10. Взаимодействие процессов в ОС GNU/Linux

Разделы:

- Принципы и механизмы взаимодействия
- Совместно используемая память
- SystemV-семафоры
- Отображаемые на память файлы
- Конвейеры
- FIFO-файлы
- Сокеты

Принципы и механизмы взаимодействия

- Совместно используемая память
- Семафоры
- Отображаемая память
- Конвейеры
- FIFO-файлы
- Сокеты

Принципы и механизмы взаимодействия

- Различия определяются следующими критериями:
  - Ограничено ли взаимодействие рамками связанных процессов или соединяются процессы, которые выполняются в одной ФС или на разных хостах
  - Ограничен ли процесс *readonly*- или *writeonly*-операциями
  - Количество взаимодействующих процессов
  - Синхронизируются ли взаимодействующие процессы

Совместно используемая память

- Linux использует понятие виртуальной памяти, которая процессу выделяется страницами
- У процесса есть таблица страниц, в которой устанавливается соответствие между виртуальными и физическими адресами
- Разными процессам разрешается ссылаться на одни и те же страницы
- Это **разделяемая память**
- Размер разделяемого (совместно используемого) сегмента кратен **размеру страницы** виртуальной памяти (в Linux эта величина равна 4 Кб, но проверить это значение можно вызовом функции *getpagesize()*)
- Для использования функций работы с сегментами должны подключаться файлы *sys/shm.h* и *sys/stat.h*.

Совместно используемая память

- Операция выделения совместной памяти выполняется функцией *shmget()*
- Первый аргумент – целочисленный ключ, которым идентифицируется сегмент
- Если несвязанные процессы хотят получить доступ к одному и тому же сегменту, то они указывают одинаковый ключ
- Второй аргумент – размер сегмента (в байтах), который затем округляется, чтобы быть кратным размеру страницы виртуальной памяти

Совместно используемая память

- Операция выделения совместной памяти выполняется функцией *shmget()*
- Третий аргумент – набор битовых флагов:
  - *IPC_CREAT* – указывает на создание нового сегмента, которому присваивается заданный ключ
  - *IPC_EXCL* – всегда используется с *IPC_CREAT* и вынуждает функцию создать новый совместно используемый сегмент памяти либо вернуть идентификатор существующего сегмента, если такой ключ есть в системе
  - *IPC_PRIVATE* – всегда создавать новый сегмент
  - **Флаги режима** из 9 флагов, задающих права доступа для владельца, группы и остальных пользователей

```
int segId = shmget(shmKey, getpagesize()
                   , IPC_CREAT | S_IRUSR | S_IWUSR);
```

Совместно используемая память

- Для подключения используется функция *shmat()*
- Первый аргумент – идентификатор сегмента
- Второй – указатель на место в адресном пространстве процесса создается привязка на совместно используемую память
- Третий аргумент – флаги:
  - *SHM_RND*
  - *SHM_RDONLY*
- Для отключения используется *shmdt()*

Совместно используемая память

- Функция *shmctl()* возвращает информацию о совместно используемом сегменте, а также может модифицировать его
- Первый аргумент – идентификатор сегмента
- Для получения информации о сегменте в качестве второго аргумента передается *IPC_STAT*, а третий аргумент – указатель на структуру *shmid_ds*
- Для удаления сегмента в качестве второго аргумента передается *ICP_RMID*, а третий аргумент – *NULL*

Совместно используемая память

```
#include <stdio.h>
#include <sys/shm.h>
#include <sys/stat.h>
int main ()
{
  int segmentId;
  char* shared_memory;
  struct shmid_ds shmBuffer;
  int segmentSize;
  const int sharedSegmentSize = 0x6400;
  segmentId = shmget(IPC_PRIVATE, sharedSegmentSize
                     , IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
  sharedMemory = (char*)shmat(segment_id, 0, 0);
  printf("shared memory attached at address %p\n", sharedMemory);
  shmctl(segmentId, IPC_STAT, &shmBuffer);
  segmentSize = shmBuffer.shm_segsz;
  printf ("segment size: %d\n", segmentSize);
  sprintf(sharedMemory, "Hello, world.");
  shmdt(sharedMemory);

  sharedMemory = (char*)shmat(segmentId, (void*)0x5000000, 0);
  printf("shared memory reattached at address %p\n", sharedMemory);
  printf("%s\n", sharedMemory);
  shmdt(sharedMemory);

  shmctl(segmentId, IPC_RMID, 0);
  return 0;
}
```

