

ОСНОВЫ ОПЕРАЦИОННЫХ СИСТЕМ

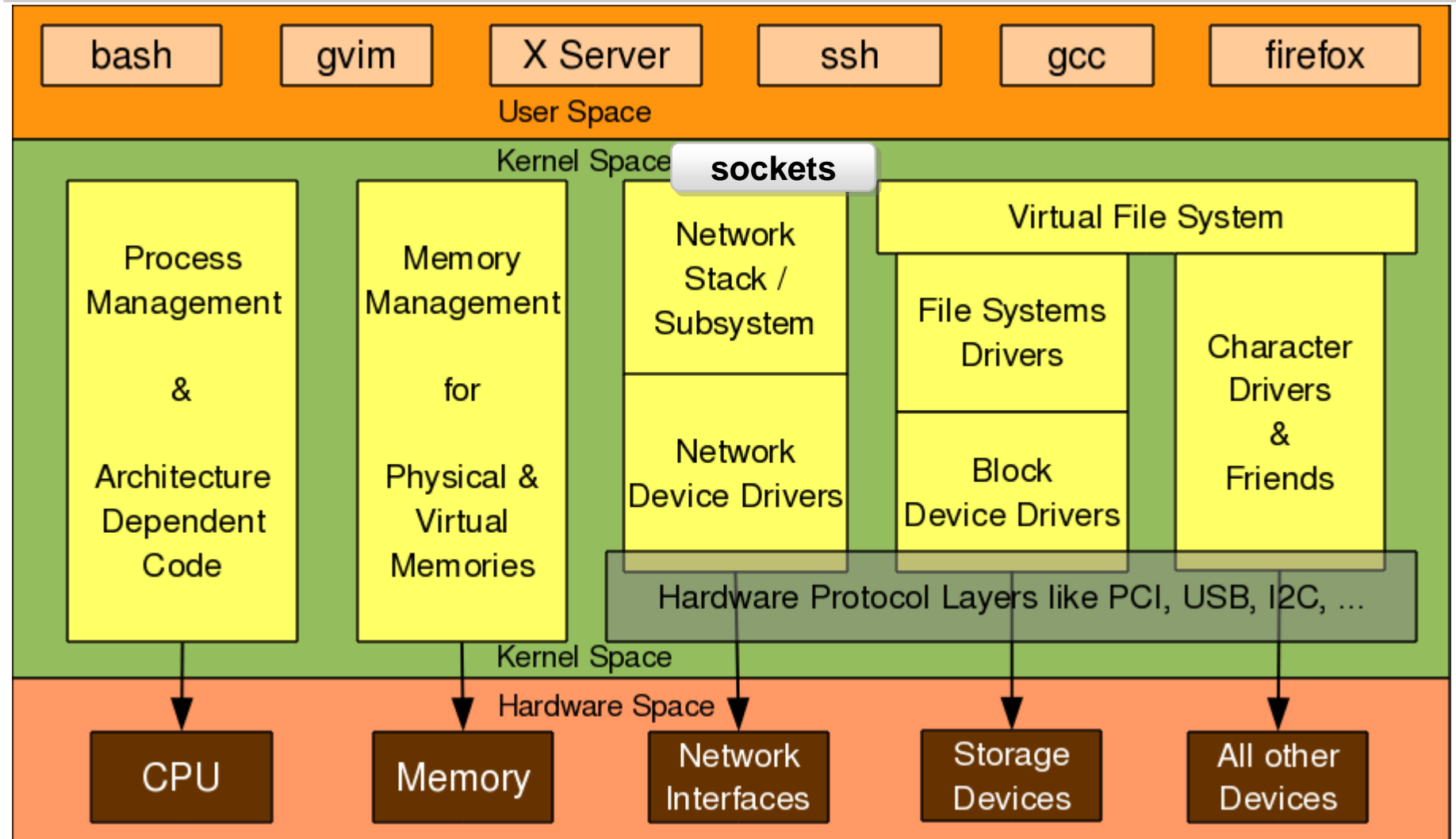
Ефанов Николай Николаевич,
кандидат физико-математических наук, доцент

