Прикладные физико-технические и компьютерные методы исследований

Семинар 7

Повторение

- Классификация средств коммуникации

Очереди сообщений

- Сообщения имеют определённую структуру -> ясно, где начинаются, а где заканчиваются
- Двунаправленная передача между несколькими неродственными процессами
- Организации синхронизации процессов благодаря встроенным механизмам взаимоисключения и блокировки при чтении из пустого буфера и записи в переполненный буфер

Очереди сообщений

- Пространство имен. Адресация в System V IPC. Функция ftok(). (Что было для файлов, pipe`ов и FIFO?)
- Дескрипторы System V IPC. (Где хранятся? Сколько живут?)
- Реализованы в виде однонаправленных списков
- У каждого сообщения есть тип

Извлечение сообщений из очереди (receive)

- В порядке FIFO, независимо от типа сообщения.
- В порядке FIFO для сообщений конкретного типа.
- Первым выбирается сообщение с минимальным типом, не превышающим некоторого заданного значения, пришедшее ранее всех других сообщений с тем же типом.

Примитив **send**

- Обеспечивает скрытое от пользователя взаимоисключение во время помещения сообщения в очередь.
- Блокировку процесса при попытке добавить в очередь сообщение, в которой нет свободного места.

Примитив **receive**

- Обеспечивает скрытое от пользователя взаимоисключение во время получения сообщения из очереди.
- Блокировка процесса при попытке выполнить примитив над пустой очередью или очередью, в которой отсутствуют сообщения запрошенного типа.

Получение дескриптора очереди

- **int msgget(key_t key, int msgflg);**
- Все ли процессы могут добавлять и получать сообщения в очередь?

Отправка сообщения в очередь

- int msgsnd(int msqid, struct msgbuf *ptr, int length, int flag);
- struct msgbuf { // по умолчанию
  long mtype;
  char mtext[1];
  };
- struct MyMsgBuf {
  long mtype;
  char mtext[1024];
  } mybuf;

Отправка сообщения в очередь

- struct MyMsgBuf {

```
long messageType;
struct {
        int intInfo;
        float floatInfo;
} info;
} myBuf;
```

- **messageType > 0**
- **sizeof(myBuf.info) -> length <** 4080 в ОС Linux
- **flag = 0**, либо **IPC_NOWAIT**

Получение сообщений из очереди

- int msgrcv(int msqid, struct msgbuf *ptr, int length, long type, int flag);
- type = 0, либо +n, либо –n.
- В случае удачи, системный вызов копирует выбранное сообщение из очереди сообщений по адресу, указанному в параметре **ptr**, одновременно удаляя его из очереди сообщений.

Удаление очереди сообщений

- ipcs -l (просмотр максимальной длины сообщения в консоли)
- ipcs –q (просмотр очередей)
- ipcrm msg <IPC идентификатор>
- msgctl()

Пример пересылки текстовой информации из учебника

https://dl.dropboxusercontent.com/u/96739039/ex0.c

https://dl.dropboxusercontent.com/u/96739039/ex1.c

Упражнение 1

- Модифицируйте предыдущий пример для пересылки числовой информации
- Размещение данных в памяти с выравниванием на определенные адреса

```
struct {
    short sinfo;
    float finfo;
} info;
```

sizeof(info) >= sizeof(short) + sizeof(float)

Упражнение 2 (Двусторонняя связь через одну очередь)

- Отправляем число N
- Получаем N!
- Типы отправляемого и получаемого сообщения разные

Мультиплексирование сообщений.

- получение сообщений одним процессом от множества других процессов через одну очередь сообщений и отправку им ответов через ту же очередь сообщений

Модель клиент-сервер.

- Сервер, работает постоянно, на всем протяжении жизни приложения, а клиенты могут работать эпизодически.
- Сервер ждет запроса от клиентов, инициатором же взаимодействия выступает клиент.
- Клиент обращается к одному серверу за раз, в то время как к серверу могут одновременно поступить запросы от нескольких клиентов.
- Клиент должен знать, как обратиться к серверу (например, какого типа сообщения он воспринимает) перед началом организации запроса к серверу, в то время как сервер может получить недостающую информацию о клиенте из пришедшего запроса.

Упражнение 3 (если не успели на семинаре, доделываем дома)

- Реализация модели клиент-сервер
  - Клиенты посылают серверу два числа. В ответ получают их произведение.
  - Каждая задача на сервере выполняется в отдельном потоке/процессе.
  - Одновременно может исполняться не более N задач.