# Map Batch Processing

Yonghong Song

# Use Case

- Potential performance gain.
- Performance improvement of batched lookup/lookup_and_delete vs. bpf_map_get_next_key() approach, batching 100 keys saving more than 70% wallclock time based on synthetic benchmark.
- Typical use cases:
  - Batched lookup
  - Batched lookup and delete
  - Delete all elements in a map

# Two Approaches

- Simple batch processing, no special treatment for any particular key
  - For example, if user says to lookup the first 100 elements, then the first 100 elements will be returned to user.
- Programmable (via secondary BPF program) batch processing
  - A filter bpf program runs through the key/value pair to decide what to do.
  - A dumper bpf program runs through every to-be-deleted key/value pair.
- Both approaches need to ensure
  - Avoid the issue of current bpf_map_get_next_key() when the prev_key is not in the hashmap.
  - Minimizing/no duplication in lookup
  - No omission in deletion w.r.t. the map state when the operation starts

# Bucket Based Iteration

- Suggested by Alexei
- User space won’t iterate through keys, but through a opaque batch id
- This should work since we do not do rehashing after map creation.

```
For (batch = 0; batch < htab->n_buckets; batch++) {
    b = &htab->buckets[batch];
    raw_spin_lock_irqsave(&b->lock, flags);
    /* check number of elements in this bucket */
    If (cannot perform operation, e.g., no enough space */) {
        raw_spin_unlock_irqrestore(&b->lock, flags);
        break;
    }
    /* do lookup/delete/update of elements, dump to user */
   raw_spin_unlock_irqrestore(&b->lock, flags);
}
```

# Simple Batch Processing UAPI

```
struct {
        __u64 batch; /* input/output */
        __aligned_64 keys;
        __aligned_64 values;
        __u32 count; /* input/output */
        __u32 map_fd;
        __u64 elem_flags;
        __u64 flags;
}
```

Lookup
Lookup_and_delete
Update
Delete

# Programmable Batch Processing

```
/* From Jakub */
LIST_HEAD(deleted);
for entry in map {
        struct map_op_ctx {
                .key    = entry->key,
                .value= entry->value,
        };
        act = BPF_PROG_RUN(filter, &map_op_ctx);
        if (act & ~ACT_BITS)
                return -EINVAL;

        if (act & DELETE) {
                map_unlink(entry);
                list_add(entry, &deleted);
        }
        if (act & STOP)
                break;
}
synchronize_rcu();
```

```
/* continued */
for entry in deleted {
        struct map_op_ctx {
                .key    = entry->key,
                .value = entry->value,
        };

        BPF_PROG_RUN(dumper, &map_op_ctx);
        map_free(entry);
}
```