Почаев М. Н. Вариант 21

Разработать программы, состоящие из нескольких взаимодействующих параллельных процессов. Количество разрабатываемых программ и их характеристики определяются в соответствии с требованиями выполнения на соответствующую оценку.

Задача о нелюдимых садовниках. Имеется пустой участок земли (двумерный массив размером M ×N) и план сада, разбитого на отдельные квадраты. От 10 до 30 процентов (задается случайно) площади сада заняты прудами или камнями. То есть недоступны для ухаживания. Эти квадраты располагаются на плане произвольным (случайным) образом. Ухаживание за садом выполняют два садовника, которые не хотят встречаться друг другом (то есть, одновременно появляться в одном и том же квадрате). Первый садовник начинает работу с верхнего левого угла сада и перемещается слева направо, сделав ряд, он спускается вниз и идет в обратном направлении, пропуская обработанные участки. Второй садовник начинает работу с нижнего правого угла сада и перемещается снизу вверх, сделав ряд, он перемещается влево и также идет в обратную сторону. Если садовник видит, что участок сада уже обработан другим садовником или является необрабатываемым, он идет дальше. Если по пути какой-то участок занят другим садовником, то садовник ожидает когда участок освободится, чтобы пройти дальше на доступный ему необработанный участок. Садовники должны работать одновременно со скоростями, определяемыми как параметры задачи. Прохождение через любой квадрат занимает некоторое время, которое задается константой, меньшей чем времена обработки и принимается за единицу времени. Создать многопроцессное приложение, моделирующее работу садовников. Каждый садовник — это отдельный процесс.

Я написал 4 программы. Первая выводит визуализацию сада и запускает садовников в одной консоли. Вторая - отдельно инициализация сада в одной консоли и садовников в других (визуализация работы сада в первой консоли). В третьей добавил наблюдателя ко второй программе. В четвертой - реализовал несколько наблюдателей.

Во всех программах одна модель сада, которая написана в garden.c garden.h.

Каждая клетка сада описывается структурой cell_t. Она имеет 4 состояния: свободна и не обработана, свободна и обработана или заблокирована, занята. При инициализации сада задаются размеры MxN, случайно выбираются доля заблокированных клеток и отмечаются CELL_BLOCKED, вычисляется количество обрабатываемых клеток, счетчик обработанных устанавливается как 0, координаты садовников устанавливаются -1. Вычисление направления движения садовников написал тоже в этом модуле функциями gardener1_start, gardener2_start, gardener1_next, gardener2_next.

Программа 1
Реализована в файле main.c,
Для **запуска** нужно в консоли ввести
gcc -Wall -Wextra -O2 main.c garden.c -o gardeners -lrt -pthread
./gardeners M N move_time_us process_g1_us process_g2_us

- M — количество строк сада;
- N — количество столбцов сада;
- move_time_us — время прохода через одну клетку (в микросекундах), одинаковое для обоих садовников;
- process_g1_us — время обработки клетки садовником 1 (в микросекундах);
- process_g2_us — время обработки клетки садовником 2.

Например:
./gardeners_p1 10 20 100000 300000 300000

Еще я обработал ситуацию с дедлоком в правом верхнем углу, когда оба садовника ждут друг друга, ожидая, что следующая клетка освободится. Если садовник 1 ждет больше двух времен обработки, то отскакивает на 2 клетки назад, чтобы второй садовник продвинулся. Это единственное, мне кажется место, где садовники двигаются вопреки условию, но другого решения, не противоречащего условию я не смог найти.

Программа 2
Во второй программе модель сада и логика работы садовников остаются такими же, меняется архитектура приложения меняется. Теперь три оддельные программы: init, gardener1, gardener2. Процессы обмениваются через POSIX Shared Memory, синхронизируются с именованным семафором. Для работы с разделяемой памятью, семафором и очередью сообщений для прооограм 3 и 4 написал отдельный модуль ipc.c ipc.h. В нём определены имя сегмента разделяемой памяти, например, #define SHM_NAME "/garden_shm_v2", имя именованного семафора #define SEM_NAME "/garden_sem_v2" и функции-обёртки:
shared_data_t *attach_shared(int create); sem_t *open_mutex(int create); void destroy_shared(void); void destroy_mutex(void);

shared_data_t — та же структура, что и в программе 1, но уже лежащая в именованном сегменте.

Функция attach_shared(create):

при create != 0:

вызывает shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666), делает ftruncate до нужного размера, подключает сегмент через mmap(MAP_SHARED)

при create == 0:

открывает уже существующий сегмент shm_open(SHM_NAME, O_RDWR, ...) и также делает mmap.

Функция open_mutex(create):

при create != 0:

создаёт именованный семафор через sem_open(SEM_NAME, O_CREAT, 0666, 1)

при create == 0:

открывает уже существующий семафор sem_open(SEM_NAME, 0).

Функции destroy_shared и destroy_mutex позже используются в отдельной утилите cleanup для удаления IPC-объектов (shm_unlink, sem_unlink).

Программа init отвечает за подготовку общей структуры данных, с которой дальше будут работать садовники и наблюдатели:

Подключается к разделяемой памяти и при необходимости создаёт её:

shared_data_t *sh = attach_shared(1);

Открывает (или создаёт) именованный семафор:

sem_t *mutex = open_mutex(1);

Вызывает инициализацию сада:

garden_init(&sh->garden, M, N, 10, 30);

Здесь M и N передаются как аргументы командной строки, а часть заблокированных клеток (от 10 до 30%) выбирается случайно внутри garden_init.

init может один раз вывести стартовое состояние сада или в цикле периодически его печатать, чтобы визуально видеть прогресс обработки.

