пНациональный исследовательский университет “Высшая школа экономики”.

Факультет компьютерных наук. Программная инженерия.

Операционные системы.

Индивидуальное домашнее задание №2 студента группы БПИ213 Абрамова Александра Сергеевича.

Вариант 14.

14. **Задача о гостинице – 3 (дамы и джентльмены).** В гостинице 10 номеров рассчитаны на одного человека и 15 номеров рассчитаны на двух человек. В гостиницу случайно приходят клиенты–дамы и клиенты–джентльмены, и конечно они могут провести ночь в номере только с представителем своего пола. Если для клиента не находится подходящего номера, он уходит искать ночлег в другое место. Клиенты порождаются динамически и уничтожаются при освобождении номера или уходе из гостиницы при невозможности поселиться. **Создать приложение, моделирующее работу гостиницы.** *Каждого клиента реализовать в виде отдельного процесса.*

СОДЕРЖАНИЕ

СЦЕНАРИЙ РЕШАЕМОЙ ЗАДАЧИ

В условии задачи представлено две сущности: отель и посетитель, между которыми происходит взаимодействие следующим образом:

1. Процесс отеля должен быть запущен до посетителей. При этом происходит инициализация всех комнат, а также создание примитивов синхронизации процессов, если программа реализует взаимодействие независимых процессов (в случае дочерних процессов за создание синхропримитивов “отвечает” родительский процесс).
2. По окончании запуска отель открывает дверь для входа путём разблокировки семафора $door\_in$ и начинает ждать одного посетителя на ресепшене с помощью семафора $reception\_in$.
3. Каждый посетитель должен остановиться у двери и проверить, открыта ли она, с помощью системного вызова sem_wait (или аналогичного в зависимости от реализации) для семафора $door\_in$. Если дверь открыта, то посетитель проходит. Иначе посетитель встаёт в очередь и ждёт, когда завершится обработка всех посетителей до него.
4. После прохода в отель посетитель должен сообщить свой запрос. Для этого он размещает соответствующие данные в разделяемой памяти или в очереди сообщений и подходит на ресепшн путём разблокировки семафора $reception\_in$ с помощью системного вызова sem_post (или аналогичного в зависимости от реализации). Для организации ожидания ответа используется семафор $reception\_out$, на котором пользователь дожидается информации от отеля.
5. Отель производит обработку запроса пользователя - поиск подходящей комнаты или определение, что таковой нет, размещает ответ в разделяемой памяти или в очереди сообщений и разблокирует семафор $reception\_out$, сообщая пользователю о готовности ответа и возможности его чтения. При этом отель дожидается сообщения от клиента о том, что ответ прочитан, с помощью семафора $door\_out$.
6. Посетитель, пропущенный разблокировкой семафора $reception\_out$, производит чтение ответа из разделяемой памяти или очереди сообщений и увеличивает значение семафора $door\_out$, таким образом сообщая отелю о получении ответа и уходе. При этом отель приступает к ожиданию следующего посетителя.
7. По окончании обработки ответа посетитель либо уходит, так как для него не нашлось подходящей комнаты, либо заселяется в отель и живёт там некоторое время.
8. Когда посетитель хочет освободить комнату, он повторяет описанный ранее процесс взаимодействия с ресепшеном отеля, изменяя лишь тип запроса.

9. По получении от отеля успешного ответа посетитель уходит, что отражается завершением соответствующего процесса.

Дополнительные комментарии по реализации описанного протокола представлены в исходном коде программ.

СОДЕРЖАНИЕ РАБОТЫ

1. Каталоги `4 - 10` содержат программы на соответствующую оценку. Каждый из них содержит директорию `bin`, где сохраняются исполняемые файлы программы, а также директорию `output`, где представлены выходные данные программы на соответствующих тестах.
2. Файлы `small_test.txt` и `big_test.txt` содержат наборы данных, на которых тестировалась программа. Каждый файл содержит описания посетителей в следующем формате: первая строка указывает пол посетителя (m для мужчин и f для женщин), а вторая строка - время его пребывания в отеле в секундах (от 1 до 15).
3. Файлы `test.sh`, `small.sh` и `big.sh`, собирающие программу, номер которой указывается единственным аргументом командной строки при исполнении скрипта. Исполняемые файлы сохраняются в директории `bin` соответствующей программы. Файл `test.sh` также производит запуск программ `4 - 6` в режиме ожидания входных данных от пользователя с сохранением результата в файл `output.txt` директории `output` соответствующей программы, а `small.sh` и `big.sh` дополнительно запускают все программы на соответствующих наборах тестовых данных, сохраняя результат соответственно в папки `small` и `big` или в одноимённые текстовые файлы директории `output` соответствующей программы.
   3.1. Для компиляции программы используется утилита `gcc` со следующими параметрами:
      3.1.1. `-O2`, позволяющий компилятору производить оптимизации программы для более эффективной работы
      3.1.2. `-fsanitize=address,undefined`, указывающий компилятору на необходимость добавления в код программы дополнительных инструкций, позволяющих отлавливать ошибки во время её выполнения, которые могут приводить к некорректному поведению
      3.1.3. `-fno-sanitize-recover=all`, требующий немедленного завершения программы с ненулевым кодом возврата в случае обнаружения любой ошибки во время выполнения
      3.1.4. `-std=gnu17`, указывающий версию спецификации языка C, которую необходимо использовать при компиляции - `C17`
      3.1.5. `-Werror` для прекращения процесса сборки программы с ошибкой в случае обнаружения любого предупреждения компилятора
      3.1.6. `-Wall` и `-Wextra` для включения всех предупреждений компилятора.

