ОС UNIX, лекция 7: схема серверного приложения

Пожидаев М. С.

10 марта 2020 г.

Создание серверного сокета

1. `socket()` — создание сокета.
2. `bind()` — назначение сокету порта TCP.
3. `listen()` — перевод сокета в режим ожидания соединений.
4. `accept()` — ожидание и принятие входящего соединения.

Код создания серверного сокета

```
int fd;
fd=socket(AF_INET, SOCK_STREAM, 0);
struct sockaddr_in addr;
bzero(&addr, sizeof(struct sockaddr_in));
addr.sin_family=AF_INET;
addr.sin_addr.s_addr=htonl(INADDR_ANY);
addr.sin_port=htons(the_port_you_want);
bind(fd, (struct sockaddr*)&addr, sizeof(struct sockaddr_in));
 listen (fd, 512);
```

Порядок работы

1. Создание сокета.
2. Регистрация обработчика сигнала.
3. Блокирование сигнала.
4. Выполнение основного цикла приложения, в котором ключевую роль играет функция `pselect()`.
5. Завершение работы.

Структура основного цикла

1. Подготовка списка файловых дескрипторов, в который должны войти основной сокет для приёма новых соединений и все установленные соединения.
2. Вызов функции `pselect()`, которая временно разблокирует необходимый сигнал и дождётся одного из трёх событий:
   - получение сигнала;
   - новый запрос на установление соединения;
   - новые данные в установленном соединении.
3. Производится обработка произошедшего события в зависимости от логики приложения.
4. Цикл повторяется для выполнения нового вызова функции `pselect()`.

Порядок работы функции pselect()

1. Временное восстановление блокированного сигнала.
2. Ожидание изменений состояния файловых дескрипторов или прерывания из-за получения сигнала с выполнением его обработчика.
3. Повторное блокирование сигнала вне зависимости от причины завершения работы.

Фрагмент описания функции pselect()

The reason that `pselect()` is needed is that if one wants to wait for either a signal or for a file descriptor to become ready, then an atomic test is needed to prevent race conditions. (Suppose the signal handler sets a global flag and returns. Then a test of this global flag followed by a call of `select()` could hang indefinitely if the signal arrived just after the test but just before the call. By contrast, `pselect()` allows one to first block signals, handle the signals that have come in, then call `pselect()` with the desired sigmask, avoiding the race.)

Объявление обработчика сигнала

```
volatile sig_atomic_t wasSigHup = 0;

void sigHupHandler(int r)
{
  wasSigHup = 1;
}
```

Регистрация обработчика сигнала

```
struct sigaction sa;
 sigaction (SIGHUP, NULL, &sa);
sa.sa_handler = sigHupHandler;
sa.sa_flags |= SA_RESTART;
 sigaction (SIGHUP, &sa, NULL);
```

Блокировка сигнала

```
sigset_t  blockedMask;
sigemptyset(&blockedMask);
sigaddset(&blockedMask, SIGHUP);
sigprocmask(SIG_BLOCK, &blockedMask, &origMask);
```

Работа основного цикла

```
fd_set fds;
FD_ZERO(&fds);
FD_SET(socket, &fds);
for( clientIt = clients.begin(); clientIt != clients.end(); clientIt ++)
  FD_SET(*clientIt, &fds);
if (pselect(maxFd + 1, &fds, NULL, NULL, NULL, origSigMask) == -1) {
  if (errno == EINTR)
    //some actions on receiving the signal
//for the main socket and for every established connection
if (FD_ISSET(fd, &fds))
//some actions on the descriptor activity
```

Пример альтернативного решения

Функция `signalfd()` — трансляция приходящих сигналов в файловый дескриптор.

1. Сигналы для `signalfd()` должны быть также заблокированы через вызов `sigprocmask()`.
2. Полученный файловый дескриптор можно также добавлять в вызовы `select()`.

Потоки POSIX

Библиотека `pthread` — переносимая реализация потоков для UNIX-подобных систем.

1. Поддерживается использование общего адресного пространства и прочих ресурсов, обычно изолируемых процессами (например, текущий каталог).
2. Потоки получают только собственный стек и состояние регистров.
3. Существенно сокращается время на переключение потоков при разделении времени процессора и расходование оперативной памяти.

Функции pthread

- `pthread_create()` — создание потока;
- `pthread_exit()` — завершить работу потока;
- `pthread_join()` — дождаться завершения потока;
- `pthread_mutex_lock()` и `pthread_mutex_unlock()` — операции с мьютексами;
- `pthread_cond_signal()`, `pthread_cond_broadcast()` и `pthread_cond_wait()` — операции с условной переменной.

Спасибо за внимание!

Веб-сайт: http://marigostra.ru/
E-mail: msp@luwrain.org