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 №2**

по дисциплине: «Операционные системы»

на тему: «Процессы и потоки в ОС Linux (Ubuntu): сравнение, механизмы синхронизации. Парадигмы межпроцессорного взаимодействия.»

Выполнил: ст. группы ПВ-223

Дмитриев Андрей Александрович

Проверили:

доц. Островский Алексей Мичеславович,

асс. Четвертухин Виктор Романович

Белгород, 2024

**Цель работы:** Изучить различия между процессами и потоками в ОС Linux (Ubuntu), а также освоить механизмы синхронизации и межпроцессорного взаимодействия для обеспечения корректной работы программ в многозадачной среде.

**Условие индивидуального задания:**

Змей Горыныч имеет три головы, каждая из которых независимо от других голов ест продукцию с двух кондитерских фабрик. Каждая фабрика производит разные типы кондитерских изделий (торты, пирожные, конфеты, пряники) с различной скоростью. Головам нужно получать продукцию из общего склада, но склад ограничен по объему. Задача — организовать взаимодействие между фабриками (производителями) и головами Змея Горыныча (потребителями) так, чтобы они корректно синхронизировались при производстве и потреблении продукции, избегая конфликтов, минимизируя ситуации простоя и переполнения склада.

Для реализации склада использовать, в случае процессов, именованный канал FIFO (mkfifo), для реализации потоков — очередь с блокировкой. Для синхронизации, в случае процессов, использовать семафоры (sys/sem.h), а для потоков — futex (linux/futex.h, sys/syscall.h).

**Ход выполнения работы**

**Задание 1.**

**Текст программы** на языке С с комментариями.

#define QUEUE\_SIZE 8

typedef struct

{

    void \*data[QUEUE\_SIZE];

    int head;

    int tail;

    int count;

    int futex\_count; // futex для синхронизации

} BlockingQueue;

static long futex\_wait(int \*uaddr, int val)

{

    return syscall(SYS\_futex, uaddr, FUTEX\_WAIT, val, NULL, NULL, 0);

}

static long futex\_wake(int \*uaddr, int val)

{

    return syscall(SYS\_futex, uaddr, FUTEX\_WAKE, val, NULL, NULL, 0);

}

BlockingQueue \*queue\_create()

{

    BlockingQueue \*queue = (BlockingQueue \*)malloc(sizeof(BlockingQueue));

    if (queue == NULL)

    {

        perror("malloc failed");

        exit(1);

    }

    queue->head = 0;

    queue->tail = 0;

    queue->count = 0;

    queue->futex\_count = 0;

    return queue;

}

void enqueue(BlockingQueue \*queue, void \*data)

{

    int ret;

    while (queue->count == QUEUE\_SIZE)

    {

        futex\_wait(&queue->futex\_count, queue->count);

    }

    queue->data[queue->tail] = data;

    queue->tail = (queue->tail + 1) % QUEUE\_SIZE;

    queue->count++;

    queue->futex\_count = queue->count; // Обновляем futex

    ret = futex\_wake(&queue->futex\_count, 1);

    if (ret == -1)

        perror("futex\_wake failed");

}

void \*dequeue(BlockingQueue \*queue)

{

    int ret;

    void \*data;

    while (queue->count == 0)

    {

        futex\_wait(&queue->futex\_count, queue->count);

    }

    data = queue->data[queue->head];

    queue->head = (queue->head + 1) % QUEUE\_SIZE;

    queue->count--;

    queue->futex\_count = queue->count; // Обновляем futex

    ret = futex\_wake(&queue->futex\_count, 1);

    if (ret == -1)

        perror("futex\_wake failed");

    return data;

}

void queue\_destroy(BlockingQueue \*queue)

{

    free(queue);

}

////////////////////////////////////////////////////////////

#define NUM\_OF\_PROD 4

#define MAX\_VOLUME\_OF\_STORAGE 50

// Структура Продукт

typedef struct Prod

{

    int num;

    char name[16];

} Prod;

// Структура производительность

// конкретной продукции

typedef struct Efficiency

{

    Prod prod;

    unsigned long long time;

    int amount;

