ФИО: Бюрчиев Тимур Зольванович
Группа: БПИ 248
Номер варианта: 24
Ссылка на GH

Вариант 24. Задача о программистах

В отделе работают три программиста. Каждый программист пишет свою программу и отдаёт её на проверку одному из двух оставшихся программистов, выбирая его случайно и ожидая окончания проверки.

Программист начинает проверять чужую программу, когда его собственная уже написана и передана на проверку. По завершении проверки программист возвращает программу с результатом (формируемым случайно по любому из выбранных вами законов): программа написана правильно или неправильно.

Программист «спит», если он отправил свою программу и не проверяет чужую программу. Программист «просыпается», когда получает заключение от другого программиста.

Если программа признана правильной, программист пишет другую программу. Если программа признана неправильной, программист исправляет её и отправляет на проверку тому же программисту, который её проверял. К исправлению своей программы он приступает после завершения проверки чужой программы.

При наличии в очереди проверяемых программ и при приходе заключения о неправильности собственной программы программист может выбирать любую из возможных работ. Необходимо учесть, что кто-то из программистов может получать несколько программ на проверку.

Требуется: создать многопроцессное приложение, моделирующее работу программистов.
Каждый программист — это отдельный процесс.

Программа на 4-6 баллов

1. Сценарий моделируемой системы

1.1. Участники и их роли

В отделе работают три программиста. В модели:

- каждый программист - отдельный процесс (ровно 3 дочерних процесса, порождённых одним родителем);
- каждый процесс выполняет полный цикл разработки и проверки программ:
  - пишет свою программу;
  - отправляет её на проверку одному из двух коллег, выбирая проверяющего случайно;
  - ожидает результата проверки;
  - проверяет чужие программы, если они есть в его очереди;
  - исправляет свою программу, если она признана неверной, и пересылает её тому же проверяющему.

Состояния программиста:

- `Пишет` - имитация написания новой программы;
- `Ждёт результата` - собственная программа отправлена, идёт ожидание вердикта;
- `Проверяет чужую` - в его очереди есть программы от коллег, он проводит ревью;
- `Исправляет` - собственная программа получила FAIL, программист вносит правки и снова отправляет её.

Состояние «спит» в модели фактически реализовано как часть состояния «ждёт результата», когда нет ни результатов проверки, ни задач на ревью - процесс просто делает паузы (`usleep`) и периодически проверяет состояние.

1.2. Логика поведения одного программиста

1. Пишет новую программу:
   - имитация работы (задержка ~200 мс);
   - выбирается проверяющий: один из двух остальных программистов с равной вероятностью;
   - формируется задача (`ProgramTask`) и помещается в очередь выбранного проверяющего;
   - переход в состояние `Ждёт результата` .
2. Ждёт результата:
   - пытается неблокирующе прочитать результат из собственного «почтового ящика»;
   - если результат есть:
     - `OK` → увеличивает локальный счётчик успешно принятых программ; увеличивает глобальный счётчик принятых программ; начинает писать новую программу;
     - `FAIL` → переходит в состояние `Исправляет` ;

- если результата ещё нет, но в собственной очереди есть чужие программы → переходит в `Проверяет чужую`;
- если ни результата, ни чужих задач нет → делает короткую задержку («спит») и остаётся в `Ждёт результата`.

3. Проверяет чужую программу:
   - извлекает одну задачу из своей очереди;
   - имитирует ревью (задержка);
   - случайно формирует вердикт:
     - вероятность «программа написана правильно» - 0.7;
     - вероятность «неправильно» - 0.3;
   - отправляет результат автору через его «почтовый ящик»;
   - возвращается в состояние `Ждёт результата`.
4. Исправляет свою программу:
   - имитирует процесс исправления (задержка);
   - увеличивает версию своей программы;
   - отправляет исправленный вариант тому же проверяющему, который проверял предыдущую версию;
   - переходит в `Ждёт результата`.

Таким образом, система бесконечно моделировала бы работу отдела, если бы не введён критерий завершения моделирования (см. ниже).

---

2. Используемые механизмы POSIX

Родительский процесс создаёт трёх дочерних (`fork`), все они разделяют одну область памяти,
созданную через `mmap` с флагом `MAP_SHARED`. В этой области лежат очереди задач и "почтовые
ящики" результатов.

Для синхронизации используются неименованные POSIX-семафоры (`sem_t`), инициализируемые через
`sem_init` с `pshared = 1`, что позволяет использовать их между процессами.

---

3. Завершение программы и обработка сигналов

3.1. Критерий завершения моделирования

Так как в текстовом описании задачи естественный конец работы отдела не указан, в модели введён следующий критерий завершения:

- введён глобальный порог `TOTAL_ACCEPTED_LIMIT` - общее количество программ, принятых с результатом `OK` для всех программистов;
- моделирование продолжается, пока `total_accepted < total_limit`;
- каждое успешное завершение проверки собственной программы (результат `OK`) приводит к вызову:
  - `shared_state_increment_total_accepted(state)`, который:
    - увеличивает `total_accepted` (под защитой `total_mutex`);
    - не даёт счётчику выйти за пределы `total_limit`;
    - возвращает актуальное значение `total_accepted`.
- Каждый программист в начале итерации проверяет, не достигнут ли глобальный лимит. Если достигнут — выходит из основного цикла.

Таким образом, система завершает работу, когда суммарное число принятых программ достигает заданного порога. Значение лимита задаётся в конфигурационном заголовке для блока 4-6, например:

```
#define TOTAL_ACCEPTED_LIMIT 50
```

3.2. Корректное завершение по сигналу прерывания (Ctrl+C)

В родительском процессе установлен обработчик `SIGINT`, который выставляет флаг `g_stop`. Дочерние процессы периодически проверяют этот флаг и корректно завершаются.

То есть при нажатии `Ctrl+C`:

1. глобальный флаг `g_stop` устанавливается в 1;
2. все программисты при следующей итерации цикла выходят;
3. родитель дожидается завершения всех дочерних процессов и освобождает ресурсы.

---

4. Очистка ресурсов

После завершения моделирования (по глобальному лимиту или по `SIGINT`) родительский процесс:

1. Ожидает завершения всех дочерних процессов (`waitpid`).
2. Вызывает `shared_state_destroy(state)`, где:

- для каждой очереди задач уничтожаются семафоры `mutex`, `items`, `spaces`;
- для каждого почтового ящика - семафоры `mutex`, `ready`;
- уничтожается `total_mutex`.

3. Освобождает разделяемую память вызовом `munmap`.

Дочерние процессы при выходе не выполняют явной очистки семафоров - это корректно, так как владельцем структур является родитель, который инициализировал и уничтожает их после завершения всех дочерних процессов.

---

5. Способы запуска

5.1. Сборка проекта

Из корня проекта `os-ihw`:

```
cmake -S . -B cmake-build-debug
cmake --build cmake-build-debug
```

После этого исполняемый файл для блока 4-6 располагается в подкаталоге `cmake-build-debug/4-6`:

```
cd cmake-build-debug/4-6
ls
# ... os_4_6 ...
```

5.2. Запуск программы и сохранение логов

Для запуска первой программы:

```
cd /path/os-ihw/cmake-build-debug/4-6

./os_4_6
```

Все сообщения процессов (родитель + три программиста) выводятся в стандартный поток вывода.