Семафоры

- Для использования функций работы с сегментами должны подключаться файлы *sys/ipc.h*, *sys/sem.h*, *sys/types.h*.
- Выделение и освобождение семафоров выполняются функциями *semget()* и *semctl()*
- Аргументы:
  - ключ-идентификатор группы семафоров
  - количество семафоров в группе
  - флаги прав доступа
- Последний процесс явно удаляет группу функцией *semctl()* с идентификатором группы, числом семафоров, флагом *IPC_RMID* и любым значением типа *union semun*

Семафоры

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

union semun
{
  int val;
  struct semid_ds* buf;
  unsigned short int* array;
  struct seminfo* __buf;
};

int InitializeBinarySemaphore(int semId)
{
  union semun argument;
  unsigned short values[1];
  values[0] = 1;
  argument.array = values;
  return semctl(semId, 0, SETALL, argument);
}
```

Семафоры

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

union semun
{
  int val;
  struct semid_ds* buf;
  unsigned short int* array;
  struct seminfo* __buf;
};

int BinarySemaphoreAllocation(key_t key, int semFlags)
{
  return semget(key, 1, semFlags);
}

int BinarySemaphoreDeallocate(int semId)
{
  union semun ignoredArgument;
  return semctl(semId, 1, IPC_RMID, ignoredArgument);
}
```

Семафоры

- Установка и ожидание семафора реализуются системным вызовом *semop()*
- Первый аргумент – идентификатор группы
- Второй аргумент – массив значений *struct sembuf*, задающих выполняемые операции
- Третий аргумент – размер этого массива
- Поля структуры *sembuf*:
  - *sem_num* – номер в группе.
  - *sem_op* – число, обозначающее конкретную операцию
  - *sem_flg* – значение флага
    - Если он равен *IPC_NOWAIT*, то отключается блокировка операции
    - Если флаг равен *SEM_UNDO*, то ОС автоматически отменит заданную операцию по завершению процесса

Семафоры

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int BinarySemaphoreWait(int semId)
{
  struct sembuf operations[1];
  operations[0].sem_num = 0;
  operations[0].sem_op = -1;
  operations[0].sem_flg = SEM_UNDO;
  return semop(semId, operations, 1);
}

int BinarySemaphorePost(int semId)
{
  struct sembuf operations[1];
  operations[0].sem_num = 0;
  operations[0].sem_op = 1;
  operations[0].sem_flg = SEM_UNDO;
  return semop(semId, operations, 1);
}
```

Отображаемые на память файлы

- Функция *mmap()* предназначена для отображения обычного файла в памяти процесса

```
#include <sys/mman.h>
void* mmap(void* start // адрес начала отображаемой области
  , size_t length      // длина отображаемой области (в байтах)
  , int prot           // степень защиты отображаемых адресов
  , int flags          // флаги
  , int fd             // дескриптор открытого файла
  , off_t offset       // смещение от начала файла
);
```

- Третий аргумент задает степень защиты диапазона отображаемых адресов
- Может содержать объединение битовых констант *PROT_READ*, *PROT_WRITE* и *PROT_EXEC*

Отображаемые на память файлы

- Дополнительные флаги, задаваемые в четвертом аргументе:
  - *MAP_FIXED*. При наличии этого флага ОС Linux использует значение первого аргумента как точный адрес размещения отображаемого файла
  - *MAP_PRIVATE*. Изменения, вносимые в отображаемую память, записываются не в присоединенный файл, а в частную копию файла, принадлежащую процессу
  - *MAP_SHARED*. Изменения, вносимые в отображаемую память, немедленно фиксируются в файле, минуя буфер

Отображаемые на память файлы

- По окончании работы с отображаемым файлом его необходимо освободить с помощью функции *munmap()*

```
#include <sys/mman.h>
int munmap(void* start // начальный адрес
  , size_t length      // длина отображаемой области
);
```

- ОС при завершении программы автоматически освобождает отображаемые области

Отображаемые на память файлы

```
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <time.h>
#include <unistd.h>
#define FILE_LENGTH 0x100

int RandomRange(unsigned const low, unsigned const high)
{
  unsigned const range = high - low + 1;
  return low + (int) (((double) range) * rand() / (RAND_MAX + 1.0));
}

int main(int argc, char* const argv[])
{
  int fd;
  void* fileMemory;
  srand(time (NULL));
  fd = open(argv[1], O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
  lseek(fd, FILE_LENGTH+1, SEEK_SET);
  write(fd, "", 1);
  lseek(fd, 0, SEEK_SET);
  fileMemory = mmap(0, FILE_LENGTH, PROT_WRITE, MAP_SHARED, fd, 0);
  close (fd);
  sprintf((char*) fileMemory, "%d\n", RandomRange(-100, 100));
  munmap(fileMemory, FILE_LENGTH);
  return 0;
}
```

Отображаемые на память файлы

