Домашняя работа №4 (2015)

Problem H41: Синхронное чтение-2

Условие этой задачи практически дословно повторяет условие задачи `H32`, только вместо сигналов должны быть использованы семафоры.

Напишите программу, которая обрабатывает текстовый ввод (со стандартного потока ввода) при помощи параллельных процессов указанным ниже образом.

Программа создает два сыновних процесса (назовем их "первым" и "вторым") с выделением необходимых IPC-ресурсов. Первый процесс обрабатывает нечетные строки ввода, второй - четные строки ввода. Строки нумеруются с единицы. Первый процесс делает регистр каждой буквы строки верхним (a -> A, A -> A, 2 -> 2, # -> #), второй процесс - нижним (a -> a, A -> a, 2 -> 2, # -> #). Изменению регистра подвергаются только латинские символы. Каждый процесс сразу печатает на стандартный поток вывода полученную строку. По окончании ввода первый и второй процесс завершаются, а с ними завершается и отцовский процесс.

Необходимо реализовать следующее взаимодействие процессов. В то время, когда один сыновний процесс обрабатывает очередную строку, другой сыновний процесс должен ожидать окончания этой обработки. После успешного чтения и обработки строки сыновний процесс передает управление своему брату при помощи семафоров. Если перед чтением строки сыновний процесс определил, что ввод завершен, он завершает свою работу с кодом 0. После этого отцовский процесс удаляет созданные IPC-ресурсы, что приводит к корректному "вежливому" завершению другого сыновнего процесса с кодом 0. Отцовский процесс дожидается завершения работы этого процесса и сам завершается с кодом 0, если не было каких-либо ошибок, приводящих к другому коду завершения (об этом ниже).

Если отцовский процесс не смог создать хотя бы один IPC-ресурс или дочерний процесс, он сообщает об этом на стандартный поток ошибок при помощи функций `strerror` или `perror` и завершается с кодом 1. Если при этом уже существуют какие-нибудь дочерние процессы, они должны быть завершены (с кодом 1) посылкой им соответствующего сигнала.

Необходимо явно освобождать занимаемые ресурсы (файлы, динамическую память и т.п.). В таблице процессов не должны оставаться "зомби". Ожидание процессов должно быть пассивным, т.е. не тратить ресурсы процессора.

Можно предполагать, что отцовский процесс запускается в отдельной группе процессов.

Отцовский процесс должен завершаться последним.

Выберите алгоритм создания ключа IPC-объектов таким, чтобы допускалась независимая параллельная работа нескольких копий отцовского (и сыновних) процессов.

Для реализации взаимодействия при чтении нельзя использовать сигналы.

Problem H42: Синхронное чтение-3

Условие этой задачи практически дословно повторяет условие задачи `H32`, только для синхронизации процессов вместо сигналов должны быть использованы сообщения с семантикой блокирующего чтения/получения.

Напишите программу, которая обрабатывает текстовый ввод (со стандартного потока ввода) при помощи параллельных процессов указанным ниже образом.

Программа создает два сыновних процесса (назовем их "первым" и "вторым") с выделением необходимых IPC-ресурсов. Первый процесс обрабатывает нечетные строки ввода, второй - четные строки ввода. Строки нумеруются с единицы. Первый процесс делает регистр каждой буквы строки верхним (a -> A, A -> A, 2 -> 2, # -> #), второй процесс - нижним (a -> a, A -> a, 2 -> 2, # -> #). Изменению регистра подвергаются только латинские символы. Каждый процесс сразу печатает на стандартный поток вывода полученную строку. По окончании ввода первый и второй процесс завершаются, а с ними завершается и отцовский процесс.