} Efficiency;

// Структура данных, которые

// поступают в очередь, для

// изменения склада

typedef struct QueueData

{

    int num\_obj;

    int prod\_num;

    int add\_val;

} QueueData;

int storage[NUM\_OF\_PROD]; // склад

BlockingQueue \*queue;     // очередь

int futex\_val;            // значение для передачи в futex

// sleep в миллисекундах

void sleep\_ms(int milliseconds) { usleep(milliseconds \* 1000); }

// Функция для определения наличия свободного пространства на складе

int get\_free\_storage\_space()

{

    // requierd lock storage

    int sum = 0;

    for (size\_t i = 0; i < NUM\_OF\_PROD; i++)

    {

        sum += storage[i];

        if (sum > MAX\_VOLUME\_OF\_STORAGE)

        {

            return 0;

        }

    }

    return MAX\_VOLUME\_OF\_STORAGE - sum;

}

// Поток фабрики (писатель),

// производит одну продукцию

void run\_fabr(

    const int fabr\_num,

    const Efficiency eff)

{

    printf("efficiency of fabric number %d\n", fabr\_num);

    printf("\tname: %s, amount: %d, time: %llu\n",

           eff.prod.name, eff.amount, eff.time);

    while (1)

    {

        sleep\_ms(eff.time);

        QueueData \*queueData = (QueueData \*)malloc(sizeof(QueueData));

        queueData->num\_obj = fabr\_num;

        queueData->prod\_num = eff.prod.num;

        queueData->add\_val = eff.amount;

        enqueue(queue, queueData);

        printf("fabr %d do %s in amount of %d\n",

               fabr\_num, eff.prod.name, eff.amount);

    }

}

// Поток головы (писатель), случайно

// съедает единицу указанной продукции

void run\_head(

    const int head\_num,

    const Prod eating\_prod[],

    const int amount\_eating\_prod,

    const unsigned long long eating\_time)

{

    printf("head %d eat:\n\t", head\_num);

    for (size\_t i = 0; i < amount\_eating\_prod; i++)

        printf("%s, ", eating\_prod[i].name);

    printf("\b\b - eating per %llu\n", eating\_time);

    while (1)

    {

        srand(time(0));

        sleep\_ms(eating\_time);

        int index = rand() % amount\_eating\_prod;

        printf("eating prod - %s\n", eating\_prod[index].name);

        QueueData \*queueData = (QueueData \*)malloc(sizeof(QueueData));

        queueData->num\_obj = head\_num;

        queueData->prod\_num = eating\_prod[index].num;

        queueData->add\_val = -1;

        enqueue(queue, queueData);

    }

}

void run\_storage\_manager()

{

    while (1)

    {

        QueueData \*queueData = dequeue(queue);

        printf("- input queue data: %d %d\n",

               queueData->prod\_num, queueData->add\_val);

        storage[queueData->prod\_num] += queueData->add\_val;

        if (storage[queueData->prod\_num] < 0 || !get\_free\_storage\_space())

        {

            storage[queueData->prod\_num] -= queueData->add\_val;

            if (queueData->add\_val < 0)

                printf("head %d cant eat prodnum: %d\n", queueData->num\_obj, queueData->prod\_num);

            else

                printf("fabr %d cant put prodnum: %d\n", queueData->num\_obj, queueData->prod\_num);

        }

else

        {

            if (queueData->add\_val < 0)

                printf("1 head %d eat prodnum: %d\n", queueData->num\_obj, queueData->prod\_num);

            else

                printf("1 fabr %d put prodnum: %d\n", queueData->num\_obj, queueData->prod\_num);

        }

        printf("- storage state: ");

        for (size\_t i = 0; i < NUM\_OF\_PROD; i++)

            printf("%d ", storage[i]);

        printf("\n");

        free(queueData);

    }

}

int main()

{

    queue = queue\_create();

    Prod PROD\_PIE = {0, "pie"};

    Prod PROD\_CAKE = {1, "cake"};

    Prod PROD\_CANDY = {2, "candy"};

    Prod PROD\_BREAD = {3, "bread"};