3.1.7. `-lpthread` и `-lrt` для подключения соответствующих библиотек ОС Linux.

3.2. Программы `4 - 6` компилируются одинаково путём сборки `utils.c`, `log.c`, `hotel.c`, `visitor.c` и `index.c` соответствующей программы в исполняемый файл `index.exe` директории `bin`.

3.3. Остальные программы также компилируются одинаково путём сборки `utils.c`, `log.c` и `hotel.c` соответствующей программы в исполняемый файл `hotel.exe` директории `bin`, а также `utils.c`, `log.c` и `visitor.c` соответствующей программы в исполняемый файл `visitor.exe` директории `bin`.

4. Для удобства тестирования программ создан скрипт `run.mjs`, который запускает указанную первым аргументом командной строки программу на входных данных, указываемых вторым аргументом командной строки, а также завершает работу отправкой сигнала завершения $SIGINT$ через 10 секунд для маленького теста и 100 секунд для большого.

4.1. Для программ `4 - 6` происходит запуск одного исполняемого файла `index.exe`, которому затем отправляется сигнал $SIGINT$.

4.2. Для программ `7 - 10` происходит запуск процесса отеля с помощью исполняемого файла `hotel.exe`, а также после небольшого ожидания - все необходимые в соответствии с тестовыми данными процессы посетителей с помощью исполняемого файла `visitor.exe`. При этом происходит сбор выходных данных процессов с последующей их записью в соответствующие файлы директории `output` тестируемой программы. По истечение времени процессу отеля отправляется сигнал завершения $SIGINT$.

5. Для удобства запуска был также создан скрипт `run_all.sh`, который запускает все реализованные программы на всех тестах.

ПРОГРАММА 4

Программа `4` состоит из следующих элементов:

1. Подпрограмма `log`, реализующая ряд методов для потокобезопасного вывода сообщений в текстовый файл.
   1.1. Функция *set_log_file* производит открытие файла для вывода данных и создание семафора для синхронизации записи.
   1.2. Функция *stop_logging* закрывает файл и семафор, а также удаляет семафор, если функция вызывается тем же процессом, который создал семафор.
   1.3. Функция *log_time*, производящая вывод текущего времени в файл для более удобной организации выходных данных.
   1.4. Макро-функция *log*, производящая потокобезопасный вывод очередного сообщения в файл. Одновременно запись в файл может производить только один процесс.
2. Подпрограмма `utils`, реализующая ряд методов для удобства взаимодействия отеля и посетителей и позволяющая производить изменение используемых API ОС Linux без внесения значительных изменений в остальную программу.
   2.1. Файл `protocol.h` определяет структуру сообщения, передаваемого между участниками взаимодействия.
   2.2. Файл `utils.h` определяет структуру “состояния системы” - набор синхропримитивов, которые необходимы для организации взаимодействия процессов, - имена семафоров и разделяемой памяти, а также ряд функций для управления ими.
   2.3. Файл `utils.c` реализует следующие функции:
      2.3.1. *sleep_milliseconds* “усыпляет” текущий процесс на указанное количество миллисекунд.
      2.3.2. Функции *create_semaphore*, *wait_semaphore*, *post_semaphore*, *close_semaphore* и *delete_semaphore* предоставляют интерфейс для удобного взаимодействия с именованными POSIX семафорами операционной системы Linux.
      2.3.3. Функции *create_memory*, *write_memory*, *read_memory*, *close_memory* и *delete_memory* предоставляют интерфейс для удобного взаимодействия с разделяемой памятью в стандарте POSIX.
      2.3.4. Функции *init_state*, *close_state* и *clear_state* предоставляют интерфейс для работы с “состоянием системы” в целом, что позволяет полностью изолировать реализации синхропримитивов.

