MSG_ZEROCOPY

Willem de Bruijn
willemb@google.com

Copying

```
perf record netperf -t TCP_STREAM -H $host

Samples: 42K of event 'cycles', Event count (approx.): 21258597313
 79.41%         33884  netperf  [kernel.kallsyms]  [k] copy_user_generic_string
  3.27%          1396  netperf  [kernel.kallsyms]  [k] tcp_sendmsg
  1.66%           694  netperf  [kernel.kallsyms]  [k] get_page_from_freelist
  0.79%           325  netperf  [kernel.kallsyms]  [k] tcp_ack
  0.43%           188  netperf  [kernel.kallsyms]  [k] __alloc_skb
```

Page cache

```
ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count);
```

Page cache

```
ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count);
```

```
/* link page into frags[] */
get_page(page)
skb_shinfo(skb)->frags[i].page.p      = page
skb_shinfo(skb)->tx_flags            |= SKBTX_SHARED_FRAG
```

```
/* copy on payload access */
if (skb_has_shared_frag(skb))
     __skb_linearize(skb)
```

Page cache

```
ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count);
```

```
/* check eligibility */
sk->sk_route_caps & NETIF_F_SG && sk_check_csum_caps(sk)
```

```
/* link page into frags[] */
get_page(page)
skb_shinfo(skb)->frags[i].page.p      = page
skb_shinfo(skb)->tx_flags            |= SKBTX_SHARED_FRAG
```

```
/* copy on payload access */
if (skb_has_shared_frag(skb))
     __skb_linearize(skb)
```

splice

```
ssize_t splice(int fd_in, loff_t *off_in, int fd_out,
               loff_t *off_out, size_t len, unsigned int flags);

SPLICE_F_MOVE
```

splice + vmsplice

```
ssize_t splice(int fd_in, loff_t *off_in, int fd_out,
               loff_t *off_out, size_t len, unsigned int flags);

SPLICE_F_MOVE


ssize_t vmsplice(int fd, const struct iovec *iov,
                        unsigned long nr_segs, unsigned int flags);

SPLICE_F_GIFT
```

splice + vmsplice

```
ssize_t splice(int fd_in, loff_t *off_in, int fd_out,
               loff_t *off_out, size_t len, unsigned int flags);

SPLICE_F_MOVE


ssize_t vmsplice(int fd, const struct iovec *iov,
                        unsigned long nr_segs, unsigned int flags);

SPLICE_F_GIFT



man vmsplice(2):
  "The user pages are a gift to the kernel.
   The application may not modify this memory ever"
```

virtio-net

guest kernel
  virtio-net driver

qemu
  virtio-net dev
  send(tap_fd)

host kernel
  tun_sendmsg

virtio-net

guest kernel
  virtio-net driver

qemu
  virtio-net dev
  send(tap_fd)

host kernel
  tun_sendmsg

vhost-net

guest kernel
  virtio-net driver

host kernel
  vhost dev
  sock->ops->sendmsg
  tun_sendmsg

vhost-net zerocopy

```
/* vhost handle_tx */
msg.iov = iov;
msg.msg_control = uarg;
sock->ops->sendmsg(sock, &msg, len);
```

vhost-net zerocopy

```
/* vhost handle_tx */
msg.iov = iov;
msg.msg_control = uarg;
sock->ops->sendmsg(sock, &msg, len);

struct ubuf_info {
     void (*callback)(struct ubuf_info *, bool zerocopy_succes);
     void *ctx;
     unsigned long desc;
}

static void vhost_zerocopy_callback(struct ubuf_info *ubuf, bool success);
```

vhost-net zerocopy

```
/* vhost handle_tx */
msg.iov = iov;
msg.msg_control = uarg;
sock->ops->sendmsg(sock, &msg, len);

/* tun_sendmsg -> tun_get_user */
if (msg.msg_control && ..)
     zerocopy_sg_from_iter(skb, from);
     sbk_shinfo(skb)->tx_flags |= SKBTX_DEV_ZEROCOPY;
     skb_shinfo(skb)->destructor_arg = uarg;
else
     skb_copy_data_from_iter(skb, from)
```

