Komunikacja międzyprocesowa i międzywątkowa

Moduł 2
Komunikacja międzyprocesowa

mgr inż. Tomasz Pawelec

Agenda

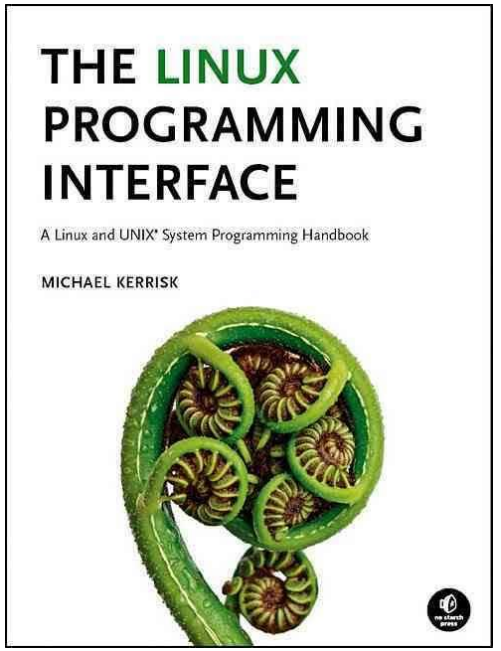
1. Literatura
2. Taksonomia IPC (ang. ***Interprocess communication***)
3. Potoki
    - potoki nienazwane (ang. ***Pipes***)
    - potoki nazwane (ang. ***FIFOs***)
4. Kolejki komunikatów (ang. ***Message queues***)
5. Pamięć współdzielona (ang. ***Shared memory***)
6. Gniazda (ang. ***Sockets***)
7. Zadanie

1. Literatura

Linux man (manual)
http://man7.org/linux/man-pages/index.html

- man 7 pipe   // potoki nienazwane
- man 7 fifo   // potoki nazwane
- man 7 mq_overview   // kolejki komunikatów
- man 7 shm_overview // pamięć współdzielona
- man 7 socket   // gniazda

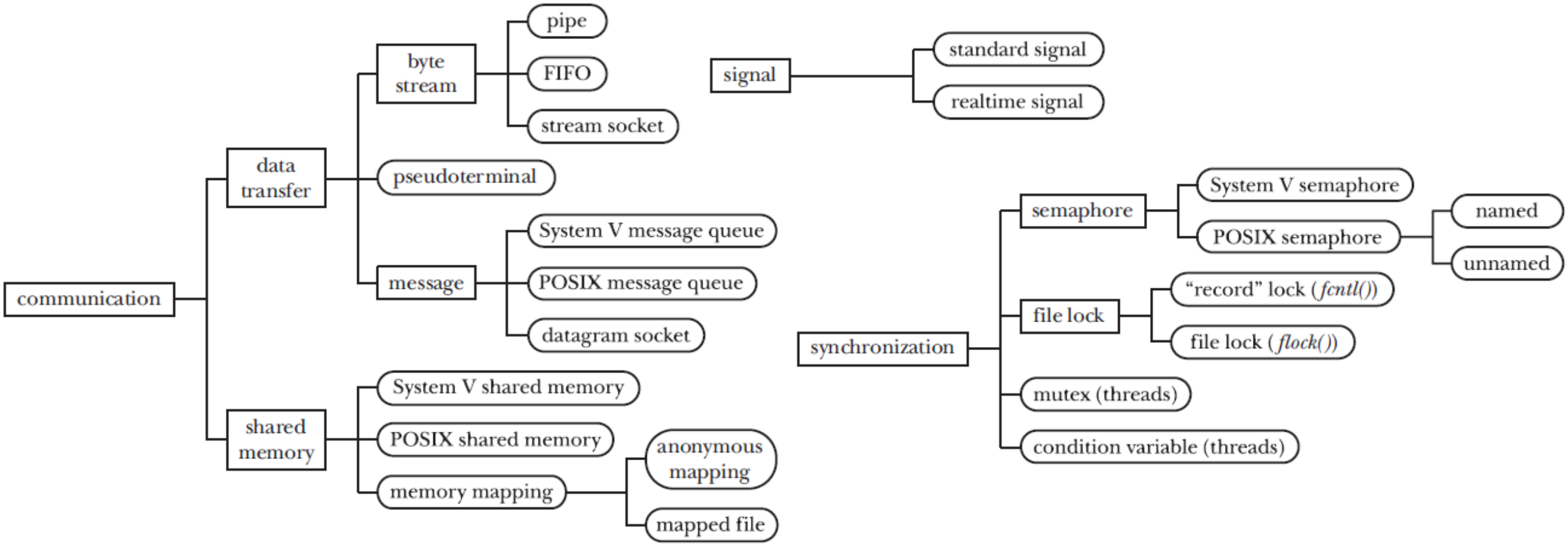
The Linux programming Interface, Michael Kerrisk



