Shared Memory Control

Системный вызов **shmctl**():

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int **shmctl** *(int shmid, int cmd, struct shmid_ds *buf));*

дает возможность производить ряд операций по управлению *существующим* сегментом разделяемой памяти и системной структурой данных **shmid_ds** этого сегмента. Имеет три аргумента.

shmid - идентификатор разд. сегмента памяти, сгенерированный ранее **shmget**().

cmd - определяет операцию, которую должен выполнить **shmctl**().

buf - указатель на структуру типа **shmid_ds**.

Shared Memory Control

Вызов **shmctl**() исполняет следующие операции *cmd* :

- IPC_RMID удалить системную структуру данных с указанным идентификатором разделяемого сегмента *shmid.* На месте аргумента *buf* указывают 0, приведённый к нужному типу *(shmid_ds *).*
- IPC_STAT - возвращает текущее значение структуры **shmid_ds** для разделяемого сегмента с идентификатором shmid.

Процесс должен иметь право на чтение данного разделяемого сегмента.

- SHM_LOCK - блокирование в памяти (запрет свопинга) сегмента разделяемой памяти, соответствующий *semid* аргументу
- SHM_UNLOCK - разблокирование сегмента (разрешение свопинга) разделяемой памяти, на который указывает *semid*.

Операции SHM_LOCK и SHM_UNLOCK может выполнить процесс, имеющий эффективный идентификатор сегмента или ID суперпользователя.

Shared Memory Operations Synchronization

При работе с разделяемой памятью необходимо **синхронизировать** выполнение взаимодействующих процессов:
когда один из процессов записывает данные в разделяемую память, остальные процессы ожидают завершения операции.
Обычно синхронизация обеспечивается с помощью *семафоров*.
назначение и число которых определяется конкретным использованием разделяемой памяти.

При обмене данными между процессами (клиент и сервер) для синхронизации обычно используется группа из 2-х семафоров.
Первый семафор нужен для блокирования доступа к разделяемой памяти.
Второй семафор служит для сигнализации серверу о том, что клиент начал работу, заблокировал разделяемую память и начал записывать данные в эту область.

Messages Exchange Example

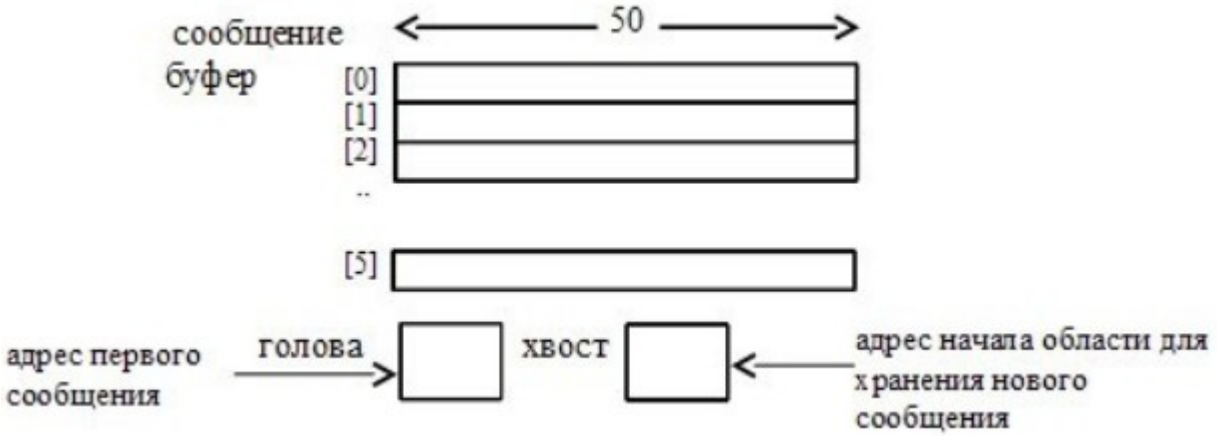
Обеспечение взаимодействия между двумя процессами (*producer* и *consumer*), которые могут выполняться с различными скоростями,

используется массив (обслуживается как очередь FIFO) с шестью буферами для сообщений. Для координации доступа к разделяемому сегменту данных используются два семафора.