Необходимо реализовать следующее взаимодействие процессов. В то время, когда один сыновний процесс обрабатывает очередную строку, другой сыновний процесс должен ожидать окончания этой обработки. После успешного чтения и обработки строки сыновний процесс посылает сообщение своему брату (брат ждет получения этого сообщения и без него не начинает чтение). Если перед чтением строки сыновний процесс определил, что ввод завершен, он завершает свою работу с кодом 0. После этого отцовский процесс удаляет созданные IPC-ресурсы, что приводит к корректному "вежливому" завершению другого сыновнего процесса с кодом 0. Отцовский процесс дожидается завершения работы этого процесса и сам завершается с кодом 0, если не было каких-либо ошибок, приводящих к другому коду завершения (об этом ниже).

Если отцовский процесс не смог создать хотя бы один IPC-ресурс или дочерний процесс, он сообщает об этом на стандартный поток ошибок при помощи функций `strerror` или `perror` и завершается с кодом 1. Если при этом уже существуют какие-нибудь дочерние процессы, они должны быть завершены (с кодом 1) посылкой им соответствующего сигнала.

Необходимо явно освобождать занимаемые ресурсы (файлы, динамическую память и т.п.). В таблице процессов не должны оставаться "зомби". Ожидание процессов должно быть пассивным, т.е. не тратить ресурсы процессора.

Можно предполагать, что отцовский процесс запускается в отдельной группе процессов.

Отцовский процесс должен завершаться последним.

Выберите алгоритм создания ключа IPC-объектов таким, чтобы допускалась независимая параллельная работа нескольких копий отцовского (и сыновних) процессов.

Для реализации взаимодействия при чтении нельзя использовать сигналы.

Problem H43: Крестики-нолики: клиент

В этом задании вам необходимо реализовать клиент для игры в "Крестики-нолики". В другом задании вам необходимо реализовать сервер для этой же игры. Для тестирования клиента рекомендуется использовать как свой сервер, так и сервер своих коллег по группе.

Целью игры является составление линии из трех фигур одного типа на поле 3x3. Один игрок выставляет "крестики", другой игрок - "нолики". Допустимы горизонтальные, вертикальные или диагональные линии. Игра заканчивается победой одного из игроков или ситуацией ничьи.

Сервер контролирует несколько параллельно идущих игр. Клиенты обрабатывают ходы пользователей. Пользователи играют друг с другом в парах при помощи клиентов. Вначале случайным образом выбираются два игрока и они совершают ходы до возникновения финальной ситуации (выигрыша одного из игроков или ничьи) или прерывания игры одним из игроков.

Запуск клиента

Клиент должен поддерживать следующий формат командной строки: `СИМВОЛ ИМЯ`, где `СИМВОЛ` - это одна из букв `X`, `x`, `O`, `o`, означающих фигуру игрока, `ИМЯ` - это последовательность латинских букв, означающих имя игрока (размер имени ограничен только возможностями командной строки и может отличаться в разных ОС).

Сразу после запуска клиент выполняет инициализацию. Цель инициализации - получение информации об игре и о сопернике (игроке противоположной фигуры).

Клиент подключается к серверу при помощи сокета по имени файла `/tmp/ttt` (сокет во внутреннем домене потокового типа, протокол выбирается автоматически). Клиент посылает на сервер (записывает в сокет) имя своего пользователя (последовательность байтов до нулевого байта) и его фигуру (1 байт - `'X'` или `'O'`). Затем клиент ждет, пока сервер поместит в сокет имя пользователя-соперника (целое число типа `int` с длиной строки и последовательность байтов в количества длины строки). Если не удалось подключиться к серверу, отправить или считать данные, клиент пишет строку с причиной ошибки, полученной при помощи функций `strerror` или `perror`, и завершается с кодом 1.

Далее клиент подключается к следующим существующим IPC-объектам, используя имя сокета и число 1 для вычисления их ключа:

1. разделяемая память размера 9 целых чисел типа `int` (для игрового поля);
2. массив семафоров из 1 семафора (для синхронизации доступа к игровому полю);
3. очередь сообщений (для обмена информацией о ходах между клиентами во время игры).

Процесс игры

Начинают крестики.

