**Межпроцессное взаимодействие в системе linux**

**Общее описание**

В каждом варианте лабораторной работы требуется написать одну или две программы на языке С++, которые выполняют требуемые действия с помощью нескольких видов межпроцессного взаимодействия в системе Linux. Файлы работы должны собираться с помощью Makefile. Программы должны корректно обрабатывать ошибки и выводить соответствующие сообщения (рекомендуется вывод в лог), кроме того, должны выводиться статусные сообщения (присоединился, завершился, включился и тп).

**Типы процессов**

В результате в каждом варианте может получиться один или два процесса/программы двух видов: 

* **Хост.** Есть в любом варианте. Является инициатором (процессом/программой, которая создает все виды взаимодействия, семафоры и т.д.). Запускается всегда первым. Файл с функцией main хоста должен называться host.cpp. 
* **Клиент.** В некоторых вариантах присутствует в виде ребенка хоста, в других — отдельным независимым процессом/программой. Если является отдельной программой, файл с функцией main должен называться client.cpp.

**Дополнительные операции**

**Знакомство(handshake)**

Производится перед установкой соединения, используется для синхронизации идентификаторов соединения.

1. Без знакомства. Никаких дополнительных действий не производится.
2. Знакомство с помощью сигналов. Клиенту при запуске передается pid хоста, на этот pid клиент отправляет сигнал SIGUSR1.
3. Двустороннее знакомство с помощью сигнала. Данный тип используется в случае, когда на момент соединения клиента, канал для связи ещё не создан и хост должен оповестить клиента о факте этого создания. Данного оповещения клиент синхронно ожидает после отправки своего первого сообщения хосту. Кроме того, в ответе от хоста (можно обойтись и без этого) передаётся id клиента в сессии, с помощью которого клиент получает нужные идентификаторы дескрипторов для связи. Например, первый клиент получает id 1 и открывает файл pong\_fifo1, второй - pong\_fifo2 и тд.

**Синхронизация**

Используется для установки очередности операций записи/чтения в соединении.

1. Локальные семафоры. sem\_init.
2. Глобальные семафоры. sem\_open.

Все запросы на ожидание в семафорах и функциях ожидания сигнала (за исключением случаев запланированного ожидания, например, при ожидании ввода от пользователя) должны иметь тайм-ауты 5 секунд, в случае превышения таймаута связь должна прерываться, а клиент — останавливать работу.

**Типы и виды конфигураций**

В работе могут быть реализованы различные виды отношений хост-клиент(-ы), а так же два типа, реализация которых зависит от вида. Виды конфигурации: 

* “Родственные”. Клиенты являются процессами-потомками хоста, порожденными функцией fork. Используется первый вид знакомства (“Без знакомства”). Типы:
  + Один к одному. Каждый хост запускает ровно одного клиента. Синхронизация локальным семафором.
  + Один ко многим. Каждый хост запускает (в теории) неограниченное количество клиентов. Для работы с каждым клиентом выделяется отдельный поток.
* “Независимые”. Клиенты являются независимыми процессами, при запуске им передается pid хоста. Здесь необходимо использовать содержательные типы знакомства. Типы:
  + Один к одному. Каждый хост работает ровно с одним клиентом. Синхронизация глобальным семафором. Знакомство: знакомство с помощью сигнала.
  + Один ко многим. Каждый хост работает (в теории) с неограниченным количеством клиентов. Для работы с каждым клиентом выделяется отдельный поток. Синхронизация с помощью глобального семафора. Знакомство: двустороннее знакомство с помощью сигнала.

**Требования к исходным файлам и CMakeLists**

1. Различные типы взаимодействия должны быть реализованы в различных файлах с префиксом “conn\_”, за которым следует кодовое название типа взаимодействия. Исходный код каждого типа должен быть вынесен в класс, все классы должны иметь одинаковый интерфейс. Пример интерфейса:

class Conn

{

public:

Conn(SomeType id , bool create);

void/bool Read(void \*buf, size\_t count);

void/bool Write(void \*buf, size\_t count);

~Conn();

}

1. Каждый файл с расширением \*.cpp (\*.c,..) должен автоматически компилироваться в объектный файл без изменения CMakeLists
2. Код функции main хоста и клиента (в случае независимой конфигурации) должен находиться в файлах host.cpp и client.cpp соответственно
3. Файлы должны компилироваться с флагами -Wall и –Werror
4. Для каждого исходного файла (\*.cpp/\*.c) с префиксом “conn\_\*”, автоматически должны генерироваться исполняемые файлы “host\_\*” и “client\_\*” (в случае независимой конфигурации), где \* - кодовое слово типа взаимодействия (TYPE\_CODE);
5. Вызов команды make в папке проекта должен приводить к тому, что будут собраны все исполняемые файлы host\* и client\* для всех видов взаимодействия, реализованных в файлах “conn\_\*”;
6. Необходимо завернуть вызов cmake в скрипт build.sh. Данный скрипт должен очищать все промежуточные файлы, сгенерированные cmake-ом.