    Efficiency eff1 = {

        PROD\_PIE,

        50,

        2};

    Efficiency eff2 = {

        PROD\_CAKE,

        70,

        1};

    Efficiency eff3 = {

        PROD\_CANDY,

        40,

        3};

    Efficiency eff4 = {

        PROD\_BREAD,

        60,

        1};

    pthread\_t

        thr\_storage\_manager,

        thr\_head1,

        thr\_head2, thr\_head3,

        thr\_fabr11, thr\_fabr12,

        thr\_fabr21, thr\_fabr22;

    void \*fabr\_1\_1()

    {

        run\_fabr(1, eff1);

        return 0;

    };

    pthread\_create(&thr\_fabr11, 0, fabr\_1\_1, 0);

    void \*fabr\_1\_2()

    {

        run\_fabr(1, eff2);

        return 0;

    };

    pthread\_create(&thr\_fabr12, 0, fabr\_1\_2, 0);

    void \*fabr\_2\_1()

    {

        run\_fabr(2, eff3);

        return 0;

    };

    pthread\_create(&thr\_fabr21, 0, fabr\_2\_1, 0);

    void \*fabr\_2\_2()

    {

        run\_fabr(2, eff4);

        return 0;

    };

    pthread\_create(&thr\_fabr22, 0, fabr\_2\_2, 0);

    void \*storage\_manager()

    {

        run\_storage\_manager();

        return 0;

    }

    pthread\_create(&thr\_storage\_manager, 0, storage\_manager, 0);

    void \*head\_1()

    {

        Prod eating\_prod[] = {PROD\_CAKE, PROD\_CANDY};

        run\_head(1, eating\_prod, 2, 80);

        return 0;

    };

    pthread\_create(&thr\_head1, 0, head\_1, 0);

    void \*head\_2()

    {

        Prod eating\_prod[] = {PROD\_PIE, PROD\_BREAD};

        run\_head(2, eating\_prod, 2, 90);

        return 0;

    };

    pthread\_create(&thr\_head2, 0, head\_2, 0);

    void \*head\_3()

    {

        Prod eating\_prod[] = {PROD\_PIE, PROD\_CAKE, PROD\_CANDY, PROD\_BREAD};

        run\_head(3, eating\_prod, 4, 70);

        return 0;

    };

    pthread\_create(&thr\_head3, 0, head\_3, 0);

    while (1) { }

    return 0;

}

**Протоколы, логи, скриншоты, графики.**

В программе есть основные функции для выполнения в потоке. Функция run\_fabr получает информацию о продукции (внутри фабрики создание разной продукции происходит параллельно). Функция run\_head получает список «поедаемой продукции», выбор элемента происходит случайно. Функция run\_storage\_manager считывает изменения на складе и или их принимает, или отклоняет в зависимости от свободного места.

Передача данных происходит через самописную блокирующую очередь, которая блокируется с помощью futex.

При производстве (или «поедании») не учитывается размерность склада, поэтому продукция зачастую «выкидывается» (или «не съедается»).

Вывод программы:

efficiency of fabric number 1

name: pie, amount: 2, time: 50

efficiency of fabric number 1

name: cake, amount: 1, time: 70

head 1 eat:

cake, candy - eating per 80

head 2 eat:

pie, bread - eating per 90

head 3 eat:

pie, cake, candy, bread - eating per 70

efficiency of fabric number 2

name: bread, amount: 1, time: 60

efficiency of fabric number 2

name: candy, amount: 3, time: 40

fabr 2 do candy in amount of 3

- input queue data: 2 3

1 fabr 2 put prodnum: 2

- storage state: 0 0 3 0

fabr 1 do pie in amount of 2

- input queue data: 0 2

1 fabr 1 put prodnum: 0

- storage state: 2 0 3 0

- input queue data: 3 1

1 fabr 2 put prodnum: 3

- storage state: 2 0 3 1

fabr 2 do bread in amount of 1

eating prod - cake

- input queue data: 1 -1

0 head 3 cant eat prodnum: 1

- storage state: 2 0 3 1

fabr 1 do cake in amount of 1

- input queue data: 1 1

1 fabr 1 put prodnum: 1

- storage state: 2 1 3 1

eating prod - candy

- input queue data: 2 -1

1 head 1 eat prodnum: 2

- storage state: 2 1 2 1

**Замеры.** С помощью счётчика объявленного, как volatile был произведён замер съеденной продукции при 1, 3, 6 ядрах за 10 секунд. В каждом замере по 5 экспериментов