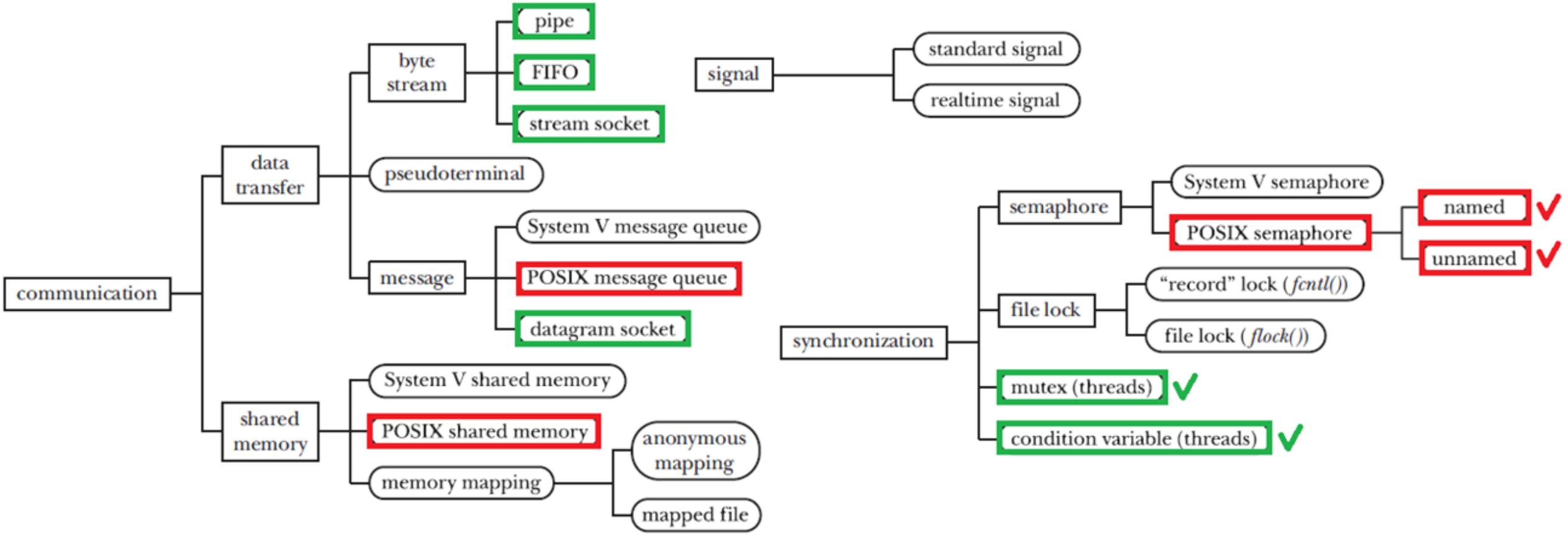
2. Taksonomia IPC

- **Komunikacja** (ang. ***Communication***) – mechanizmy służące do wymiany danch pomiędzy procesami/wątkami
- **Synchronizacja** (ang. ***Synchronization***) – mechanizmy służące do synchronizacji procesów/wątków
- **Sygnały** (ang. ***Signals***) – mechanizmy programowych przerwań mogące być wykorzystwane w roli komunikacji lub synchronizacji

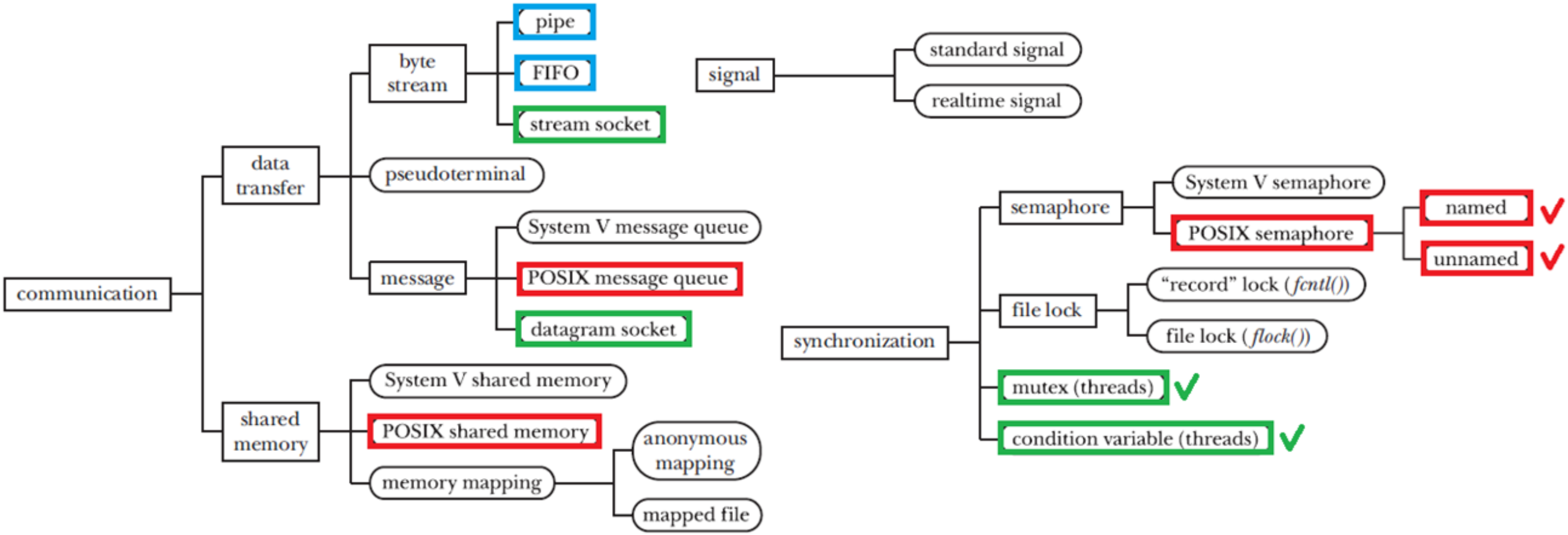
2. Taksonomia IPC



2. Taksonomia IPC



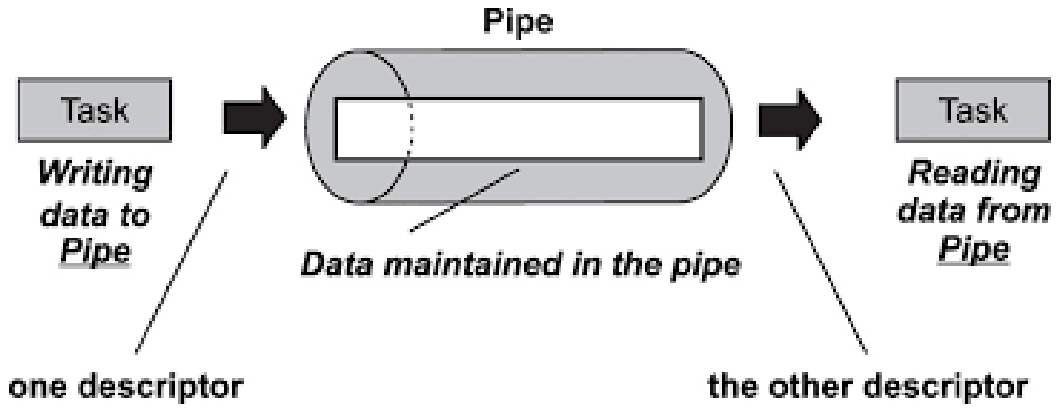
3. Potoki (ang. *Pipes*, *FIFOs*)



