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
Игнатьев Артур Олегович

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.

Вводится дополнительное условие в логику задачи. После заполнения склада производство кондитерской продукции останавливается и возобновляется только тогда, когда головы Змея Горыныча полностью съедят продукцию и склад очистится. Для реализации склада использовать, в случае процессов, разделяемую память (sys/shm.h), а для реализации потоков — связанный список. Для синхронизации, в случае процессов, использовать семафоры (sys/sem.h), а для потоков — мьютексы (pthread_mutex_t), барьерные синхронизации (pthread_barrier_t) и условные переменные (pthread_cond_t), если необходимо.

Ход выполнения работы

Решение с потоками (Pthreads)

1. Основные компоненты

- **Склад**: Реализован как связанный список. Содержит добавляемую продукцию.
- **Производители (фабрики)**:
  - Фабрика 1 производит торты и пирожные.
  - Фабрика 2 производит конфеты и пряники.
- **Потребители (головы)**:
  - Голова 1 потребляет торты и конфеты.
  - Голова 2 потребляет пирожные и пряники.
  - Голова 3 потребляет любые виды продукции.

2. Синхронизация

Используются следующие механизмы из библиотеки Pthreads:

- **Мьютексы** (pthread_mutex_t) защищают доступ к складу, чтобы несколько потоков одновременно не изменяли его состояние.
- **Условные переменные** (pthread_cond_t) позволяют потокам "засыпать", если склад заполнен (для производителей) или пуст (для потребителей).

3. Логика работы

1. **Производители**:
   - Если склад заполнен, производитель засыпает, ожидая освобождения места.
   - Когда появляется место, производитель добавляет продукцию и сигнализирует, что на складе есть товары.
2. **Потребители**:
   - Если склад пуст, потребитель засыпает, ожидая поступления продукции.
   - Когда появляется продукция, потребитель забирает её и сигнализирует, что появилось свободное место.
3. **Очистка склада**:
   - Склад наполняется до предела, после чего фабрики "засыпают".
   - Головы потребляют продукцию до тех пор, пока склад не опустеет, после чего фабрики возобновляют работу.

Ключевые особенности программы с потоками

- Потоки "засыпают", избегая активного ожидания, что экономит ресурсы CPU.
- Реализация проста благодаря общему доступу потоков к памяти.

Код программы:

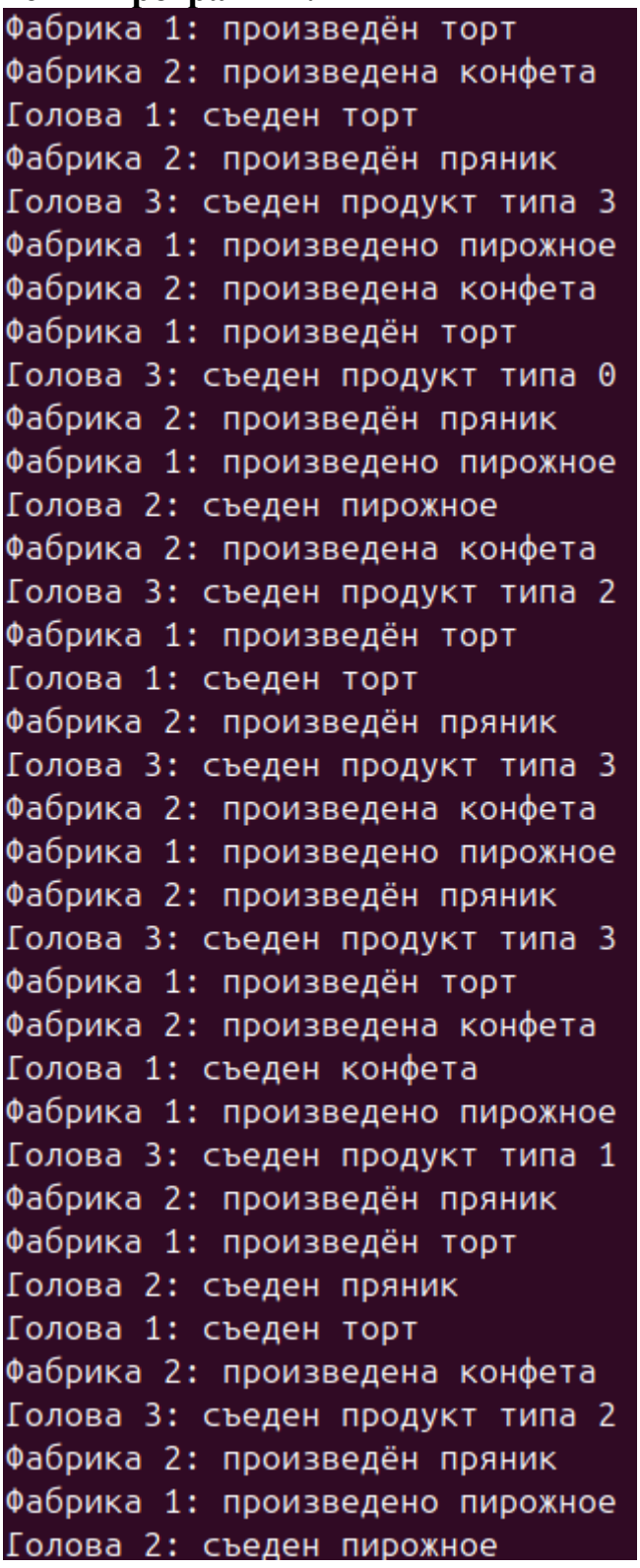
```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

#define MAX_STORAGE 10

// Продукция
typedef enum { CAKE, PASTRY, CANDY, GINGERBREAD } ProductType;

// Узел для связанного списка
typedef struct Node {
    ProductType product;
    struct Node* next;
} Node;

// Глобальные переменные
Node* storage = NULL;
int storage_count = 0;

// Синхронизация
pthread_mutex_t storage_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t storage_not_full = PTHREAD_COND_INITIALIZER;
pthread_cond_t storage_not_empty = PTHREAD_COND_INITIALIZER;

// Добавить продукцию на склад
void add_product(ProductType product) {
    Node* new_node = (Node*)malloc(sizeof(Node));
    new_node->product = product;
    new_node->next = NULL;

    if (!storage) {
        storage = new_node;
    } else {
        Node* temp = storage;
        while (temp->next) temp = temp->next;
        temp->next = new_node;
    }
    storage_count++;
}

// Убрать продукцию со склада
ProductType remove_product() {
```

```
    if (!storage) return -1;

    Node* temp = storage;
    ProductType product = temp->product;
    storage = storage->next;
    free(temp);
    storage_count--;

    return product;
}

// Фабрика 1 (торты и пирожные)
void* factory1(void* arg) {
    while (1) {
        pthread_mutex_lock(&storage_mutex);
        while (storage_count >= MAX_STORAGE)
            pthread_cond_wait(&storage_not_full, &storage_mutex);

        add_product(CAKE);
        printf("Фабрика 1: произведён торт\n");
        pthread_cond_signal(&storage_not_empty);
        pthread_mutex_unlock(&storage_mutex);
        usleep(70000);

        pthread_mutex_lock(&storage_mutex);
        while (storage_count >= MAX_STORAGE)
            pthread_cond_wait(&storage_not_full, &storage_mutex);

        add_product(PASTRY);
        printf("Фабрика 1: произведено пирожное\n");
        pthread_cond_signal(&storage_not_empty);
        pthread_mutex_unlock(&storage_mutex);
        usleep(50000);
    }
}

// Фабрика 2 (конфеты и пряники)
void* factory2(void* arg) {
    while (1) {
        pthread_mutex_lock(&storage_mutex);
        while (storage_count >= MAX_STORAGE)
            pthread_cond_wait(&storage_not_full, &storage_mutex);

        add_product(CANDY);
        printf("Фабрика 2: произведена конфета\n");
```

```
        pthread_cond_signal(&storage_not_empty);
        pthread_mutex_unlock(&storage_mutex);
        usleep(40000);

        pthread_mutex_lock(&storage_mutex);
        while (storage_count >= MAX_STORAGE)
            pthread_cond_wait(&storage_not_full, &storage_mutex);

        add_product(GINGERBREAD);
        printf("Фабрика 2: произведён пряник\n");
        pthread_cond_signal(&storage_not_empty);
        pthread_mutex_unlock(&storage_mutex);
        usleep(60000);
    }
}

// Голова Змея 1 (торты и конфеты)
void* head1(void* arg) {
    while (1) {
        pthread_mutex_lock(&storage_mutex);
        while (storage_count == 0)
            pthread_cond_wait(&storage_not_empty, &storage_mutex);

        ProductType product = remove_product();
        if (product == CAKE || product == CANDY) {
            printf("Голова 1: съеден %s\n", product == CAKE ? "торт" : "конфета");
        }
        pthread_cond_signal(&storage_not_full);
        pthread_mutex_unlock(&storage_mutex);
        usleep(80000);
    }
}

// Голова Змея 2 (пирожные и пряники)
void* head2(void* arg) {
    while (1) {
        pthread_mutex_lock(&storage_mutex);
        while (storage_count == 0)
            pthread_cond_wait(&storage_not_empty, &storage_mutex);

        ProductType product = remove_product();
        if (product == PASTRY || product == GINGERBREAD) {
            printf("Голова 2: съеден %s\n", product == PASTRY ? "пирожное" :
"пряник");
        }
```

```
        pthread_cond_signal(&storage_not_full);
        pthread_mutex_unlock(&storage_mutex);
        usleep(90000);
    }
}

// Голова Змея 3 (любой продукт)
void* head3(void* arg) {
    while (1) {
        pthread_mutex_lock(&storage_mutex);
        while (storage_count == 0)
            pthread_cond_wait(&storage_not_empty, &storage_mutex);

        ProductType product = remove_product();
        printf("Голова 3: съеден продукт типа %d\n", product);
        pthread_cond_signal(&storage_not_full);
        pthread_mutex_unlock(&storage_mutex);
        usleep(70000);
    }
}

int main() {
    pthread_t f1, f2, h1, h2, h3;

    pthread_create(&f1, NULL, factory1, NULL);
    pthread_create(&f2, NULL, factory2, NULL);
    pthread_create(&h1, NULL, head1, NULL);
    pthread_create(&h2, NULL, head2, NULL);
    pthread_create(&h3, NULL, head3, NULL);

    pthread_join(f1, NULL);
    pthread_join(f2, NULL);
    pthread_join(h1, NULL);
    pthread_join(h2, NULL);
    pthread_join(h3, NULL);

    return 0;
}
```