2.3.5. Функции $request$, $get\_response$, $get\_request$ и $response$ реализуют получение и отправку сообщений между процессами. При этом перед размещением данных в разделяемой памяти установлено ожидание в 250 миллисекунд для имитации взаимодействия, приближенного к реальности, что упрощает отлов ошибок взаимодействия процессов.

2.4. Подпрограмма `hotel`, реализующая процесс отеля в соответствии с описанным ранее сценарием.

2.4.1. Структура $Room$ описывает комнату отеля.

2.4.2. Функция $stop$ используется для обработки сигнала $SIGINT$ завершения программы и осуществляет завершение работы отеля с принудительной остановкой всех известных посетителей и закрытием синхропримитивов.

2.4.3. Функция $findRoomForGender$ осуществляет поиск комнаты, подходящей для посетителя указанного пола.

2.4.4. Функция $registerRoom$ производит заселение посетителя в указанную комнату.

2.4.5. Функция $handleCome$ обрабатывает запрос на заселение посетителя: производит поиск подходящей комнаты с помощью метода $findRoomForGender$, заселение в подходящую комнату, если таковая есть, с помощью метода $registerRoom$ и ответ посетителю с помощью соответствующего метода подпрограммы `utils`.

2.4.6. Функция $handleLeave$ обрабатывает запрос на выселение из гостиницы: производит поиск комнаты, в которой проживал посетитель, её освобождение и отправку ответа с помощью соответствующего метода подпрограммы `utils`.

2.4.7. Функция $hotel$ является основной функцией отеля. Она производит установку обработчика сигнала $SIGINT$, инициализацию комнат, а также ожидание запросов и вызов соответствующих методов их обработки.

2.5. Подпрограмма `visitor`, реализующая процесс посетителя в соответствии с описанным ранее сценарием.

2.5.1. Функция $stop$ используется для обработки сигнала $SIGINT$ завершения программы и осуществляет завершение работы посетителя с закрытием синхропримитивов.

2.5.2. Функция $come\_to\_hotel$ осуществляет запрос отелю на заселение с помощью соответствующих методов подпрограммы `utils`.

2.5.3. Функция $live$ осуществляет “проживание” посетителя в отеле с помощью метода $sleep\_milliseconds$ подпрограммы `utils`.

2.5.4. Функция $leave\_hotel$ с помощью соответствующих методов подпрограммы `utils` осуществляет запрос на выселение из отеля.

2.5.5. Функция $visitor$ является основной функцией посетителя. Она производит установку обработчика сигнала $SIGINT$, а также “сопровождает” посетителя по сценарию.

2.6. Программа `index.c` реализует родительский процесс, организующий работу отеля и посетителей.

2.6.1. Функция $stop$ используется для обработки сигнала $SIGINT$ завершения программы и осуществляет принудительное завершение работы отеля с удалением синхропримитивов.

2.6.2. Функция $main$ является точкой входа в программу и реализует установку обработчика сигнала $SIGINT$ завершения программы, инициализацию синхропримитивов, создание процесса отеля, а также последовательное считывание данных о посетителях из командной строки и создание посетителей.

При запуске программе в качестве единственного аргумента командной строки должно быть передано имя файла для выходных данных. При вводе посетителей во время работы программы необходимо следовать указаниям программы: сначала должен быть введён пол посетителя одной буквой (m для мужчин и f для женщин), а затем время работы в секундах (от 1 до 15 включительно). Для корректного завершения программы необходимо передать родительскому процессу сигнал завершения $SIGINT$.

ПРОГРАММА 5

Структура программы 4 позволяет с помощью незначительных изменений подпрограммы `utils` получить программу 5:

1. Функция $create\_semaphore$ теперь создаёт неименованные POSIX семафоры с помощью системного вызова sem_init. Семафоры располагаются в разделяемой памяти, также создаваемой функцией $create\_semaphore$ для каждого семафора с помощью метода $create\_memory$.
2. Неименованные POSIX семафоры не требуют закрытия, поэтому функция $close\_semaphore$ лишь закрывает соответствующую разделяемую память (что, на самом деле, тоже является “пустой” функцией, так как не требуется).
3. Функция $delete\_semaphore$ модифицирована для работы с неимнованными POSIX семафорами: помимо удаления семафора системным вызовом $sem\_destroy$ дополнительно происходит удаление соответствующей разделяемой памяти с помощью метода $delete\_memory$.