3. Potoki - wstęp

- **potoki nienazwane** (ang. ***pipes***) oraz **potoki nazwane** (ang. ***FIFO***) zapewniają jednokierunkową (ang. ***unidirectional***) metodę komunikacji pomiędzy procesami polegającą na strumieniu bajtów (ang. ***byte stream***)
- domyślnie odczyt(pusty) / zapis(pełny) jest blokujący

- **potoki nazwane** posiadają nazwę w systemie plików:
  - procesy niepowiązane/niespokrewnione (ang. ***unrelated***)
  - procesy powiązane/spokrewnione (ang. ***related***)
- **potoki nienazwane** można zleźć w **/proc/<pid>/fd**
  - procesy powiązane/spokrewnione (ang. ***related***)



Źródło:
http://www.embeddedlinux.org.cn/rtconforembsys/5107final/LiB0050.html

3. Potoki - konfiguracja

Zapis <= **PIPE_BUF** jest atomowy

- POSIX – at least 512B

  cat **/usr/include/x86_64-linux-gnu/bits/posix1_lim.h** | grep _PIPE_BUF

  #define _POSIX_PIPE_BUF 512

- Linux – 4096B

  cat **/usr/include/linux/limists** | grep PIPE_BUF

  #define PIPE_BUF 4096

/proc/sys/fs/pipe-max-size (since Linux 2.6.35)

- maksymalny rozmiar w bajtach jaki pojedynczy potok może posiadać (1 048 576 - 1 MiB)

3. Potoki - interfejs

Potoki nienazwane:

```
int pipe(int pipefd[2]);
int pipe2(int pipefd[2], int flags);
    // pipe[0] – read  (input)
    // pipe[1] – write (output)
```

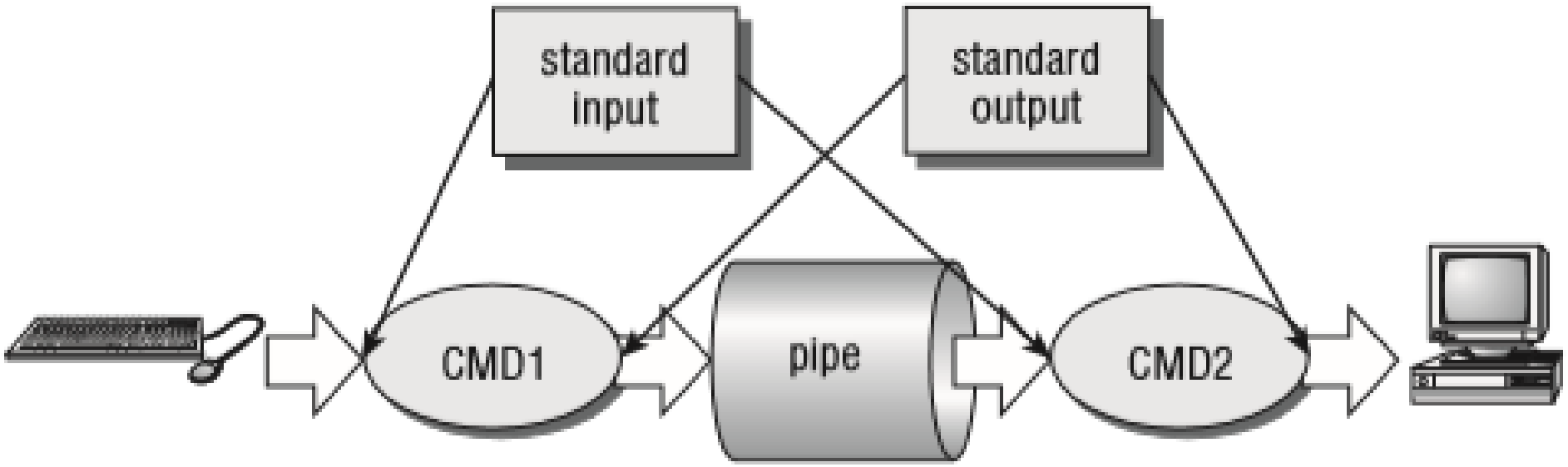
Potoki nazwane:

```
int mkfifo(const char *pathname, mode_t mode);
```