---

6. Формат и пример результатов работы

6.1. Формат логов

Каждая строка лога имеет вид:

```
[<время>][<идентификатор>] <сообщение>
```

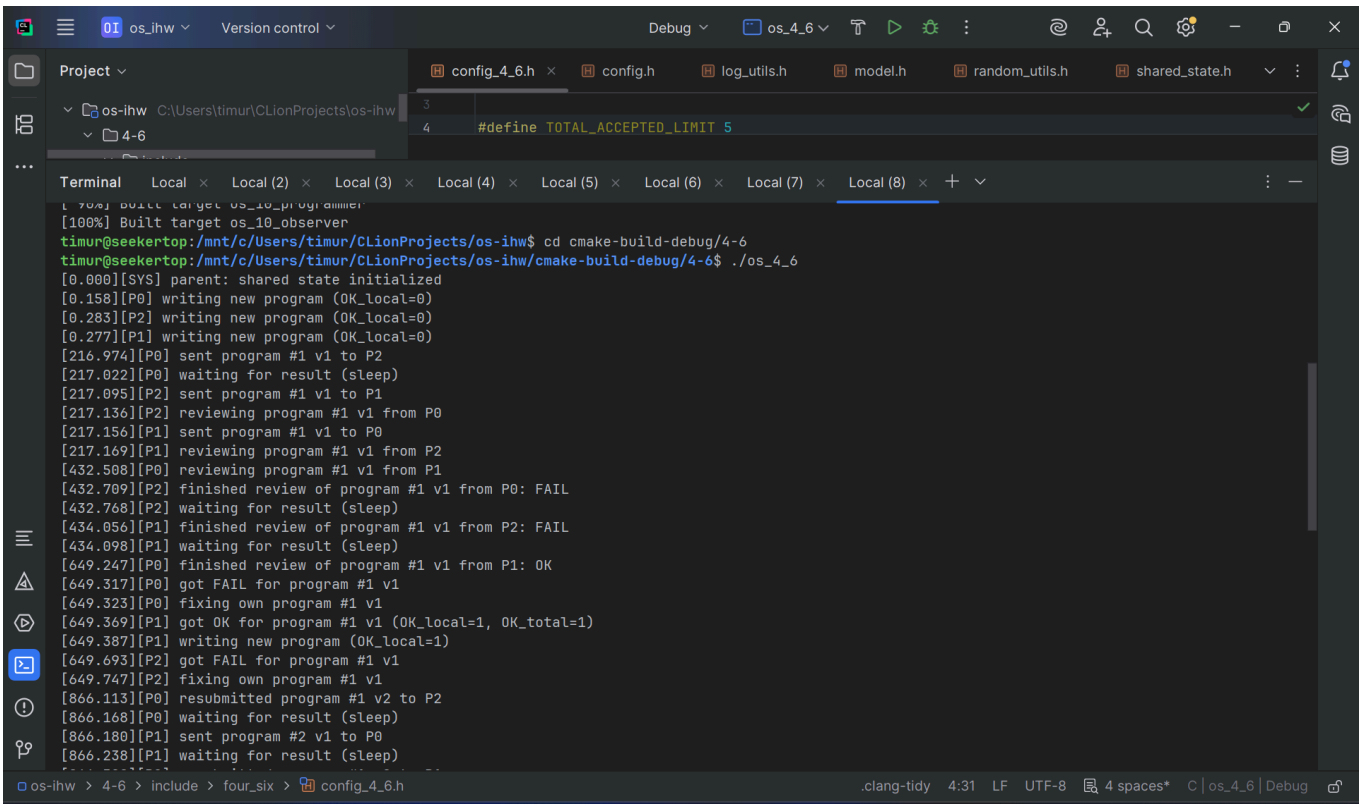
- `<время>` - метка времени в секундах с точностью до миллисекунд (реализация через `gettimeofday`);
- `<идентификатор>`:
  - `SYS` - сообщения родительского процесса (инициализация, завершение);
  - `P0`, `P1`, `P2` - сообщения процессов-программистов;
- `<сообщение>` - текстовое описание события.

Примеры фрагмента лога для небольшого лимита `TOTAL_ACCEPTED_LIMIT = 5`:

```
[0.000][SYS] parent: shared state initialized
[0.158][P0] writing new program (OK_local=0)
[0.283][P2] writing new program (OK_local=0)
[0.277][P1] writing new program (OK_local=0)
[216.974][P0] sent program #1 v1 to P2
[217.022][P0] waiting for result (sleep)
[217.095][P2] sent program #1 v1 to P1
[217.136][P2] reviewing program #1 v1 from P0
[217.156][P1] sent program #1 v1 to P0
[217.169][P1] reviewing program #1 v1 from P2
[432.508][P0] reviewing program #1 v1 from P1
[432.709][P2] finished review of program #1 v1 from P0: FAIL
[432.768][P2] waiting for result (sleep)
...
[1949.580][P2] got OK for program #1 v3 (OK_local=1, OK_total=5)
[1949.635][P2] global limit reached (TOTAL_ACCEPTED_LIMIT=5), local OK=1,
exiting
[1949.639][P2] stopping, local OK=1
[2152.137][P0] sent program #3 v1 to P2
[2152.204][P0] global limit reached (TOTAL_ACCEPTED_LIMIT=5), local OK=2,
exiting
[2152.208][P0] stopping, local OK=2
[2165.934][P1] sent program #3 v1 to P0
[2165.969][P1] global limit reached (TOTAL_ACCEPTED_LIMIT=5), local OK=2,
exiting
[2165.971][P1] stopping, local OK=2
[2166.084][SYS] parent: all children finished, total accepted = 5
```

Из лога видно, что:

- каждый программист пишет программы, отправляет их случайным коллегам, получает результаты;
- часть программ получает `OK` с вероятностью 0.7, часть — `FAIL` и затем исправляется;
- глобальный счётчик `OK_total` растёт при каждом успешном принятии собственной программы;
- когда `OK_total` достигает `TOTAL_ACCEPTED_LIMIT`, все процессы корректно завершают работу.



Программа на 7-8 баллов

Отличие второй программы от первой:

1. Процессы запускаются независимо: каждый программист - это отдельный процесс, запускаемый в своей консоли.
2. Для синхронизации используются именованные POSIX-семафоры, общие для всех процессов.
3. Разделяемое состояние хранится в POSIX shared memory (`shm_open`), доступной всем запущенным процессам-программистам.

Код второй программы находится в каталоге `7-8/` и собирает два исполняемых файла: `os_7_8_init` и `os_7_8_programmer`. Общие структуры вынесены в `common/`.

Во второй программе используется та же структура общего состояния (очереди задач, почтовые ящики, глобальный счётчик), что и в первой. Отличие только в том, что она размещается в объекте POSIX shared memory, созданном через `shm_open` и отображаемом в память всех процессов.