Тема 13

Сети и сетевые операционные системы

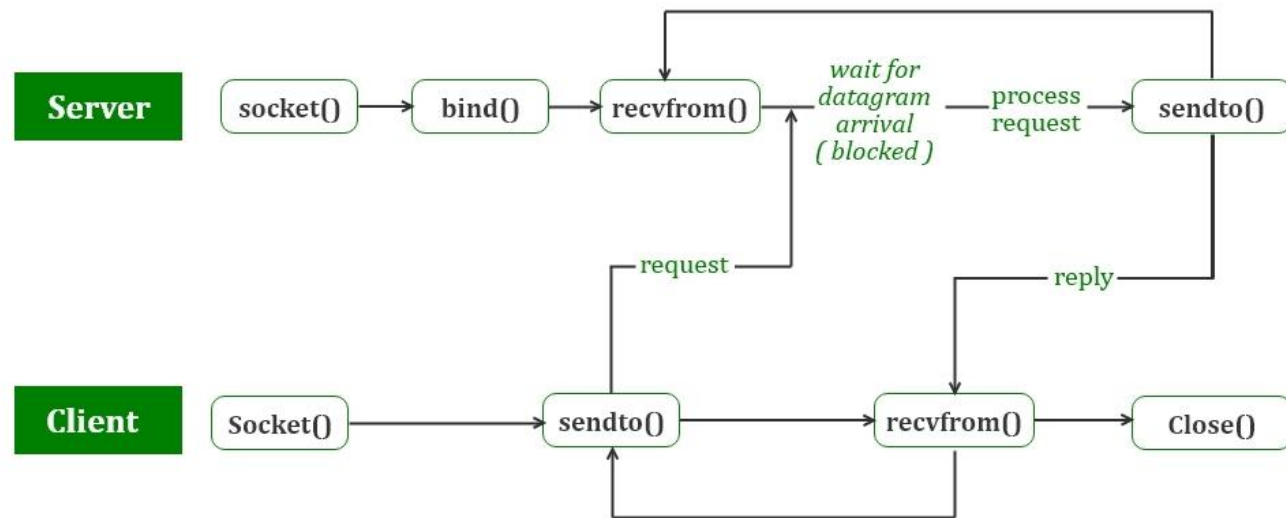
Сети и сетевые ОС

Вспомогательный рисунок



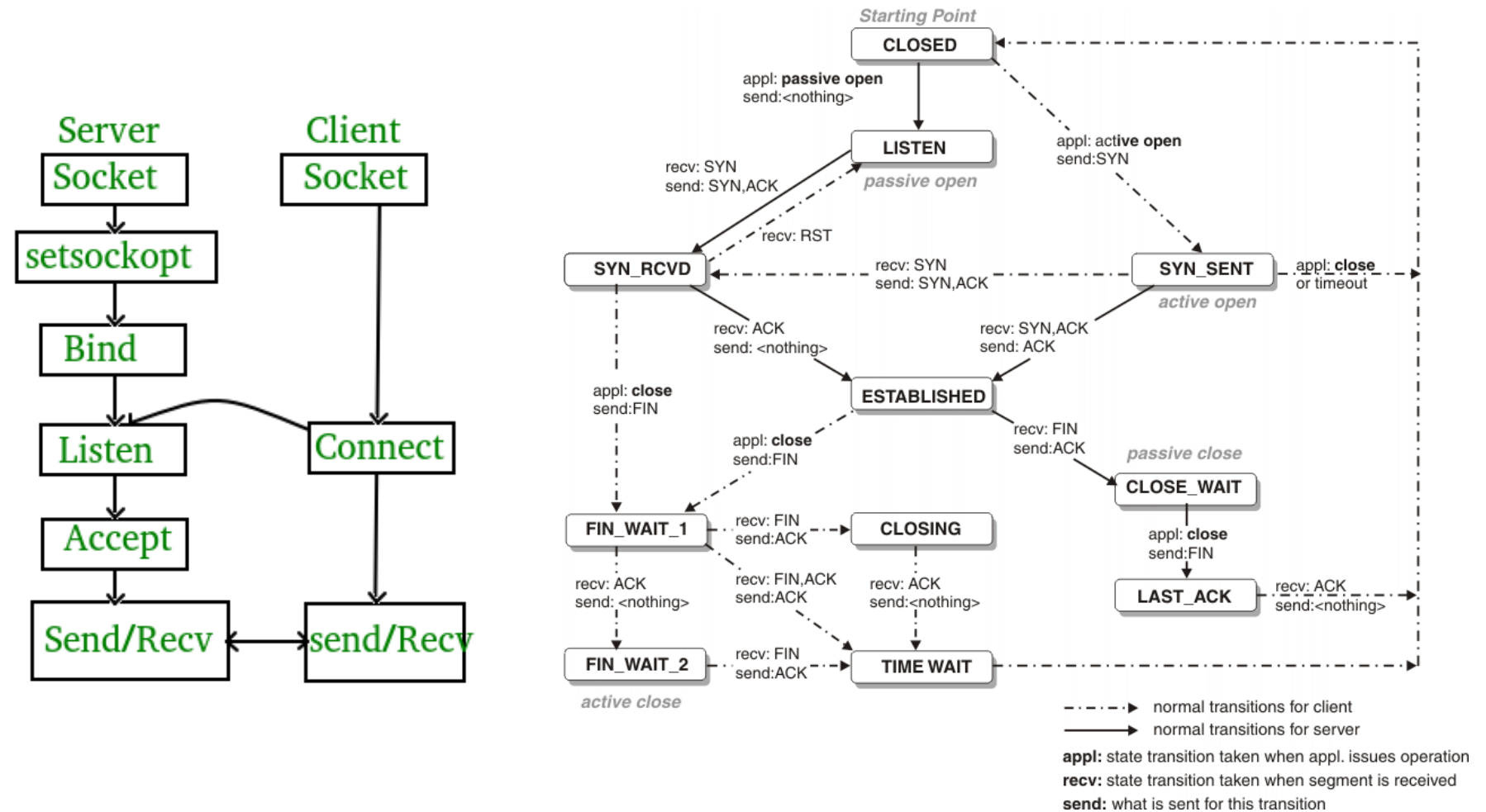
Сети и сетевые ОС

Интерфейсы: Сокеты UDP



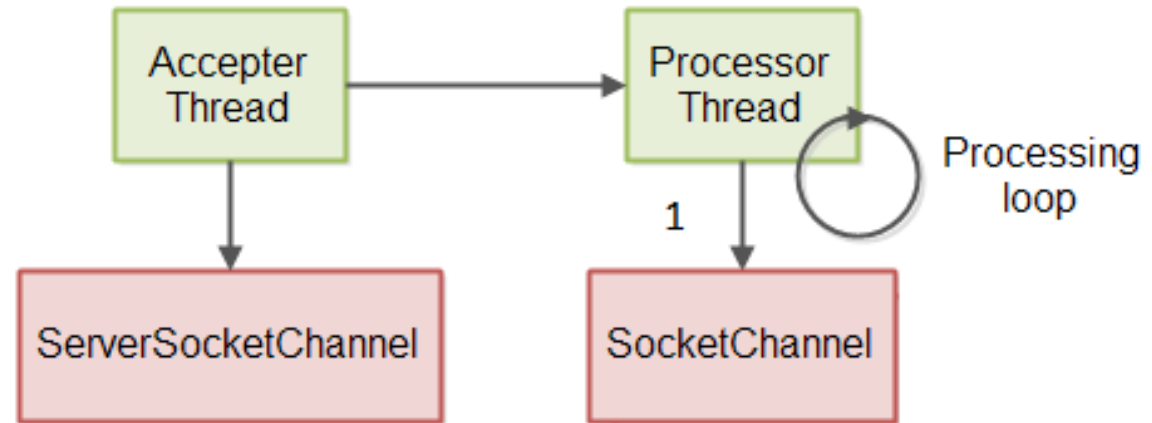
Сети и сетевые ОС

Интерфейсы: Сокеты и протокол TCP



Сети и сетевые ОС

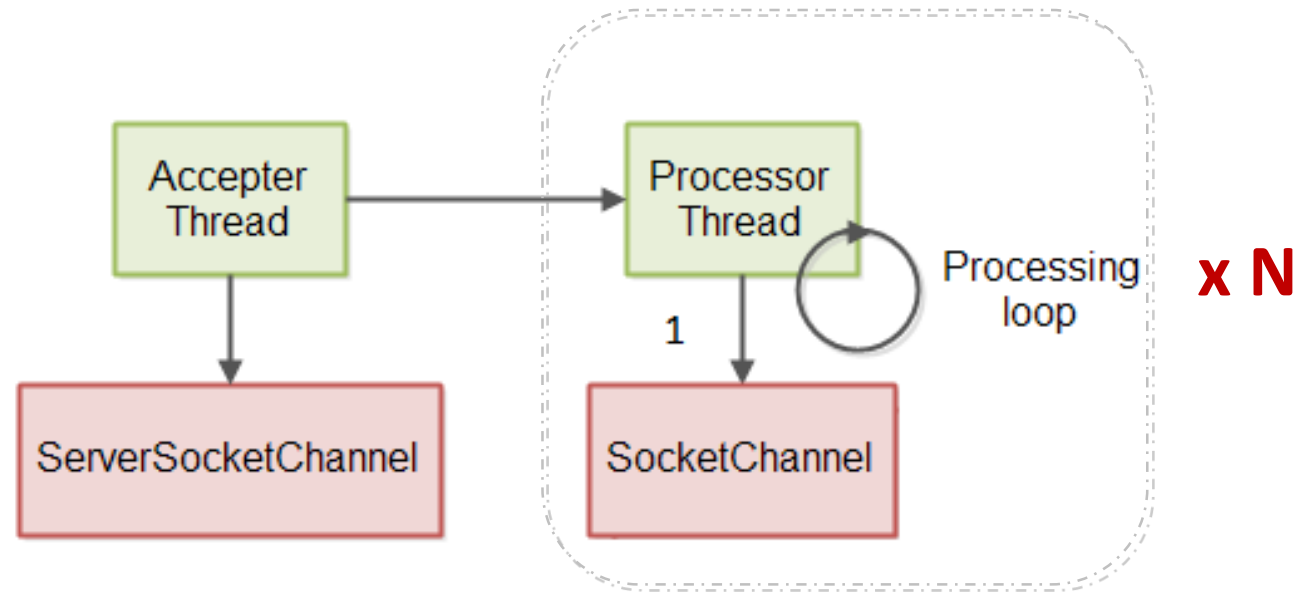
ТСР-сервер: 1 соединение



Узкое место: только 1 соединение вряд ли кому-нибудь интересно