open() – domyślnie jest zsynchronizowany (blokujący), druga strona musi otworzyć

3. Potoki - przykłady

```
ls | sort -r
history | grep ps
```



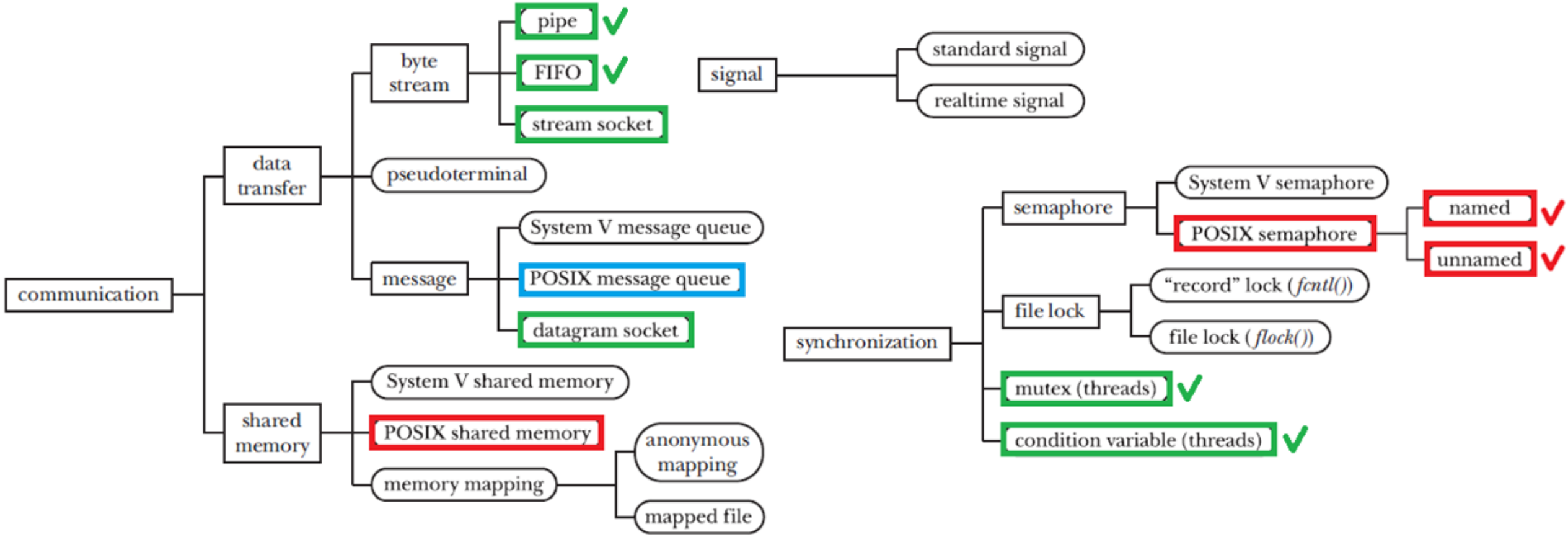
3. Potoki - podsumowanie

	pipe	fifo	mq	shm	sockets
komunikacja - typ	byte stream	byte stream			
komunikacja – kierunek	unidirectional	unidirectional			
zapis (pełny)	blokujący	blokujący			
odczyt (pusty)	blokujący	blokujący			
procesy powiązane	tak	tak			
procesy niepowiązane	nie	tak			
procesy zdalne	nie	nie			
specjalne właściwości	dostęp atomowy jeżeli < PIPE_BUF	dostęp atomowy jeżeli < PIPE_BUF			

3. Potoki - przykłady

<przykłady w kodzie>

4. Kolejki komunikatów (ang. *message queues*)



4. Kolejki komunikatów - wstęp

- metoda komunikacji pomiędzy procesami oparta na wiadomościach (ang. ***message oriented communication***)
  - odbiorca odczytuje całe wiadomości, brak możliwości częściowego odczytu wiadomości
  - wiadomości mają swoją strukturę i priorytet
    - 0 - najniższy
    - 31 – najwyższy (POSIX)
    - _SC_MQ_PRIO_MAX (32768) - najwyższy(Linux)
  - można wykorzystać mechanizm notyfikacji (ang. ***notification***) jeżeli odbiorca dostanie wiadomość
  - można konfigurować maksymalny rozmiar kolejki oraz maksymalną liczbę wiadomości w kolejce
- domyślnie odczyt(pusty) / zapis(pełny) jest blokujący
- tworzone w wirtualnym systemie plików: **/dev/mqueue**
  - procesy niepowiązane/niespokrewnione (ang. ***unrelated***)
  - procesy powiązane/spokrewnione (ang. ***related***)
- trzeba dolinkować bibliotekę real-time ***-lrt***

4. Kolejki komunikatów - konfiguracja

/proc/sys/fs/mqueue/msg_max

- maksymalna ilość wiadomości w kolejce (sufit dla attr->mq_maxmsg)
- minimalna wartość - 1
- maksymalna wartość – 65 536
- domyślna wartość - 10

/proc/sys/fs/mqueue/msgsize_max

- maksymalny rozmiar rozmiar pojedynczej wiadomości (sufit dla attr->mq_msgsize)
- minimalna wartość – 128B
- maksymalna wartość – 16 777 215 B (16MB)
- domyślna wartość – 8192B

/proc/sys/fs/mqueue/queues_max

- Maksymalna liczba kojek komunikatów, która może być stworzona naraz
- minimalna wartość – 0
- maksymalna wartość – INT_MAX (256)
- domyślna wartość – 256

4. Kolejki komunikatów - interfejs

Library interface	**System call**
mq_close(3)	close(2)
mq_getattr(3)	mq_getsetattr(2)
mq_notify(3)	mq_notify(2)
mq_open(3)	mq_open(2)
mq_receive(3)	mq_timedreceive(2)
mq_send(3)	mq_timedsend(2)
mq_setattr(3)	mq_getsetattr(2)
mq_timedreceive(3)	mq_timedreceive(2)
mq_timedsend(3)	mq_timedsend(2)
mq_unlink(3)	mq_unlink(2)

