Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет «Высшая школа экономики»

Факультет компьютерных наук

Департамент программной инженерии

Отчет к домашнему заданию
По дисциплине
«Архитектура вычислительных систем»

Работу выполнил:

Студент группы БПИ-193 Штанько Е.О.

Москва 2020

Задание

Задача о каннибалах.

Племя из n дикарей ест вместе из большого горшка, который вмещает m кусков тушеного миссионера. Когда дикарь хочет обедать, он ест из горшка один кусок, если только горшок не пуст, иначе дикарь будит повара и ждет, пока тот не наполнит горшок. Повар, сварив обед, засыпает. Создать многопоточное приложение, моделирующее обед дикарей. При решении задачи пользоваться семафорами.

Решение

В данной задаче можно узнать типичную проблему "producer-consumer"(производитель-потребитель). Есть два процесса, которые обмениваются информацией через буфер ограниченного размера. Производитель (в нашем случае, повар) закладывает информацию(пищу) в буфер(горшок), а потребитель(дикарь) извлекает ее оттуда. Если буфер(горшок) заполнен, то производитель(повар) должен ждать, пока в нем появится место, чтобы положить туда новую порцию информации(пищи). Если буфер(горшок) пуст, то потребитель(дикарь) должен дожидаться нового сообщения(новой порции).

Решение данной задачи было реализовано через семафоры.

Краткое описание основных моментов

В программе имеются два семафора: full и empty.

- семафор full будем использовать для гарантии того, что потребитель(дикарь) будет ждать, пока в буфере(горшке) появится информация(пища).

- семафор empty будем использовать для организации ожидания производителя(повара) при заполненном буфере(непустом горшке).

Также, в программе применяются mutex-ы (для организации взаимоисключения на критических участках, которыми являются действия put_item и get_item (операции "положить информацию(пищу)" и "взять информацию(пищу)")

Тогда схематично решение можно описать так:

Semaphore empty = 1; // так как горшок у нас один, и заполняется за раз полностью.
Semaphore full = 0;

Producer:
while(не достигнуто введенное кол-во повторов) {
pthread_mutex_lock;
sem_wait(empty);
put_item;
для каждого дикаря sem_post(full);
pthread_mutex_unlock;
}

Consumer:
while(не достигнуто введенное кол-во повторов) {
pthread_mutex_lock;
sem_wait(full);
get_item;
если горшок пуст sem_post(empty);
pthread_mutex_unlock;
}

Подробное описание по шагам

1. Запуск

Приложение запускается из командной строки.

Аргументы для запуска приложения из командной строки:

1. argv[1] - количество дикарей в племени(n)(количество потребителей). Если введено число меньше 1 - сообщение об ошибке:

```
Число каннибалов не может быть меньше 1!
Process finished with exit code 0
```

2. argv[2] - вместимость горшка(m)(объем буфера). Если введено число меньше 1 - сообщение об ошибке:

```
Что это за горшок, в который и один кусок не помещается!
Process finished with exit code 0
```

3. argv[3] - то, на сколько порций(полных горшков) хватит миссионера(сколько раз будет происходить заполнение горшка)(numberOfServings). Если введено число меньше 1 - сообщение об ошибке:

```
Если Вы ни разу не накормите каннибалов - они съедят Вас!
Process finished with exit code 0
```

2. Начало работы программы

При попытке создать неименованный семафор я столкнулась с проблемой: неименованные семафоры не поддерживаются mac. Поэтому, мне пришлось использовать именованные(named) семафоры.

Программа начинается с попытки удалить семафоры, которые могли остаться в системе при некорректном завершении предыдущего сеанса работы с программой.

```
sem_unlink("emptySem");
sem_unlink("fullSem");
```

Происходит инициализация мьютексов

```
pthread_mutex_init(&mutexSecond, nullptr);
pthread_mutex_init(&mutexFirst, nullptr);
```

Создание семафоров (используется #include <semaphore.h>)

(O_CREAT - если семафор еще не существует - создается, 0600 - (permissions to be placed on the new semaphore: read / write) , 1/0 - начальные значения)

```
empty = sem_open("emptySem", O_CREAT, 0600, 1);
full = sem_open("fullSem", O_CREAT, 0600, 0);
```

Создание потоков (учитывается наличие основного потока) (используется #include <pthread.h>)

```
// создаем повара
pthread_t threadP;
pthread_create(&threadP, nullptr, Producer, (void *) nullptr);

// создаем каннибалов
// учитываем наличие основного потока
pthread_t threadC[n - 1];
int consumers[n - 1];

for (int i = 0; i < n - 1; i++) {
    consumers[i] = i + 1;
    pthread_create(&threadC[i], nullptr, Consumer, (void *) (consumers + i));
}
```

int body - буфер (куски тушеного миссионера в горшке)

3. Producer

Я постаралась сделать комментарии к коду качественными, описав в них и теоретические аспекты.

