Лабораторная работа №1 «Реализация межпроцессного взаимодействия посредством сообщений»

Введение

Для изучения различных аспектов функционирования распределенных систем часто используют несколько моделей распределенной обработки информации. При этом та или иная модель выбирается в зависимости от исследуемой задачи из области распределенных вычислений. Взаимодействие процессов в моделях обычно происходит посредством обмена сообщениями.

Распределенное вычисление удобно рассматривать в виде совокупности дискретных событий, каждое из которых вызывает небольшое изменение состояния всей системы. Система становится «распределенной» благодаря тому обстоятельству, что каждое событие приводит к изменению только части глобального состояния всей системы. А именно, при наступлении события изменяется лишь локальное состояние одного процесса и, возможно, состояние одного или нескольких каналов связи.

В данной лабораторной работе необходимо реализовать библиотеку межпроцессного взаимодействия посредством обмена сообщениями, которая будет использована в дальнейшем для изучения основных свойств распределенных вычислительных систем.

Исходные данные

Число N процессов, составляющих распределенную систему. При проверке используются значения от 1 до 10.

Постановка задачи

Распределенная система состоит из конечного множества N независимых процессов $\{P_1, P_2, \ldots, P_N\}$. Если процесс P_i может напрямую отправлять сообщения процессу P_j, то мы будем говорить, что между процессом P_i и процессом P_j имеется канал C_{ij}. Для удобства все каналы будем считать однонаправленными. Чтобы два процесса имели возможность обмениваться сообщениями, между ними необходимо установить два разнонаправленных канала, каждый из которых служит для передачи сообщений в одном направлении. Топология сети распределенной системы является полносвязной, т.е. каждый процесс может взаимодействовать со всеми другим процессами. В данной лабораторной работе используется асинхронный обмен сообщениями.

Теоретическая модель процесса определяется в виде множества его возможных состояний (в т.ч. начальных состояний) и множества дискретных событий. Событие e, происходящее в процессе P, представляет собой атомарное действие, которое может изменить состояние самого процесса P и состояние канала C, инцидентного этому процессу: состояние C будет изменено при отправке сообщения по этому каналу или при получении сообщения из этого канала. Поэтому все события могут быть классифицированы как внутренние события, события отправки и события получения сообщения. Следует отметить, что в качестве одного атомарного события также часто рассматривают отправку сразу нескольких сообщений по нескольким каналам связи, инцидентным P, например при широковещательной или групповой рассылке. При этом для удобства мы будем полагать, что получение сообщения не может совпадать с отправкой или получением других сообщений в виде одного события.

Задание

Необходимо реализовать библиотеку межпроцессного взаимодействия для распределенной системы, описанной выше. В качестве процессов системы используются процессы ОС Linux, и обмен сообщениями осуществляется с помощью неименованных каналов (*pipes*). Родительский процесс создает все дочерние процессы при помощи функции *fork()* (завершение дочерних процессов отслеживается при помощи *wait()*), каналы открываются функцией *pipe()*.

Количество создаваемых дочерних процессов в полносвязной топологии определяется параметром командной строки «*-p X* », где *X* — количество процессов. При создании процессов следует не забывать, в каком процессе происходит выполнение, что позволяет предотвратить создание 2^X процессов. Таким образом, общее число процессов в распределенной системе $N = X + 1$.

Процессы обмениваются сообщениями посредством записи и чтения из каналов. Каждое сообщение состоит из заголовка и тела сообщения. В заголовок (структура *MessageHeader*) входят следующие поля:

1. «магическая подпись», которая используется при автоматической проверке лабораторных работ (константа *MESSAGE_MAGIC*);
2. длина тела сообщения;
3. тип сообщения;
4. метка времени.

Таким образом, максимальная длина сообщения составляет 64 Кб. В работе необходимо использовать структуру заголовка и константы из прилагаемого заголовочного файла *ipc.h*.