```
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <unistd.h>
#define FILE_LENGTH 0x100

int main(int argc, char* const argv[])
{
  int fd;
  void* fileMemory;
  int integer;

  fd = open(argv[1], O_RDWR, S_IRUSR | S_IWUSR);
  fileMemory = mmap(0, FILE_LENGTH, PROT_READ | PROT_WRITE,
               MAP_SHARED, fd, 0);
  close(fd);
  sscanf(fileMemory, "%d", &integer);
  printf("value: %d\n", integer);
  sprintf((char*) fileMemory, "%d\n", 2 * integer);
  munmap(file_memory, FILE_LENGTH);
  return 0;
}
```

Отображаемые на память файлы

- Функция переноса содержимого буфера в дисковый файл:

```
#include <sys/mman.h>
int msync(mem_addr    //адрес начала отображаемой области
  , mem_length        //длина отображаемой области
  , MS_SYNC | MS_INVALIDATE
);
```

- Третий параметр может содержать следующие флаги:
  - *MS_ASYNC*
  - *MS_SYNC*
  - *MS_INVALIDATE*
- Пример вызова:

```
msync(memAddr, memLength, MS_SYNC | MS_INVALIDATE);
```

Конвейеры

```
int pipe_fds[2];
int read_fd;
int write_fd;

pipe(pipe_fds);
read_fd = pipe_fds[0];
write_fd = pipe_fds[1];
```

Конвейеры

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

void writer(const char* message, int count, FILE* stream)
{
  for (; count > 0; --count)
  {
    fprintf(stream, "%s\n", message);
    fflush(stream);
    sleep(1);
  }
}
void reader(FILE* stream)
{
  char buffer[1024];
  while (!feof(stream) && !ferror(stream)
    && fgets(buffer, sizeof (buffer), stream) != NULL
   )
    fputs(buffer, stdout);
}
```

Конвейеры

```
int main()
{
  int fds[2];
  pid_t pid;
  pipe(fds);
  pid = fork();
  if (pid == (pid_t) 0)
  {
    FILE* stream;
    close(fds[1]);
    stream = fdopen(fds[0], "r");
    reader(stream);
    close(fds[0]);
  }
  else
  {
    FILE* stream;
    close(fds[0]);
    stream = fdopen(fds[1], "w");
    writer("Hello, goodbye and farewell!", 4, stream);
    close(fds[1]);
  }
return 0;
}
```

Конвейеры

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
int main()
{
  int fds[2];
  pid_t pid;
  pipe(fds);
  pid = fork();
  if (pid == (pid_t)0)
  {
    close(fds[1]);
    dup2(fds[0], STDIN_FILENO);
    execlp("sort", "sort", 0);
  }
...
```

Конвейеры

```
...
  else
  {
    FILE* stream;
    close(fds[0]);
    stream = fdopen (fds[1], "w");
    fprintf(stream, "3. I don't know what it is that I like about
   you, but I like it a lot.\n");
    fprintf(stream, "1. Hey, girl, stop what you're doin'!\n");
    fprintf(stream, "5. Communication Breakdown, It's always the
   same,\n");
    fprintf(stream, "2. Hey, girl, you'll drive me to ruin.\n");
    fprintf(stream, "6. I'm having a nervous breakdown, Drive me
   insane!\n");
    fprintf(stream, "4. Won't let me hold you, Let me feel your
   lovin' charms.\n");
    fflush (stream);
    close (fds[1]);
    waitpid(pid, NULL, 0);
  }

  return 0;
}
```

Конвейеры

```
#include <stdio.h>
#include <unistd.h>

int main()
{
  FILE* stream = popen("sort", "w");
  fprintf(stream, "3. I don't know what it is that I like about you, but I like it a
    lot.\n");
  fprintf(stream, "1. Hey, girl, stop what you're doin'!\n");
  fprintf(stream, "5. Communication Breakdown, It's always the same,\n");
  fprintf(stream, "2. Hey, girl, you'll drive me to ruin.\n");
  fprintf(stream, "6. I'm having a nervous breakdown, Drive me insane!\n");
  fprintf(stream, "4. Won't let me hold you, Let me feel your lovin' charms.\n");

  return pclose(stream);
}
```

FIFO-файлы

- Файл FIFO можно создать с помощью функции *mkfifo()*

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char *path, mode_t mode);
```

- Пример вызова:

```
int status = mkfifo("/home/user/abrakadabra"
          , S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH);
```