|  |  |  |
| --- | --- | --- |
| количество ядер | значения | среднее значение |
| 1 | 322, 326, 353, 326, 331 | 331,6 |
| 3 | 323, 253, 363, 352, 350 | 328,2 |
| 6 | 257, 296, 329, 307, 292 | 296,2 |

Заметен интересный парадокс, что чем больше ядер, тем меньше эффективность, хотя предполагался обратный результат (особенно с 3мя ядрами, количество нередко было не больше 90). Возможно, что на результат повлияло наличие других процессов, которые в зависимости от свободных ядер совершали сервисные действия. Также выборка была довольно небольшая.

**Задание 2.**

**Текст программы** на языке С с комментариями.

#define NUM\_OF\_PROD 4

#define MAX\_VOLUME\_OF\_STORAGE 50

#define CHANNEL\_NAME "fifo.txt"

// Структура Продукт

typedef struct Prod

{

    int num;

    char name[16];

} Prod;

// Структура производительность

// конкретной продукции

typedef struct Efficiency

{

    Prod prod;

    unsigned long long time;

    int amount;

} Efficiency;

// Структура данных, которые

// поступают в очередь, для

// изменения склада

typedef struct Data

{

    int num\_obj;

    int prod\_num;

    int add\_val;

} Data;

int storage[NUM\_OF\_PROD]; // склад

sem\_t sem;

int fd\_fifo;

// sleep в миллисекундах

void sleep\_ms(int milliseconds) { usleep(milliseconds \* 1000); }

// Функция для определения наличия свободного пространства на складе

int get\_free\_storage\_space()

{

    // requierd lock storage

    int sum = 0;

    for (size\_t i = 0; i < NUM\_OF\_PROD; i++)

    {

        sum += storage[i];

        if (sum > MAX\_VOLUME\_OF\_STORAGE)

        {

            return 0;

        }

    }

    return MAX\_VOLUME\_OF\_STORAGE - sum;

}

// Поток фабрики (писатель),

// производит одну продукцию

void run\_fabr(

    const int fabr\_num,

    const Efficiency eff)

{

    printf("efficiency of fabric number %d\n", fabr\_num);

    printf("\tname: %s, amount: %d, time: %llu\n",

           eff.prod.name, eff.amount, eff.time);

    while (1)

    {

        sleep\_ms(eff.time);

        Data \*data = (Data \*)malloc(sizeof(Data));

        data->num\_obj = fabr\_num;

        data->prod\_num = eff.prod.num;

        data->add\_val = eff.amount;

        printf("fabr %d do %s in amount of %d\n",

               fabr\_num, eff.prod.name, eff.amount);

        sem\_wait(&sem);

        fd\_fifo = open(CHANNEL\_NAME, O\_WRONLY | O\_NONBLOCK);

        write(fd\_fifo, data, sizeof(data));

        close(fd\_fifo);

        sem\_post(&sem);

    }

}

// Поток головы (писатель), случайно

// съедает единицу указанной продукции

void run\_head(

    const int head\_num,

    const Prod eating\_prod[],

    const int amount\_eating\_prod,

    const unsigned long long eating\_time)

{

    printf("head %d eat:\n\t", head\_num);

    for (size\_t i = 0; i < amount\_eating\_prod; i++)

        printf("%s, ", eating\_prod[i].name);

    printf("\b\b - eating per %llu\n", eating\_time);

    while (1)

    {

        sleep\_ms(eating\_time);

        srand(time(0));

        int index = rand() % amount\_eating\_prod;