---

1. Именованные POSIX-семафоры

Для взаимодействия независимых процессов используются именованные семафоры (`sem_open`, `sem_close`, `sem_unlink`) со следующей логикой:

1. Очереди задач (`ReviewQueue7_8`) для каждого программиста `i`:
   - `queue_mutex[i]` - двоичный семафор-мьютекс для операций с очередью;
   - `queue_items[i]` - счётчик количества элементов в очереди;
   - `queue_spaces[i]` - счётчик свободных мест (по схеме producer/consumer).
2. Почтовый ящик результатов (`ResultMailbox7_8`):
   - `mb_mutex[i]` - мьютекс для записи/чтения ящика;
   - `mb_ready[i]` - семафор «в ящике есть новый результат».
3. Глобальный счётчик принятых программ:
   - `total_mutex` - мьютекс для атомарного чтения/изменения `total_accepted` и проверки порогового значения `total_limit`.

Имена семафоров формируются по префиксам (задаются в `config_7_8.h`), например:

```
"/os_ihw_7_8_q_mutex_0"
"/os_ihw_7_8_q_items_1"
"/os_ihw_7_8_mb_ready_2"
"/os_ihw_7_8_total_mutex"
```

Создание и открытие всех семафоров скрыто внутри функций `ipc_7_8.c`:

- при инициализации (`os_7_8_init`) - создание (`O_CREAT | O_EXCL`) и установка начальных значений (1 / 0 / `QUEUE_CAPACITY`);
- при подключении программистов - простое открытие существующих семафоров.

---

2. Процессы и их роли

Во второй программе используются три типа процессов:

1. Инициализатор IPC (`os_7_8_init`):
   - удаляет (на всякий случай) предыдущие объекты IPC: `shm_unlink + sem_unlink` ;
   - создаёт новый объект shared memory и инициализирует `SharedState7_8` ;
   - создаёт и инициализирует все именованные семафоры;
   - после успешной инициализации завершает работу.
2. Процессы-программисты (`os_7_8_programmer`):
   - запускаются независимо в отдельных консолях;
   - при старте подключаются к уже созданной shared memory и семафорам;
   - выполняют конечный автомат программиста (пишут, отправляют, проверяют, исправляют, спят);
   - логируют свои действия в свою консоль;
   - завершаются при достижении глобального лимита или по сигналу `SIGINT` .
3. Сервис очистки - логика удаления объектов IPC реализована внутри инициализатора: повторный запуск `os_7_8_init` перед новым экспериментом полностью очищает и создаёт ресурсы заново.

---

3. Завершение работы и очистка ресурсов

Критерий логического завершения (глобальный лимит успешных программ) совпадает с первой программой; отличие только в том, что счётчик хранится в `POSIX shared memory` , а доступ к нему синхронизируется именованным семафором `total_mutex` .

Обработка `SIGINT` реализована так же, как в первой программе.

3.1. Очистка объектов IPC между экспериментами

Чтобы запустить модель с нуля после предыдущего эксперимента, используется инициализатор `os_7_8_init` . Он:

1. Сначала пытается удалить старые объекты IPC (`ipc7_8_unlink_all`):
   - `shm_unlink(SHM_NAME_7_8)` ;
   - `sem_unlink` для всех семафоров очередей, почтовых ящиков и глобального мьютекса.
2. Затем создаёт новый объект shared memory и заново инициализирует все именованные семафоры.

---

4. Способ запуска

4.1. Сборка

Из корня проекта `os-ihw`:

```
cd /mnt/c/Users/timur/CLionProjects/os-ihw
cmake -S . -B cmake-build-debug
cmake --build cmake-build-debug
```

4.2. Полный цикл эксперимента

1. Инициализация IPC:

   ```
   cd /mnt/path/os-ihw/cmake-build-debug/7-8
   ./os_7_8_init
   ```

2. Запуск процессов-программистов в трёх консолях:

   В трёх отдельных окнах терминала:

   ```
   cd /mnt/path/os-ihw/cmake-build-debug/7-8
   ./os_7_8_programmer 0
   ./os_7_8_programmer 1
   ./os_7_8_programmer 2
   ```

   Каждый процесс будет логировать свою работу в собственную консоль.

3. Завершение эксперимента:
   - процессы автоматически завершатся при достижении глобального лимита `TOTAL_ACCEPTED_LIMIT_7_8` (по умолчанию 50 успешных программ);
   - либо могут быть прерваны по `Ctrl+C` в соответствующей консоли.

4. Перезапуск с чистого состояния:

   Для нового эксперимента достаточно снова выполнить:

   ```
   ./os_7_8_init
   ```

---

5. Демонстрация результата

Типичный фрагмент лога для одного из процессов-программистов:

```
[65848.194][P0] resubmitted program #14 v2 to P1
[65848.261][P0] waiting for result (sleep)
[66065.061][P0] waiting for result (sleep)
[66281.895][P0] waiting for result (sleep)
[66498.750][P0] got OK for program #14 v2 (OK_local=14, OK_total=44)
[66498.794][P0] writing new program (OK_local=14)
[66715.611][P0] sent program #15 v1 to P2
[66715.659][P0] waiting for result (sleep)
[66931.276][P0] waiting for result (sleep)
[67148.171][P0] reviewing program #17 v1 from P2
[67365.014][P0] finished review of program #17 v1 from P2: OK
[67365.088][P0] got FAIL for program #15 v1
[67365.092][P0] fixing own program #15 v1
[67576.730][P0] resubmitted program #15 v2 to P2
[67576.776][P0] waiting for result (sleep)
[67776.925][P0] waiting for result (sleep)
[67977.115][P0] got OK for program #15 v2 (OK_local=15, OK_total=49)
[67977.152][P0] writing new program (OK_local=15)
[68184.548][P0] sent program #16 v1 to P1
[68184.603][P0] global limit reached (TOTAL_ACCEPTED_LIMIT_7_8=50), local
OK=15, exiting
[68184.606][P0] stopping (7-8), local OK=15
```

По этим логам можно:

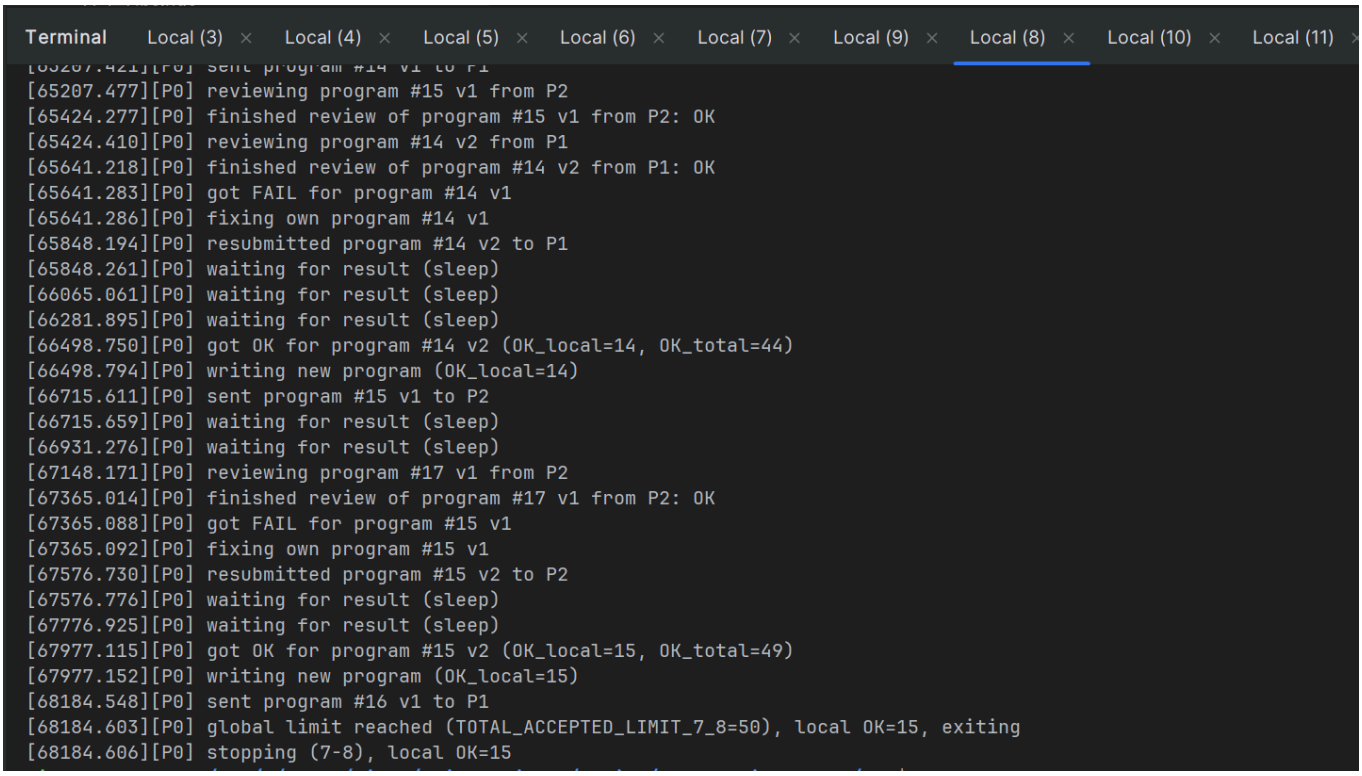
- проследить передачу программ между программистами (кто кому отправляет задачи);
- увидеть, как формируются результаты `OK` / `FAIL` и как происходит исправление;
- проверить достижение глобального лимита и корректное завершение работы каждого процесса.

Terminal Local (3) × Local (4) × Local (5) × Local (6) × Local (7) × Local (9) × Local (8) × Local (10) × Local (11) ×

```
[39856.751][P1] waiting for result (sleep)
[40073.562][P1] got OK for program #14 v2 (OK_local=14, OK_total=42)
[40073.625][P1] writing new program (OK_local=14)
[40290.403][P1] sent program #15 v1 to P2
[40290.437][P1] reviewing program #16 v1 from P2
[40507.255][P1] finished review of program #16 v1 from P2: FAIL
[40507.316][P1] reviewing program #14 v2 from P0
[40724.150][P1] finished review of program #14 v2 from P0: OK
[40724.220][P1] got OK for program #15 v1 (OK_local=15, OK_total=43)
[40724.228][P1] writing new program (OK_local=15)
[40940.139][P1] sent program #16 v1 to P2
[40940.178][P1] reviewing program #16 v2 from P2
[41156.964][P1] finished review of program #16 v2 from P2: OK
[41157.015][P1] waiting for result (sleep)
[41373.827][P1] got OK for program #16 v1 (OK_local=16, OK_total=46)
[41373.875][P1] writing new program (OK_local=16)
[41590.664][P1] sent program #17 v1 to P2
[41590.715][P1] waiting for result (sleep)
[41807.517][P1] waiting for result (sleep)
[42016.446][P1] got OK for program #17 v1 (OK_local=17, OK_total=48)
[42016.473][P1] writing new program (OK_local=17)
[42216.613][P1] sent program #18 v1 to P0
[42216.662][P1] reviewing program #18 v1 from P2
[42416.760][P1] finished review of program #18 v1 from P2: OK
[42416.795][P1] waiting for result (sleep)
[42633.378][P1] global limit reached (TOTAL_ACCEPTED_LIMIT_7_8=50), local OK=17, exiting
[42633.419][P1] stopping (7-8), local OK=17
```

Terminal Local (3) × Local (4) × Local (5) × Local (6) × Local (7) × Local (9) × Local (8) × Local (10) × Local (11)

```
[13373.411][P2] sent program #16 v1 to P1
[13373.450][P2] waiting for result (sleep)
[13590.253][P2] reviewing program #15 v1 from P1
[13807.091][P2] finished review of program #15 v1 from P1: OK
[13807.148][P2] got FAIL for program #16 v1
[13807.152][P2] fixing own program #16 v1
[14022.551][P2] resubmitted program #16 v2 to P1
[14022.583][P2] waiting for result (sleep)
[14239.394][P2] reviewing program #16 v1 from P1
[14456.224][P2] finished review of program #16 v1 from P1: OK
[14456.280][P2] got OK for program #16 v2 (OK_local=16, OK_total=45)
[14456.283][P2] writing new program (OK_local=16)
[14673.097][P2] sent program #17 v1 to P0
[14673.136][P2] reviewing program #15 v1 from P0
[14889.908][P2] finished review of program #15 v1 from P0: FAIL
[14889.947][P2] reviewing program #17 v1 from P1
[15105.621][P2] finished review of program #17 v1 from P1: OK
[15105.661][P2] got OK for program #17 v1 (OK_local=17, OK_total=47)
[15105.663][P2] writing new program (OK_local=17)
[15314.394][P2] sent program #18 v1 to P1
[15314.427][P2] reviewing program #15 v2 from P0
[15514.527][P2] finished review of program #15 v2 from P0: OK
[15514.560][P2] waiting for result (sleep)
[15716.055][P2] got OK for program #18 v1 (OK_local=18, OK_total=50)
[15716.096][P2] global limit reached (TOTAL_ACCEPTED_LIMIT_7_8=50), local OK=18, exiting
[15716.097][P2] stopping (7-8), local OK=18
[15716.098][P2] P: exiting (7-8)
```



Программа на 9 баллов

Программа на 9 баллов расширяет модель 7-8 за счёт добавления отдельного процесса-наблюдателя, который:

- запускается в своей консоли независимо от программистов;
- получает все события от программистов через именованный канал (FIFO);
- интегрирует их и отображает в одном потоке, позволяя наблюдать поведение всей системы сверху.