FIFO-файлы

```
/* низкий уровень */
int fd = open(fifoPath, O_WRONLY);
write(fd, dataBuffer, dataLength);
close(fd);
-----------------------------------
/* высокий уровень */
FILE fifo = fopen(fifoPath, "r");
fscanf(fifo, "%s", dataBuffer);
fclose(fifo);
```

Сокеты

- При взаимодействии *с установлением соединения* гарантируется доставка и оригинальный порядок пакетов
- Сокеты называются *потоковыми*

- При передаче *дейтаграмм* (*без установки соединения*) не гарантируется доставка и порядок пакетов
- Сокеты называются *дейтаграммными*

Сокеты

- При работе с сокетами используются следующие функции:
  - *socket()* — создает сокет;
  - *close()* — уничтожает сокет;
  - *connect()* — устанавливает соединение между двумя сокетами;
  - *bind()* — назначает серверному сокету адрес;
  - *listen()* — переводит сокет в режим приема запросов на подключение;
  - *accept()* — принимает запрос на подключение и создает новый сокет, который будет обслуживать данное соединение;
  - *write()* — для отправки данных;
  - *read()* — для получения данных

Сокеты

- Константы, определяющие пространство имен, начинаются с префикса *AF_*
- Константы *AF_LOCAL* и *AF_UNIX* соответствуют локальному пространству имен
- Константа *AF_INET* — пространству имен Internet

- Константы, определяющие тип взаимодействия, начинаются с префикса *SOCK_*
- Сокетам, ориентированным на установку соединения, соответствует константа *SOCK_STREAM*, а дейтаграммным сокетам — константа *SOCK_DGRAM*

Сокеты

- Жизненный цикл «сервера»:
  1. создание сокета, ориентированного на соединения, с помощью *socket()*;
  2. назначение сокету адреса привязки функцией *bind()*;
  3. перевод сокета в режим ожидания запросов с помощью *listen()*;
  4. прием поступающих запросов на подключение функцией *accept()*;
  5. чтение пакетов/дейтаграмм с помощью функций *read()* или *recv()*;
  6. закрытие сокета функцией *close()*.

Сокеты

- Чтобы прочитать данные из сокета, не удалив их из входящей очереди, можно использовать функцию *recv()*
- Она принимает те же аргументы, что и функция *read()*, а также дополнительный аргумент – *flags*
- Флаг *MSG_PEEK* задает режим неразрушающего чтения, при котором прочитанные данные остаются в очереди

```
#include <sys/types.h>
#include <sys/socket.h>

int recv(int s, void *buf, size_t len, int flags);
```

Сокеты

- Имя сокета задается в структуре типа *sockaddr_un* (из <*sys/un.h*> и <*unistd.h*>)
- В поле *sun_family* необходимо записать константу *AF_LOCAL*
- Поле *sun_path* содержит полное имя файла и не может превышать 108 байтов
- Длина структуры *sockaddr_un* вычисляется с помощью макроса *SUN_LEN*
- Допускается любое имя файла, но процесс должен иметь право записи в каталог, где находится файл
- При подключении к сокету процесс должен иметь право чтения файла
- Только процессам, работающим в пределах одного хоста, разрешается взаимодействовать друг с другом посредством локальных сокетов

Сокеты

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
int server(int clientSocket)
{
  while (!0)
  {
    int length;
    char* text;
    if (0 == read(clientSocket, &length, sizeof (length)))
      return 0;
    text = (char*) malloc(length);
    read(clientSocket, text, length);
    printf("%s\n", text);
    if (0 == strcmp(text, "adios amigo"))
    {
      free(text);
      return !0;
    }
    free(text);
  }
  return 0;
}
```

Сокеты

```
int main(int argc, char* const argv[])
{
  if (argc < 2)
  {
    fprintf(stderr, "Too few parameters.\n");
    return EXIT_FAILURE;
  }
  const char* const socketName = argv[1];
  int socketFileDescriptor;
  struct sockaddr_un name;
  int clientSentQuitMessage;
  socketFileDescriptor = socket(AF_LOCAL, SOCK_STREAM, 0);
  name.sun_family = AF_LOCAL;
  strcpy (name.sun_path, socketName);
  bind(socketFileDescriptor, (const struct sockaddr *)&name, SUN_LEN(&name));
  listen(socketFileDescriptor, 4);
  do
  {
    struct sockaddr_un clientName;
    socklen_t clientNameLength;
    int clientSocketFileDescriptor;
    clientSocketFileDescriptor = accept(socketFileDescriptor
                  , (__SOCKADDR_ARG)&clientName, &clientNameLength);
    clientSentQuitMessage = server(clientSocketFileDescriptor);
    close(clientSocketFileDescriptor);
  } while (!clientSentQuitMessage);
  close(socketFileDescriptor);
  unlink(socketName);
  return 0;
}
```

Сокеты

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
void WriteText(int socketFileDescriptor, const char* text)
{
  int length = strlen(text)+1;
  write(socketFileDescriptor, &length, sizeof (length));
  write (socketFileDescriptor, text, length);
}
int main(int argc, char* const argv[])
{
  if (argc != 3)
  {
    fprintf(stderr, "Socketname and text messages expected.\n");
    return EXIT_FAILURE;
  }
  const char* const socket_name = argv[1];
  const char* const message = argv[2];
  int socketFileDescriptor;
  struct sockaddr_un name;
  socketFileDescriptor = socket(AF_LOCAL, SOCK_STREAM, 0);
  name.sun_family = AF_LOCAL;
  strcpy(name.sun_path, socket_name);
  connect(socketFileDescriptor, (const struct sockaddr *)&name, SUN_LEN (&name));
  WriteText(socketFileDescriptor, message);
  close(socketFileDescriptor);
  return 0;
}
```