4. Kolejki komunikatów - podsumowanie

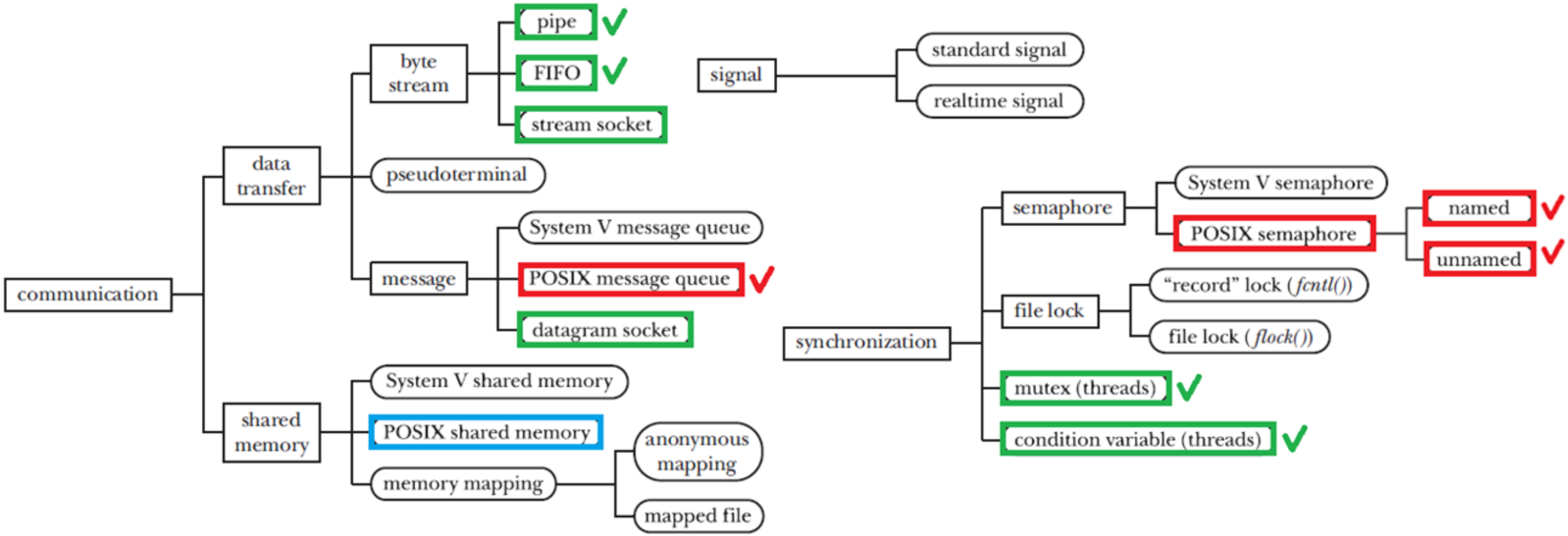
	pipe	fifo	mq	shm	sockets
komunikacja - typ	byte stream	byte stream	message		
komunikacja - kierunek	unidirectional	unidirectional	bidirectional		
zapis (pełny)	blokujący	blokujący	blokujący		
odczyt (pusty)	blokujący	blokujący	blokujący		
procesy powiązane	tak	tak	tak		
procesy niepowiązane	nie	tak	tak		
procesy zdalne	nie	nie	nie		

specjalne właściwości	dostęp atomowy jeżeli < PIPE_BUF	dostęp atomowy jeżeli < PIPE_BUF	• konfiguracja
			  ○ msg_max
			  ○ msgsize_max
			  ○ queues_max
			• priorytety
			• notyfikacje		

4. Kolejki komunikatów - przykłady

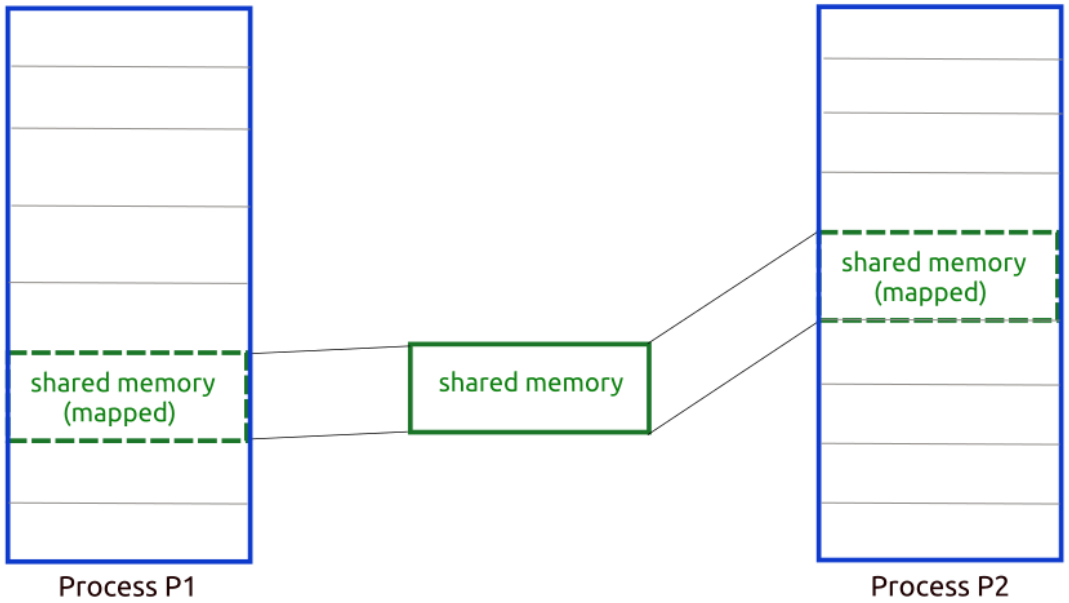
<przykłady w kodzie>

5. Pamięć współdzielona (ang. *Shared memory*)



5. Pamięć współdzielona

- meteda komunikacji pomiędzy procesami oparta na współdzieleniu tego samego obszaru pamięci
- najszybsza metoda komunikacji ponieważ nie ma transferu danych:
  - user space => kernel space => user space
- nie zapewnia synchronizacji
- tworzona w wirtualnym systemie plików: **/dev/shm**
- trzeba dolinkować bibliotekę real-time ***-lrt***



Zródła:
https://www.softprayog.in/programming/interprocess-communication-using-posix-shared-memory-in-linux

5. Pamięć współdzielona - interfejs

shm_open(3)	stworzenie/otwarcie obiektu SHM
ftruncate(2)	ustawienie rozmiaru obiektu SHM
mmap(2)	zmapowanie obiektu SHM do wirtualnego obszaru pamieci procesu
munmap(2)	odmapowanie obiektu SHM od wirtualnego obszaru pamieci procesu
shm_unlink(3)	usunięcie obiektu SHM
close(2)	zamknięcie deskryptora do obiektu SHM
fstat(2)	odczytanie iformacji o obiekcie SHM
fchown(2)	zmiana właściciela obiektu SHM
fchmod(2)	zmiana uprawnień obiektu SHM