Сети и сетевые ОС

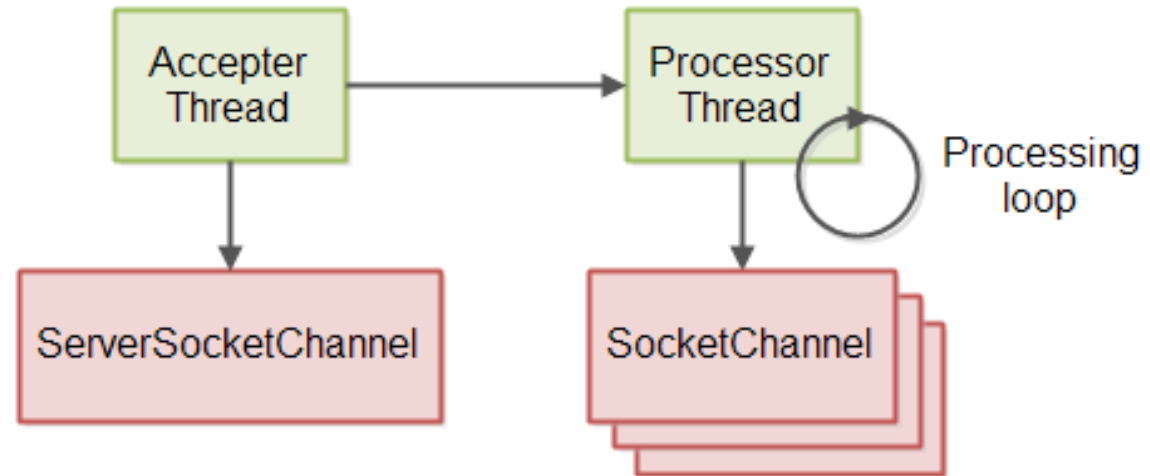
ТСР-сервер: 1 соединение - 1 поток



- Узкое место:
- большое число потоков == накладные расходы
 - проблемы с синхронизацией при работе с разделяемыми ресурсами

Сети и сетевые ОС

ТСР-сервер: N соединений на поток



Узкое место: как опрашивать соединения в `ProcessorThread`?
`ProcessorThread` в каждый момент времени фокусируется на 1 соединении*

