROW_PROCESS), (j + 1) * (columns/COLUMNS_PROCESS),
            shmid, rows, columns);
            return 0;
        }

        pids[i*COLUMNS_PROCESS + j] = res;
    }
}

sem_post(&sem);

for (int i = 0; i < ROW_PROCESS * COLUMNS_PROCESS; ++i)
{
    if(waitpid(pids[i], NULL, WUNTRACED) == -1)
    {
        perror("waitpid error");
        exit(1);
    }
}

printf("Output from main process\n");

for(int i = 0; i < rows; ++i)
{
    for(int j = 0; j < columns; ++j)
    {
        printf("%.3f ", arr[i*columns + j]);
    }
    printf("\n");
}

sem_close(&sem);
sem_destroy(&sem);
shmdt(mem);
shmctl(shmid, IPC_RMID, NULL);

return 0;
}

```