Сокеты

- Адрес Internet-сокета состоит из двух частей: адреса хоста и номера порта
- Эта информация хранится в структуре типа *sockaddr_in*
- В поле *sin_family* необходимо указать константу *AF_INET*
- В поле *sin_addr* хранится IP-адрес хоста в виде 32-битного целого числа
- Благодаря номерам портов можно различать сокеты, создаваемые на одном хосте
- В разных системах многобайтные значения могут храниться с разным порядком следования байтов, поэтому с помощью *htons()* необходимо преобразовать номер порта в число с *сетевым порядком следования байтов*

Сокеты

```
#include <stdlib.h>
#include <stdio.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/socket.h>
#include <unistd.h>
#include <string.h>

void GetHomePage(int socketFileDescriptor)
{
  char buffer[10000];
  ssize_t numberCharactersRead;
  sprintf(buffer, "GET /\n");
  write(socketFileDescriptor, buffer, strlen(buffer));
  while (!0)
  {
    numberCharactersRead = read(socketFileDescriptor, buffer, 10000);
    if (numberCharactersRead == 0)
      return;
    fwrite(buffer, sizeof (char), numberCharactersRead, stdout);
  }
}
```

Сокеты

```
int main(int argc, char* const argv[])
{
  int socketFileDescriptor;
  struct sockaddr_in name;
  struct hostent* hostinfo;
  socketFileDescriptor = socket(AF_INET, SOCK_STREAM, 0);
  name.sin_family = AF_INET;
  hostinfo = gethostbyname(argv[1]);
  if (hostinfo == NULL)
    return 1;
  else
    name.sin_addr = *((struct in_addr *) hostinfo->h_addr);
  name.sin_port = htons(80);
  if (connect(socketFileDescriptor, (const struct sockaddr *)&name
              , sizeof (struct sockaddr_in)) == -1
             )
  {
    perror("connect");
    return !0;
  }
  GetHomePage(socketFileDescriptor);
return 0;
}
```

Пары сокетов

- Функция *socketpair()* создает два дескриптора для двух родственных сокетов, находящихся на одном хосте
- Первые три параметра функции *socketpair()* такие же, как и у функции *socket()*, т.е. пространство имен (всегда *AF_LOCAL*), тип взаимодействия и протокол
- Четвертый параметр – это массив из двух целых чисел, в которые записываются дескрипторы сокетов, подобно функции *pipe()*

See also

- Delve into UNIX process creation - http://www.ibm.com/developerworks/aix/library/au-unixproce
- LXF83:Unix API - http://wiki.linuxformat.ru/index.php/LXF83:Unix_API
- Лав, Р. Linux. Системное программирование/ Р.Лав. – СПб.: Питер, 2008. – 416 с.
- Программирование сокетов в Linux - http://gzip.rsdn.ru/article/unix/sockets.xml
- Протокол IP - http://www.citforum.ru/internet/tifamily/ipspec.shtml
- Протокол TCP - http://www.citforum.ru/internet/tifamily/tcpspec.shtml
- Протокол UDP - http://www.citforum.ru/internet/tifamily/udpspec.shtml

See also

- Развитие стека TCP/IP: протокол IPv.6 - http://www.citforum.ru/nets/ip/glava_9.shtml
- Стек протоколов TCP/IP - http://www.agpu.net/fakult/ipimif/fpiit/kafinf/umk/el_lib/calc_system/lab_work_net/kulgin_3.htm
- Руководство программиста для Linux - http://citforum.ru/operating_systems/linux_pg/lpg_03.shtml
- byte order - http://encyclopedia2.thefreedictionary.com/byte+or