**Типы взаимодействия**

В случае реализации *родственной* конфигурации необходимо использовать специальные модификации типов, предназначенные для использования родственными процессами, если таковые предусмотрены во всех перечисленных в варианте видах.

1. Общая память (mmap). *TYPE\_CODE* - *mmap*. Anonymous mapping.
2. Общая память (shmget). *TYPE\_CODE* - *seg*. Заполнение родительским процессом данных в сегмент должно происходить уже после создания дочернего процесса.
3. Общие файлы (shmopen+mmap). *TYPE\_CODE* - *shm*. Здесь, в отличие от пункта 1, работа с общей памятью должна производиться в каждом процессе уже после создания дочернего (так, будто это несвязанные процессы).
4. Очереди сообщений (mq\_open). *TYPE\_CODE* - *mq*. Работа с очередью должна производиться в каждом процессе уже после создания дочернего (так, будто это несвязанные процессы).
5. Программные каналы (pipe). *TYPE\_CODE* - *pipe*.
6. Именованные каналы (mkfifo). *TYPE\_CODE* - *fifo*. Работа с именованным каналом должна производиться в каждом процессе уже после создания дочернего (так, будто это несвязанные процессы). После окончания работы родительский процесс должен удалять файл именованного канала.
7. Сокеты (socket). *TYPE\_CODE* – *sock*

**Варианты заданий**

1. Тематическая пошаговая игра **“Волк и семеро козлят”**.Хост - волк, клиент(ы) - козлята, n- количество козлят. На каждом ходу волк выбрасывает число от 1 до 100, живые козлята делают то же самое, а мертвые выбрасывают число от 1 до 50. Если число живого козленка отличается от числа волка не более, чем на , то считается, что он спрятался. Козлята, которые не спрятались, попадаются и считаются мертвыми. Если число, полученное мертвым козленком, отличается не более, чем на — он воскресает. До подключения козлят ничего не происходит, игра заканчивается, если в течение 2 ходов подряд все козлята мертвы. За логику отвечает хост, клиенты шлют ему числа козлят, а в ответ получают статус козленка (жив, мертв).

Варианты задания:

|  |  |  |  |
| --- | --- | --- | --- |
| **Номер** | **Конфигурация** | **Типы взаимодействия** | **Баллы** |
| 01 | Родственные. Один к одному. | 1, 3, 5 | 10 |
| 02 | Родственные. Один к одному. | 2, 4, 6 | 10 |
| 03 | Родственные. Один к одному. | 4, 6, 7 | 15 |
| 04 | Независимые. Один к одному. | 2, 6, 7 | 15 |
| 05 | Независимые. Один к одному. | 2, 4, 6 | 15 |
| 06 | Независимые. Один к одному. | 4, 6, 7 | 15 |
| 07 | Родственные. Один ко многим. | 3, 4, 5 | 15 |
| 08 | Родственные. Один ко многим. | 5, 6, 7 | 15 |
| 09 | Независимые. Один ко многим. | 2, 4, 6 | 15 |
| 10 | Независимые. Один ко многим. | 2, 6, 7 | 20 |

1. **«Прогноз погоды».** Пользователь на хосте отправляет клиентам дату в произвольном формате (определяется хостом). Клиенты должны «предсказать» температуру на улице в этот день. В качестве результата предсказания следует использовать псевдослучайные числа. Результаты должны быть воспроизводимыми (для одного и того же дня клиент должен предсказывать одинаковый результат) и разные клиенты должны давать разные предсказания.

Варианты задания:

|  |  |  |  |
| --- | --- | --- | --- |
| **Номер** | **Конфигурация** | **Типы взаимодействия** | **Баллы** |
| 11 | Родственные. Один к одному. | 1, 3, 5 | 10 |
| 12 | Родственные. Один к одному. | 2, 4, 6 | 10 |
| 13 | Родственные. Один к одному. | 4, 6, 7 | 15 |
| 14 | Независимые. Один к одному. | 2, 6, 7 | 10 |
| 15 | Независимые. Один к одному. | 2, 4, 6 | 15 |
| 16 | Независимые. Один к одному. | 4, 6, 7 | 15 |
| 17 | Родственные. Один ко многим. | 3, 4, 5 | 15 |
| 18 | Родственные. Один ко многим. | 5, 6, 7 | 15 |
| 19 | Независимые. Один ко многим. | 2, 4, 6 | 15 |
| 20 | Независимые. Один ко многим. | 2, 6, 7 | 20 |

Допускается реализация не всех типов взаимодействия. Баллы за нереализованный вид взаимодействия будут выставляться следующим образом:

|  |  |  |  |
| --- | --- | --- | --- |
| Число нереализованных взаимодействий / базовый балл | 10 | 15 | 20 |
| 1 | 7 | 10 | 13 |
| 2 | 3 | 5 | 7 |