---

1. Архитектура решения

1.1. Задействованные процессы

В рамках решения на 9 баллов используются следующие процессы:

1. Инициализатор IPC (`os_7_8_init`)
2. Программисты (`os_9_programmer`)
   - реализуют ту же логики, что и в решении на 7-8 баллов:
   - каждый процесс выводит лог в свою консоль и дополнительно отправляет копию логов наблюдателю.
3. Наблюдатель (`os_9_observer`)
   - создаёт именованный канал (FIFO) и открывает его на чтение;

- читает все строки, отправленные программистами, и выводит их в свою консоль с префиксом `[OBS]`;
- завершается при получении сигнала `SIGINT` или при закрытии всех писателей (EOF на FIFO).

1.2. Канал связи с наблюдателем

Для передачи информации наблюдателю используется именованный канал (FIFO):

```
#define OBSERVER_FIFO_PATH_9 "/tmp/os_ihw_9_observer_fifo"
```

Механика:

- Наблюдатель при старте:
  - вызывает `mkfifo(OBSERVER_FIFO_PATH_9, 0666)` (если канал уже есть - ошибка `EEXIST` игнорируется);
  - открывает FIFO на чтение (`open(..., O_RDONLY)`), при необходимости создаёт фиктивного писателя, чтобы `read` не возвращал EOF, пока программисты не запущены.
- Программисты (`os_9_programmer`):
  - при старте пробуют открыть тот же FIFO на запись: `open(OBSERVER_FIFO_PATH_9, O_WRONLY | O_NONBLOCK);`
  - если открыть получилось, в `log_utils` вызывается `log_set_observer_fd(fd)`;
  - после этого каждая строка лога, сформированная `log_event`, дублируется:
    - в stdout процесса-программиста;
    - в FIFO, откуда её читает наблюдатель.

Если наблюдатель не запущен или FIFO недоступен, программист пишет соответствующее сообщение и продолжает работу без внешнего наблюдения (лог остаётся только в своей консоли).

---

2. Поведение наблюдателя

Основной цикл наблюдателя:

1. Выводит сообщение о старте:
   ```
   [0.000][SYS] observer (9) started, listening on /tmp/os_ihw_9_observer_fifo
   ```
2. В цикле читает данные из FIFO:

- если прочитано `n > 0` байт - дописывает `'\0'` и выводит строку с префиксом `[OBS]`:

  ```
  [OBS] [208.715][P0] sent program #1 v1 to P2
  ```

- если `read` вернул `0` (EOF) - все писатели закрыты, наблюдатель аккуратно завершает работу;
- при `SIGINT` (`Ctrl+C` в консоли наблюдателя) выставляется флаг `g_stop`, и цикл также корректно завершается.

---

3. Способ запуска

Ниже приведён рекомендуемый порядок запуска в отдельных терминалах WSL.

Терминал 1 - сборка и инициализация IPC

```
cd /mnt/path/os-ihw
cmake -S . -B cmake-build-debug
cmake --build cmake-build-debug

cd cmake-build-debug/7-8
./os_7_8_init
```

Терминал 2 - наблюдатель

```
cd /mnt/path/os-ihw/cmake-build-debug/9
./os_9_observer
```

Терминал 3 - программист P0

```
cd /mnt/path/os-ihw/cmake-build-debug/9
./os_9_programmer 0
```

Терминал 4 - программист P1

```
cd /mnt/path/os-ihw/cmake-build-debug/9
./os_9_programmer 1
```

Терминал 5 - программист P2

```
cd /mnt/path/os-ihw/cmake-build-debug/9
./os_9_programmer 2
```

---

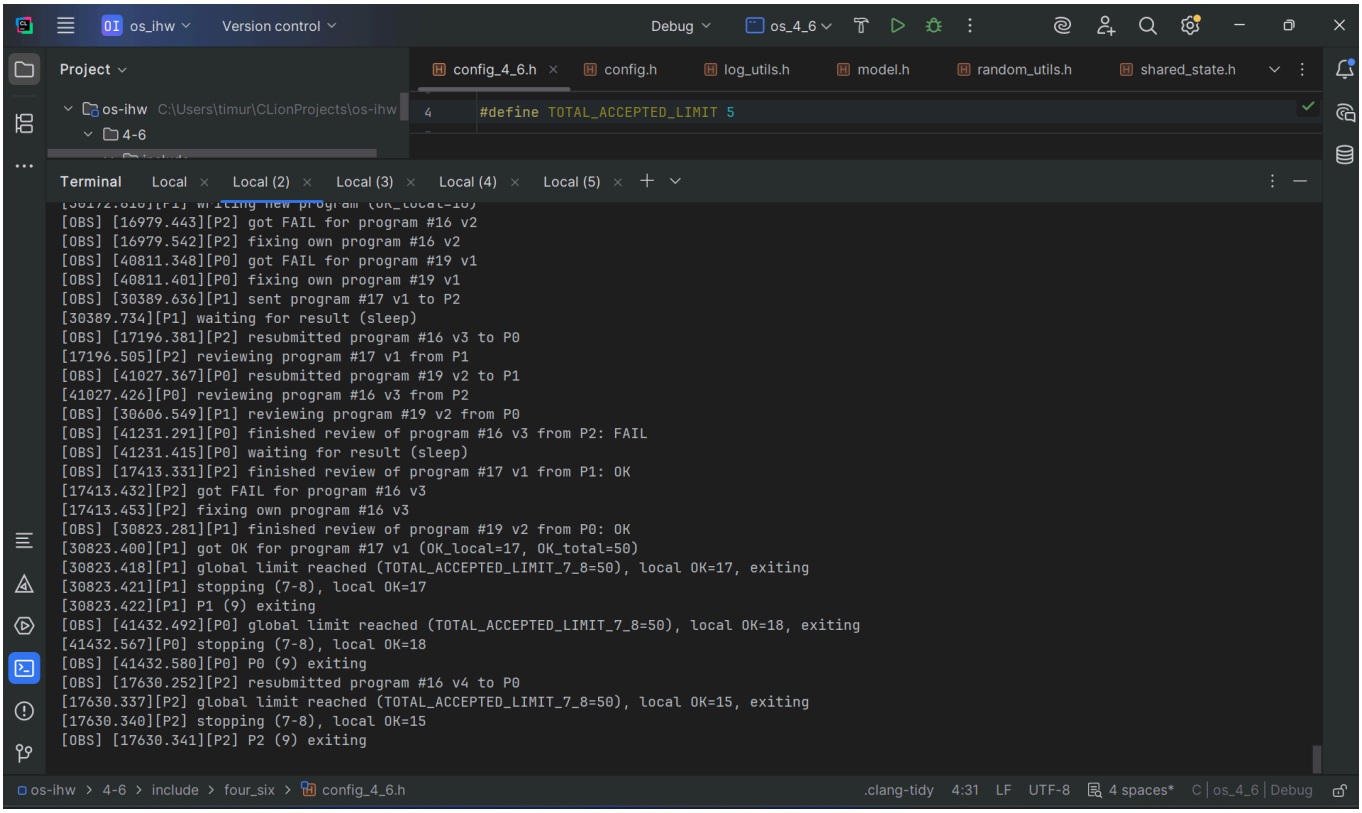
4. Пример результатов работы программы (наблюдатель)