*согласно нашим текущим знаниям и условию запрета использования корутин

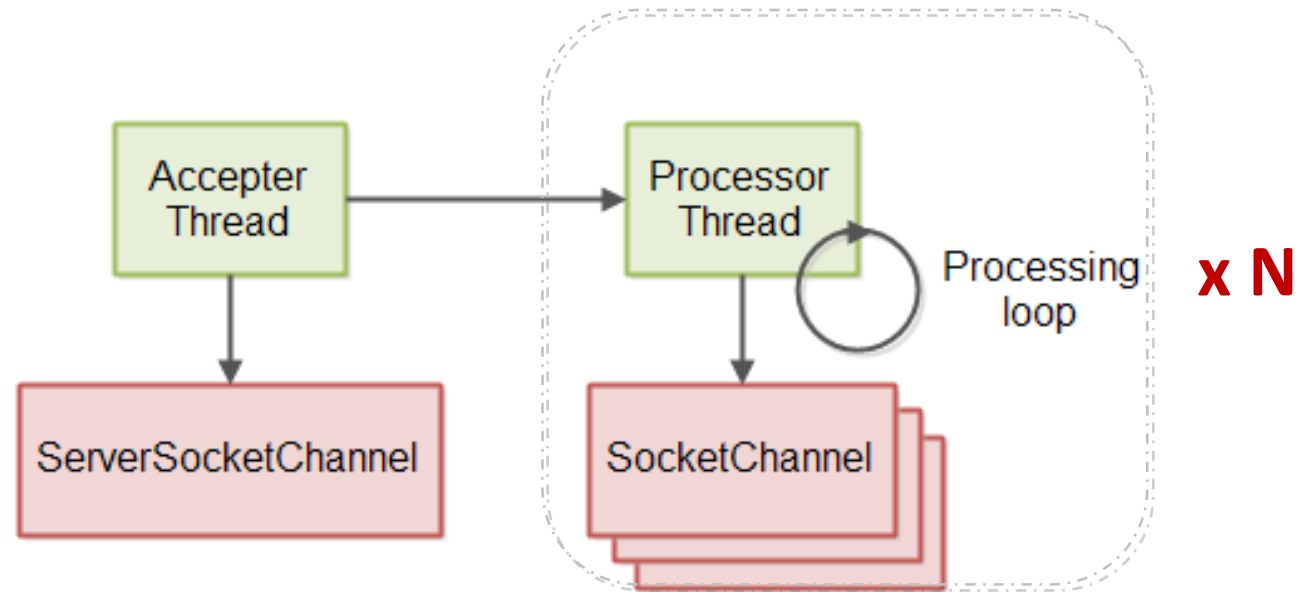
Сети и сетевые ОС

ТСР-сервер: Ввод / Вывод

- Блокирующийся – опрос дескрипторов сокетов (IPC, файлов) по списку, дожидаясь готовности
- Неблокирующийся – опрос дескрипторов сокетов (IPC, файлов) по списку
- Асинхронный – отдать на откуп корутине (что есть + отдельный поток в ядре)

Сети и сетевые ОС

ТСР-сервер: N соединений на поток



Узкое место: как опрашивать соединения в `ProcessorThread`?
`ProcessorThread` в каждый момент времени фокусируется на 1 соединении*

*согласно нашим текущим знаниям и условию запрета использования корутин

Сети и сетевые ОС

ТСР-сервер: Ввод / Вывод

- Блокирующийся – опрос дескрипторов сокетов (IPC, файлов) по списку, дожидаясь готовности
- Неблокирующийся – опрос дескрипторов сокетов (IPC, файлов) по списку
- Асинхронный – отдать на откуп корутине (что есть + отдельный поток в ядре)

Нужен вариант множественного опроса набора дескрипторов на готовность ввода/вывода без блокировки!

Сети и сетевые ОС

ТСР-сервер: Ввод / Вывод

- Блокирующийся – опрос дескрипторов сокетов (IPC, файлов) по списку, дожидаясь готовности
- Неблокирующийся – опрос дескрипторов сокетов (IPC, файлов) по списку
- Асинхронный – отдать на откуп корутине (что есть + отдельный поток в ядре)
- **Множественный (мультиплексируемый) I/O**

Сети и сетевые ОС

Множественный ввод / вывод

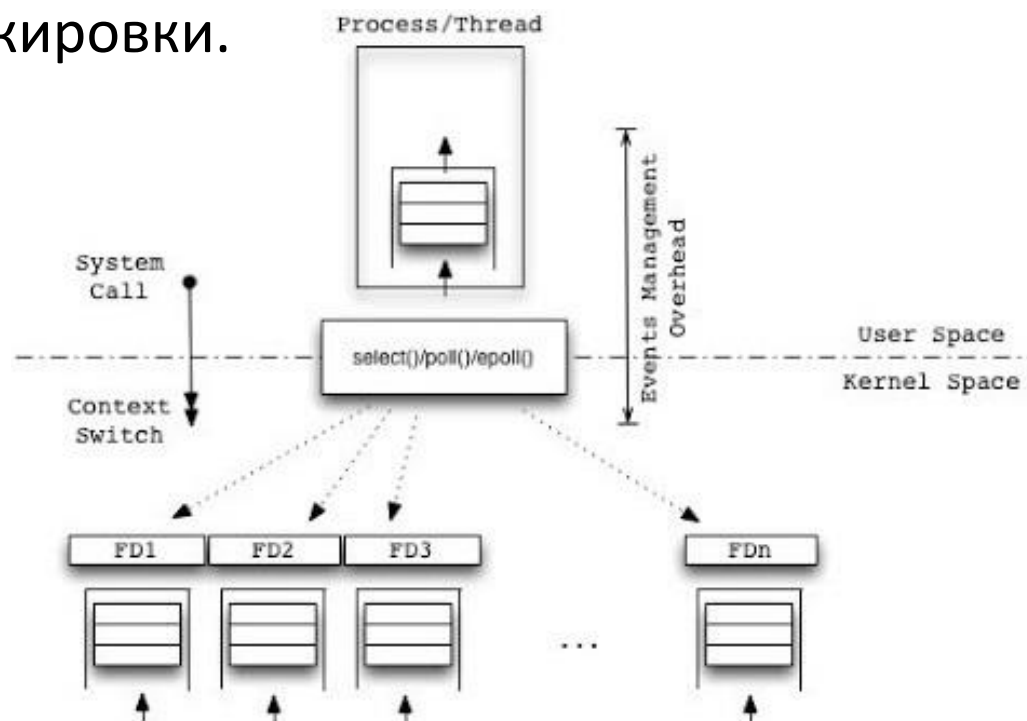
Множественный ввод-вывод позволяет приложениям параллельно блокировать несколько файловых дескрипторов и получать уведомления, как только любой из них будет готов к чтению или записи без блокировки, примерно по следующему принципу:

- 1/ [syscall](#): сообщите мне, когда любой из этих файловых дескрипторов будет готов к операции ввода-вывода.
- 2/ Ни один не готов? Перехожу в спящий режим до готовности одного или нескольких дескрипторов.
- 3/ Проснулся! Где готовый дескриптор?
- 4/ Обрабатываю без блокировки все файловые дескрипторы, готовые к вводу-выводу.
- 5/ Возвращаюсь к шагу 1

Сети и сетевые ОС

Множественный ввод / вывод

Множественный ввод-вывод позволяет приложениям параллельно блокировать несколько файловых дескрипторов и получать уведомления, как только любой из них будет готов к чтению или записи без блокировки.



Системные вызовы: Select / Poll / Epoll

Сети и сетевые ОС

Select/Poll/Epoll: сравнение

SELECT

- 1) -модификация fd_set
- 2) -MAX_FD == 1024
- 3) -UB при изменении дескриптора другим потоком
- 4) -неизвестен дескриптор, на котором произошло событие
- 5) -требуется вычислить max_num == MAX(fds)+1;
- 6) +портируемость
- 7) +высокая точность (~мкс при соответствующих условиях)

POLL

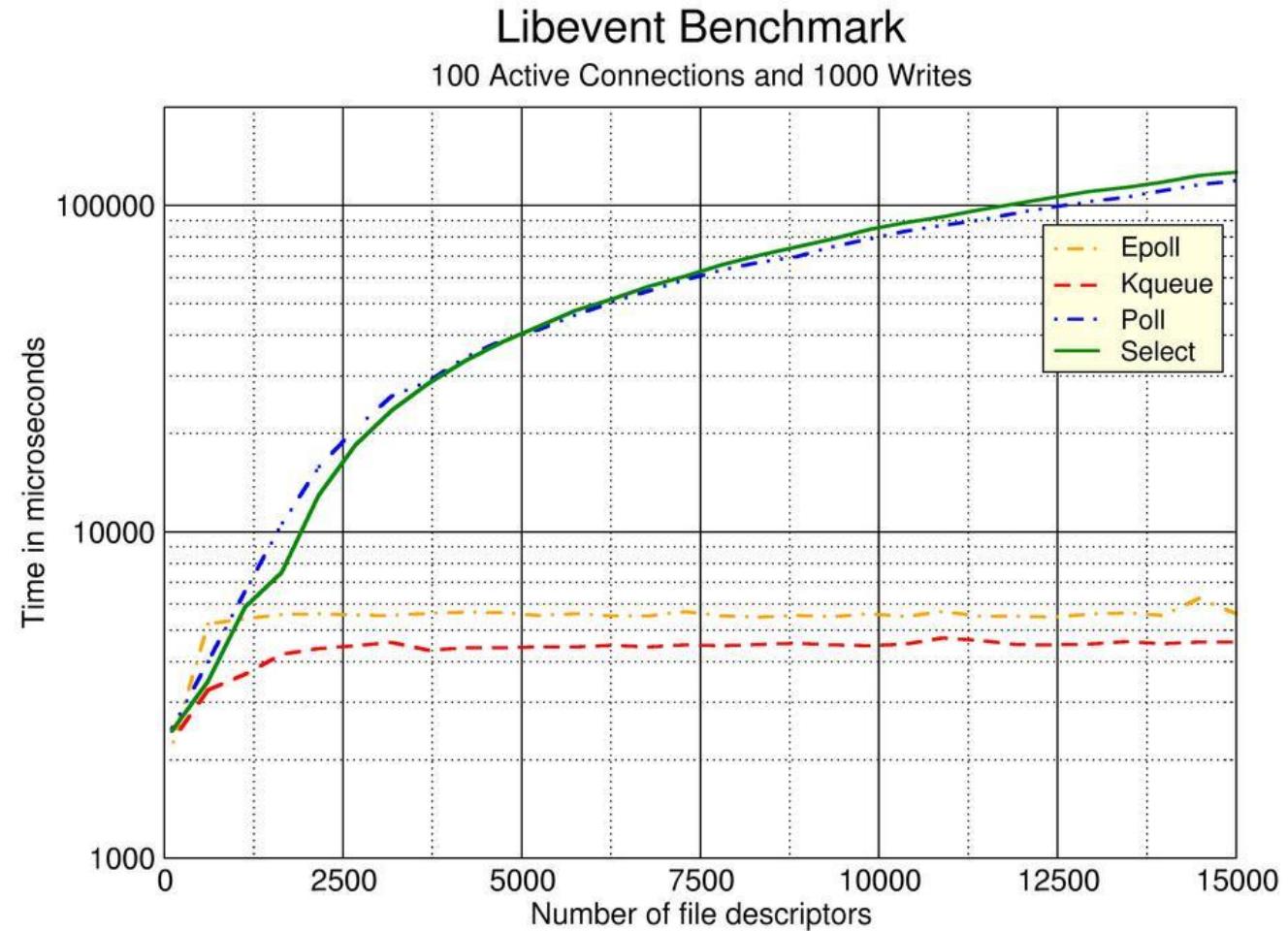
- 1) + лучшая структурированность pollfd
- 2) + не изменяет pollfd[]
- 3) + нет лимита на число дескрипторов в 1024
- 4) - точность (1мс, недостаток ликвидирован в Linux-специфичном **PPOLL**)
- 5) - портируемость
- 6) - определимость дескрипторов без обхода
- 7) - невозможность переопределить дескрипторы автоматически