5. Pamięć współdzielona - podsumowanie

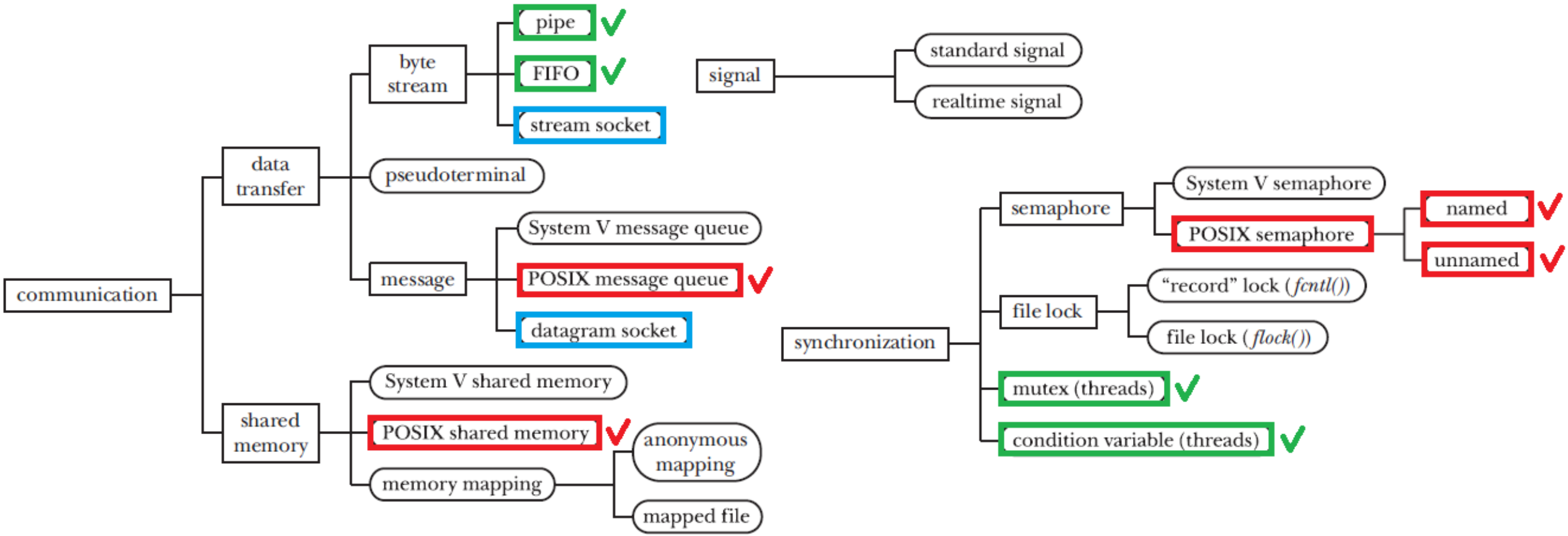
	pipe	fifo	mq	shm	sockets
komunikacja - typ	byte stream	byte stream	message		
komunikacja - kierunek	unidirectional	unidirectional	bidirectional	bidirectional	
zapis (pełny)	blokujący	blokujący	blokujący		
odczyt (pusty)	blokujący	blokujący	blokujący		
procesy powiązane	tak	tak	tak	tak	
procesy niepowiązane	nie	tak	tak	tak	
procesy zdalne	nie	nie	nie	nie	

specjalne właściwości	dostęp atomowy jeżeli < PIPE_BUF	dostęp atomowy jeżeli < PIPE_BUF	• konfiguracja
			  ○ msg_max
			  ○ msgsize_max
			  ○ queues_max
			• priorytety
			• notyfikacje		

5. Pamięć współdzielona - przykłady

<przykłady w kodzie>

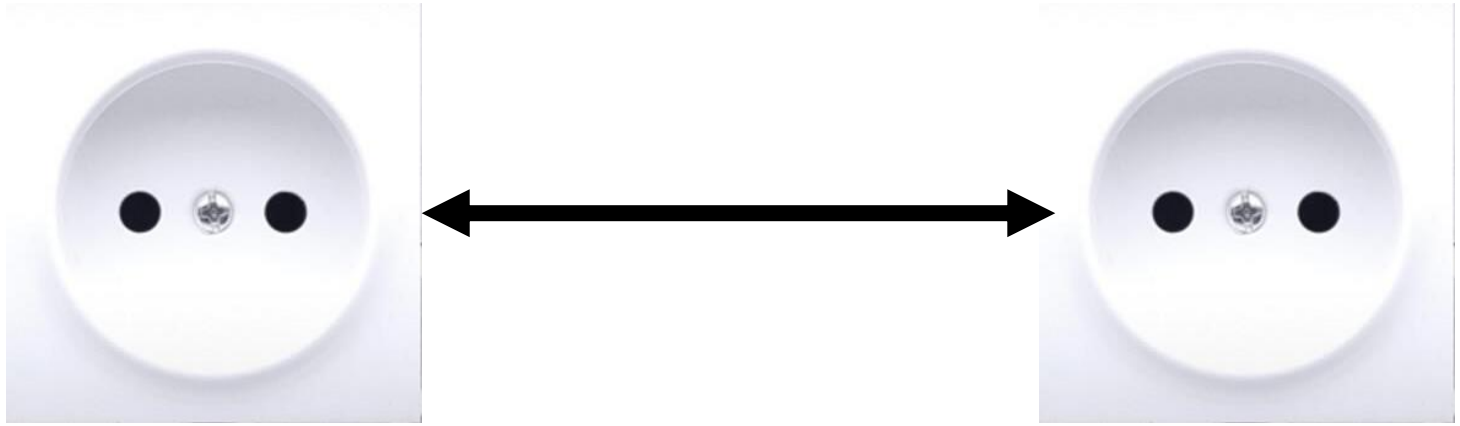
6. Gniazda(ang. *Sockets*)



6. Gniazda - wstęp

- najbardziej złożony mechanizm kumunikacji międzyprocesowej pozwalający również na komunikację zdalną (pomiędzy różnymi hostami)
- omawiane będę wybrane elementy tego mechanizmu