Ниже приведён фрагмент лога из консоли наблюдателя:

```
[0.000][SYS] observer (9) started, listening on /tmp/os_ihw_9_observer_fifo
[OBS] [0.000][P0] P0 (9) started
[0.131][P0] writing new program (OK_local=0)
[OBS] [203.542][P0] sent program #1 v1 to P1
...
[OBS] [41432.492][P0] global limit reached (TOTAL_ACCEPTED_LIMIT_7_8=50),
local OK=18, exiting
[41432.567][P0] stopping (7-8), local OK=18
[OBS] [41432.580][P0] P0 (9) exiting
[OBS] [17630.252][P2] resubmitted program #16 v4 to P0
[17630.337][P2] global limit reached (TOTAL_ACCEPTED_LIMIT_7_8=50), local
OK=15, exiting
[17630.340][P2] stopping (7-8), local OK=15
[OBS] [17630.341][P2] P2 (9) exiting
```

Из этого лога видно, что:

- наблюдатель получает события от всех трёх программистов (`P0` , `P1` , `P2`);
- глобальный счётчик принятых программ `OK_total` достигает значения `50` ;
- каждый программист корректно завершает работу после достижения глобального лимита;
- локальные результаты распределились, например:
  - P0: `local OK = 15` ;
  - P1: `local OK = 19` ;
  - P2: `local OK = 16` .



Программа на 10 баллов

Программа на 10 баллов расширяет эту идею 9:

- теперь можно запустить несколько наблюдателей, каждый - в своей консоли;
- все наблюдатели получают один и тот же поток событий от программистов;
- каждый наблюдатель ведёт свой независимый вывод, но информация у них согласованная.

---

1. Архитектура решения

1.1. Задействованные процессы

1. Инициализатор IPC (`os_7_8_init`)
2. Программисты (`os_10_programmer`)
   - логика работы программиста полностью наследует решение 7-8
   - каждый программист:
     - пишет лог в свою консоль (stdout процесса);
     - дублирует те же строки логов во все подключённые наблюдатели.
3. Наблюдатели (`os_10_observer`)
   - каждый наблюдатель запускается с собственным `observer_id` :

        - `os_10_observer 0`
        - `os_10_observer 1`
        - … (до заданного `MAX_OBSERVERS_10`);
    - каждый наблюдатель:
        - создаёт свой именованный канал FIFO;
        - читает из него все события, которые программисты ему отправляют;
        - выводит их в свою консоль с префиксом, различающим наблюдателей (например, `[OBS#0]`, `[OBS#1]`).

Таким образом, программисты не знают точного числа наблюдателей и их текущего состояния: они просто пытаются подключиться к всем потенциальным FIFO и рассылают события во все открытые каналы.

1.2. Каналы связи с несколькими наблюдателями

Для каждого наблюдателя используется свой именованный канал (FIFO).

- наблюдатель с `observer_id = 0`:
    - `/tmp/os_ihw_10_observer_fifo_0`
- наблюдатель с `observer_id = 1`:
    - `/tmp/os_ihw_10_observer_fifo_1`
- и так далее, до `MAX_OBSERVERS_10`.

1.3. Расширение модуля логирования

Базовый модуль логирования `log_utils` был расширен для поддержки нескольких наблюдателей:

- `log_set_observer_fd(int fd)` — сбрасывает список и устанавливает один наблюдатель;
- `log_add_observer_fd(int fd)` — добавляет ещё один дескриптор наблюдателя (чтобы подключать нескольких наблюдателей одновременно);
- `log_clear_observers()` — очищает список наблюдателей в текущем процессе.

1.4. Завершение наблюдателей

Каждый наблюдатель завершает работу, когда:

1. Пользователь посылает ему `SIGINT`
2. Все программисты закрыли свои дескрипторы FIFO, и `read` возвращает `0` (EOF):
    - наблюдатель логирует событие вида:

```
observer #<id> (10): EOF on fifo, exiting
```

- и завершает выполнение.

Наблюдатели работают одинаково и независимо друг от друга: каждый читает свой FIFO, но, так как программисты рассылают строки во все открытые FIFO, содержимое логов у наблюдателей синхронно.

---

2. Способ запуска

Ниже приведён рекомендуемый порядок запуска в отдельных терминалах WSL.

Терминал 1 - сборка и инициализация IPC

```
cd /mnt/path/os-ihw
cmake -S . -B cmake-build-debug
cmake --build cmake-build-debug

cd cmake-build-debug/7-8
./os_7_8_init
```

Терминал 2 - наблюдатель 0

```
cd /mnt/path/os-ihw/cmake-build-debug/10
./os_10_observer 0
```

Терминал 3 - наблюдатель 1

```
cd /mnt/path/os-ihw/cmake-build-debug/10
./os_10_observer 1
```

Терминал 4 - программист P0

```
cd /mnt/path/os-ihw/cmake-build-debug/10
./os_10_programmer 0
```

Терминал 5 - программист P1

```
cd /mnt/path/os-ihw/cmake-build-debug/10
```

```
./os_10_programmer 1
```

Терминал 6 - программист P2

```
cd /mnt/path/os-ihw/cmake-build-debug/10
./os_10_programmer 2
```

Каждый программист логирует события как в свою консоль, так и во все доступные FIFO наблюдателей.

Оба наблюдателя (`0` и `1`) в своих консолях видят одинаковый интегрированный лог всей системы.

---

3. Пример результатов работы программы (несколько наблюдателей)

Ниже приведён схематичный фрагмент из двух консолей наблюдателей. Для наглядности используется префикс `[OBS#id]` :

Наблюдатель 0

```
[0.000][SYS] observer #0 (10) started, listening on
/tmp/os_ihw_10_observer_fifo_0
[OBS#0] [0.000][P0] P0 (10) started
[OBS#0] [0.012][P1] P1 (10) started
[OBS#0] [0.018][P2] P2 (10) started
[OBS#0] [0.120][P0] writing new program (OK_local=0)
[OBS#0] [0.135][P1] writing new program (OK_local=0)
[OBS#0] [0.148][P2] writing new program (OK_local=0)
...
[OBS#0] [15567.782][P2] got OK for program #16 v1 (OK_local=16, OK_total=50)
[OBS#0] [15567.857][P2] global limit reached (TOTAL_ACCEPTED_LIMIT_7_8=50),
local OK=16, exiting
[OBS#0] [52277.019][P0] stopping (7-8), local OK=15
[OBS#0] [31036.216][P1] P1 (10) exiting
[OBS#0] observer #0 (10): EOF on fifo, exiting
```

Наблюдатель 1

```
[0.000][SYS] observer #1 (10) started, listening on its fifo
[OBS#1] [0.000][P0] P0 (10) started
```

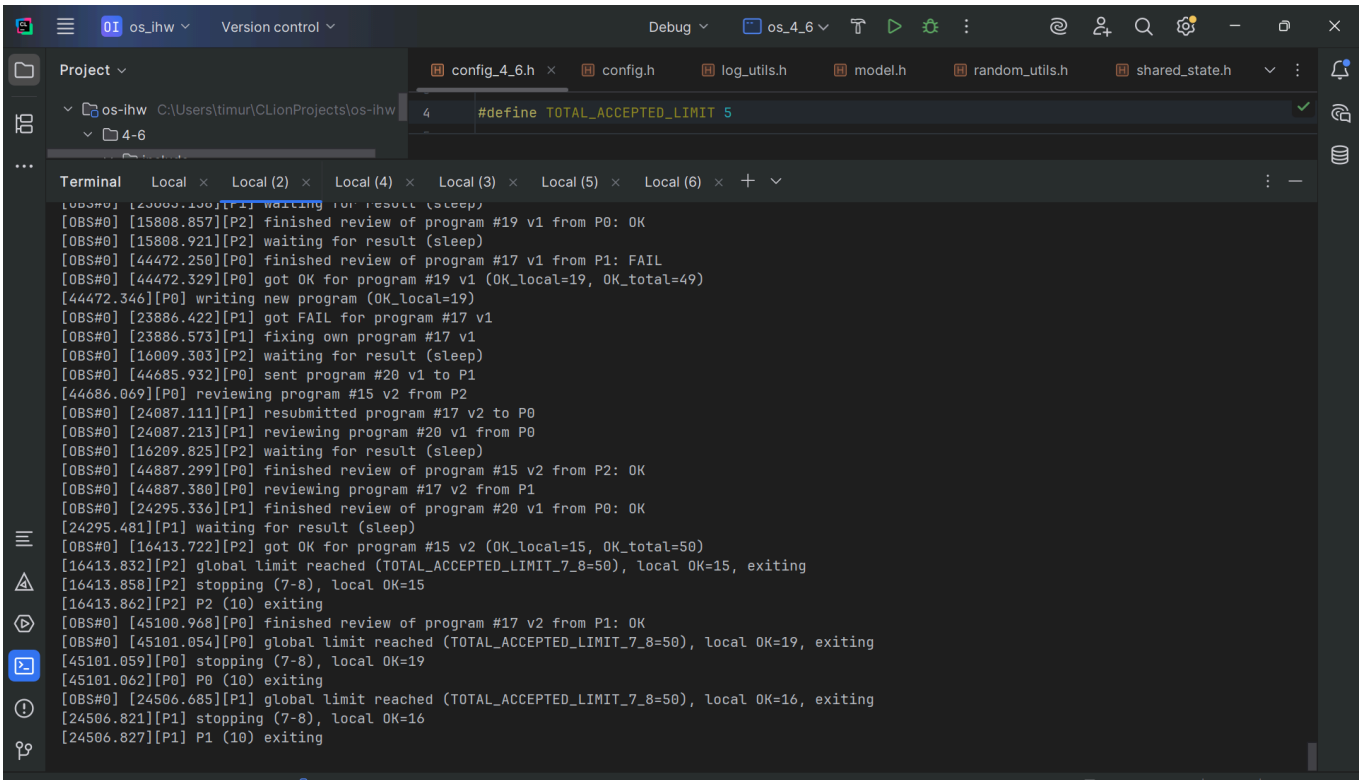
```
[0.048][P0] writing new program (OK_local=0)
[OBS#1] [213.682][P0] sent program #1 v1 to P2
[213.800][P0] waiting for result (sleep)
[OBS#1] [430.634][P0] waiting for result (sleep)
...
[OBS#1] [24295.336][P1] finished review of program #20 v1 from P0: OK
[24295.481][P1] waiting for result (sleep)
[OBS#1] [16413.722][P2] got OK for program #15 v2 (OK_local=15, OK_total=50)
[16413.832][P2] global limit reached (TOTAL_ACCEPTED_LIMIT_7_8=50), local
OK=15, exiting
[16413.858][P2] stopping (7-8), local OK=15
[16413.862][P2] P2 (10) exiting
[OBS#1] [45100.968][P0] finished review of program #17 v2 from P1: OK
[45101.054][P0] global limit reached (TOTAL_ACCEPTED_LIMIT_7_8=50), local
OK=19, exiting
[45101.059][P0] stopping (7-8), local OK=19
[45101.062][P0] P0 (10) exiting
[OBS#1] [24506.685][P1] global limit reached (TOTAL_ACCEPTED_LIMIT_7_8=50),
local OK=16, exiting
[24506.821][P1] stopping (7-8), local OK=16
[24506.827][P1] P1 (10) exiting
```

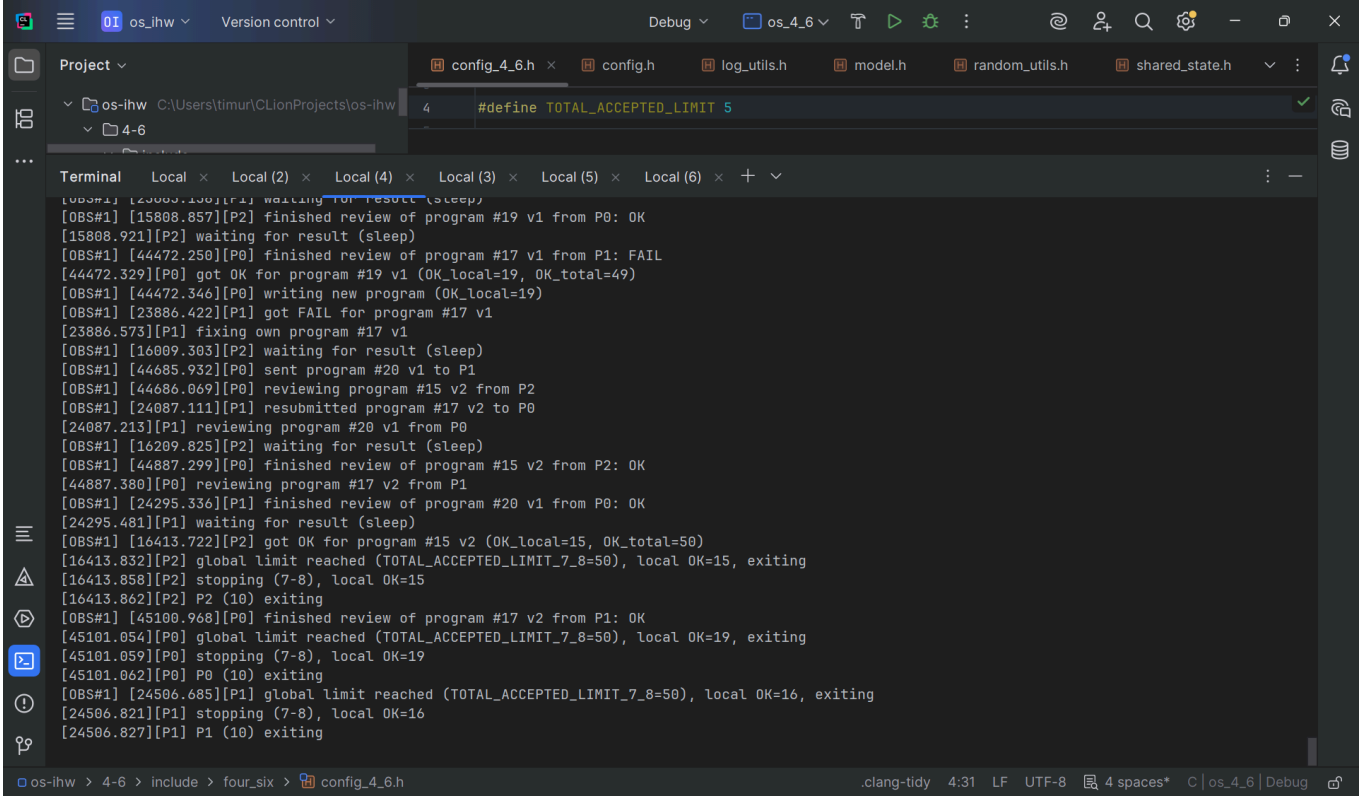
Из этих фрагментов видно, что:

- оба наблюдателя получают одни и те же события от программистов;
- порядок сообщений и значения (id программистов, номера программ, значения `OK_local` и `OK_total`) совпадают;
- все программисты корректно завершают работу при достижении глобального лимита, после чего наблюдатели получают EOF по своим FIFO и также завершаются.

Таким образом, программа на 10 баллов:

- использует те же подходы к завершению, что и предыдущие решения;
- допускает подключение нескольких наблюдателей;
- обеспечивает одинаковую и согласованную работу всех наблюдателей, каждый в своей консоли, с интегрированным представлением работы всех программистов.


os_ihw
Version control
Debug
os_4_6
Project
os-ihw C:\Users\timur\CLionProjects\os-ihw
4-6
config_4_6.h
config.h
log_utils.h
model.h
random_utils.h
shared_state.h
4 #define TOTAL_ACCEPTED_LIMIT 5
Terminal
Local
Local (2)
Local (4)
Local (3)
Local (5)
Local (6)
[OBS#0] [15808.857][P2] finished review of program #19 v1 from P0: OK
[OBS#0] [15808.921][P2] waiting for result (sleep)
[OBS#0] [44472.250][P0] finished review of program #17 v1 from P1: FAIL
[OBS#0] [44472.329][P0] got OK for program #19 v1 (OK_local=19, OK_total=49)
[44472.346][P0] writing new program (OK_local=19)
[OBS#0] [23886.422][P1] got FAIL for program #17 v1
[OBS#0] [23886.573][P1] fixing own program #17 v1
[OBS#0] [16009.303][P2] waiting for result (sleep)
[OBS#0] [44685.932][P0] sent program #20 v1 to P1
[44686.069][P0] reviewing program #15 v2 from P2
[OBS#0] [24087.111][P1] resubmitted program #17 v2 to P0
[OBS#0] [24087.213][P1] reviewing program #20 v1 from P0
[OBS#0] [16209.825][P2] waiting for result (sleep)
[OBS#0] [44887.299][P0] finished review of program #15 v2 from P2: OK
[OBS#0] [44887.380][P0] reviewing program #17 v2 from P1
[OBS#0] [24295.336][P1] finished review of program #20 v1 from P0: OK
[24295.481][P1] waiting for result (sleep)
[OBS#0] [16413.722][P2] got OK for program #15 v2 (OK_local=15, OK_total=50)
[16413.832][P2] global limit reached (TOTAL_ACCEPTED_LIMIT_7_8=50), local OK=15, exiting
[16413.858][P2] stopping (7-8), local OK=15
[16413.862][P2] P2 (10) exiting
[OBS#0] [45100.968][P0] finished review of program #17 v2 from P1: OK
[OBS#0] [45101.054][P0] global limit reached (TOTAL_ACCEPTED_LIMIT_7_8=50), local OK=19, exiting
[45101.059][P0] stopping (7-8), local OK=19
[45101.062][P0] P0 (10) exiting
[OBS#0] [24506.685][P1] global limit reached (TOTAL_ACCEPTED_LIMIT_7_8=50), local OK=16, exiting
[24506.821][P1] stopping (7-8), local OK=16
[24506.827][P1] P1 (10) exiting
os-ihw > 4-6 > include > four_six > config_4_6.h
.clang-tidy 4:31 LF UTF-8 4 spaces* C | os_4_6 | Debug


os_ihw
Version control
Debug
os_4_6
Project
os-ihw C:\Users\timur\CLionProjects\os-ihw
4-6
config_4_6.h
config.h
log_utils.h
model.h
random_utils.h
shared_state.h
4 #define TOTAL_ACCEPTED_LIMIT 5
Terminal
Local
Local (2)
Local (4)
Local (3)
Local (5)
Local (6)
[OBS#1] [15808.857][P2] finished review of program #19 v1 from P0: OK
[15808.921][P2] waiting for result (sleep)
[OBS#1] [44472.250][P0] finished review of program #17 v1 from P1: FAIL
[44472.329][P0] got OK for program #19 v1 (OK_local=19, OK_total=49)
[OBS#1] [44472.346][P0] writing new program (OK_local=19)
[OBS#1] [23886.422][P1] got FAIL for program #17 v1
[23886.573][P1] fixing own program #17 v1
[OBS#1] [16009.303][P2] waiting for result (sleep)
[OBS#1] [44685.932][P0] sent program #20 v1 to P1
[OBS#1] [44686.069][P0] reviewing program #15 v2 from P2
[OBS#1] [24087.111][P1] resubmitted program #17 v2 to P0
[24087.213][P1] reviewing program #20 v1 from P0
[OBS#1] [16209.825][P2] waiting for result (sleep)
[OBS#1] [44887.299][P0] finished review of program #15 v2 from P2: OK
[44887.380][P0] reviewing program #17 v2 from P1
[OBS#1] [24295.336][P1] finished review of program #20 v1 from P0: OK
[24295.481][P1] waiting for result (sleep)
[OBS#1] [16413.722][P2] got OK for program #15 v2 (OK_local=15, OK_total=50)
[16413.832][P2] global limit reached (TOTAL_ACCEPTED_LIMIT_7_8=50), local OK=15, exiting
[16413.858][P2] stopping (7-8), local OK=15
[16413.862][P2] P2 (10) exiting
[OBS#1] [45100.968][P0] finished review of program #17 v2 from P1: OK
[45101.054][P0] global limit reached (TOTAL_ACCEPTED_LIMIT_7_8=50), local OK=19, exiting
[45101.059][P0] stopping (7-8), local OK=19
[45101.062][P0] P0 (10) exiting
[OBS#1] [24506.685][P1] global limit reached (TOTAL_ACCEPTED_LIMIT_7_8=50), local OK=16, exiting
[24506.821][P1] stopping (7-8), local OK=16
[24506.827][P1] P1 (10) exiting
os-ihw > 4-6 > include > four_six > config_4_6.h
.clang-tidy 4:31 LF UTF-8 4 spaces* C | os_4_6 | Debug