Информацию обо всех открытых дескрипторах каналов (чтение / запись) необходимо вывести в файл *pipes.log*. Это помогает обнаружить часто встречаемую ошибку: реализацию топологии «общая шина» вместо полносвязной. Кроме того, следует не забывать, что неиспользуемые дескрипторы необходимо закрыть.

Каждый процесс должен иметь свой локальный идентификатор: $[0..N-1]$. Причем родительскому процессу присваивается идентификатор *PARENT_ID*, равный 0. Данные идентификаторы используются при отправке и получении сообщений.

При запуске программы родительский процесс осуществляет необходимую подготовку для организации межпроцессного взаимодействия, после чего создает *X* идентичных дочерних процессов. Функция родительского процесса ограничивается созданием дочерних процессов и дальнейшим мониторингом их работы.

Выполнение каждого дочернего процесса состоит из трех последовательных фаз:

1. процедура синхронизации со всеми остальными процессами в распределенной системе;
2. «полезная» работа дочернего процесса;
3. процедура синхронизации процессов перед их завершением.

Первая фаза работы дочернего процесса заключается в том, что при запуске он пишет в лог (все последующие действия также логируются) и отправляет сообщение типа *STARTED* всем остальным процессам, включая родительский. Затем процесс дожидается сообщений *STARTED* от других дочерних процессов, после чего первая фаза его работы считается оконченной. В данной лабораторной работе дочерние процессы не выполняют никакой «полезной» работы, поэтому сразу переходят к третьей фазе завершения собственного выполнения. В этой фазе дочерние процессы отправляют сообщение типа *DONE* всем, включая родителя. Условием завершения дочернего процесса является получение сообщений *DONE* от всех остальных дочерних процессов. В сообщениях *STARTED* и *DONE* в качестве тела сообщения следует использовать такие же строки, как были записаны в лог. Таким образом, для дочерних процессов определены следующие события (в скобках указаны имена строк форматирования для логирования):

- процесс начал выполнение работы (*log_started_fmt*);
- процесс получил сообщения о запуске всех остальных процессов (*log_received_all_started_fmt*);
- процесс окончил выполнение «полезной» работы (*log_done_fmt*);
- процесс получил сообщения о выполнении «полезной» работы всеми дочерними процессами (*log_received_all_done_fmt*).

Родительский процесс не должен отправлять сообщения дочерним процессам, однако сообщения *STARTED* и *DONE* должны быть им получены. Родительский процесс завершается при завершении всех остальных процессов.

Все события логируются на терминал и в файл *events.log*. При логировании необходимо использовать форматы сообщений из прилагаемого заголовочного файла. Стоит обратить внимание на то, что последовательности событий, регистрируемые при различных выполнениях программы, не совпадают.

В реализации запрещается использовать многопоточность: один процесс — один поток. Кроме того, нельзя использовать разделяемую память, примитивы синхронизации (семафоры и т.п.), функции *select()* и *poll()*.

Требования к реализации и среда выполнения

Реализацию необходимо выполнить на языке программирования Си с использованием предоставленных заголовочных файлов и библиотеки из архива *pa1_starter_code.tar.gz*. Архив содержит следующие файлы:

ipc.h	объявления структур данных и функций для организации межпроцессного взаимодействия. Объявленные функции необходимо реализовать;
common.h	некоторые константы;
pa1.h	форматы строк для логирования;

Заголовочные файлы содержат большое число важных комментариев, пояснений и рекомендаций. Запрещается модифицировать любые файлы из архива: при автоматической проверке они заменяются на оригинальные.

Работа присылается в виде архива с именем *pa1.tar.gz*, содержащим каталог *pa1*. Все файлы с исходным кодом и заголовки должны находиться в корне этого каталога. Среда выполнения — Linux (Ubuntu версии ≥ 12.10). При автоматической проверке используется следующая команда: *gcc -std=c99 -Wall -pedantic *.c*. При наличии варнингов работа не принимается. При успешном выполнении запущенные процессы не должны использовать *stderr*, код завершения программы должен быть равен 0.