- gniazdo (ang. ***socket***) jest zakończeniem kanału kmunikacyjnego
  - potrzeba dwóch gniazd
  - dwukierunkowe (ang. ***bidirectional***)



6. Gniazda - interfejs

fd = socket(domain, type, protocol)

domena (ang. domain):

- metoda identyfikacji gniazda (format adresu)
- zakres kumunikacji (lokalna, zdalna)

- **AF_UNIX/AF_LOCAL** (UNIX_domain)
  - komunikacja w obrębie tego samego hosta
  - adres - ścieżka w systemie plików
- **AF_INET** (IPv4 domain)
  - komunikacja poprzez sieć IPv4
  - adres – 32bit IPv4 + numer portu
- **AF_INET6** (IPv6) domain)
  - komunikacja poprzez sieć IPv6
  - adres – 128bit IPv6 + numer portu

- **INNE** - pominięte

6. Gniazda - interfejs

fd = socket(domain, type, protocol)

typ (ang. type):

- określa semantykę kumunikacji (ang. ***semantics of communication***)

- **SOCK_STREAM** (ang. ***stream***)
  - ciąg bajtów (ang. ***byte stream***)
  - zorientowana na połączenie (ang. ***connection oriented***)
  - niezawodna (ang. ***reliable***)
  - przydkład – **TCP**

- **SOCK_DGRAM** (ang. ***datagram***)
  - wiadomości(ang. ***message-oriented***)
  - bezpołączeniowa (ang. ***connection-less***)
  - zawodna (ang. ***reliable***)
    - duplikacja (ang. ***duplicated***)
    - poza kolejnością (ang. ***out of order***)
    - brak komunikacji o niedostarczeniu (ang. ***not at all***)
  - przykład – **UDP**

- **INNE -** pominięte

6. Gniazda - interfejs

fd = socket(domain, type, protocol)

protokół (ang. protocol):

- określa konkrenty protokół, kóry będzie użyty do komunikacji

- 0 – domyślny protokół będzie używany (zdeterminowany przez domenę i typ)
  - AF_INET, SOCK_DGRAM => IPPROTO_UDP
  - AF_INET, SOCK_STREAM => IPPROTO_TCP
- !0 – można wybrać dowolny protokół
  - **/etc/protocols** => wszystkie dostępne protokoły

6. Gniazda - interfejs

socket(3) – tworzenie gniazda

bind(2) – łaczenie deskryptora z adresem lokalnym

connect(2) – łaczenie deskryptora gniazda z adresem zdalnym

listen(2) – ustawienie gniazda jako pasywnego (do akceptacji połączeń)

accept(3) – akceptacja połączenia na gnieździe

send(2), sendto(2), sendmsg(2) – wysyłanie danych

rcv(2), rcvfrom(2), rcvmsg(2) – odbieranie danych

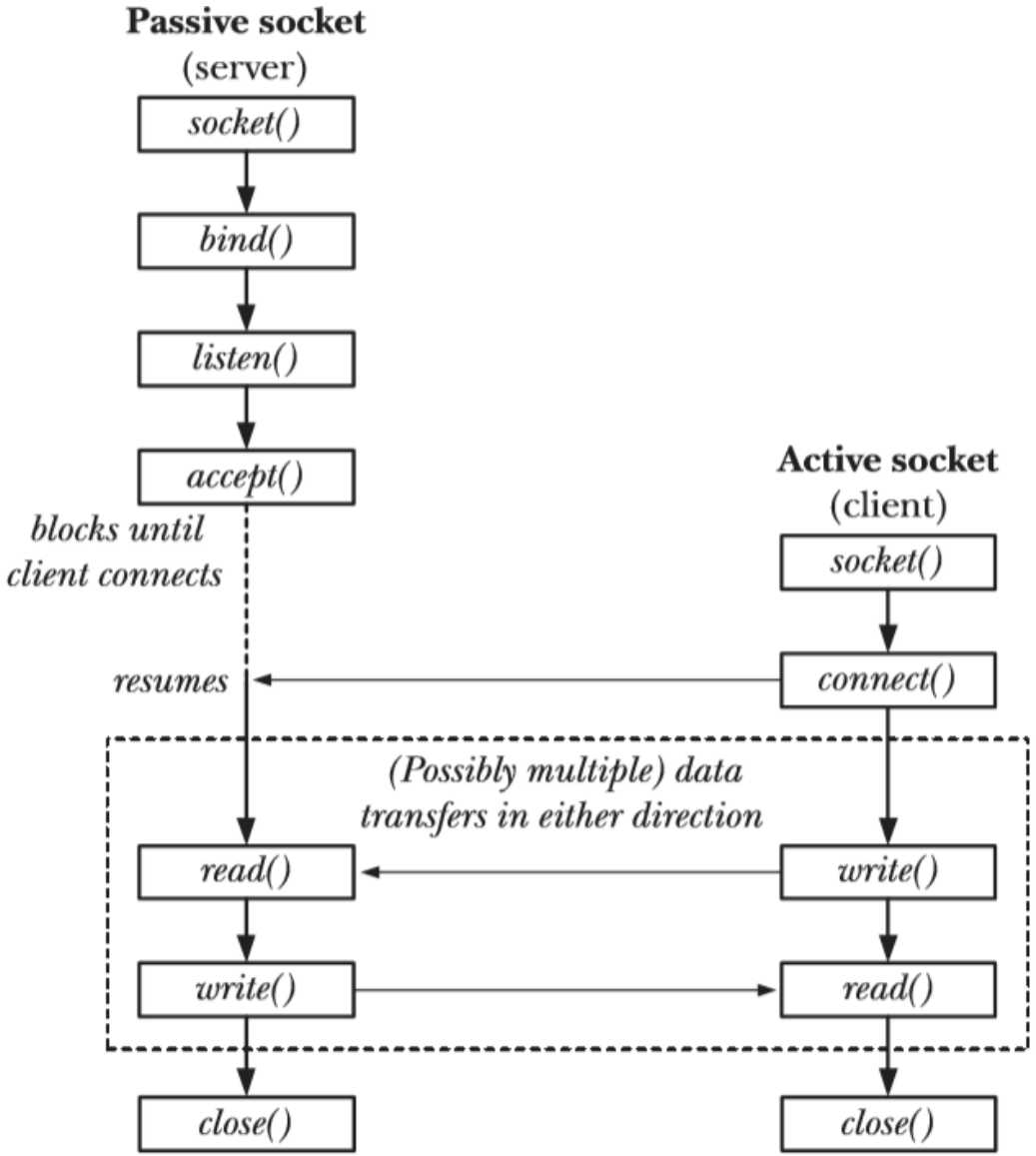
getsockname(2) – zwraca adres przywiązania do gniada

