**Отчет по ИДЗ №2**

**По предмету “Операционные системы”**

**Морина Ильи Олеговича**

**БПИ215**

**Вариант 30**

**Тема «Применение семафоров для организации взаимодействия параллельных процессов».**

**Задание:** Разработать консольное приложение, в котором единый родительский процесс запускает требуемое число дочерних процессов.

**Условие:**

Задача о Пути Кулака. На седых склонах Гималаев стоит древний буддистский монастырь: Гуань-Инь-Янь. Каждый год в день сошествия на землю боддисатвы Монахи монастыря собираются на совместное празднество и показывают свое совершенствование на 10 Пути Кулака. Всех соревнующихся монахов разбивают на пары, победители пар бьются затем между собой и так далее, до финального поединка. Монах который победил в финальном бою, забирает себе на хранение статую боддисатвы. Реализовать приложение, определяющего победителя. В качестве входных данных используется массив, в котором хранится количество энергии Ци каждого монаха. При победе монах забирает энергию Ци своего противника. Каждый бой длится некоторое случайное время. Разбивка на пары перед каждым сражением осуществляется случайным образом и асинхронно. То есть, очередную пару образуют первые освободившиеся от боя монахи. Но бои следующего круга начинаются одновременно. Монах, оставшийся без пары, удваивает свою энергию, отдохнув от поединка и является первым для последующего выбора.

(Далее слово монах будет заменено на игрок)

**В качестве победителя каждого боя и игры в целом выбирается игрок с большим количеством энергии.**

**Вариант на оценку 4:**

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

**Схема работы программы.**

Стоит отметить, что программа соответствует условию задания и следующий раунд начинается только после того, как все дочерние процессы (игроки) завершат свои действия и освободятся. Родительский процесс ожидает завершения дочерних процессов с помощью вызова функции **waitpid()** для каждого из них. После завершения всех дочерних процессов, состояние игроков обновляется, и программа определяет, можно ли начать следующий раунд или уже определен победитель.

В данном коде используются именованные семафоры. Именованный семафор создается и открывается с помощью функции sem\_open():

**sem = sem\_open("/semaphore", O\_CREAT | O\_EXCL, 0644, 1);**

"/semaphore" - это имя именованного семафора. Флаги O\_CREAT и O\_EXCL указывают, что семафор должен быть создан, и если он уже существует, то вызов sem\_open() завершится с ошибкой. В данном случае, значение именованного семафора инициализируется значением 1.

Для удаления имени именованного семафора используется функция sem\_unlink():

**sem\_unlink("/semaphore");**

sem\_unlink() удаляет имя семафора, но не закрывает сам семафор. Семафор будет закрыт и уничтожен только после того, как его закроют все процессы, которые его использовали, с помощью вызова sem\_close():

**if (sem\_close(sem) == -1) {**

**perror("sem\_close");**

**exit(EXIT\_FAILURE);**

**}**

**Для выхода из программы можно использовать два способа.**

1. Когда просят ввести число игроков предусмотрен вариант ввода команды “q”. В таком случае программа завершится до создания семафоров и разделяемой памяти, поэтому освобождать данные ресурсы не придется.
2. Ввод команды control + C возможен только во время выполнения программы (после ввода кол-ва игроков), в таком случае программа будет завершена, все необходимыe ресурсы будут завершены и закрыты.

**Обработки некорректного ввода нет !!! Корректным считается ввод числа от 1 до 1079, так как если ввести число больше уже может происходить ошибка fork: Resource temporarily unavailable. Так же пользователь может ввести команду q для выхода, или же control + C (но уже во время выполнения).**

1. Ввод количества игроков или команды для выхода из программы (энергия у игроков генерируется случайным образом от 1 – 100).
2. Создание семафора для синхронизации процессов.
3. Создание разделяемой памяти для хранения информации об игроках.
4. Инициализация структуры данных для каждого игрока (энергия, состояние и т.д.).
5. Пока есть более одного живого игрока и пользователь не прерывает программу (SIGINT):

a. Сброс состояния отдыха для игроков, которые отдыхали в предыдущем раунде.

b. Родительский процесс создает дочерние процессы для каждого активного (живого и не отдыхающего) игрока.

c. В дочерних процессах выполняется функция battle\_process:

i. Если игрок отдыхает, завершаем дочерний процесс.

ii. Генерируем случайное время боя.

iii. Если игрок не в бою и жив, он начинает искать соперника.

iv. Если соперник найден, начинается бой между игроками, и энергия переходит к победителю.

v. Если соперник не найден, игрок удваивает свою энергию и переходит в состояние отдыха.

d. Родительский процесс ожидает завершения всех дочерних процессов.

e. Сброс флагов (in\_battle и resting) для всех игроков и подсчет оставшихся живых игроков.

