Komunikacja międzywątkowa i międzyprocesowa

Marcin Kołodziej
<marcin.kolodziej1@globallogic.com>

Agenda

1. Mechanizmy komunikacji międzyprocesowej
   a. Potoki (ang. *Pipes*)
   b. Kolejki (ang. *Message queues*)
   c. Pamięć współdzielona (ang. *Shared memory*)
   d. Gniazda (ang. *Sockets*)
2. Boost
   a. `boost::interprocess`
   b. `boost::asio` (część odpowiedzialna za sockety TCP/UDP)
   c. `boost::process`
3. Valgrind
   a. Helgrind
   b. DRD

`man (manual)`

Najlepszy przyjaciel programisty linux

Przydatne strony

```
man 7 pipe          // potoki
man 7 fifo          // nazwane potoki
man mq_overview     // kolejki
man shm_overview    // pamięć współdzielona
man 7 socket        // gniazda
man 7 exec          // podmienianie procesów
```

Mechanizmy komunikacji międzyprocesowej

Teoria

- Istnieje kilka różnych implementacji komunikacji międzyprocesowej:
  - System V
  - POSIX
  - BSD
- Każdy obiekt IPC posiada unikalny identyfikator, który używany jest w jądrze do dostępu.
- Aplikacje z userspace'a korzystają z interfejsów wystawionych przez jądro systemu pod postacią specjalnych plików.
- Do przeglądania zasobów IPC służy komenda `ipcs`.

Potoki

Teoria

- Potoki i FIFO (nazwane potoki) pozwalają na jednokierunkową komunikację międzyprocesową.
- Domyślnie operacje zapisu oraz odczytu są blokujące, ale można zmienić ustawienia na nieblokujące przez otwarcie z flagą `O_NONBLOCK`.
- Odczyt i zapis odbywa się poprzez wywołania *read(2)* oraz *write(2)* typowe dla operacji na plikach.
- W przypadku próby zapisu do potoku, który nie jest odczytywany przez żaden proces następuje emisja sygnału `SIGPIPE`.
- Potoki nienazwane możemy znaleźć katalogu `/proc/<pid>/fd/`.

Funkcje do obsługi potoków

```
#include <unistd.h>
int pipe(int fd[2]); // flags == 0
int pipe(int fd[2], int flags);
```

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

Konfiguracja potoków

- `/proc/sys/fs/pipe-max-size`
  Maksymalny rozmiar (w bajtach), który użytkownik może nadać dla stworzonego potoku.

- `/proc/sys/fs/pipe-user-pages-hard`
  `/proc/sys/fs/pipe-user-pages-soft`
  Maksymalny sumaryczny rozmiar (w stronach) wszystkich potoków stworzonych przez nieuprzywilejowanego użytkownika. Dostępne od wersja 4.5 jądra.

Kolejki

Teoria

- Kolejki wiadomości są bardzo podobne do nazwanych potoków, ale:
  - Posiadają wewnętrzną strukturę (rozmiar, priorytet).
  - Mogą mieć różny priorytet odczytu.
  - W trakcie tworzenia kolejki można ustalić dokładną maksymalną liczbę wiadomości oraz ich rozmiar.
  - Proces jest w stanie odczytać stan kolejki.
- Reszta właściwości (m.in. czytanie jak pliki) jest taka sama.
- Aby funkcje `mq_*` działały w tle musi działać `Mqueue`.
- Kolejki wiadomości są tworzone w wirtualnym systemie plików w lokalizacji `/dev/mqueue`.
- Stworzone kolejki istnieją do momentu ich ręcznego usunięcia lub wyłączenia systemu.
- Podczas kompilacji trzeba dołączyć bibliotekę `rt` (`-lrt`).

Funkcje do obsługi kolejek wiadomości

Funkcje biblioteczne

```
mq_close(3)
mq_getattr(3)
mq_notify(3)
mq_open(3)
mq_receive(3)
mq_send(3)
mq_setattr(3)
mq_timedreceive(3)
mq_timedsend(3)

mq_unlink(3)
```

Wywołania systemowe

```
close(2)
mq_getsetattr(2)
mq_notify(2)
mq_open(2)
mq_timedreceive(2)
mq_timedsend(2)
mq_getsetattr(2)
mq_timedreceive(2)
mq_timedsend(2)

mq_unlink(2)
```

Konfiguracja kolejek wiadomości

- `/proc/sys/fs/mqueue/msg_default`
  Domyślna wartość `mq_maxmsq` dla nowych kolejek.
- `/proc/sys/fs/mqueue/msg_max`
  Maksymalna wartość `mq_maxmsq` dla nowych kolejek.
- `/proc/sys/fs/mqueue/msgsize_default`
  Domyślna wartość `mq_msgsize` dla nowych kolejek.
- `/proc/sys/fs/mqueue/msgsize_max`
  Maksymalna `mq_msgsize` dla nowych kolejek.
- `/proc/sys/fs/mqueue/queues_max`
  Maksymalna ilość kolejek w systemie.

Pamięć współdzielona

Teoria

- Pamięć współdzielona to specjalnie wyznaczony obszar pamięci procesu, który może być udostępniany jak plik innym procesom.
  - Możliwość nadawania praw dostępu.
  - Dostęp przy pomocy `open`, `close`, `read`, `write`.
  - Może być też mapowana na wirtualny adres procesu, dzięki czemu mamy dostęp do całości przez zwykłe wskaźniki.
- Same z siebie nie zapewniają bezpiecznego dostępu z wielu wątków.
- Obszary pamięci współdzielonej są tworzone w wirtualnym systemie plików (przeważnie /dev/shm) i istnieją do czasu ręcznego usunięcia po odmapowaniu przez wszystkie procesy lub wyłączenia systemu.
- Najszybsza z form komunikacji międzyprocesowej.
- Podczas kompilacji trzeba dołączyć bibliotekę `rt` (`-lrt`).