Основные функции:

- sem_wait(empty);

Уменьшаем значение семафора. Если значение семафора равно нулю, то вызов блокируется до тех пор, пока не станет возможным выполнить уменьшение (пока не произойдет вызов sem_post каннибалом).

- sem_post(full);

Увеличиваем значение семафора и снимаем блок с ожидающих потоков(каннибалов)

- sem_close(empty);

Закрываем именованный семафор, на который указывает sem, позволяя освободить все ресурсы, которые система выделила под семафор вызывающему процессу.

- sem_unlink("emptySem");

Функция sem_unlink() удаляет именованный семафор.

```
// В задаче всего один продьюсер - повар
void *Producer(void *param) {
    while (i < numberOfServings) {
        pthread_mutex_lock(&mutexSecond);
        // Уменьшаем значение семафора
        // Если значение семафора равно нулю, то вызов блокируется
до тех пор, пока не станет возможным выполнить уменьшение,
        // (пока не произойдет вызов sem_post каннибалом)
        sem_wait(empty);
        printf("Время: %.3lf. Повар приготовил тушеного миссионера!
Теперь в горшочке целых %d лакомых кусочков.\n",
               clock() / 1000.0, m);
        body = m;  // Заполняем горшок(put item)
        for (int i = 0; i < m; i++) {
            // Эта функция увеличивает значение семафора и
разблокирует ожидающие потоки(каннибалов)
            sem_post(full);
        }
        i++;
        pthread_mutex_unlock(&mutexSecond);
    }
    sem_close(empty);
    // Когда все процессы закончили использовать семафор, мы его
удаляем из системы
    sem_unlink("emptySem");
    return nullptr;
}
```

3. Consumer

- sem_wait(full);

Уменьшаем значение семафора. Если значение семафора равно нулю, то вызов блокируется до тех пор, пока не станет возможным выполнить уменьшение (пока не произойдет вызов sem_post поваром).

- sem_post(empty);

Увеличиваем значение семафора и снимаем блок с ожидающего потока(повара).

- sem_close(full);

- sem_unlink("fullSem");

```
// В задаче n консьюмеров - каннибалы
void *Consumer(void *param) {
    int cNum = *((int *) param); // номер канибала
    while (stop) {
        // время проголодаться(от 0 до 15 сек)
        sleep(rand() % 16);
        pthread_mutex_lock(&mutexFirst);
        if (j < numberOfServings * m) {
            // Уменьшаем значение семафора
            // Если значение семафора равно нулю, то вызов
блокируется до тех пор, пока не станет возможным выполнить
уменьшение,
            // (пока не произойдет вызов sem_post поваром)
            sem_wait(full);
            printf("Время: %.3lf. Каннибал #%d съел кусок.
Осталось: %d кусков.\n", clock() / 1000.0, cNum + 1,
                   body - 1);
            // Уменьшаем число кусков в горшке(get item)
            body--;
            // Если еда закончилась
            if (body == 0) {
                // Эта функция увеличивает значение семафора и
разблокирует ожидающий поток(повора)
                sem_post(empty);
            }
            j++;
        } else {
            stop = false;
        }
        pthread_mutex_unlock(&mutexFirst);
    }
    sem_close(full);
    // Когда все процессы закончили использовать семафор, мы его
удаляем из системы
    sem_unlink("fullSem");
    return nullptr;
}
```

Тестирование

1. Выше я уже описала и привела скрины работы программы случае некорректных введенных данных.

2. Рассмотрим примеры корректной работы программы.

В командную строку CLion введено: 5 4 2

```
Время: 9.614. Повар приготовил тушеного миссионера! Теперь в горшочке целых 4 лакомых кусочков.
Время: 9.727. Каннибал #4 съел кусок. Осталось: 3 кусков.
Время: 9.764. Каннибал #2 съел кусок. Осталось: 2 кусков.
Время: 9.867. Каннибал #1 съел кусок. Осталось: 1 кусков.
Время: 9.982. Каннибал #3 съел кусок. Осталось: 0 кусков.
Время: 10.047. Повар приготовил тушеного миссионера! Теперь в горшочке целых 4 лакомых кусочков.
Время: 10.157. Каннибал #3 съел кусок. Осталось: 3 кусков.
Время: 10.221. Каннибал #1 съел кусок. Осталось: 2 кусков.
Время: 10.306. Каннибал #5 съел кусок. Осталось: 1 кусков.
Время: 10.427. Каннибал #2 съел кусок. Осталось: 0 кусков.

Process finished with exit code 0
```