ПРОГРАММА 6

Программа 6 также не требует значительных модификаций:

1. Функции *create_semaphore*, *wait_semaphore*, *post_semaphore*, *close_semaphore* и *delete_semaphore* были изменены соответствующим образом для работы с семафорами в стандарте Unix System V.
2. Функции *create_memory*, *write_memory*, *read_memory*, *close_memory* и *delete_memory* также были изменены соответствующим образом для работы с разделяемой памятью в стандарте Unix System V.
3. Функции *init_state*, *close_state* и *clear_state*, структура *State* и подпрограмма `log` были откорректированы для поддержки нового интерфейса синхропримитивов.

ПРОГРАММА 7

Программа 7 реализует взаимодействие независимых процессов с помощью именованных POSIX семафоров через разделяемую память в стандарте POSIX, поэтому за основу реализации была взять программа 4, в которой проделаны следующие изменения:

1. Выходные данные программ печатаются в стандартный поток вывода. Для этого подпрограмма `log` была изменена: семафор $log\_semaphore$, указатель файла log_file и функции set_log_file и $stop\_logging$ были удалены, а макро-функция log была изменена для печати в стандартный поток вывода без необходимости синхронизации. Также с учётом нового способа сбора выходных данных сообщения были незначительно изменены.
2. Для организации корректного завершения всех процессов по отправке сигнала $SIGINT$ одному из них в структуру $State$ было добавлено поле $hotel\_pid$ - разделяемая память, содержащая единственное значение - идентификатор процесса отеля, который записывается единственный раз - отелем при запуске, - и считывается посетителем при получении сигнала завершения $SIGINT$.
3. Программа `index.c` была удалена, а в программу `hotel.c` было добавлено создание набора синхропримитивов и установка значения разделяемой памяти, указываемой полем $hotel\_pid$ структуры $State$.
4. Для передачи входных данных программе `visitor` принято решение использовать аргументы командной строки. Таким образом, при запуске программы посетителя требуется с учётом описанных ранее ограничений указать его пол и время пребывания в качестве первого и второго аргументов командной строки соответственно. Для поддержки этого изменения в программу был добавлен алгоритм разбора аргументов командной строки и проверки их корректности.
5. В программу `visitor` также был добавлен вызов функции $init\_state$ для получения информации о синхропримитивах, используемых отелем.
6. В программу `visitor` добавлена обработка сигнала $SIGTERM$, который используется отелем для завершения работы одного посетителя, во избежание отправки лишних сигналов при завершении программы. Таким образом, при получении любым посетителем сигнала $SIGINT$ завершается весь процесс, включая других посетителей и отель, а при получении сигнала $SIGTERM$ происходит завершение лишь того посетителя, который получил сигнал. Также соответствующим образом была модифицирована функция $stop$.
7. Функция $hotel$ программы `hotel.c` и функция $visitor$ программы `visitor.c` были переименованы в $main$ для обеспечения возможности запуска программ как независимых процессов.

Для корректной работы приложения допускается запуск ровно одного процесса отеля, что должно быть сделано до запуска посетителей. Для корректного завершения работы необходимо передать любому процессу сигнал завершения $SIGINT$.

ПРОГРАММА 8

Программа 8 реализует взаимодействие независимых процессов с помощью семафоров в стандарте Unix System V через разделяемую память в стандарте Unix System V. Программа 8 была получена из программы 7 модификаций подпрограммы `utils`:

1. Функции *create_semaphore*, *wait_semaphore*, *post_semaphore*, *close_semaphore* и *delete_semaphore* были изменены соответствующим образом для работы с семафорами в стандарте Unix System V.
2. Функции *create_memory*, *write_memory*, *read_memory*, *close_memory* и *delete_memory* также были изменены соответствующим образом для работы с разделяемой памятью в стандарте Unix System V.
3. Функции *init_state*, *close_state* и *clear_state*, а также структура *State* были откорректированы для поддержки нового интерфейса синхропримитивов.

ПРОГРАММА 9

Программа 9 реализует взаимодействие независимых процессов с помощью POSIX семафоров через очереди сообщений POSIX. Аналогично другим программам, программа 9 была получена путём модификации функций $create\_memory$, $write\_memory$, $read\_memory$, $close\_memory$ и $delete\_memory$ подпрограммы `utils` программы 7 для работы с очередями сообщений вместо разделяемой памяти. Модификаций вне подпрограммы `utils` не производилось.

ПРОГРАММА 10

Программа 10 реализует взаимодействие независимых процессов с помощью семафоров Unix System V через очереди сообщений Unix System V. Аналогично другим программам, программа 10 была получена путём модификации функций *create_memory*, *write_memory*, *read_memory*, *close_memory* и *delete_memory* подпрограммы `utils` программы 8 для работы с очередями сообщений вместо разделяемой памяти. Модификаций вне подпрограммы `utils` не производилось.