Скриншот выполнения программы:



Программа с процессами

1. Основные компоненты

- **Склад:** Реализован через разделяемую память (shmget, shmat), где хранится массив продукции и текущее количество элементов.
- Производители (фабрики):
  - Фабрика 1 и фабрика 2 создают продукцию с заданной скоростью и помещают её на склад.
- **Потребители (головы):**
  - Голова 1 потребляет торты и конфеты.
  - Голова 2 потребляет пирожные и пряники.
  - Голова 3 универсальна и потребляет любой продукт.

2. Синхронизация

Семафоры (sys/sem.h) используются для управления доступом к разделяемому ресурсу:

- **Мьютекс (семафор 0):** Обеспечивает атомарный доступ к складу.
- **Семафор свободных мест (семафор 1):** Указывает, сколько мест свободно на складе.
- **Семафор заполненных мест (семафор 2):** Указывает, сколько продукции доступно для потребления.

3. Логика работы

1. **Производители:**
   - Если склад заполнен, производитель блокируется на семафоре свободных мест.
   - Производитель добавляет продукцию в массив и увеличивает счётчик заполненных мест.
   - После добавления продукции сигнализирует, что склад не пуст.
2. **Потребители:**
   - Если склад пуст, потребитель блокируется на семафоре заполненных мест.
   - Потребитель забирает продукцию из массива и увеличивает счётчик свободных мест.
   - После потребления сигнализирует, что на складе появилось свободное место.
3. **Очистка склада:**
   - Когда склад наполняется до максимума, производители "засыпают".
   - После полного потребления продукции головы разблокируют производителей.

Ключевые особенности программы с процессами

- Использование разделяемой памяти для передачи данных между процессами.
- Семафоры обеспечивают строгую синхронизацию.
- Процессы работают независимо, что позволяет масштабировать программу на нескольких ядрах CPU.

Код программы:

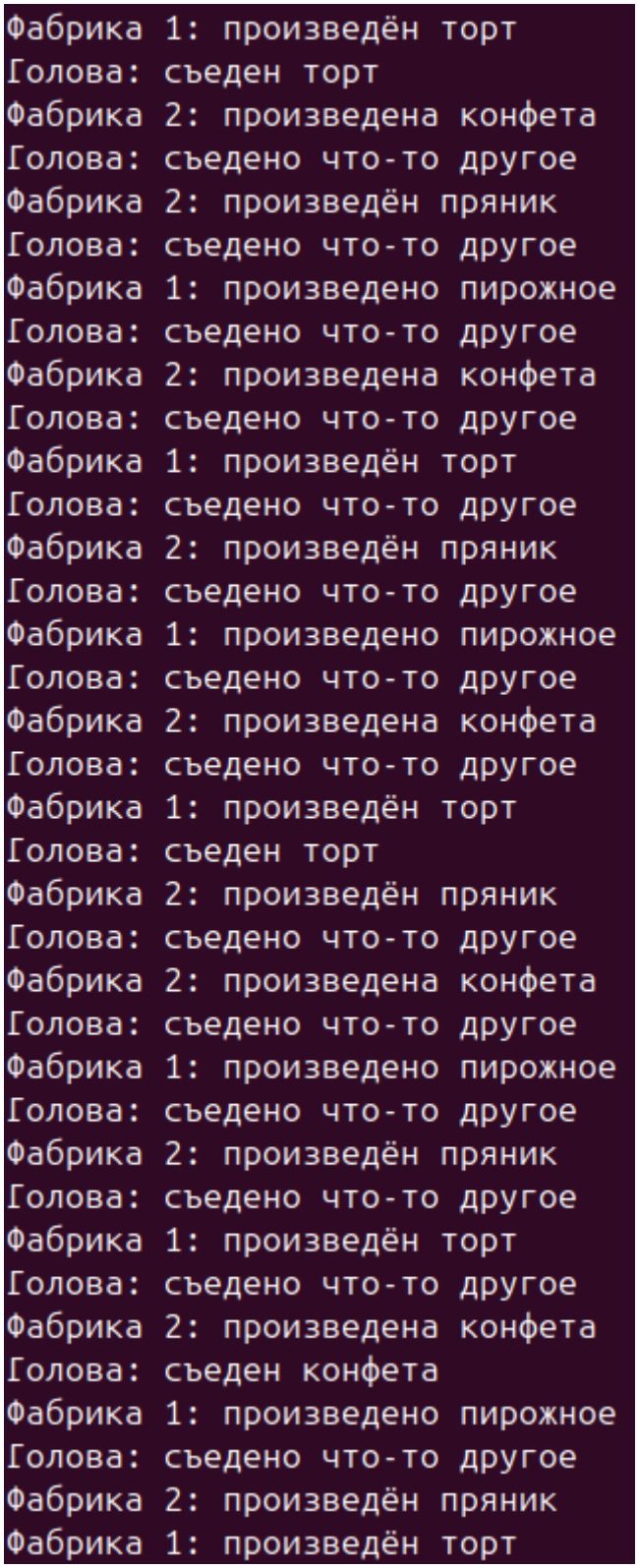
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <string.h>
#include <time.h>

#define MAX_STORAGE 10

typedef enum { CAKE, PASTRY, CANDY, GINGERBREAD } ProductType;

// Структура склада
typedef struct {
    ProductType storage[MAX_STORAGE];
    int count;
} Storage;

// Функции для работы с семафорами
void sem_wait(int sem_id, int sem_num) {
    struct sembuf op = {sem_num, -1, 0};
    semop(sem_id, &op, 1);
}

void sem_signal(int sem_id, int sem_num) {
    struct sembuf op = {sem_num, 1, 0};
    semop(sem_id, &op, 1);
}

void factory1(int shm_id, int sem_id) {
    Storage* storage = (Storage*)shmat(shm_id, NULL, 0);
    if (storage == (void*)-1) {
        perror("shmat");
        exit(1);
    }

    while (1) {
        // Производство торта
        sem_wait(sem_id, 1); // Ждем место на складе
        sem_wait(sem_id, 0); // Мьютекс
        storage->storage[storage->count++] = CAKE;
```

```
        printf("Фабрика 1: произведён торт\n");
        sem_signal(sem_id, 0); // Освобождаем мьютекс
        sem_signal(sem_id, 2); // Увеличиваем заполненность склада
        usleep(70000);

        // Производство пирожного
        sem_wait(sem_id, 1);
        sem_wait(sem_id, 0);
        storage->storage[storage->count++] = PASTRY;
        printf("Фабрика 1: произведено пирожное\n");
        sem_signal(sem_id, 0);
        sem_signal(sem_id, 2);
        usleep(50000);
    }
}

void factory2(int shm_id, int sem_id) {
    Storage* storage = (Storage*)shmat(shm_id, NULL, 0);
    if (storage == (void*)-1) {
        perror("shmat");
        exit(1);
    }

    while (1) {
        // Производство конфет
        sem_wait(sem_id, 1); // Ждем место на складе
        sem_wait(sem_id, 0); // Мьютекс
        storage->storage[storage->count++] = CANDY;
        printf("Фабрика 2: произведена конфета\n");
        sem_signal(sem_id, 0); // Освобождаем мьютекс
        sem_signal(sem_id, 2); // Увеличиваем заполненность склада
        usleep(40000);