        Data \*data = (Data \*)malloc(sizeof(Data));

        data->num\_obj = head\_num;

        data->prod\_num = eating\_prod[index].num;

        data->add\_val = -1;

        printf("eating prod - %s\n", eating\_prod[index].name);

        sem\_wait(&sem);

        fd\_fifo = open(CHANNEL\_NAME, O\_WRONLY | O\_NONBLOCK);

        write(fd\_fifo, data, sizeof(Data \*));

        close(fd\_fifo);

        sem\_post(&sem);

    }

}

void run\_storage\_manager()

{

    fd\_fifo = open(CHANNEL\_NAME, O\_RDONLY);

    while (1)

    {

        Data\* data = 0;

        sem\_wait(&sem);

        read(fd\_fifo, data, sizeof(data));

        sem\_post(&sem);

        if (data != 0)

        {

            printf("- input queue data: %d %d\n",

                   data->prod\_num, data->add\_val);

            storage[data->prod\_num] += data->add\_val;

            if (storage[data->prod\_num] < 0 || !get\_free\_storage\_space())

            {

                storage[data->prod\_num] -= data->add\_val;

                if (data->add\_val < 0)

                    printf("0 head %d cant eat prodnum: %d\n", data->num\_obj, data->prod\_num);

                else

                    printf("0 fabr %d cant put prodnum: %d\n", data->num\_obj, data->prod\_num);

            }

            else

            {

                if (data->add\_val < 0)

                    printf("1 head %d eat prodnum: %d\n", data->num\_obj, data->prod\_num);

                else

                    printf("1 fabr %d put prodnum: %d\n", data->num\_obj, data->prod\_num);

            }

            free(data);

            printf("- storage state: ");

            for (size\_t i = 0; i < NUM\_OF\_PROD; i++)

                printf("%d ", storage[i]);

            printf("\n");

        }

        else

        {

            printf("stmg nothing in fifo\n");

            sleep(1);

        }

    }

    close(fd\_fifo);

}

int main()

{

    Prod PROD\_PIE = {0, "pie"};

    Prod PROD\_CAKE = {1, "cake"};

    Prod PROD\_CANDY = {2, "candy"};

    Prod PROD\_BREAD = {3, "bread"};

    Efficiency eff1 = {

        PROD\_PIE,

        50,

        2};

    Efficiency eff2 = {

        PROD\_CAKE,

        70,

        1};

    Efficiency eff3 = {

        PROD\_CANDY,

        40,

        3};

    Efficiency eff4 = {

        PROD\_BREAD,

        60,

        1};

    unlink(CHANNEL\_NAME);

    if ((mkfifo(CHANNEL\_NAME, 0777)) == -1)

    {

        fprintf(stderr, "Невозможно создать fifo\n");

        exit(0);

    }

    sem\_init(&sem, 0, 1);

    if (fork() == 0)

        run\_fabr(1, eff1);

    if (fork() == 0)

        run\_fabr(1, eff2);

    if (fork() == 0)

        run\_fabr(2, eff3);

    if (fork() == 0)

        run\_fabr(2, eff4);

    if (fork() == 0)

    {

        Prod eating\_prod1[] = {P ROD\_CAKE, PROD\_CANDY};

        run\_head(1, eating\_prod1, 2, 80);

    }

    if (fork() == 0)

    {

        Prod eating\_prod2[] = {PROD\_PIE, PROD\_BREAD};

        run\_head(2, eating\_prod2, 2, 90);

    }

    if (fork() == 0)

    {

        Prod eating\_prod3[] = {PROD\_PIE, PROD\_CAKE, PROD\_CANDY, PROD\_BREAD};

        run\_head(3, eating\_prod3, 4, 70);

    }

    if (fork() == 0)

        run\_storage\_manager();

    while (1)

    {

    }

    return 0;

}

**Протоколы, логи, скриншоты, графики.**

К сожалению, не удалось выполнить передачу с помощью именованного канала, предположительно из-за особенностей его работы.

**Выводы**

В ходе лабораторной работы было выполнено индивидуальное задание. Изучены различия между процессами и потоками в ОС Linux (Ubuntu), а также освоены некоторые механизмы синхронизации и межпроцессорного взаимодействия для обеспечения корректной работы программ в многозадачной среде.