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);

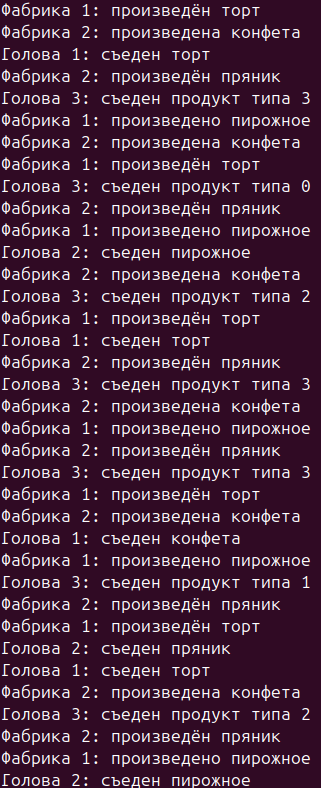
pthread\_join(h2, NULL);

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);

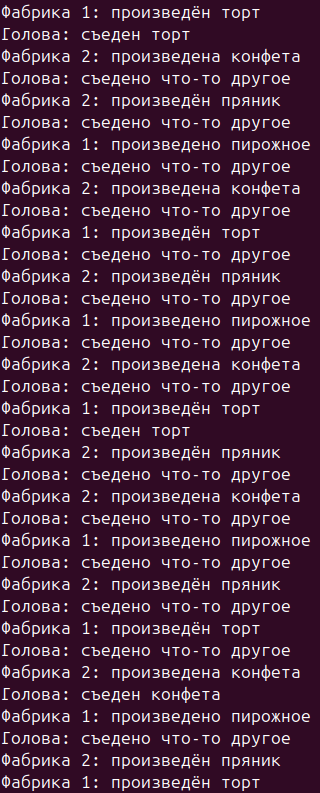
shmctl(shm\_id, IPC\_RMID, NULL);

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 C для организации взаимодействия производителей и потребителей с использованием потоков (Pthreads) и использованием процессов. Произвели сравнение положительных и отрицательных сторон реализаций.