getpeername(2) – zwraca adres połaczonego klienta

getsockopt(2) – opcje gniazda

close(2), shutdown(2) – zamykanie gniazda

6. Gniazda – przykłady (TCP)



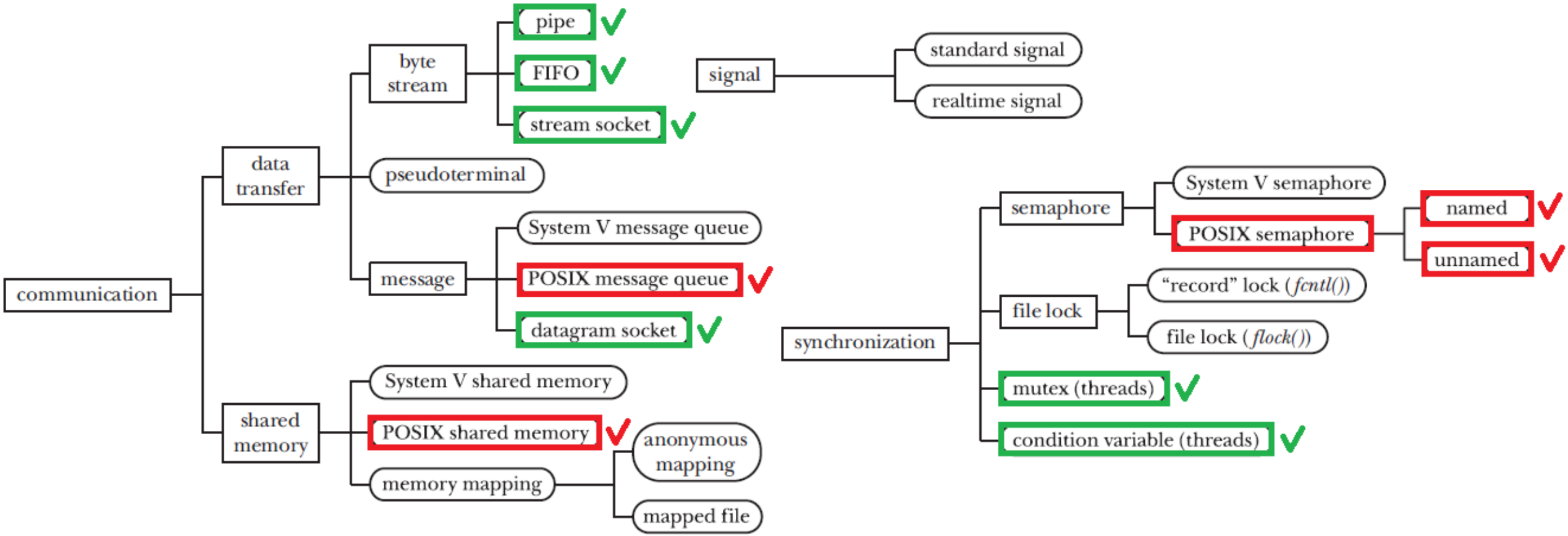
6. Gniazda - podsumowanie

	pipe	fifo	mq	shm	sockets
komunikacja - typ	byte stream	byte stream	message		message
komunikacja - kierunek	unidirectional	unidirectional	bidirectional	bidirectional	bidirectional
zapis (pełny)	blokujący	blokujący	blokujący		blokujący
odczyt (pusty)	blokujący	blokujący	blokujący		blokujący
procesy powiązane	tak	tak	tak	tak	tak
procesy niepowiązane	nie	tak	tak	tak	tak
procesy zdalne	nie	nie	nie	nie	tak

specjalne właściwości	dostęp atomowy jeżeli < PIPE_BUF	dostęp atomowy jeżeli < PIPE_BUF	• konfiguracja
			  ○ msg_max
			  ○ msgsize_max
			  ○ queues_max
			• priorytety
			• notyfikacje		

6. Gniazda - przykłady

<przykłady w kodzie>


communication
data transfer
byte stream
pipe
FIFO
stream socket
pseudoterminal
message
System V message queue
POSIX message queue
datagram socket
shared memory
System V shared memory
POSIX shared memory
memory mapping
anonymous mapping
mapped file
signal
standard signal
realtime signal
synchronization
semaphore
System V semaphore
POSIX semaphore
named
unnamed
file lock
"record" lock (fcntl())
file lock (flock())
mutex (threads)
condition variable (threads)

7. Zadanie

1. Zaimplementuj stos przy wykorzystaniu tablicy o rozmiarze **N** zawierające dane typu **unsigned int**.
2. Zaimplementuj metody dodawania/usuwania do/z stosu:

int push_front()

- jeżeli stos jest pełny, wówczas zwracana wartość ma wynosić **-1**

int pop_front()

- jeżeli stos jest pusty, wówczas zwracana wartość ma wynosić **-1**

3. Instancja tego stosu ma znajdować się w pamięci współdzielonej dla dwóch powiązanych procesów (producent, konsument).
4. Zidetyfikuj sekcje krytyczne w programie i odpowiednio je zabezpiecz.
5. Jeżeli stos jest pusty, wówczas czytający proces (konsument) ma być uśpiony.
6. Jeżeli stos jest pełny, wówczas zapisujący proces (producent) ma być uśpiony.

schemat 1-producent, 1-konsument, k-zasobów