1. Вывод информации о победителе.
2. Очистка разделяемой памяти, закрытие семафора и освобождение динамически выделенной памяти.
3. Завершение программы.

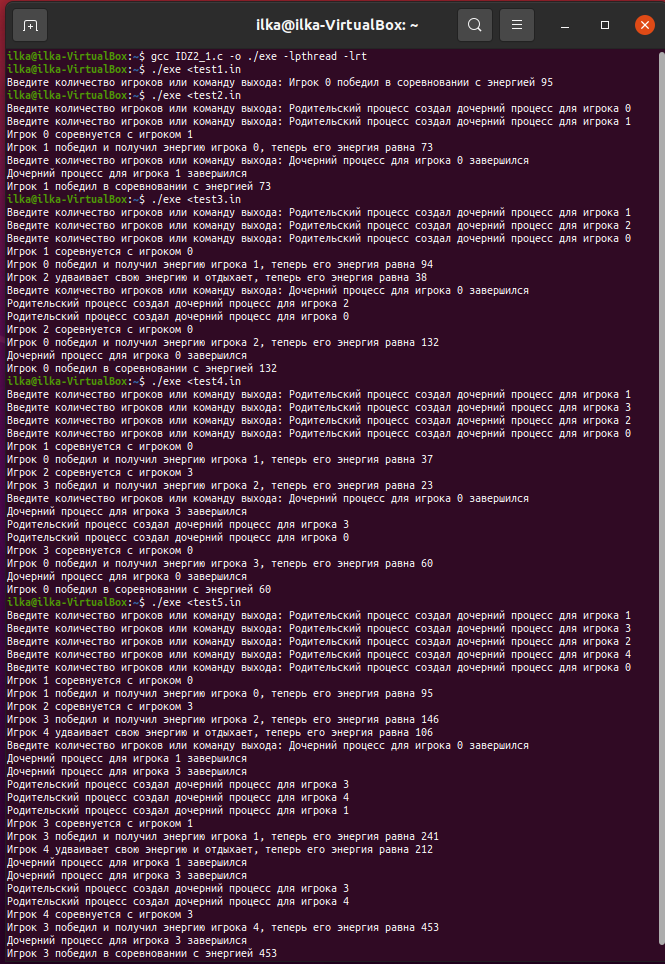
**Тестовое покрытие (для всех вариантов программы на оценку 4-6 одинаковое) :**

Так как программа принимает на вход только одно число, то напишем 5 следующих тестов:

1. Число 1 – 1 игрок (проверяем что у нас побеждает единственный созданный игрок)
2. Число 2 – 2 игрока сражаются между собой.
3. Число 3 – 3 игрока, один будет отдыхать.
4. Число 4 – 4 игрока, проверим что никто не отдыхает.
5. Число 5 – 5 игроков, проверим что один игрок не отдыхает несколько раз.

Писать файлы test.out не имеет смысла, потому что программа при каждом запуске случайным образом генерирует данные (энергию игроков).

Скомпилировали программу и прогнали тесты, видeм что все работает отлично.



**Вариант на оценку 5:**

В этом коде используются неименованные (анонимные) семафоры. Это видно из следующего фрагмента кода:

**sem\_init(&players[i].sem, 1, 1);**

Функция sem\_init() используется для инициализации анонимного семафора. Второй аргумент функции sem\_init() равен 1, что указывает на использование семафора в разделяемой памяти между процессами. Если бы этот аргумент был равен 0, это указывало бы на использование семафора в пределах одного процесса между потоками.

1. Получение количества игроков от пользователя.
2. Инициализация памяти, игроков и семафоров.
3. Пока остается больше одного живого игрока и программа не прерывается пользователем:

a. Сброс состояния отдыха у всех игроков.

b. Для каждого активного игрока создается дочерний процесс.

c. В каждом дочернем процессе происходит следующее:

i. Если игрок отдыхает, ничего не происходит.

ii. Если игрок не отдыхает, выбирается случайное время боя.

iii. Захват семафора для текущего игрока.

iv. Если игрок не в бою и жив, ищется противник.

v. Если противник найден, происходит бой, и победитель забирает энергию проигравшего.

vi. Если противник не найден, игрок отдыхает, и его энергия удваивается.

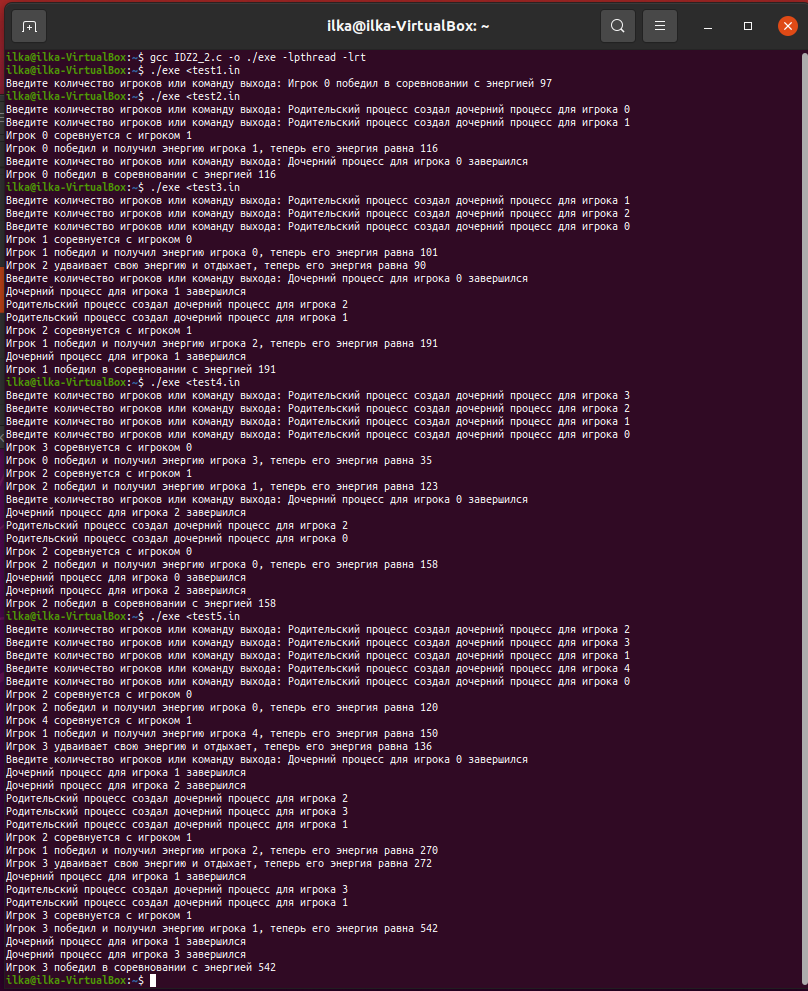
vii. Освобождение семафора для текущего игрока.

d. Родительский процесс ожидает завершения всех дочерних процессов.

e. Сброс флагов состояний и подсчет живых игроков.

1. Вывод победителя.
2. Очистка выделенных ресурсов и завершение программы.

Скомпилировали программу и прогнали тесты, видeм что все работает отлично.

**Вариант на оценку 6:**

Множество процессов взаимодействуют с использованием семафоров в стандарте UNIX SYSTEM V. Обмен данными ведется через разделяемую память в стандарте UNIX SYSTEM V.

**Схема работы программы:**

В программе используются семафоры UNIX SYSTEM V. В коде используются функции semget(), semop() и semctl() для работы с семафорами.

Создание семафора и установка начального значения:

sem\_id = semget(IPC\_PRIVATE, 1, IPC\_CREAT | 0666);

if (sem\_id == -1) {

perror("semget");

exit(EXIT\_FAILURE);

}

semctl(sem\_id, 0, SETVAL, 1);

Функции для блокировки и разблокировки семафора:

void sem\_lock() {

struct sembuf op;

op.sem\_num = 0;

op.sem\_op = -1;

op.sem\_flg = 0;

semop(sem\_id, &op, 1);

}

void sem\_unlock() {

struct sembuf op;

op.sem\_num = 0;

op.sem\_op = 1;

op.sem\_flg = 0;

semop(sem\_id, &op, 1);

}

И удаление семафора при завершении работы программы:

if (semctl(sem\_id, 0, IPC\_RMID) == -1) {

perror("semctl");

exit(EXIT\_FAILURE);

Таким образом, в коде используются семафоры System V для синхронизации процессов и разделяемая память для хранения информации об игроках.

В остальном же схема работы программы не отличается от предыдущих двух вариантов.

Скомпилировали программу и прогнали тесты, видeм что все работает отлично.