Клиент работает в следующем цикле с пользователем. Клиент пишет приглашение для ввода команды пользователю, пользователь пишет команду на стандартный ввод клиенту, клиент выполняет команду. Если команда не меняет состояние игрового поля, клиент

пишет приглашение и запрашивает следующую команду, а в противном случае клиент проверяет наличие финальной ситуации, отправляет клиенту соперника сделанный ход или признак окончания игры (см. ниже), получает от него ход, сделанный соперником (если игра не закончилась), и пишет его на стандартный вывод (формат описан ниже).

Приглашение состоит из имени пользователя, двоеточия и 1 символа пробела.

Клиент принимает 2 команды (ввод, не являющийся одной из этих команд, игнорируется, клиент пишет приглашение для ввода следующей команды):

1. команда для совершения хода. Чтобы дать эту команду, пользователь пишет только координату клетки, т.е. только два числа от 1 до 3, разделенных пробелами, табуляциями, символами перевода строки и т.п. Первым идет номер строки игрового поля, вторым - номер столбца. Клиент проверяет, свободна ли указанная клетка. Если она занята, клиент пишет на стандартный вывод сообщение `Occupied` и пишет приглашение для ввода следующей команды. Если клетка свободна, клиент меняет ее значение.
2. команда для печати текущего состояния игры. Чтобы дать эту команду, пользователь пишет число 0. Клиент печатает 3 строки, в каждой строке только 3 символа. Первая строка - это первая строка поля, вторая - вторая, третья - третья. Крестику соответствует символ X, нолику - символ O, отсутствию знака - символ точка.

Поскольку оба клиента работают параллельно с одной разделяемой памятью, необходимо синхронизовать их работу с разделяемой памятью. А именно, доступ к разделяемой памяти должен быть помещен в "семафорные скобки". Иными словами, только один из процессов может в один момент времени работать с разделяемой памятью.

Выполнив команду, изменившую состояние игрового поля, клиент должен проверить наличие финальной ситуации.

Если после хода игрока реализовалась финальная ситуация, клиент пишет на стандартный вывод одно из сообщений `You win!`, `You lose`, `We tied`, отправляет клиенту соперника два целых числа типа `int`, равных нулю (это признак окончания игры), закрывает сокет с сервером и завершается с кодом 0.

Если же не реализовалась финальная ситуация, клиент посылает сделанный ход клиенту соперника (два целых числа типа `int`) и ожидает от него ответный ход. В момент ожидания клиент не реагирует на ввод. Ответом будет координата точки, куда соперник сделал ход (два целых числа типа `int`), или два целых числа типа `int`, равных нулю, если игра окончена. Если ход соперника получен (а не нули), клиент пишет на стандартный вывод строку из имени противника, двоеточия, пробела и хода, сделанного противником (два целых числа через пробел), и принимает следующую команду игрока. Если получен признак окончания игры, клиент сообщает о финальной ситуации (как, написано в предыдущем абзаце) и завершается с кодом 0.

Пользователь в любой момент может прервать клиент нажатием клавиш `Ctrl+C`. В этом случае клиент ничего не печатает на стандартный вывод, разрывает сокет с сервером и завершается с кодом 1. Аналогичная реакция клиента должна быть в том случае, когда он обнаруживает окончание ввода перед очередным чтением ввода.

Сервер может в любой момент завершиться. Клиент узнает об этом при попытке воспользоваться IPC-объектом, который будет в этот момент уже удален. В этом случае

клиент печатает на стандартный вывод сообщение `Server disconnected` и завершается с кодом 1.

Ограничения реализации

В коде клиента запрещается использовать функции `system`, `exec***`. При корректном завершении процессы должны освобождать занимаемые ими ресурсы (файлы, динамическую память и т.п.). Можно предполагать, что клиент запускается в отдельной группе процессов.