EPOLL

- 1) + масштабируемость
- 2) - только для Linux
- 3) - некоторая громоздкость API

Сети и сетевые ОС

Select/Poll/Epoll: сравнение (1)



Сети и сетевые ОС

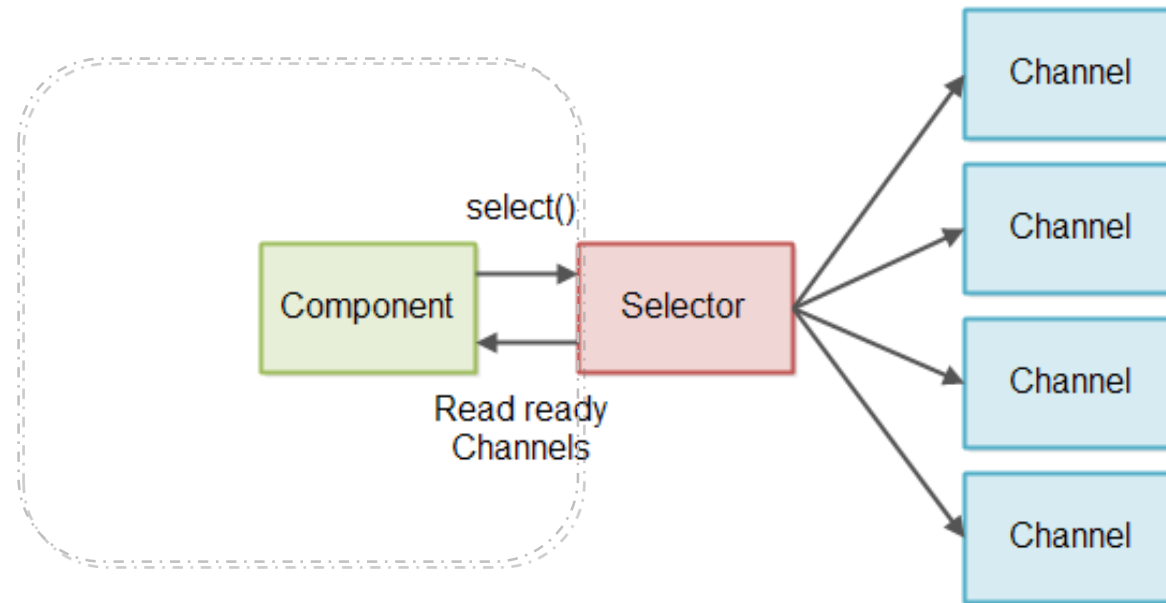
Select/Poll/Epoll: сравнение (2)

Number of File Descriptors	poll() CPU time	select() CPU time	epoll() CPU time
10	0.61	0.73	0.41
100	2.9	3	0.42
1000	35	35	0.53
10000 *	990	930	0.66

* Кто видит ошибку у авторов?

Сети и сетевые ОС

TCP Server: ProcessingThread + Select



Тема 14

Атрибуты и
группировка процессов

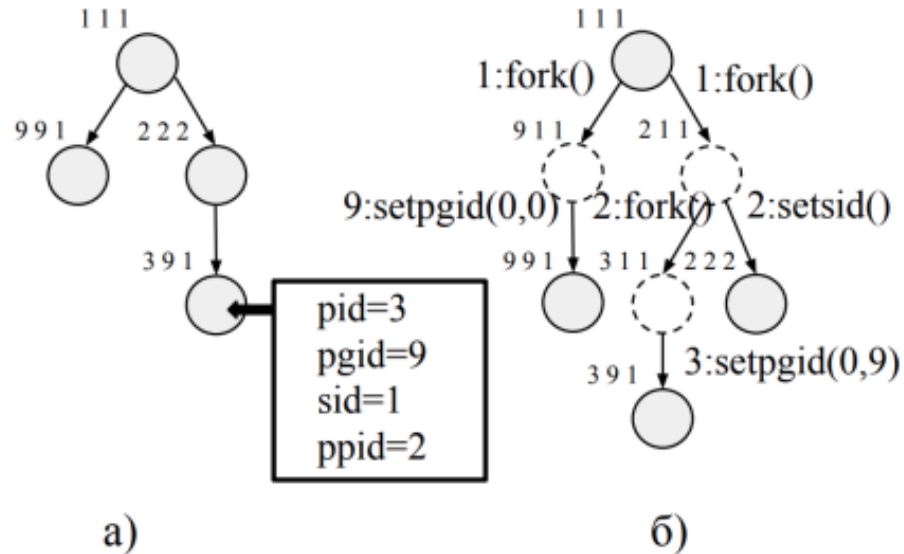
Атрибуты процессов

Атрибуты процессов Linux (credentials)