vhost-net zerocopy: callback

```
/* vhost handle_tx */
msg.iov = iov;
msg.msg_control = uarg;
sock->ops->sendmsg(sock, &msg, len);

/* tun_sendmsg -> tun_get_user */
if (msg.msg_control && ..)
     zerocopy_sg_from_iter(skb, from);
     sbk_shinfo(skb)->tx_flags |= SKBTX_DEV_ZEROCOPY;
     skb_shinfo(skb)->destructor_arg = uarg;
else
     skb_copy_data_from_iter(skb, from)


/* skb_release_data */
if (shinfo->tx_flags & SKBTX_DEV_ZEROCOPY)
     struct ubuf_info *uarg = shinfo->destructor_arg;
     uarg->callback(uarg, true);
```

vhost-net zerocopy: callback

```
/* vhost handle_tx */
msg.iov = iov;
msg.msg_control = uarg;
sock->ops->sendmsg(sock, &msg, len);

/* tun_sendmsg -> tun_get_user */
if (msg.msg_control && ..)
     zerocopy_sg_from_iter(skb, from);
     sbk_shinfo(skb)->tx_flags |= SKBTX_DEV_ZEROCOPY;
     skb_shinfo(skb)->destructor_arg = uarg;
else
     skb_copy_data_from_iter(skb, from)
     if (uarg) uarg->callback(uarg, false);

/* skb_release_data */
if (shinfo->tx_flags & SKBTX_DEV_ZEROCOPY)
     struct ubuf_info *uarg = shinfo->destructor_arg;
     uarg->callback(uarg, true);
```

MSG_ZEROCOPY

copy avoidance for sockets with remote peers

```
TCP, UDP, RAW
PF_INET, PF_INET6, PF_PACKET
```

mix copying and copy avoidance

```
send(fd, header, hlen, 0);
send(fd, payload, plen, MSG_ZEROCOPY);
```

notifications

```
SOF_TIMESTAMPING_TX_HARDWARE
SOF_TIMESTAMPING_OPT_TSONLY
SOF_TIMESTAMPING_OPT_ID
```

MSG_ZEROCOPY

```
ret = send(fd, buf, sizeof(buf), MSG_ZEROCOPY);
if (ret != sizeof(buf))
  error(1, errno, "send");
```

MSG_ZEROCOPY

```
ret = send(fd, buf, sizeof(buf), MSG_ZEROCOPY);
if (ret != sizeof(buf))
  error(1, errno, "send");
```

```
ret = recvmsg(fd, &msg, MSG_ERRQUEUE);
if (ret == -1)
  error(1, errno, "recvmsg");


read_notification(msg);
```

MSG_ZEROCOPY

```
ret = send(fd, buf, sizeof(buf), MSG_ZEROCOPY);
if (ret != sizeof(buf))
  error(1, errno, "send");


pfd.fd = fd;
pfd.events = 0;
if (poll(&pfd, 1, -1) != 1 ||
    pfd.revents & POLLERR == 0)
  error(1, errno, "poll");


ret = recvmsg(fd, &msg, MSG_ERRQUEUE);
if (ret == -1)
  error(1, errno, "recvmsg");


read_notification(msg);
```

MSG_ZEROCOPY

```
uint32_t read_notification(struct msghdr *msg)
{
  struct sock_extended_err *serr;
  struct cmsghdr *cm;

  cm = CMSG_FIRSTHDR(msg);
  if (cm->cmsg_level != SOL_IP &&
      cm->cmsg_type != IP_RECVERR)
    error(1, 0, "cmsg");

  serr = (void *) CMSG_DATA(cm);
  if (serr->ee_origin != SO_EE_ORIGIN_ZEROCOPY)
    error(1, 0, "serr");

  return serr->ee_data;
}
```

implementation: notification range

```
raw_sendmsg
     uarg = sock_zerocopy_alloc(sk, datalen);
```

```
sock_zerocopy_alloc(sk, len)
     uarg->id = atomic_inc_return(&sk->sk_zckey);
     uarg->len = 1;
     uarg->kbytelen = len >> 10;
```

implementation: notification range

```
tcp_sendmsg
     skb = tcp_write_queue_tail(sk);
     uarg = sock_zerocopy_realloc(sk, len, skb ? skb->uarg : NULL);
```

```
sock_zerocopy_realloc(sk,len, uarg)
     if (!uarg)
          return sock_zerocopy_alloc(sk, len);

     uarg->len++;
     uarg->kbytelen += len >> 10;
     atomic_inc(&sk->sk_zckey);
```