В командную строку CLion введено: 1 5 2

```
Время: 13.538. Повар приготовил тушеного миссионера! Теперь в горшочке целых 5 лакомых кусочков.
Время: 13.614. Каннибал #1 съел кусок. Осталось: 4 кусков.
Время: 13.666. Каннибал #1 съел кусок. Осталось: 3 кусков.
Время: 13.714. Каннибал #1 съел кусок. Осталось: 2 кусков.
Время: 13.764. Каннибал #1 съел кусок. Осталось: 1 кусков.
Время: 13.806. Каннибал #1 съел кусок. Осталось: 0 кусков.
Время: 13.836. Повар приготовил тушеного миссионера! Теперь в горшочке целых 5 лакомых кусочков.
Время: 13.905. Каннибал #1 съел кусок. Осталось: 4 кусков.
Время: 13.951. Каннибал #1 съел кусок. Осталось: 3 кусков.
Время: 14.013. Каннибал #1 съел кусок. Осталось: 2 кусков.
Время: 14.073. Каннибал #1 съел кусок. Осталось: 1 кусков.
Время: 14.146. Каннибал #1 съел кусок. Осталось: 0 кусков.

Process finished with exit code 0
```

В командную строку CLion введено: 2 100 100

```
Время: 7.360. Повар приготовил тушеного миссионера! Теперь в горшочке целых 100 лакомых кусочков.
Время: 7.503. Каннибал #1 съел кусок. Осталось: 99 кусков.
Время: 7.567. Каннибал #2 съел кусок. Осталось: 98 кусков.
Время: 7.611. Каннибал #2 съел кусок. Осталось: 97 кусков.
Время: 7.641. Каннибал #1 съел кусок. Осталось: 96 кусков.
Время: 7.708. Каннибал #2 съел кусок. Осталось: 95 кусков.
Время: 7.762. Каннибал #1 съел кусок. Осталось: 94 кусков.
Время: 7.815. Каннибал #2 съел кусок. Осталось: 93 кусков.
Время: 7.866. Каннибал #2 съел кусок. Осталось: 92 кусков.
Время: 7.911. Каннибал #1 съел кусок. Осталось: 91 кусков.
Время: 7.956. Каннибал #2 съел кусок. Осталось: 90 кусков.
Время: 8.049. Каннибал #1 съел кусок. Осталось: 89 кусков.
Время: 8.095. Каннибал #2 съел кусок. Осталось: 88 кусков.
Время: 8.149. Каннибал #1 съел кусок. Осталось: 87 кусков.
Время: 8.221. Каннибал #1 съел кусок. Осталось: 86 кусков.
Время: 8.280. Каннибал #2 съел кусок. Осталось: 85 кусков.
Время: 8.343. Каннибал #1 съел кусок. Осталось: 84 кусков.
Время: 8.404. Каннибал #2 съел кусок. Осталось: 83 кусков.
Время: 8.469. Каннибал #1 съел кусок. Осталось: 82 кусков.
Время: 8.513. Каннибал #1 съел кусок. Осталось: 81 кусков.
Время: 8.588. Каннибал #2 съел кусок. Осталось: 80 кусков.
```

В командную строку CLion введено: 100 2 2

```
Время: 9.903. Повар приготовил тушеного миссионера! Теперь в горшочке целых 2 лакомых кусочков.
Время: 10.350. Каннибал #5 съел кусок. Осталось: 1 кусков.
Время: 10.755. Каннибал #28 съел кусок. Осталось: 0 кусков.
Время: 10.937. Повар приготовил тушеного миссионера! Теперь в горшочке целых 2 лакомых кусочков.
Время: 11.128. Каннибал #48 съел кусок. Осталось: 1 кусков.
Время: 11.216. Каннибал #67 съел кусок. Осталось: 0 кусков.

Process finished with exit code 0
```

В командную строку CLion введено: 1 1 1

```
Время: 8.751. Повар приготовил тушеного миссионера! Теперь в горшочке целых 1 лакомых кусочков.
Время: 8.823. Каннибал #1 съел кусок. Осталось: 0 кусков.

Process finished with exit code 0
```

Список используемых источников

1. Задача с семинара "Многопоточность. Синхронизация потоков. Методы синхронизации" " о кольцевом буфере. Применении мьютексов и семафоров для обеспечения корректной работы программы" (http://www.softcraft.ru/edu/comparch/practice/thread/02-sync/readwriters01/main.cpp) Просмотрено: 6.12.2020
2. Сайт с описанием решения проблемы producer-consumer с помощью семафоров (https://studfile.net/preview/1200771/) Просмотрено 6.12.2020
3. Сайт "MANPAGES.org" с обзором на возможностей <semaphore.h> (http://ru.manpages.org), а именно (http://ru.manpages.org/sem_open/3), (http://ru.manpages.org/sem_unlink/3), (http://ru.manpages.org/sem_close/3) Просмотрено: 6.12.2020