THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
EUROPE

Implementing PID allocation with the IDR API

Gargi Sharma, *Outreachy Intern*

THE LINUX FOUNDATION

Process IDs

- Every process has a unique identifier that represents it, called the process ID (pid).
  - The first process that the kernel runs is called the idle task and has the pid 0.
  - The first process that runs after booting is called the init process and has the pid 1.

PID Namespaces

- A key point in understanding PIDs is to understand their use in namespaces.
  - –PID Namespaces isolate the PID number space.
    - allow containers to suspend/resume processes in the container.
    - migrate to a new host while maintaining the same PIDs.
  - –Hierarchically nested in parent child relationship.
  - –Process has a different PID in each layer.

How were process IDs allocated?

- Each namespace has an associated bitmap.
- alloc_pid allocates the PIDs serially.
  - alloc_pid searches the bitmap for the last allocated PID and allocate PID sequentially.
  - If PID reaches the maximum limit, assignment wraps around.

PID lookup and deletion

- To make the process of looking up PIDs faster, PIDs are added to a hashlist.
  - Iterate over the hashlist to find the PID that is being looked for.
- Iterate through all the namespaces where the PID is visible and free it in each namespace.
  - The PID is also deleted from the hashlist (used for lookup).

Replacing the bitmap implementation with the IDR API

IDR API

- IDR: a generic mechanism to associate an integer with a pointer.
- Internal implementation done using a radix tree
  - convenient to associate an integer and pointer.
  - high search efficiency.

Why use the IDR API?

- Simplify the kernel code.
  - Replace custom code with a generic API.
- Reduce the kernel size.
- Make PID allocation faster.
  - IDR API has an underlying Radix tree implementation, hence is faster than a bitmap + hashlist (used for lookup).

Kernel size - Before and After

- pid_namespace.o

	text	data	bss	dec	hex
Before	5692	1842	192	7726	1e2e
After	2854	216	16	3086	c0e

- 60.05% decrease.

Kernel size - Before and After

- pid.o

	text	data	bss	dec	hex
Before	8447	3894	64	12405	3075
After	3397	304	0	3701	e75

- 70.16% decrease.

Performance - Before and After

- ps with 10,000 processes

	With IDR API	With bitmap
User	0m0.052s	0m0.060s
Sys	0m0.392s	0m0.516s
User+Sys	0m0.444s	0m0.576s

- 22.92% faster than bitmap implementation.

Performance - Before and After

- pstree with 10,000 processes

	With IDR API	With bitmap
User	0m0.536s	0m0.612s
Sys	0m0.184s	0m0.264s
User+Sys	0m0.720s	0m0.876s

- 17.81% faster than bitmap implementation.

Performance - Before and After

- Calling readdir on /proc with 10,000 processes

	With IDR API	With bitmap
User	0m0.004s	0m0.004s
Sys	0m0.012s	0m0.016s
User+Sys	0m0.016s	0m0.020s

- 20.00% faster than bitmap implementation.

IDR API interface

- idr_alloc{_cyclic}(struct idr *idp, void *ptr,

  int start, int end, gfp_t gfp_mask)
- idr_remove(struct idr *idp, int id)
- idr_find(struct idr *idp, int id)
- idr_replace(struct idr *idp, void *ptr, int id)
- idr_destroy(struct idr *idp)

Allocation using the IDR API

- Associate an IDR structure with each namespace.
- Call idr_alloc_cyclic(idr, NULL, pid_min, pid_max, GFP_ATOMIC) followed by a call to idr_replace(idr, pid, nr).
- idr_replace() is called so that find_pid_ns() does not find a non initialised pid.

Lookup & deletion using the IDR API

- Lookup: idr_find(idr, nr)
- Deletion: idr_remove(idr, nr).
- To destroy a namespace, each of the individual pages in the bitmap had to be freed.
  - Replaced with a call to idr_destroy(struct *idr).

Simplification of the kernel code

Before

```
struct pid
*find_ge_pid(int nr, struct pid_namespace *ns)
 {
       struct pid *pid;
       do {
              pid = find_pid_ns(nr, ns);
              if (pid)
                     break;
              nr = next_pidmap(ns, nr);
       } while (nr > 0);
       return pid;
 }
```

After

```
struct pid
*find_ge_pid(int nr, struct pid_namespace *ns)
 {
       return idr_get_next(&ns->idr, &nr);
 }
```

Experience as an Outreachy intern

- Status: Patches applied to Andrew Morton’s -mm tree.
- By far the most exciting thing I have done as a software engineer!
- Had great mentors who were always there. Thank you, Rik and Julia!
- Learnt more about operating systems, version control, etc
- Became friends with really cool former interns!
- Read more about my internship at:
  - medium.com/@gargi_sharma

THE LINUX FOUNDATION

OPEN SOURCE SUMMIT

EUROPE

Děkuji! :)

THE LINUX FOUNDATION