**Условные переменные**

1. Небольшое объяснение задачи

Здесь используется два потока:

Поток-производитель получает данные от оборудования (симулируется с помощью sleep(1)) и передает их в общий ресурс.

Поток-потребитель обрабатывает данные, которые ему передает поток-производитель.

Проблема заключается в синхронизации потоков, чтобы избежать "гонок" за ресурсы (то есть конфликтов, когда оба потока пытаются изменить один и тот же ресурс одновременно) и эффективно передавать данные.

2. Главные переменные

Программа использует следующие ключевые переменные:

data\_read - это флаг, который показывает, есть ли данные для обработки. Если флаг равен 1, то данные готовы для потребителя. Если флаг равен 0, то данных нет.

pthread\_mutex\_t mutex - это мьютекс, который защищает доступ к флагу data\_ready, гарантируя, что только один поток одновременно может читать или изменять его значение.

pthread\_cond\_t condvar - это условная переменная, которая позволяет одному потоку "разбудить" другой, когда наступает нужное событие (например, когда данные готовы для обработки).

3. Поток-потребитель

Поток-потребитель выполняет следующую работу:

Потребитель заходит в бесконечный цикл обработки данных.

Внутри цикла потребитель сначала блокирует мьютекс, чтобы получить эксклюзивный доступ к флагу data\_ready.

pthread\_mutex\_lock(&mutex);

Это предотвращает одновременное изменение флага двумя потоками (гонка за данные).

Потребитель проверяет значение флага. Если флаг равен 0, данные еще не готовы, и потребитель вызывает функцию ожидания условной переменной pthread\_cond\_wait().

while (!data\_ready) {

    pthread\_cond\_wait(&condvar, &mutex);

}

Что происходит в этой функции:

Мьютекс освобождается (потому что потребителю не нужно проверять данные до тех пор, пока производитель их не предоставит).

Поток переходит в состояние ожидания (sleep) до тех пор, пока производитель не вызовет функцию pthread\_cond\_signal(), чтобы разбудить его.

Как только поток будет разбужен, pthread\_cond\_wait() автоматически снова заблокирует мьютекс.

Когда данные готовы (флаг становится равным 1), потребитель обрабатывает данные, сбрасывает флаг обратно в 0, чтобы показать, что данные были обработаны.

data\_ready = 0;

После обработки данных поток освобождает мьютекс.

pthread\_mutex\_unlock(&mutex);

4. Поток-производитель

Поток-производитель выполняет следующую работу:

Производитель получает данные от оборудования (симулируется через вызов sleep(1), чтобы задержать поток).

sleep(1);

Производитель блокирует мьютекс, чтобы получить доступ к флагу data\_ready.

pthread\_mutex\_lock(&mutex);

Если флаг data\_ready уже установлен в 1 (то есть данные еще не обработаны потребителем), производитель вызывает pthread\_cond\_wait(), ожидая, пока потребитель освободит флаг.

Как только флаг сброшен, производитель устанавливает его в 1, что означает, что данные готовы для обработки.

data\_ready = 1;

Производитель использует условную переменную, чтобы разбудить потребителя и дать ему сигнал, что данные готовы.

pthread\_cond\_signal(&condvar);

После этого поток освобождает мьютекс, позволяя другим потокам получить к нему доступ.

pthread\_mutex\_unlock(&mutex);

5. Основные концепции

Мьютексы - используются для защиты общих ресурсов (в данном случае флага data\_ready), гарантируя, что только один поток может изменять флаг в любой момент времени.

Условные переменные - используются для эффективной синхронизации между потоками. Поток-потребитель не должен бесконечно проверять флаг data\_ready — он может просто "заснуть" и ждать, пока производитель не разбудит его с помощью pthread\_cond\_signal().

Когда поток-производитель устанавливает флаг data\_ready в 1, он "пробуждает" поток-потребитель, используя функцию pthread\_cond\_signal(). Поток-потребитель проверяет флаг и начинает обработку данных.

Пример кода:

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

int data\_ready = 0;

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_cond\_t condvar = PTHREAD\_COND\_INITIALIZER;

void\* consumer(void \*p) {

printf("This is the consumer thread...\n");

while (1) {

pthread\_mutex\_lock(&mutex);

while (!data\_ready) {

pthread\_cond\_wait(&condvar, &mutex);

}

// Process the data

printf("Consumer: received data from producer\n");

data\_ready = 0;

pthread\_cond\_signal(&condvar);

pthread\_mutex\_unlock(&mutex);

}

return NULL;

}

void\* producer(void \*p) {

printf("This is the producer thread...\n");

while (1) {

sleep(1); // Simulate receiving data from hardware

printf("Producer: received data from h/w\n");

pthread\_mutex\_lock(&mutex);

while (data\_ready) {

pthread\_cond\_wait(&condvar, &mutex);

}

data\_ready = 1;

pthread\_cond\_signal(&condvar);

pthread\_mutex\_unlock(&mutex);

}

return NULL;

}

int main() {

pthread\_t prod\_thread, cons\_thread;

printf("Starting producer and consumer example...\n");

// Create producer and consumer threads

pthread\_create(&prod\_thread, NULL, producer, NULL);

pthread\_create(&cons\_thread, NULL, consumer, NULL);

// Give threads time to work

sleep(20);

return 0;

}

Результат:

