Национальный исследовательский университет «Высшая школа экономики»
Факультет компьютерных наук
Программная инженерия

Операционные системы

Отчёт по индивидуальному домашнему заданию№2
Вариант 8

Работу выполнила:
М. В. Царахова
Группа: БПИ-213
Преподаватель:
А. И. Легалов

Москва
2023

Содержание

Постановка задачи

Военная задача. Анчуария и Тарантерия — два крохотных латиноамериканских государства, затерянных в южных Андах. Диктатор Анчуарии, дон Федерико, объявил войну диктатору Тарантерии, дону Эрнандо. У обоих диктаторов очень мало солдат, но очень много снарядов для минометов, привезенных с последней американской гуманитарной помощью. Поэтому армии обеих сторон просто обстреливают наугад территорию противника, надеясь поразить минометы противника. Стрельба ведется хаотично до полного уничтожения всех минометов противника, размещенных на некоторой прямоугольной площадке размером N*N. То есть координаты целей, хоть и случайные, согласуются между разными минометчиками одной армии (это предохраняет от стрельбы в одну и ту же точку). Интервалы между выстрелами (задержки) зависят от расстояния до точки попадания, формируемой случайно. Повторная стрельба по одним и тем же координатам не производится.
Количество минометов у каждой стороны одинаково и задается аргументом командной строки. Размер поля также задается в командной строке. Создать приложение, моделирующее военные действия.

Каждая страна — отдельный клиент. Сервер отвечает за прием координат от клиентов и передает эти координаты другим клиентам. Он также получает информацию от клиентов об их уничтожении. Расположение минометов порождается каждым клиентом случайно.

Программы должны быть написаны на языке программирования C и выполняться в среде ОС Linux

4 балла

Множество процессов взаимодействуют с использованием неименованных POSIX семафоров расположенных в разделяемой памяти. Обмен данными ведется через разделяемую память в стандарте POSIX.

Аргументами командной строки являются - размер поля и количество минометов. Если аргументов недостаточно программа завершается и показывает сообщение с подсказкой о том какие аргументы необходимо передать.

Затем с помощью функции strtol мы получаем число из текста и записываем размер поля и количество минометов в переменные. Миномет в программе обозначается как gun.

Мы будем использовать разделенную память с названием memory_name, в которой будут храниться два поля - территории государств.

Открываем разделенную память с помощью shm_open. Далее устанавливаем размер памяти через ftruncate. И инициализируем первое и второе поле через mmap. Храним поля в переменных first_field и second_field.

Поле состоит из N*N точек. Точка описывается структурой point_t. В этой программе мы будем использовать неименнованные семафоры, поэтому в каждой точке есть gun_sem и manager_sem. Также в каждой точке хранится её тип. У нас есть 4 типа и 5ый дополнительный. 2 типа для обозначения обычной точки - empty если в неё ещё не стреляли, used если уже стреляли. Alive gun - если эта пушка ещё не была поражена, dead gun - если поражена. И finish чтобы обозначить что все удары были завершены. Также в точке хранится target_coordinate что означает номер точки в которую будут стрелять. Если это не gun или dead gun, то номер равен -1.

После открытия разделенной памяти мы инициализируем объекты - функция fill_field. Для каждой точки (для обоих полей) изначально устанавливаем тип empty point и выставляем target -1. После этого необходимо сгенерировать данное количество gun-ов - для этого используется функция rand(), которая также учитывает чтобы мы ставили gun только если он уже там не стоит.

Следующее действие - инициализация всех семафоров. Для каждого действующего gun'a мы выполняем функцию sem_init для его gun и manager. Мы также выполняем это для обоих полей.

Затем мы делаем fork для каждого действующего gun'a. И в этом процессе запускаем функцию gun для точки. Функция gun ждёт пока gun_sem получит post а затем выполняет стрельбу (если есть target). А затем отправляет post для manager_sem, чтобы оповестить что стрельба закончилась.

Когда gun получает post и видит что его тип это finish это означает что весь процесс закончился и об этом нужно оповестить отправив post для manager_sem.

Функция manager - вызывается после того как были созданы все процессы gun и управляет процессом войны.

Для того чтобы понимать в каком состоянии сейчас война есть функция check_status, которая проверяет что у каждой стороны остался хотя бы один действующий gun.

Затем в цикле функция manager начинает войну. Первое государство начинает и сначала генерирует цели с помощью функции generate_targets. Эта функция берет точку только если это alive gun и также рандомно выбирает target из тех точек в которые ещё не было стрельбы. Затем делаем post для gun_sem чтобы отправить команду о том что нужно выполнить стрельбу из этого gun'a - её также отправляем только для alive guns. Затем ждём (wait) от функции gun когда она отправит post для manager_sem что будет означать что стрельба была произведена. После этого проверяем что война не завершена. А далее выполняем всё тоже самое только для второго государства.

После того как одно из государств было поражено мы печатаем оба поля, чтобы увидеть

в каком они окончательном состоянии. И затем начинаем переходить к процессу завершения. Мы выставляем тип FINISH для всех точек которые gun (alive или dead), и затем для каждой такой точки отправляем запрос post gun_sem чтобы gun мог определить конец и завершиться. gun в свою очередь отправляет post manager_sem. И manager в самом конце ждёт чтобы все gun с статусом finish вернули ему результат (сделали post).

А затем из main мы вызываем destroyer который закрывает все семафоры и разделяемую память.

Также есть функция handler, которая нужна для того чтобы отлавливать ctrl c и корректно завершать программу - она также вызывает destroyer чтобы всё закрыть.

В файле helper.h расположены дополнительные функции для основной программы.

Вот некоторый пример работы программы:

```
Field size = 3
Number of guns = 1
Memory opened: name = shared memory, id = 0x3
Memory size set - 1296
All semaphores are inited
Generated targets for first side
Gun from first side in coordinate: 3
Making shot from 3: into 3 by 1 side
Gun from second side in coordinate: 4
Shot made by first side from 3 into 3. Now this point is used
Generated targets for second side
Making shot from 4: into 3 by 2 side
Shot made by second side from 4 into 3. Now this point type is dead gun
SECOND SIDE IS WINNER
Side number 1
***
D**
***
Side number 2
***
xG*
***
Point: 3 is dead
Point: 4 is dead
Closed all semaphores
Shared memory is unlinked
```

5 баллов

Множество процессов взаимодействуют с использованием именованных POSIX семафоров. Обмен данными ведется через разделяемую память в стандарте POSIX.

В данном случае мы будем использовать именованные семафоры. Чтобы получить имя семафора мы используем функцию get_semaphore_name, которая записывает gun или manager, id и номер государства. Также мы сохраняем все семафоры в переменных -

```
sem_t **gun_semaphores_pointer_first = NULL;
sem_t **gun_semaphores_pointer_second = NULL;
sem_t **manager_semaphores_pointer_first = NULL;
sem_t **manager_semaphores_pointer_second = NULL;
```

Инициализация семафоров происходит следующим образом:

```
sem_t *gun_semaphores_first[field_size * field_size];
sem_t *gun_semaphores_second[field_size * field_size];
for (int i = 0; i < field_size * field_size; ++i) {
    char* semaphore_name = get_semaphore_name(i, 1, gun_semaphore_name);
    gun_semaphores_first[i] = sem_open(semaphore_name, O_CREAT, 0666, 0);
    free(semaphore_name);

    semaphore_name = get_semaphore_name(i, 2, gun_semaphore_name);
    gun_semaphores_second[i] = sem_open(semaphore_name, O_CREAT, 0666, 0);
    free(semaphore_name);
}
gun_semaphores_pointer_first = gun_semaphores_first;
gun_semaphores_pointer_second = gun_semaphores_second;
```

И тоже самое для manager.

Далее во всех местах мы обращаемся к семафору по адресу который мы сохранили в массиве. А в gun мы например сразу передаем нужный семафор по индексу.

Также мы иначе уничтожаем семафор в функции destroy:

```
char* semaphore_name;
for (int i = 0; gun_semaphores_pointer_first != NULL && i < field_size *
    field_size; ++i) {
    sem_destroy(gun_semaphores_pointer_first[i]);
    semaphore_name = get_semaphore_name(i, 1, gun_semaphore_name);
    if (sem_unlink(semaphore_name) == -1) {
        perror("sem_unlink");
        system_error("server: error getting pointer to semaphore");
    }
    free(semaphore_name);
}
```

Всё остальное остаётся таким же. Меняется только использование семафора.

Пример работы программы:

```
Field size = 3
Number of guns = 1
Memory opened: name = shared memory, id = 0x3
Memory size set - 1296
All semaphores are inited
Gun from second side in coordinate: 2
Generated targets for first side
Gun from first side in coordinate: 5
Making shot from 5: into 4 by 1 side
Shot made by first side from 5 into 4. Now this point is used
Generated targets for second side
Making shot from 2: into 8 by 2 side
Shot made by second side from 2 into 8. Now this point type is used
Generated targets for first side
Making shot from 5: into 3 by 1 side
Shot made by first side from 5 into 3. Now this point is used
Generated targets for second side
Making shot from 2: into 5 by 2 side
Shot made by second side from 2 into 5. Now this point type is dead gun
SECOND SIDE IS WINNER
Side number 1
***
**D
**x
Side number 2
**G
xx*
***
Point: 2 from field 2 is finished
Point: 5 from field 1 is finished
Gun semaphores are closed
Manager semaphores are closed
Shared memory is unlinked
```