- Process ID (PID)
Сохраняется при `exec(2)`.
- Parent process ID (PPID)
MAN: “A process's parent process ID identifies the process that created this process using `fork(2)`. A process can obtain its PPID using `getppid(2)`. A PPID is represented using the type `pid_t`”.
Сохраняется при `exec`.
- Process group (PGID) and session (SID) IDs
Наследуются при `fork`, сохраняются при `exec`.
- Текущая директория
- Сигналы
- UID, GID, EUID, EGID – ID и группа пользователя , в том числе эффективные
- Другое

Атрибуты процессов

PID, PPID, SID, PGID (а) и их эволюция (б)



СПИСОК ВЫЗОВОВ:

- 1) 1:fork()
- 2) 1:fork()
- 3) 2:fork()
- 4) 2:setsid()
- 5) 9:setpgid(0,0)
- 6) 3:setpgid(0,9)

- - конечное состояние процесса
○ - промежуточное состояние процесса
↗ - иерархические связи и системные вызовы

Атрибуты процессов

PID, PPID, SID, PGID - возможности

- SID объединяет процессы в сеансе пользователя
 - С сеансом, как правило, ассоциирован терминал
 - Возникает возможность отсоединения от терминала,
 - А также -- создания фоновой задачи – «демона» (daemon)
- PGID объединяет процессы в наборы заданий (jobs) внутри сеанса
 - Возникает возможность переключать управляющий терминал между группами процессов
 - Возникает возможность рассылать сигналы наборам процессов
- PID/PPID задают общую иерархию процессов в дереве
- Сегментация процессов по этим атрибутам позволяет восстановить историю их назначения (как граф системных вызовов) – возможность сохранения / восстановления состояния набора*

* Данная идея раскрывается в следующих лекциях в подразделе «миграция процессов»

Атрибуты процессов

UID, GID, EUID, EGID

- Процесс запущен от имени пользователя
 - GID, UID наследуются
 - Но есть возможность их изменить («изменить владельца»)
- Устанавливают идентификаторы, с которыми процесс «действует» –
 - EGID, EUID
 - Есть возможность их изменить, например, запустив через **sudo**:

```
nefanov@NB500-341:/mnt/c/Users/ICPC-NB-XX/clang$ id
uid=1000(nefanov) gid=1000(nefanov) groups=1000(nefanov),
46(plugdev),117(netdev)
nefanov@NB500-341:/mnt/c/Users/ICPC-NB-XX/clang$ sudo id
uid=0(root) gid=0(root) groups=0(root)
```
- То есть ниже некоторого процесса в дереве процессы могут вести себя как root. *

* Данная идея раскрывается в следующих лекциях в подразделе «пространства имён»

Атрибуты процессов

Атрибуты процессов Linux (credentials)

Резюме: возникает ряд идей, как оперировать процессами, сгруппированными по их атрибутам. Данные идеи легли в основу технологии контейнерной виртуализации.

Атрибуты процессов

Атрибуты процессов Linux (credentials)

Резюме: возникает ряд идей, как оперировать процессами, сгруппированными по их атрибутам. Данные идеи легли в основу технологии контейнерной виртуализации.