После запуска init в системе существуют: именованный сегмент SHM_NAME с инициализированным садом, именованный семафор SEM_NAME.

К этим объектам затем подключаются садовники и наблюдатели.

Программы садовников содержат минимум собственной логики. Для обоих используется общий модуль runner.c, в котором реализована функция run_gardener.

Файл gardener1.c подключается к общей памяти и семафору:

shared_data_t *sh = attach_shared(0);
sem_t *mutex = open_mutex(0);
парсит параметры командной строки (времена прохода и обработки);
вызывает run_gardener(1, sh, mutex, move_time_us, process_time_us).

Файл gardener2.c делает то же самое, но с id = 2.

Внутри run_gardener используется та же логика, что и в программе 1.

Запуск программы 2:

сборка:

gcc -Wall -Wextra -O2 -c garden.c

gcc -Wall -Wextra -O2 -c ipc.c

gcc -Wall -Wextra -O2 -c runner.c

gcc -Wall -Wextra -O2 init.c garden.o ipc.o runner.o -o init -lrt -pthread

gcc -Wall -Wextra -O2 gardener1.c garden.o ipc.o runner.o -o gardener1 -lrt -pthread

gcc -Wall -Wextra -O2 gardener2.c garden.o ipc.o runner.o -o gardener2 -lrt -pthread

запускаем в трех отдельных терминалах:

./init 20 20

./gardener1 100000 300000

./gardener2 100000 300000

20 20 - размер сада, 100000 300000 - время перехода нс другой участок и время обработки.

Программа 3 то же самое только добавляется процесс-наблюдатель observer. Его задача - не вмешиваясь в работу садовников, получать информацию об их действиях и на основе этой информации отображать текущее состояние сада в своей консоли. Для этого используется ещё один механизм межпроцессного взаимодействия очередь сообщений POSIX.

Каждый садовник при выполнении шага посылает в очередь сообщений короткое лог-сообщение фиксированного размера. Для него введена структура:

```
typedef struct {
   int  gardener_id;  // номер садовника: 1 или 2
   int  row;          // строка клетки
   int  col;          // столбец клетки
   char action;       // тип действия: 'S', 'M', 'P', 'F' и др.
} log_msg_t;
```

Поле action кодирует вид события:

- 'S' — старт работы садовника (первая позиция);
- 'M' — перемещение в новую клетку;
- 'P' — обработка клетки (из состояния EMPTY в DONE_G1 / DONE_G2);
- 'F' — завершение работы садовника.

Наблюдателю по большому счёту важны только координаты и сам факт активности садовника — лог-коды позволяют удобно печатать в консоли, что происходит.

Программа observer:

Подключается к уже созданным shared memory и семафору:

```
shared_data_t *sh = attach_shared(0);
sem_t *mutex      = open_mutex(0);
```

Создаёт/открывает очередь сообщений:

```
mqd_t mq = open_log_queue(1);
```

В цикле читает сообщение mq_receive, интерпретирует его как log_msg_t, под семафором вызывает print_garden_safe(sh, mutex) - выводит текущее состояние сада, периодически проверяет done_cells и total_work_cells и завершает работу, когда сад полностью обработан.

Состояние сада берётся не из логов, а из shared memory. Логи служат просто «сигналом», что что-то произошло, и можно перерисовать картинку.

Запуск программы 3:

Собираем observer

gcc -Wall -Wextra -O2 observer.c garden.o ipc.o runner.o -o observer -lrt -pthread

Запускаем в разных терминалах:

./init 20 20

./observer

./gardener1 100000 300000

./gardener2 100000 400000

(именно в таком порядке)

В **программе 4** добавляется возможность запускать несколько наблюдателей одновременно. Для этого реализована отдельная программа multi_observer. В отличие от observer, который получает события из очереди сообщений, multi_observer:

напрямую подключается к shared memory и именованному семафору:

```
shared_data_t *sh = attach_shared(0);
sem_t *mutex      = open_mutex(0);
```

периодически (с заданным интервалом) вызывает:

print_garden_safe(sh, mutex);

после каждого вывода проверяет done_cells, total_work_cells и флаг stop и завершает работу, когда сад полностью обработан.

Программу multi_observer можно запускать несколько раз, каждый экземпляр будет выводить состояние сада в своём терминале. Интервал обновления можно задавать параметром командной строки.

Сборка:

```
gcc -Wall -Wextra -O2 -c garden.c
gcc -Wall -Wextra -O2 -c ipc.c
gcc -Wall -Wextra -O2 -c runner.c
gcc -Wall -Wextra -O2 multi_observer.c garden.o ipc.o runner.o -o multi_observer -lrt -pthread
```

Пример запуска:

```
./init 20 20
./gardener1 100000 300000
./gardener2 100000 400000

./multi_observer  // наблюдатель 1
```

./multi_observer 500000 // наблюдатель 2 с более редким обновлением

Дополнительно была реализована утилита cleanup, которая вызывает shm_unlink, sem_unlink и mq_unlink и позволяет корректно удалять все IPC-объекты после экспериментов.