Funkcje do obsługi pamięci współdzielonej

- `shm_open(3)` // otwarcie obiektu pamięci współdzielonej
- `ftruncate(2)` // zmiana rozmiaru obiektu shm
- `mmap(2)` // mapowanie do pamięci wirtualnej
- `munmap(2)` // usuwanie z mapy pamięci wirtualnej
- `msync(2)` // synchronizacja pamięci z systemem plików
- `mprotect(2)` // ustawia ochronę regionu pamięci
- `shm_unlink(3)` // usunięcie obiektu pamięci współdzielonej
- `close(2)` // zamknięcie deskryptora pliku
- `fstat(2)` // odczytanie informacji o pliku
- `fchown(2)` // zmiana właściciela pliku
- `fchmod(2)` // zmiana uprawnień do pliku

Gniazda

Teoria

- Tworząc gniazda wybieramy domenę protokołów oraz typ gniazda.
- Operujemy na zwróconym przez funkcję deskryptorze pliku, który opisuje punkt końcowy protokołu.
- Istnieje wiele połączeń protokołów i typu, ale zostaną omówione tylko niektóre.
- Całość znajduje się w bibliotece `<sys/socket.h>`.

Funkcje do obsługi gniazd

- `socket(2)  // tworzenie gniazda`
- `connect(2) // łączenie deskryptora do adresu zdalnego`
- `bind(2)    // łączenie deskryptora do adresu lokalnego`
- `listen(2)  // nasłuchiwanie połączenia na gnieździe`
- `accept(2)  // zaczyna akceptować połączenia na gnieździe`
- `send(2), sendto(2), sendmsg(2)`
- `recv(2), recvfrom(2), recvmsg(2)`
- `poll(2), select(2) // czeka aż deskryptor będzie gotów`
- `getsockname(2)     // zwraca adres przywiązania gniazda`
- `getpeername(2)     // zwraca adres połączonego klienta`
- `getsockopt(2), setsockopt(2) // opcje gniazda`
- `close(2), shutdown(2)        // zamykanie gniazda`

Obsługiwane domeny protokołów

- `AF_UNIX, AF_LOCAL` – komunikacja lokalna
- `AF_INET` – IP w wersji 4
- `AF_INET6` – IP w wersji 6
- `AF_IPX` – IPX - protokoły Novell
- `AF_NETLINK` – komunikacja między jądrem a userspace
- `AF_X25` – ITU-T X.25 / ISO-8208
- `AF_AX25` – Amateur radio AX.25
- `AF_ATMPVC` – dostęp do “surowych” ATM PVCs
- `AF_APPLETALK` – AppleTalk
- `AF_PACKET` – niskopoziomowy interfejs pakietowy
- `AF_ALG` – interfejs do funkcji kryptograficznych jądra

Rodzaje gniazd

- `SOCK_STREAM` – sekwencjonowany, rzetelny, dwukierunkowy strumień bajtowy
- `SOCK_DGRAM` – bezpołączeniowy, nierzetelny, o ustalonej wielkości maksymalnej
- `SOCK_SEQPACKET` – sekwencjonowane, rzetelne, dwukierunkowe połączenie dla datagramów o ustalonej wielkości maksymalnej
- `SOCK_RAW` – “surowy” protokół sieciowy
- `SOCK_RDM` – bezpołączeniowy, rzetelny, protokół nie gwarantujący sekwencyjności
- `SOCK_PACKET` – nieużywany

Biblioteka boost

boost

- `boost` - kolekcja bibliotek programistycznych dodająca warstwę abstrakcji do języka C++, ułatwiająca korzystanie z wielu złożonych mechanizmów.
- `boost::interprocess` - część odpowiadająca z komunikację międzyprocesową, przede wszystkim:
  - pamięć współdzieloną
  - pliki mapowane do pamięci
  - kolejki wiadomości
  - mechanizmy synchronizacji
- `boost::process` - nowość w boost_1.64, udostępnia warstwę abstrakcji na tworzenie procesów z poziomu C++.

Elementy boost::interprocess

1. Współdzielenie pamięci
   `shared_memory_object, file_mapping`
2. Mechanizmy synchronizacji
   `interprocess_mutex, named_mutex, scoped_lock`
3. Zmienne warunkowe
   `interprocess_condition, named_condition`
4. Semafory
   `interprocess_semaphore, named_semaphore`
5. Kolejka wiadomości
   `message_queue`

Elementy boost::process

1. Uruchamianie procesów
   `boost::process::system, boost::process::spawn`
2. Monitorowanie procesów potomnych
   `boost::process::child`
3. Komunikacja synchroniczna / asynchroniczna
4. Grupowanie procesów
   `boost::process::group`
5. Edycja zmiennych środowiskowych
   `boost::process::environment`

Pytania?

Bibliografia

- http://www.tldp.org
- http://man7.o
- rg/linux/man-pages/index.html
- http://www.boost.org/doc/libs/1_64_0/doc/html/interprocess.html
- http://www.boost.org/doc/libs/1_64_0/doc/html/process.html
- http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2006/n2044.html
- https://github.com/lzoslav/wizut-ipc