        // Производство пряников
        sem_wait(sem_id, 1);
        sem_wait(sem_id, 0);
        storage->storage[storage->count++] = GINGERBREAD;
        printf("Фабрика 2: произведён пряник\n");
        sem_signal(sem_id, 0);
        sem_signal(sem_id, 2);
        usleep(60000);
    }
}
```

```
void head(int shm_id, int sem_id, ProductType preferred1, ProductType
preferred2, int delay) {
    Storage* storage = (Storage*)shmat(shm_id, NULL, 0);
    if (storage == (void*)-1) {
        perror("shmat");
        exit(1);
    }

    while (1) {
        sem_wait(sem_id, 2); // Ждем наличие продукции на складе
        sem_wait(sem_id, 0); // Мьютекс
        ProductType product = storage->storage[--storage->count];
        sem_signal(sem_id, 0); // Освобождаем мьютекс
        sem_signal(sem_id, 1); // Увеличиваем место на складе

        if (product == preferred1 || product == preferred2) {
            printf("Голова: съеден %s\n",
                   product == CAKE ? "торт" :
                   product == PASTRY ? "пирожное" :
                   product == CANDY ? "конфета" : "пряник");
        } else {
            printf("Голова: съедено что-то другое\n");
        }
        usleep(delay);
    }
}

int main() {
    int shm_id = shmget(IPC_PRIVATE, sizeof(Storage), IPC_CREAT | 0666);
    if (shm_id < 0) {
        perror("shmget");
        exit(1);
    }

    Storage* storage = (Storage*)shmat(shm_id, NULL, 0);
    if (storage == (void*)-1) {
        perror("shmat");
        exit(1);
    }
    storage->count = 0;

    int sem_id = semget(IPC_PRIVATE, 3, IPC_CREAT | 0666);
    if (sem_id < 0) {
        perror("semget");
        exit(1);
```

```
    }
    semctl(sem_id, 0, SETVAL, 1); // Мьютекс
    semctl(sem_id, 1, SETVAL, MAX_STORAGE); // Свободные места
    semctl(sem_id, 2, SETVAL, 0); // Заполненные места

    if (fork() == 0) {
        factory1(shm_id, sem_id);
        exit(0);
    }

    if (fork() == 0) {
        factory2(shm_id, sem_id);
        exit(0);
    }

    if (fork() == 0) {
        head(shm_id, sem_id, CAKE, CANDY, 80000); // Голова 1
        exit(0);
    }

    if (fork() == 0) {
        head(shm_id, sem_id, PASTRY, GINGERBREAD, 90000); // Голова 2
        exit(0);
    }

    if (fork() == 0) {
        head(shm_id, sem_id, -1, -1, 70000); // Голова 3 (универсальная)
        exit(0);
    }

    while (wait(NULL) > 0);
    shmctl(shm_id, IPC_RMID, NULL);
    semctl(sem_id, 0, IPC_RMID);
    return 0;
}
```

Скриншот выполнения программы:



Сравнение реализаций:

Критерий	С потоком (Pthreads)	С процессами
Синхронизация	Мьютексы и условные переменные	Семафоры
Организация склада	Связанный список в общей памяти	Массив в разделяемой памяти
Затраты на ресурсы	Лёгкие (все потоки в одном процессе)	Выше (каждый процесс имеет собственный адресный слой)
Производительность	Быстрее (меньше накладных расходов)	Медленнее (из-за межпроцессного взаимодействия)
Масштабируемость	Хорошо подходит для задач на одной машине	Может быть адаптирована для работы на разных машинах
Простота реализации	Легче реализовать из-за общего пространства памяти	Сложнее из-за необходимости использования семафоров и памяти
Подходящие задачи	Подходит для программ, работающих в одном процессе	Подходит для сложных приложений с разделением ресурсов

Вывод: в ходе лабораторной работы были разработаны две программы на языке С для организации взаимодействия производителей и потребителей с использованием потоков (Pthreads) и использованием процессов. Произвели сравнение положительных и отрицательных сторон реализаций.