Первый семафор-счётчик хранит число доступных для записи слотов. Пока его значение отлично от нуля, процесс может продолжать запись сообщений в сегмент разделяемой памяти. При инициализации счётчик доступных слотов устанавливается равным 6.

Второй семафор-счётчиком указывает число слотов, доступных для чтения.

Messages Exchange Example

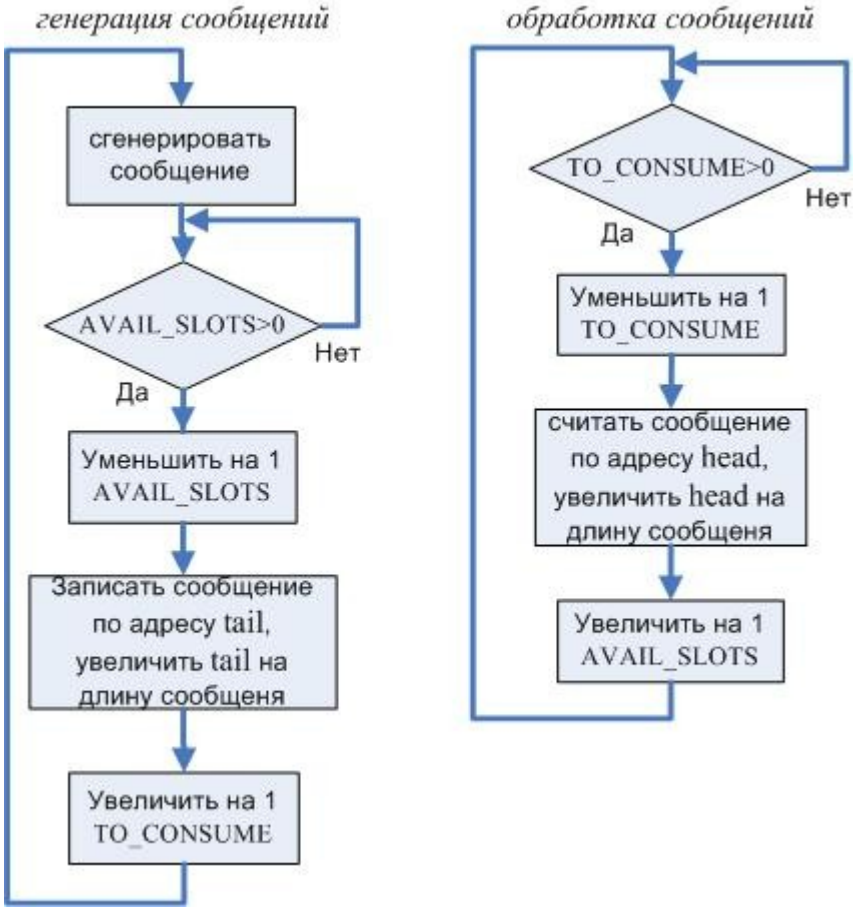


Процесс-parent создает и инициализирует разделяемый сегмент памяти и два семафора. Затем запускает вызовом fork() двух потомков: процесс-генератор (producer) и процесс-обработчик сообщений (consumer).

Messages Exchange Example

При запуске программы из командной строки задаются 2 аргумента.

Этими аргументами являются максимальное время ожидания процесса (*sleep time*) во время выполнения для процесса-генератора и для процесса-обработчика, соответственно.
Варьируя значения этих аргументов, можно моделировать взаимодействие генератора и обработчика, работающих с *разными скоростями*.



Messages Exchange Example

Исходный код примера содержится в файлах *parent.cpp, producer.cpp, consumer.cpp и local.h*.

Для снижения объёма кода используется локальный файл заголовка, *local.h*, содержащий все директивы include и объявления переменных, требуемых каждой из программ (*parent, producer и consumer*), составляющих данный пример

Thanks for your attention

Спасибо за внимание !

vladimir.shmakov.2012@gmail.com