Implementation: reference counting

```
tcp_sendmsg
     while msg_data_left(msg)
          skb = sk_stream_alloc_skb

tcp_transmit_skb
     skb_clone
     ip_queue_xmit

tcp_gso_segment
     skb_segment
          *nskb_frag = *frag

dev_queue_xmit_nit
     packet_rcv
          skb_clone
```

```
refcnt = N

refcnt = N

refcnt = N * m

refcnt = N * (2 * m)
```

Implementation: notification

- allocation failure
  - at send()
  - combined alloc
- allocation budget
  - optmem
  - TCP Small Queues
- recv coalescing
  - in-order delivery

Reliability: pinned memory

- kernel
  - ulimit -l
  - do not loop onto receive path -> only to remote peers
- process
  - stop passing MSG_ZEROCOPY
  - mremap / munmap
  - disconnect: connect AF_UNSPEC
- notification byte limit
  - tcp skb append

Security: r/w memory

Avoid TOCTTOU

- always copy headers
  - MAX_HEADER
- copy on payload access
  - NETIF_F_SG && NETIF_F_..CSUM
  - skb_orphan_frags
    - skb_checksum_help
    - ipsec
    - bpf, u32
- copy to hardware

Evaluation

```
NETPERF=./netperf -t TCP_STREAM -H $host -T 2 -l 30 -- -m $size

perf stat -e cycles $NETPERF
perf stat -C 2,3 -a -e cycles $NETPERF
```

	--process cycles--			----cpu cycles----		
	std	zc	%	std	zc	%
4K	27,609	11,217	41	49,217	39,175	79
16K	21,370	3,823	18	43,540	29,213	67
64K	20,557	2,312	11	42,189	26,910	64
256K	21,110	2,134	10	43,006	27,104	63
1M	20,987	1,610	8	42,759	25,931	61

Evaluation

```
perf record netperf -t TCP_STREAM -H $host

Samples: 42K of event 'cycles', Event count (approx.): 21258597313
 79.41%         33884  netperf  [kernel.kallsyms]  [k] copy_user_generic_string
  3.27%          1396  netperf  [kernel.kallsyms]  [k] tcp_sendmsg
  1.66%           694  netperf  [kernel.kallsyms]  [k] get_page_from_freelist
  0.79%           325  netperf  [kernel.kallsyms]  [k] tcp_ack
  0.43%           188  netperf  [kernel.kallsyms]  [k] __alloc_skb
```

Evaluation

```
perf record netperf -t TCP_STREAM -H $host

Samples: 42K of event 'cycles', Event count (approx.): 21258597313
Samples:  1K of event 'cycles', Event count (approx.):  1439509124
 30.36%           584  netperf  [kernel.kallsyms]  [k] gup_pte_range
 14.63%           284  netperf  [kernel.kallsyms]  [k] __zerocopy_sg_from_iter
  8.03%           159  netperf  [kernel.kallsyms]  [k] skb_zerocopy_add_frags_iter
  4.84%            96  netperf  [kernel.kallsyms]  [k] __alloc_skb
  3.10%            60  netperf  [kernel.kallsyms]  [k] kmem_cache_alloc_node
```

Evaluation

Tensorflow

BM_RPC(2B):	+15% wall clock
BM_RPC(98KB):	-23%
Mixed workload:	-7.5%
CDN:	+7% qps
Storage:	+2% qps

Questions

MSG_ZEROCOPY

- copy avoidance for sockets
- TCP, UDP, RAW and packet
- notifications over MSG_ERRQUEUE

Latest patchset at

- https://lwn.net/Articles/715279/
- https://github.com/wdebruij/linux/tree/zerocopy-sendmsg-rfcv2

Evaluation

```
snd_zerocopy_lo

-t       rx=437082   (27275 MB)   tx=437082  txc=0
-t -z    rx=1028184  (64163 MB)   tx=1028184 txc=1028156

-u       rx=756588   (47214 MB)   tx=756588  txc=0
-u -z    rx=1628376 (101618 MB)   tx=1628376 txc=1628338

-r       rx=429126   (26779 MB)   tx=429126  txc=0
-r -z    rx=1408782  (87914 MB)   tx=1408782 txc=1408744
```