Problem H44: Крестики-нолики: сервер

В этом задании вам необходимо реализовать сервер для игры в "Крестики-нолики". В другом задании вам необходимо реализовать клиент для этой же игры. Для тестирования сервера рекомендуется использовать как свой клиент, так и клиент своих коллег по группе.

Целью игры является составление линии из трех фигур одного типа на поле 3x3. Один игрок выставляет "крестики", другой игрок - "нолики". Допустимы горизонтальные, вертикальные или диагональные линии. Игра заканчивается победой одного из игроков или ситуацией ничьи.

Сервер контролирует несколько параллельно идущих игр. Его задача - принимать запросы клиентов на игру, искать партнеров, создавать IPC-объекты перед их игрой и удалять их после окончания игры.

В начале своей работы сервер открывает слушающий сокет по имени файла `/tmp/ttt` (сокет во внутреннем домене потокового типа, протокол выбирается автоматически) и принимает запросы на подключение к нему от клиентов. Приняв запрос, сервер считывает имя пользователя клиента и его фигуру (формат смотрите в описании клиента) и печатает на стандартный поток вывода строку вида `Connect: ФИГУРА ИМЯ`, где ФИГУРА - это символ X, если клиент будет играть за крестиков, и O, если за ноликов, ИМЯ - это имя пользователя клиента.

Как только к серверу подключились два произвольных клиента с разными фигурами, которые еще не соединены в пару игроков, сервер создает IPC-объекты для их игры и, если это успешно, посылает каждому в их сокеты имена их соперника, а на стандартный поток вывода пишет строку вида `Start: ИМЯ-КРЕСТИКА ИМЯ-НОЛИКА` (после слова Started идет имя пользователя, играющего крестиками, затем идет имя пользователя, играющего ноликами). Набор IPC-объектов игры перечислен в описании клиента.

Если не удалось создать хотя бы один из IPC-объектов, уже созданные IPC-объекты для данной игры должны быть удалены, причина ошибки создания должна быть написана на стандартный поток ошибок при помощи функций `strerror` или `perror`, сокеты с клиентами закрыты.

Сокет должен обрабатывать одновременно до 10 запросов от клиентов. Количество игр, которые параллельно может поддерживать сервер, ограничено только размерами системных таблиц ресурсов ядра ОС.

Как только оба сокета клиентов закрылись, сервер удаляет все IPC-объекты игры, пишет на стандартный вывод строку `Finish: ИМЯ-КРЕСТИКА ИМЯ-НОЛИКА` и считает игру оконченной.

В тот момент, когда клиент закрывает сокет с сервером (по запросу пользователя клиента или по окончании игры), сервер пишет строку `Disconnect: ИМЯ`, где ИМЯ - это имя пользователя клиента. При завершении игры сначала пишется строка Finish, а потом строки Disconnect.

Сервер должен реагировать на приход сигнала `SIGTERM`. В этом случае он удаляет все созданные IPC-объекты, разрывает все сокеты и завершается с кодом 0.

Сервер должен реагировать на приход сигнала `SIGINT`. В этом случае он распечатывает список текущих игр и список ожидающих клиентов. Список текущих игр должен состоять из строк, каждая строка должны быть парой имен через пробел: имя пользователя, играющего крестиками, затем пробел, затем имя пользователя, играющего ноликами. Последовательность строк в этом списке должна совпадать с последовательностью печати строк, начинающихся со Start. Список ожидающих клиентов тоже состоит из строк, каждая строка начинается с фигуры клиента (символ 'X' или 'O'), затем пробел, затем имя пользователя. Последовательность строк в этом списке должна совпадать с последовательностью печати строк, начинающихся с Connect. На время печати списка игр и клиентов начало и окончание игр, а также прием новых клиентов должны быть отложены.

В коде сервера запрещается использовать функции `system`, `exec***`. При корректном завершении процессы должны освобождать занимаемые ими ресурсы (файлы, динамическую память и т.п.). Можно предполагать, что сервер запускается в отдельной группе процессов. Отцовский процесс должен завершаться последним.