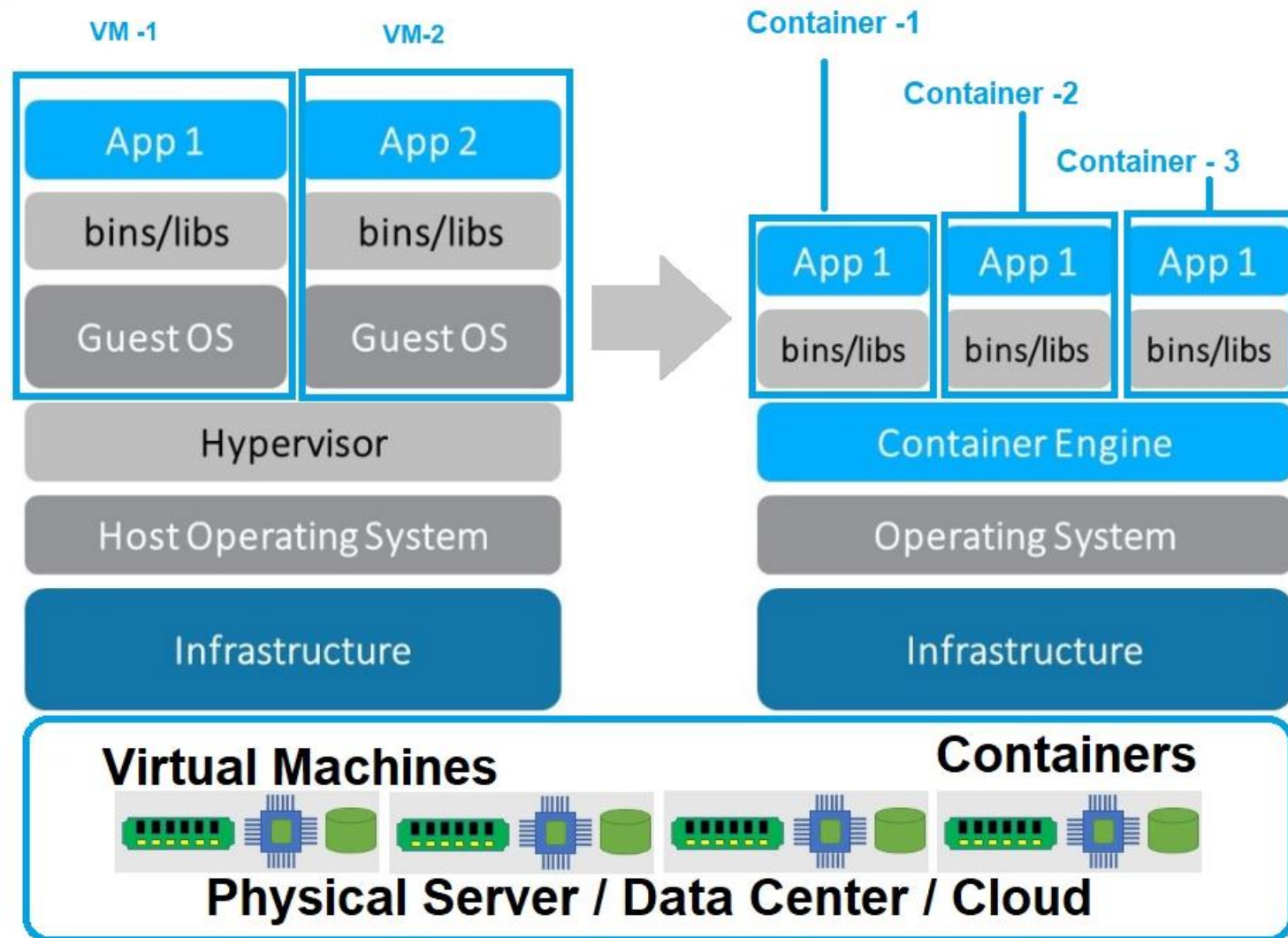
Контейнерная виртуализация

Терминология

- Контейнеры – изолированные пространства пользователя на едином ядре
- Linux namespaces -- механизм изоляции и группировки пространств идентификаторов ресурсов процессов: PID NS, FS NS, etc
- control groups -- механизм изоляции ресурсов ядра

Контейнерная виртуализация

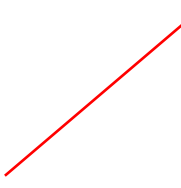
Hypervisor vs Containers



Контейнерная виртуализация

Пространства имён: представление в ядре

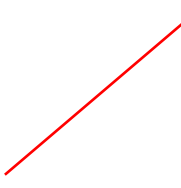
```
727 struct task_struct {
728 #ifdef CONFIG_THREAD_INFO_IN_TASK
729     /*
730      * For reasons of header soup (see current_thread_info()),
731      * must be the first element of task_struct.
732      */
733     struct thread_info    thread_info;
734 #endif
735     unsigned int          __state;
736
737 #ifdef CONFIG_PREEMPT_RT
738     /* saved state for "spinlock sleepers" */
739     unsigned int          saved_state;
740 #endif
741
742     pid_t                 pid;
743     pid_t                 tgid;
744
745     /* Namespaces: */
746     struct nsproxy        *nsproxy;
747
748     /*
749      * A structure to contain pointers to all per-process
750      * namespaces - fs (mount), uts, network, sysvipc, etc.
751      *
752      * The pid namespace is an exception -- it's accessed using
753      * task_active_pid_ns. The pid namespace here is the
754      * namespace that children will use.
755      *
756      * 'count' is the number of tasks holding a reference.
757      * The count for each namespace, then, will be the number
758      * of nsproxies pointing to it, not the number of tasks.
759      *
760      * The nsproxy is shared by tasks which share all namespaces.
761      * As soon as a single namespace is cloned or unshared, the
762      * nsproxy is copied.
763      */
764     struct nsproxy {
765         atomic_t count;
766         struct uts_namespace *uts_ns;
767         struct ipc_namespace *ipc_ns;
768         struct mnt_namespace *mnt_ns;
769         struct pid_namespace *pid_ns_for_children;
770         struct net            *net_ns;
771         struct time_namespace *time_ns;
772         struct time_namespace *time_ns_for_children;
773         struct cgroup_namespace *cgroup_ns;
774     };
775     extern struct nsproxy init_nsproxy;
```



Контейнерная виртуализация

Пространства имён: представление в ядре

```
727 struct task_struct {
728 #ifdef CONFIG_THREAD_INFO_IN_TASK
729     /*
730      * For reasons of header soup (see current_thread_info()),
731      * must be the first element of task_struct.
732      */
733     struct thread_info    thread_info;
734 #endif
735     unsigned int          __state;
736
737 #ifdef CONFIG_PREEMPT_RT
738     /* saved state for "spinlock sleepers" */
739     unsigned int          saved_state;
740 #endif
741
742     pid_t                 pid;
743     pid_t                 tgid;
744
745     /* Namespaces: */
746     struct nsproxy        *nsproxy;
747
748     /*
749      * A structure to contain pointers to all per-process
750      * namespaces - fs (mount), uts, network, sysvipc, etc.
751      *
752      * The pid namespace is an exception -- it's accessed using
753      * task_active_pid_ns. The pid namespace here is the
754      * namespace that children will use.
755      *
756      * 'count' is the number of tasks holding a reference.
757      * The count for each namespace, then, will be the number
758      * of nsproxies pointing to it, not the number of tasks.
759      *
760      * The nsproxy is shared by tasks which share all namespaces.
761      * As soon as a single namespace is cloned or unshared, the
762      * nsproxy is copied.
763      */
764     struct nsproxy {
765         atomic_t count;
766         struct uts_namespace *uts_ns;
767         struct ipc_namespace *ipc_ns;
768         struct mnt_namespace *mnt_ns;
769         struct pid_namespace *pid_ns_for_children;
770         struct net *net_ns;
771         struct time_namespace *time_ns;
772         struct time_namespace *time_ns_for_children;
773         struct cgroup_namespace *cgroup_ns;
774     };
775     extern struct nsproxy init_nsproxy;
```



To be continued...